Bullsh*t Jobs

01 April 2023

development management newsletter startups teams

Doing meaningless work is not a new concept. After automation started to get wider adoption in the second part of the 20th century, people were eagerly waiting to work for only 15 hours a week. What happened instead was that people continued working the same amount of hours doing absolutely meaningless work. This is how we arrived at the concept of a bullshit job.

> *“a form of paid employment that is so completely pointless, unnecessary, or pernicious that even the employee cannot justify its existence even though, as part of the conditions of employment, the employee feels obliged to pretend that this is not the case.”*
>
> *The Book Bullshit Jobs*

There's a whole book on this phenomenon. Graeber argues that half of the jobs today are meaningless, and it's highly destructive when paired with people who associate work with self-worth.

For example, in retail and hospitality, it's popular not to allow your employees to sit. If you're sitting means you're not doing any work, so you need to find work for the work's sake only — looking busy is more important than doing work that matters. It's a subset of meaningless work — doing something without purpose to look busy to others.

The software industry is not immune to this, especially in the big corporate world. I've already written about how Big Corps embraces the cycle of self-promotion via the loop: kickoff -> release -> get promotion -> move on to the new product. I call this the bullshit product loop, where products get released only for the sake of internal promotions — this goes hand in hand with meaningless jobs because while it will bring value to the one getting a promotion (more $$$), it will serve no purpose for other team members and the end users. The product will be forgotten and discontinued eventually, and somebody will be held accountable for its demise — which will not be the person who started it, as they will be long gone from the project.

But there are more scenarios where meaningless work thrives.

The rigid culture

Corporate culture, or in other words, “rigid” culture, is a type of working environment where you follow along the dotted line, and any deviation from that line is considered bad. The dotted line guarantees a specific, though quite low, output level across all teams/departments and can be easily measured in good-looking reports.

A good example of rigid culture is when a company switches to agile development and follows the methodology like a dogma. They hire consultants, coaches, and sprint gurus to make all the teams adhere to sprint-based development. Which, in the end, creates even more rigidness:

- Someone new joins during the sprint. Too bad, sit down and do nothing until the next sprint starts or “go ask Bob.”
- Someone finishes their tasks before the end of the sprint — too bad, sit down and do nothing until the next sprint or “support your teammates.”
- None of the devs have any blockers and have already discussed collaboration in chat yesterday evening — too bad you got daily standups to tell your manager what you’ve been doing.
- Something is not clear? Let’s have a 15-people 2-hour zoom call to discuss the vision again and see what everyone has to say about it.

Rigid culture means more meta work than real work — work for the sake of appearances rather than the output. The best companies I’ve seen who use agile development the right way are those that take the best parts of the SCRUM methodology, grind them into fitting their existing flows, and become more output-oriented rather than process-oriented.

Zombie Projects

Another great example of a bullshit job is the so-called “zombie projects.” It’s a term used to describe projects that are still ongoing within a company, despite having zero value to the company's current objectives or goals. It’s one of those projects an ambitious manager has kicked off and has gotten approved, but it was never really finished. Teams have changed multiple times, documentation is in the heads of the previous devs, and the project is alive because no manager wants to be the one who “fails” this project.

There are several reasons why zombie projects happen:

When a company is large and complex, it can be difficult for leaders to clearly understand all ongoing projects and their respective contributions to the company's overall strategy. As a result, projects that are no longer useful can continue to receive funding and resources, sustained by bureaucracy, inertia, and/or politics rather than by actual business needs or opportunities.

Another reason is the fear of failure. Some employees hesitate to report that their work is bullshit and brings no value. They may be concerned about potential consequences or negative perceptions of their abilities. And this leads to a culture of complacency — employees see that resources are wasted on projects that do not contribute to the company's success. They become less motivated to work hard or to suggest new ideas or improvements.

The next reason is that people are comfortable doing bullshit jobs. They know they are working on a zombie project, and they're OK with it. There's no stress and no deadlines. You rewrite the code from Python to Rust, add some new frameworks for the 1000th time, increase your skill, and go home to your family. The only fear is when this will all come crumbling down, but in a company with tens of thousands of employees, it can take many years for the house of cards to go down.

Doomed bloatedness

Here's a cool story I've read on HN recently.

> *"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."*
>
> *source*

This story will probably resonate with you. As a Software Developer, you might've been in the same shoes — a task is brought up, an estimation is asked, and a number is presented. The goal with these estimates is to not shoot yourself in the foot because if you can't deliver on your promise, the whole project plan will go haywire. But on the other hand, you don’t want to overestimate, as then you’ll end up doing nothing for some days, especially if you’re in a big investment bank, like the story above.

