Instapaper

October 23, 2014

Why You Can’t Work at Work | Jason Fried | Big Think
bigthink.com

Why You Can’t Work at Work | Jason Fried | Big Think

What is **Big Think?**

We are Big Idea Hunters…

We live in a time of information abundance, which far too many of us see as information overload. With the sum total of human knowledge, past and present, at our fingertips, we’re faced with a crisis of attention: which ideas should we engage with, and why? Big Think is an evolving roadmap to the best thinking on the planet — the ideas that can help you think flexibly and act decisively in a multivariate world.

A word about Big Ideas and Themes — The architecture of Big Think

Big ideas are lenses for envisioning the future. Every article and video on bigthink.com and on our learning platforms is based on an emerging “big idea” that is significant, widely relevant, and actionable. We’re sifting the noise for the questions and insights that have the power to change all of our lives, for decades to come. For example, reverse-engineering is a big idea in that the concept is increasingly useful across multiple disciplines, from education to nanotechnology.

Themes are the seven broad umbrellas under which we organize the hundreds of big ideas that populate Big Think. They include *New World Order*, *Earth and Beyond*, *21st Century Living*, *Going Mental*, *Extreme Biology*, *Power and Influence*, and *Inventing the Future*.

Big Think Features:

12,000+ Expert Videos

Browse videos featuring experts across a wide range of disciplines, from personal health to business leadership to neuroscience.

Watch videos

World Renowned Bloggers

Big Think’s contributors offer expert analysis of the big ideas behind the news.

Go to blogs

Big Think Edge

Big Think’s Edge learning platform for career mentorship and professional development provides engaging and actionable courses delivered by the people who are shaping our future.

Find out more

Close

With rendition switcher

Transcript

Question: *What is your take on the typical workplace*?

Jason Fried: Yeah, my feeling is that the modern workplace is structured completely wrong. It's really optimized for interruptions. And interruptions are the enemy of work. They are the enemy of productivity, they are the enemy of creativity, they are the enemy of everything. But that's what the modern workplace is all about, it's interruptions. Everyone's calling meetings all the time, everyone's screaming people's names across the thing, there's phones ringing all the time. People are walking around. It's all about interruptions. And people go to work today, and then they end up doing most of their real work after work, or on the weekends. So, people are working longer hours, people are tired – I'm working 50-60 hours this week. It's not that there's 50 or 60 hours worth of work to do, it's because you don't work at work anymore. You go to work to get interrupted.

What happens is, is that you show up at work and you sit down and you don't just immediately begin working, like you have to roll into work. You have to sort of get into a zone, just like you don't just go to sleep, like you lay down and you go to sleep. You go to work too. But then you know, 45 minutes in, there's a meeting. And so, now you don't have a work day anymore, you have like this work moment that was only 45 minutes. And it's not really 45 minutes, it's more like 20 minutes, because it takes some time to get into it and then you've got to get out of it and you've got to go to a meeting.

Then when the meeting's over, you're probably pissed off anyway because it was a waste of time and then the meeting's over and you don't just go right back to work again, you got to kind of slowly get back into work. And then there's a conference call, and then someone calls your name, "Hey, come a check this out. Come over here." And like before you know it, it's 4:00 and you've got nothing done today. And this is what's happening all over corporate America right now. Everybody I know, I don't care what business they're in. Like when I talk to them about this, it's like "Yeah, that's my life." Like, that is my life, and it's wrong.

And so I think that has to change. If people want to get things done, they've got to get rid of interruptions. And so I think that's something we're focused on, is trying to remove every possible interruption from people's day. So they have longer and longer periods of uninterrupted time to actually get work done. And so, our whole workplace, whatever the word you want to use, the office, workplace, although we're kind of virtual anyways; it's structured around removing interruptions. And one of the best ways you can do this is to shift your collaboration between people to more passive things. Using our products or someone else's products. Things that you can put aside when I'm busy. So, if I'm busy, I don't have to look at Base Camp, I don't have to check email, I don't have to check IM. I can put those things aside and do my work. And then when I'm done with my work and I need a break, I can go check these things out.

