NATALIE MASSE HOOPER

DEEP WORK FOR SOFTWARE DEVELOPERS

TL;DR: a few simple steps to improve your focus as a software developer

Deep Work for Software Developers

By Natalie Masse Hooper

TL;DR: a few simple steps to improve your focus as a software developer

Introduction

My name is Natalie Masse Hooper, and I am a mobile software developer with 9 years experience. My professional background includes startups, large tech and non tech companies, and I currently freelance.

I have always liked doing things besides my job. Now that I have a family (my daughter is 3), staying up late at the office isn't even an option anymore. Not having it as an option is a relief actually.

What is deep work and why you should care

"Deep Work: Rules for Focused Success in a Distracted World" is a book by Cal Newport, a computer science professor. It was published in 2016.

In it, he argues that **deep work, as opposed to shallow work, is becoming a rare skill**. Thus, developing it will give any knowledge worker an edge.

Deep work can be best understood as work that **provides value and is hard to replicate** by others. It requires focus. It's not the stuff you do when you're tired.

As a software engineer, deep work is writing code, learning, teaching. It may include looking up something on Stack Overflow but if all you do is copy and paste, it's not deep work.
Not all work needs to, or even can, be deep work. We certainly cannot work deeply for 8 hours every day. But a good proportion of your working hours should be deep work. His advice is to **aim for 3 to 4 hours a day**.

You should care because with the amount of good code tutorials out there, not to mention the abundance of open source code and, of course, Stack Overflow, there is no reward for remembering something by rote mindless repetition. And this is good, because our brains really shouldn't be worrying about whether to use *length()* or *size()*. **But it means that you need to dig deeper to offer value.** For example, understanding about Android app architecture, or web performance. **And those skills cannot be developed by skimming through online content.**

What is in this eBook

Cal offers some rules in his book, with examples. Most of those are taken from his own work as a professor and writer.

I've been through the process of adapting them to my work as a software developer. I've now been practising deep work for 2 years, and I'm really pleased with what I have achieved professionally during that time, especially as I only work for about 20 hours a week. I have a young daughter who goes to pre-school in the mornings (when I do my deep work); I sometimes do a little bit of shallow work in the afternoons.

So I've compiled this eBook as advice to myself, because it's worth reminding myself about it once in a while. **And I'm sharing it with you, hoping that it can help others.** As such, the writing style is mostly "I do this and I do that". And when I say "you", I'm mostly talking to my future self.

It starts off with **understanding your value in Chapter 1**, as the whole point of doing deep work is to deliver more value. **Chapter 2 is about scheduling deep work**; then, we **track it in Chapter 3**. **Chapter 4 is about planning tomorrow**. Then, we tackle the **online/offline question in Chapter 5**, and finally, we talk about the **weekly review in Chapter 6**.

The whole eBook is very short; it's only 14 pages, so it can be read in one sitting. It's on purpose. Enabling deep work isn't complicated.

Chapter 1: Understand what your value is

The headline

Most software developers could probably summarise their value along the lines of **“creating functioning and maintainable software within the allocated timeframe and budget”**.

While this may do for you, let’s see if you can personalise it a little bit.

Make the headline yours

A few questions are worth asking yourself.

- Is your value about the **personal code** you write, **or the code you enable others to write**? A junior developer may focus on their own code, while someone more senior, or with a teaching temperament, may focus on others. If your focus includes others, you can replace “creating” with “helping other developers create” etc.

- Are you primarily focused on a **specific platform**, eg web apps? In this case, you can replace “software” with “web apps”.

- Are you especially concerned with a **specific aspect of software**, such as user experience, automated testing, animations etc? In this case, you can amend “functioning and maintainable” to include your specific interest.

- Do you have a **specific skill** that is **less common among software developers**? Perhaps, you worked as a designer before, and you are able to bridge the gap between design and development teams. Or, you are particularly good at understanding how your colleagues feel, what frustrates or motivates them, and you can facilitate communication within a team.

- Do you have a strong desire to **improve a specific aspect of the software development community**? Such as access to documentation in languages other than English, lack of diversity in many high tech companies etc

Here's what my own final version looks like:

I want to "create easy to maintain mobile and web apps, share my knowledge with others, and inspire younger women to become developers."

Here, we don't discuss your personal values. It goes without saying that you shouldn't compromise your personal values to reach your professional goals. With deep work, you're working on the craftsmanship of software development, at developing skills that will set you apart, at providing value as a software developer. Decisions such as which product to work on are separate from deep work practice. However, such important decisions should be made in a state of focus, so deep work practice may indeed help them.

Deep work activities that enable you to deliver value

Of course, writing code is something you probably should spend time doing.

But what about **reviewing your teammates' code?**

And understanding code and **writing documentation for others?**

Besides code, what else could you do? Depending on your value, things such as writing code tutorials, writing documentation, learning about a particular technology, learning about a particular aspect of software development, mentoring others etc

This is when your value definition comes in. You cannot do everything, so you must choose the specific activities that will enable you to deliver the value you want to deliver.