In a small startup or a company with healthy development culture, if you finish a task that you overestimated early, you can grab the next one and help your team deliver on the sprint goals. Everyone is happy, you’re not getting bored with doing nothing, the team moves forward toward project completion, and the management moves forward to profitability (hopefully).

The problem becomes apparent when you realize that not everyone is as invested in completing the project as you are. You might be new to the company, eager to move forward, and filled with ideas and solutions. For some people, especially those who’ve been with the company for the past 10-20 years, it’s a cushion job, and they don’t want things completed. They want to do the least amount of work in the maximum amount of time.

Happens more often than you think.

The problem becomes even more profound if you realize that the majority of the people around you are like that — that means you ended up in a company with a dying development culture. If all you see around you is throttling, you need to get out of there. If people around you tell you to “take it easy”, or “slow down, cowboy” — you better run.

I'm not saying you should be running at 100% of your capacity constantly, but if you're running way below what your potential is, why bother?

Doomed projects

> *“It was a big, multi-year project. I was hired on as an individual contributor maybe 15 months before the project was to be delivered to the customer, and it took me awhile to get my bearings. But after a few months, it really seemed to me that (a) we were not moving along as fast as the project manager Gantt charts said we had to, (b) we were doing a lot of tasks that weren’t even on the Gantt charts, and (c) we hadn’t even received all of the specifications from the customer that we would need to complete the project. Bad estimates for the*

> *work we had, lots of work that hadn’t been estimated, and incomplete requirements – the project sure seemed doomed to me.”*
>
> *source*

The big issue with software projects that are planned for X number of years is that they are never completed in X number of years. It first starts with incomplete requirements that are changed during the X number of years that the project lives. The project becomes a perpetual machine of requirements changing, developer and manager turnover, revamping of the Gantt charts, and refactoring the code numerous times. It becomes obsolete before it ever gets to production.

> *“I had recently started a job where my new employer had been working with a contractor on their “next-gen” web application for four years, and it was nowhere near production ready. Two weeks into the new gig, I looked at my boss and told him that it was a failed project and needed to be canned. I got the “are you kidding me, we’ve already got millions tied up in this project.” Fast forward two years went by, and it’s still not production ready.”*
>
> *source*

As a developer, there's nothing more infuriating than your code/ideas/algorithms not being used or being scrapped even before it gets to production.

Building the next big thing

Yes. This is also a bullshit job. Once something new and shiny popups up, a team is assembled to apply it everywhere. Recent examples in the last decade include big data, blockchain, and chatGPT. I bet there's a team at every corporation currently trying to integrate GPT into everything — from accounting to report generation.

The cycle continues — it was the same when the "Big Data" was the hype. Everyone wanted to create huge Hadoop data clusters to analyze it. After that, it was Blockchain — lets add blockchain everywhere.

Especially when there's VC money involved, the more hyped the topic is, the more inclined startups are to include "We're a Tinder for X with big data, blockchain, and chatGPT" in their Pitch decks.

> *“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.”*
>
> *source*

Don't get me wrong; it’s a great way to sharpen your skills as a developer — you get to build something with bleeding tech. You spend time fiddling with the latest technologies and building prototypes and POC. But at the end of the day, if that never sees the light of day, was it really fulfilling?

Conclusion

The amount of time that we work is still five days, 40 hours per week. The advancement in technology in the last twenty years has created pressure for people to stay productive and fill in those hours with busy work — sometimes it’s creative work, sometimes it’s projects with no purpose.

I don’t have a solution for eliminating meaningless work, nor do I think we need one. I would, though, propose small steps towards transparency and goal alignment — not every job has to be building rockets or saving people’s lives — but every job needs to have the proper people working them.

Other Newsletter Issues:

- Contracts you should never sign
- ⛓️ Implementing Atomic Habits in IT
- Software Development is very subjective

Your Name

Write a comment...

✏ Add comment

117+ people joined this week

Get actionable Insights from a CTO into your inbox every week.

At my weekly (sometimes bi-weekly) newsletter - I share my thoughts as a CTO, which some may consider insightful and practical startup advice. Some of the topics that I like writing about: building SaaS products, growing teams, scaling technology and in general being a good founder and a decent person.

💌 Go to Newsletter

LATEST

🚪 Exit. Selling your SaaS

🚦 Going live with your SaaS: The Launch day

🤑 Fundraising for SaaS Startups

🧬 Becoming a Real Business: Accounting, Taxes, and Automation

🍻 How to build a community around your SaaS

MOST POPULAR

📜 Contracts you should never sign

🥷 Things they didn’t teach you about Software Engineering

📢 The silent majority

🎓 Product Owner vs Project Managers

⚖ Build vs Buy: age old dilemma

⛓ Implementing Atomic Habits in IT

🪪 Interviews-as-a-Service: The Bad and The Ugly

Popular Tags

Long-reads
SaaS
Management
Development
Startups