But if someone's calling my name, or tapping on my shoulder, or knocking on my door, I can't ignore those things. I can quit a program, but I can't quit someone knocking on my door. I can't quit someone calling my name, or someone ringing me on the phone. So, we try and, even though we might be sitting right across from each other, we don't talk to each other, hardly at all during the day. Even though we're right there, we'll use instant messaging, or email, and if someone doesn't respond, it means they're busy. And they probably put that window away. Instead of calling, "Hey Jason, Jason, Jason" until they respond, that's interrupting somebody; that doesn't work and that's how most workplaces are.

And managers are the biggest problem because their whole world is built around interruption. That's what they do. Management means interrupting. Hey, what's going on? How's this going? Let me call a meeting because that's what I do all day, I call meetings. And so, managers are the real problems here and that's got to change too. So, as managers of our company, we don't really manage people, but we prefer people to be managers of one. Let them just figure things out on their own, and if they need our help, they can ask us for it instead of us always constantly asking them if they need help and getting in their way. So, we're all about getting rid of interruptions. And I think that if companies were more focused on getting rid of interruptions, they would get a whole lot more work done.

Question: *How does your company avoid these distractions*?

Jason Fried: So, this isn't really a plug, but we use our product called Campfire, which is a real time chat tool. That is our office. Campfire is our office, and that's a web based chat tool where there's a persistent chat room open all the time. Anyone who has a question for anyone else in the company posts it there and in real time, everyone else can see it if they're looking at it. But if they're busy, they just don't pay attention. And then if non one responds, then that means someone is busy. Not like, I'm going to keep calling their name until they turn around. That's what it's like in most offices. Or you ring someone and they're not there and so you call their name, and they're not there, so you go to their office and you bang on their door. If someone doesn't respond in Campfire, it means they're busy. And unless it's a true emergency, where you really need an answer right now, then you just let them be and they'll get back to you in three hours. And the truth of the matter is, there are almost no true emergencies in business. Everything can wait a few hours. Everything can wait a day. It's not a big deal if you get back to me later in the day for me to know right now.

And the other thing about interruptions and calling people's names, and ringing them on the phone and stuff, it's actually really an arrogant sort of move because you're saying that whatever I have to ask you is more important than what you're doing. Because I'm going to stop you from doing what you are doing for me to ask you this questions that probably doesn't matter anyway. So, we're very cognizant of this, and we make sure that we only ping people, that's what we call it, digitally and in ways that will not really get in their way if they're really busy.

And that's not always the case, but that's really what we try to do. And use Campfire and use Base Camp and use High Rise and all our products. Other people's products this well as well, but we just use our own because we built them for ourselves and we use them and they're free for us.

Question: *Does your office have a hierarchy*?

Jason Fried: Yeah. So, we don't really have hierarchy, technically. I mean, ultimately the buck stops with me, but like it doesn't get to that. We really let people make their own decisions and we give them feedback on those decisions and help them learn and make better decisions. And we have some small teams. People work in teams of three, but there are really no true leader in those teams necessarily. It's like, the leader is the product. Like the product is what leads you. It's got to be good. Quality is the leader and everyone has to understand that that's what this is all about. We're making good products here. We're not making your idea, or my idea, we're making a product that useful for our customers. So, that's kind of what guides everything. And it's surprisingly works pretty well.

We have like, big visions for things, and we all share common points of view on like what's important, but ultimately it's quality, it's the product, it's usefulness, it's clarity. Those are the things that lead us on the right direction.

More from the Big Idea for Saturday, November 30 2013

Rethinking the Workplace

More than one-third of the population will be working online by the year 2020. This is what Gary Swart, CEO of the world's largest online workplace, oDesk, tells Big Think in today's lesson. Wh… Read More…

Why You Can't Work at Work

by Jason Fried
February 3, 2010, 12:04 AM
With its constant commotion, unnecessary meetings, and infuriating wastes of time, the modern workplace makes us all work longer, less focused hours. Jason Fried explains how we can change all of

this.

Maker's Schedule, Manager's Schedule

Paul Graham

Want to start a startup? Get funded by Y Combinator.

July 2009

One reason programmers dislike meetings so much is that they're on a different type of schedule from other people. Meetings cost them more.

There are two types of schedule, which I'll call the manager's schedule and the maker's schedule. The manager's schedule is for bosses. It's embodied in the traditional appointment book, with each day cut into one hour intervals. You can block off several hours for a single task if you need to, but by default you change what you're doing every hour.