I already know how to create Android apps (with the Android SDK), I have been learning Flutter, which has enabled me to publish both iOS and Android apps. I do not know much about web apps yet.

Therefore, my focus is to get better at both Flutter and web apps, as well as explore "easy to maintain" further. In this simple concept lie many things. Easy to test, easy to amend, easy to adapt, easy to debug, easy to document etc. And also, easy for whom? Obviously, easy for me is a requirement, but I am looking beyond myself too.

I also want to share my knowledge, so writing code tutorials and in depth articles are important.

Lastly, I want to inspire younger women to become developers. I think the best way to do this is by being visible myself as a female developer. At this point in time, I am mostly using my blog for this, and shallow work activities such as tweeting etc

In summary, my deep work activities are:

Writing code for my current Android and Flutter projects

Learning about web apps

Learning about and exploring various ways to write “easy to maintain” apps

Writing code tutorials and in depth articles

Chapter 2: Schedule your deep work

How much?

Cal advises to aim for 3 to 4 hours a day max. This is mostly based on studies of virtuoso performers, but this is quite common sense. We all know we feel like our brain is “fried up” after an intense day at work. Or is it just me? ;-)

I manage about 3.5 hours per day, but not in one session. My limit for uninterrupted stretches appears to be about 2 hours. I can, however, fit 2 stretches per day, with only a 15 minutes break in between.

If you currently struggle to even fit in half an hour a day, start low in your schedule, with perhaps 2 daily session of half an hour each.

When?

In his book, Cal Newport details 4 different approaches to deep work, in terms of when to do it. The one most suitable to most of us is the **rhythmic approach**.

With that approach, you decide in advance that you will do deep work between 9.30am and 11.30am tomorrow for example. The more regular you can make it, the better. For example “every workday between 9.30am and 11.30am”. In reality, though, most of us have work meetings in our calendar, so we can’t find a slot that would work every day.

No need to fret if you don’t do your deep work at the same time every work day. It is still a rhythmic approach when **you plan to do a few hours every workday**.

Let others know

If you’re in an open plan office, it’s important that you **adopt a sign that lets your colleagues know you are busy working and not available for interruptions**. Wearing headphones (with no music) or visible ear plugs (works better with short hair) is a good tactic, as it’s easy to understand by all. If, like me, you can’t stand anything in or on your ears when focusing, wear a hat that makes it difficult for others to make eye contact with you (while still enabling you to see your computer monitor of course), or have a notebook with

some pseudo code next to you (it works effectively well at giving out a “I’m busy doing difficult work” vibe).

Always let your manager and close team members know that you plan to go off email/chat to focus on writing code. The more rhythmic you can do it, the easiest it is. For example,”in between team meetings, assume I’m offline and in deep work mode, except the half hour before lunch and the hour before I leave, during which I’ll catch up on email and messages, and get back to you”.

Bonus support

If you can **encourage one of your close colleagues** to schedule a deep work session at the same time as yours, you’ll have an extra incentive to do it (you can compare notes after).

Chapter 3: Track your deep work

This chapter is very small, because tracking your deep work should not be something that takes time. You do it and it's done. Just like reading this chapter.

What to track

When it comes to deep work, **it is all about controlling your own focus**, so it follows that you should only track what you control. There is no point in tracking the number of features delivered, because, well, it doesn't only depend on you. It depends on the design team, it depends on the business team not changing their mind every sprint, it depends on your colleague reviewing your code in a timely manner etc

Your ability to work deeply is a requirement for your long term success. So, **track your time spent in deep work**.

What qualifies as deep work? I'm sure you recognise it when you do it. It's when **you feel you've done some real good work**. Not chasing up a mock or API details. Deep work means some solid coding (not copy and paste), some solid learning (not skimming through some documentation or a blog article), or some writing (writing tweets doesn't count!).

How to track

It really doesn't matter, you can use an app, a spreadsheet, pen and paper. Whatever works best for you. At the moment, I'm using a page at the back of a notebook, with a row for each week. Put a mark of some kind for each half hour of deep work, and also link them if part of one session (eg you can put a circle around them).

Chapter 4: Plan tomorrow

Cal Newport recommends a **shut down routine**, at the end of the workday. I have actually done this for many years, roughly as follows.

15 minutes

I give myself **15 minutes to plan the next day**.

To start off, you can check your work email, Slack, and other services you regularly use to communicate with colleagues. If your input is required on something, either reply (if it takes less than 2 minutes), or add this to your list of “marbles and sands” tasks for the next day (more on that list later).

Then, go through Trello, JIRA, or whatever system your team currently uses, make sure all your tasks are up to date, and check any upcoming tasks.

During those 15 minutes, **you do not start any work**. It’s fine to push a commit if you notice you forgot, but it’s not when you start thinking about how you’re going to implement that new mock up the designer has emailed you.