When you use time that way, it's merely a practical problem to meet with someone. Find an open slot in your schedule, book them, and you're done.

Most powerful people are on the manager's schedule. It's the schedule of command. But there's another way of using time that's common among people who make things, like programmers and writers. They generally prefer to use time in units of half a day at least. You can't write or program well in units of an hour. That's barely enough time to get started.

When you're operating on the maker's schedule, meetings are a disaster. A single meeting can blow a whole afternoon, by breaking it into two pieces each too small to do anything hard in. Plus you have to remember to go to the meeting. That's no problem for someone on the manager's schedule. There's always something coming on the next hour; the only question is what. But when someone on the maker's schedule has a meeting, they have to think about it.

For someone on the maker's schedule, having a meeting is like throwing an exception. It doesn't merely cause you to switch from one task to another; it changes the mode in which you work.

I find one meeting can sometimes affect a whole day. A meeting commonly blows at least half a day, by breaking up a morning or afternoon. But in addition there's sometimes a cascading effect. If I know the afternoon is going to be broken up, I'm slightly less likely to start something ambitious in the morning. I know this may sound oversensitive, but if you're a maker, think of your own case. Don't your spirits rise at the thought of having an entire day free to work, with no appointments at all? Well, that means your spirits are correspondingly depressed when you don't. And ambitious projects are by definition close to the limits of your capacity. A small decrease in morale is enough to kill them off.

Each type of schedule works fine by itself. Problems arise when they meet. Since most powerful people operate on the manager's schedule, they're in a position to make everyone resonate at their frequency if they want to. But the smarter ones restrain themselves, if they know that some of the people working for them need long chunks of time to work in.

Our case is an unusual one. Nearly all investors, including all VCs I know, operate on the manager's schedule. But Y Combinator runs on the maker's schedule. Rtm and Trevor and I do because we always have, and Jessica does too, mostly, because she's gotten into sync with us.

I wouldn't be surprised if there start to be more companies like us. I suspect founders may increasingly be able to resist, or at least postpone, turning into managers, just as a few decades ago they started to be able to resist switching from jeans to suits.

How do we manage to advise so many startups on the maker's schedule? By using the classic device for simulating the manager's schedule within the maker's: office hours. Several times a week I set aside a chunk of time to meet founders we've funded. These chunks of time are at the end of my working day, and I wrote a signup program that ensures all the appointments within a given set of office hours are clustered at the end. Because they come at the end of my day these meetings are never an interruption. (Unless their working day ends at the same time as mine, the meeting presumably interrupts theirs, but since they made the appointment it must be worth it to them.) During busy periods, office hours sometimes get long enough that they compress the day, but they never interrupt it.

When we were working on our own startup, back in the 90s, I evolved another trick for partitioning the day. I used to program from dinner till about 3 am every day, because at night no one could interrupt me. Then I'd sleep till about 11 am, and come in and work until dinner on what I called "business stuff." I never thought of it in these terms, but in effect I had two workdays each day, one on the manager's schedule and one on the maker's.

When you're operating on the manager's schedule you can do something you'd never want to do on the maker's: you can have speculative meetings. You can meet someone just to get to know one another. If you have an empty slot in your schedule, why not? Maybe it will turn out you can help one another in some way.

Business people in Silicon Valley (and the whole world, for that matter) have speculative meetings all the time. They're effectively free if you're on the manager's schedule. They're so common that there's distinctive language for proposing them: saying that you want to "grab coffee," for example.

Speculative meetings are terribly costly if you're on the maker's schedule, though. Which puts us in something of a bind. Everyone assumes that, like other investors, we run on the manager's schedule. So they introduce us to someone they think we ought to meet, or send us an email proposing we grab coffee. At this point we have two options, neither of them good: we can meet with them, and lose half a day's work; or we can try to avoid meeting them, and probably offend them.

Till recently we weren't clear in our own minds about the source of the problem. We just took it for granted that we had to either blow our schedules or offend people. But now that I've realized what's going on, perhaps there's a third option: to write something explaining the two types of schedule. Maybe eventually, if the conflict between the manager's schedule and the maker's schedule starts to be more widely understood, it will become less of a problem.

Those of us on the maker's schedule are willing to compromise. We know we have to have some number of meetings. All we ask from those on the manager's schedule is that they understand the cost.

Thanks to Sam Altman, Trevor Blackwell, Paul Buchheit, Jessica Livingston, and Robert Morris for reading drafts of this.

Related:

Programmer Interrupted

blog.ninlabs.com

2013-01-19

I'm writing this post in an apt state: low-sleep, busy, disorientated, and interrupted. I try all the remedies: Pomodoro, working in coffee shops, headphones, and avoiding work until being distraction free in the late night.

But it is only so long before interruption finds a way to pierce my protective bubble. Like you, *I am programmer, interrupted*. Unfortunately, our understanding of interruption and remedies for them are not too far from homeopathic cures and bloodletting leeches.

But what is the evidence and what can we do about it? Every few months I still see programmers who are asked to not use headphones during work hours or are interrupted by meetings too frequently but have little defense against these claims. I also fear our declining ability to handle these mental workloads and interruptions as we age.

The costs of interruptions have been studied in office environments. An interrupted task is estimated to take twice as long and contain twice as many errors as uninterrupted tasks (Czerwinski:04). Workers have to work in a fragmented state as 57% of tasks are interrupted (Mark:05).

For programmers, there is less evidence of the effects and prevalence of interruptions. Typically, the number that gets tossed around for getting back into the "zone" is at least 15 minutes after an interruption. Interviews with programmers produce a similiar number (vanSolingen:98). Nevertheless, numerous figures have weighed in: Paul Graham stresses the differences between a maker's schedule and manager's schedule. Jason Fried says the office is where we go to get interrupted.

Interruptions of Programmers

Based on a analysis of 10,000 programming sessions recorded from 86 programmers using Eclipse and Visual Studio and a survey of 414 programmers (Parnin:10), we found:

- A programmer takes between 10-15 minutes to start editing code after resuming work from an interruption.
- When interrupted during an edit of a method, only 10% of times did a programmer resume work in less than a minute.
- **A programmer is likely to get just one uninterrupted 2-hour session in a day**

We also looked at some of the ways programmers coped with interruption:

- Most sessions programmers navigated to several locations to rebuild context before resuming an edit.
- Programmers insert intentional compile errors to force a "roadblock" reminder.
- A source diff is seen as a last resort way to recover state but can be cumbersome to review

Worst Time to Interrupt a Programmer

Research shows that the worst time to interrupt anyone is when they have the highest memory load. Using neural correlates for memory load, such as pupillometry, studies have shown that interruptions during peak loads cause the biggest disruption(Iqbal:04).

We looked at subvocal utterances during a programming tasks to find different levels of memory load during programming tasks (Parnin:11).

If an interrupted person is allowed to suspend their working state or reach a "good breakpoint", then the impact of the interruption can be reduced (Trafton:03). However, programmers often need at least 7 minutes before they transition from a high memory state to low memory state (Iqbal:07). An experiment evaluating which state a programmer less desired an interruption found these states to be especially problematic (Fogarty:05):

- During an edit, especially with concurrent edits in multiple locations.
- Navigation and search activities.
- Comprehending data flow and control flow in code.
- IDE window is out of focus.

A Memory-Supported Programming Environment

Ultimately, we cannot eliminate interruptions. In some cases interruption may even be beneficial. But we can find ways to reduce the impact on the memory failures that often result from interruption. I introduce some types of memory that get disrupted or heavily burdened during programming and some conceptual aids that can support them.

Prospective Memory

Prospective memory holds reminders to perform future actions in specific circumstances *e.g.* to buy milk on the way home from work (PM).

Various studies have described how developers have tried to cope with prospective memory failures. For example, developers often leave TODO comments in the code they are working on (Storey:08). A drawback of this mechanism is that there is no impetus for viewing these reminders. Instead, to force a prospective prompt, developers may intentionally leave a compile error to ensure they remember to perform a task (Parnin:10). A problem with compile errors is that they inhibit the ability to switch to another task on the same codebase. Finally, developers also do what other office workers do: leave sticky notes and emails to themselves (Parnin:10).

A **smart reminder** is reminder that can be triggered based on conditions such as *a teammate checking in code*, or *proximity to a reminder*. To support prospective memory, I've created a Chrome extension called `pm` that lets you place smart reminders on specific urls, and a Visual Studio extension called `attachables`, that lets you attach TODO notes as smart reminders in your code editors. Check it out!

Attentive Memory

Attentive memory holds conscious memories that can be freely attended to.

Some programming tasks require developers to make similar changes across a codebase. For example, if a developer needs to refactor code in order to move a component from one location to another or to update the code to use a new version of an API, then that developer needs to systematically and carefully edit all those locations affected by the desired change. Unfortunately, even a simple change can lead to many complications, requiring the developer to track the status of many locations in the code. Even worse, after an interruption to such as task, the tracked statuses in attentive memory quickly evaporate and the numerous visited and edited locations confound retrieval.

Touch points allow a programmer to track status across many locations in code.

Associative Memory

Associative memory holds a set of non-conscious links between manifestations of co-occurring stimuli.

Developers commonly experience disorientation when navigating to unfamiliar code. The disorientation stems from associative memory failures that arise when a developer must recall information about the places of code they are viewing or what to view next. Researchers believe the lack of rich and stable environmental cues in interface elements, such as document tabs, prevent associative memories from being recalled.

The presence of multiply modalities in a stimulus increases the ability to form an associative memory. In this sense, a *modality* refers to distinct type of perceptions that is processed by a distinct regions of the brain, such as auditory or visual pathways. Examples of different modalities include: lexical, spatial, operational, and structural. When multiple modalities are present in the same stimulus, more pathways are activated, thus increasing the chance of forming an associative memory. In contrast, a monotonous stimulus with a single modality is less likely to form an associative memory.

An **associative link** helps a programmer by situating information of multiple modalities with a program element. In particular, by improving navigating document tabs, which default configuration are especially spartan, often just showing the name of the document.

Episodic Memory

Episodic memory is the recollection of past events.

Software developers continually encounter new learning experiences about their craft. Retaining and making use of those such acquired knowledge requires that developers are able to recollect those experiences from their episodic memory. When recalling from episodic memory, developers commonly experience failures that limit their ability to recall essential details or recollect the key events. For example, a developer may forget the changes they performed for a programming task, or forget details such as a the blog post that was used for implementing a part of the task.

A **code narrative** is an episodic memory aid that helps a developer recall contextual details and the history of programming activity. Two narrative structures are currently supported: A review mode for high-level recall of events and a share mode for publishing a coding task for others.

To support code narratives, I've created `autogit`, which will take snapshots of your code everytime you save in Sublime Text or Visual Studio. `Automark` does an analysis of the local history recorded by autogit and generates a narrative summary of your recent coding history. Try it out!

Also related, `Docsight` is a Chrome extension that lets you see developer web resources you've visited in the past, in an episodic manner.

Conceptual Memory

Conceptual memory is a continuum between perceptions and abstractions. How does the brain remember objects such as a hammer and concepts such as *tool*? The brain first learns basic features of encountered stimuli such as the wood grains and metal curves of a hammer and then organizes those features into progressively higher levels of abstraction.

Developers are expected to maintain expertise in their craft throughout their careers. Unfortunately, the path to becoming an expert is not easily walked: For a novice, evidence suggests this can be a 10 year journey (Chi:82). Furthermore, for experts trying to become experts in new domains, like the desktop developer becoming a web developer, there are many concepts that must be put aside and new ones learned.

Studies examining the difference between an expert and novice find that performance differences arise from differences in brain activity. Not only do experts require less brain activity than novices, they also use different parts of their brains (Milton:07): Experts use conceptual memory whereas novices use attentive memory. That is, experts are able to exploit abstractions in conceptual memory, whereas novices must hold primitive representations in attentive memory.

Sketchlets(alpha) helps a programmer form and prime concepts by supporting abstraction and reviewing concepts that need to be refreshed.

Future

- fMRI studies of programmers. See preliminary research!
- Will future programmers take **designer nootropics** for boosting memory and attention to keep up?
- Can we predict the memory load of using a language feature or performing a particular programming tasks?
- What new tool ideas can be derived for programmers?
- What experiments need to be run?

Interested in participating in an experiment or have any ideas? Email me at `[email protected]` or comment below.

References