The big stones

A lot of the advice I have read about productivity, and life happiness, revolves around the idea of **“putting the big stones first”**, and it combines very well with the idea of deep work. Shallow work can be described as “sand”, in the sense that it is doable in small chunks and it can easily fit around your schedule as it doesn’t require a lot of mental energy.

After taking stock of all my possible tasks, **I look at my calendar** to determine how many hours of deep work I can realistically fit in the next working day, and then I make an ordered list of “big stones” items I can realistically tackle in that time. I may put some rough timings next to them if my day is particularly fragmented, but I rarely bother.

The marbles and the sand

I make 2 separate lists of smaller items, the marbles and the sand lists. I order each one, by order of energy required (ie the easiest one is at the bottom of the list). This is the list I go to when I feel tired with my deep work, or I have only 15 minutes until my next meeting or lunch.

Treat others as you would like to be treated

Another thing I do while checking my upcoming tasks is **asking myself if something is blocking me**. I then check the relevant information sources (eg Zeplin if I'm after a design) and send messages to relevant colleagues asking for the information if I can't find it.

By doing this every day, you should be able to unblock most of your tasks before you need to do them, without your colleagues feeling they have to reply to your messages straight away. Treat others as you would like to be treated.

Chapter 5: Be online, only for what you need

Cal Newport suggests you disconnect from the internet while doing deep work.

However, **as a software developer, it's not realistic to be offline**. We need to push our git commits to a repo, we need to check various online tools for data (eg Zeplin for designs), we often need to use the internet to check a specific language or framework documentation.

So, instead, **"be online, but only for what you need"**.

Use a tool to track your time

I'm a big fan of Rescue Time *(aff)*, to give me an overview of how I spend my time, both on my phone and my computer. I don't obsess about it, but I do check the weekly email I receive.

Know why you use each communication tool

The first step is to assess how you use communication tools - both your personal ones (ie the ones you are in full control of), and your professional ones (ie those imposed on you by your team).

I won't go much into choosing personal communication tools, except that I regularly prune my Facebook friends list. As I get older, I find that what matters the most to me are friends I would make an actual effort to go see. So I keep those, and family members on Facebook, and remove the others. **Cal Newport makes a big deal of assessing your social media presence in quite a drastic manner, but I think notifications is really what matters** (see next section).

Email, Slack, JIRA, Trello, other chat apps… **What tools do you use for your work, and why?**

For example, is email mostly used for announcements, or does your team use it to actively discuss features? Mileage will vary between teams - and while you may have some suggestions to reduce the noise, the first step is to understand how each tool is currently used by your colleagues.

I'm a freelancer, so I do the assessment for each client.

Disable notifications as much as possible

Unless you are actively looking for work, there is no reason to get notifications from LinkedIn. In fact, you probably don't need the app if you just check the website once a week.

A trick I use to reduce notifications and temptation is to not install the apps and actually just use the websites. Most websites, eg Twitter, work great on my phone, there is no need for the apps. It requires a bit more effort to check Twitter, as I have to open Chrome, and then start typing twitter in the address bar. This additional task can sometimes be enough to stop me in my tracks and refocus on what I know I should be doing.

Some chat apps, like Hangouts, allow you to disable notifications on a sender basis, and you can easily disable/enable. So, let's say, you can enable notification if you're meeting said friend within the next 24 hours, and disable otherwise.

And **I disable all notifications on my work desktop.**

Async as much as you can

When using a chat tool for work, a very common opening sentence is "Quick question". When you read this, you feel under pressure to answer it straight away. To avoid this, log off during your deep work hours. If you're expected to be online, change your status to something like "focusing on code, will not answer messages right away".

And, with this in mind, **be mindful of how you contact your colleagues**. If you are really stuck without their answer, do apologise for interrupting them. Even better if you can also explain how you'll make sure not to put yourself in same situation next time.

For example, perhaps you didn't check the mocks the designer sent you when he did. Now that you're looking at them, reading to write some code, you have some questions. In this case, take full responsibility for your own mistake (not checking the mocks when you received them) - don't say you were busy, or X or Y interrupted you, but actually say you didn't prioritise what you should have (checking the mocks). **Saying you made a mistake can be hard, but it will help you with not making that mistake again** (as nobody likes to admit to the same mistake twice in a row!).

The daily ritual outlined in chapter 4 will enable to unblock your blockers without needing immediate response from colleagues. And **the less you interrupt your colleagues, the less they will interrupt you**.

As long as you regularly check on your work chat, no project manager will begrudge you for being offline if you're actually doing some important work. Checking 2 or 3 times daily is a good practice. Even better if you can do it at a regular time, so they know when they can expect your reply.

Chapter 6: Weekly review

Your weekly review for deep work should be very quick. It's not about creating an administrative burden on you, so this chapter is very short.

Once a week, on Friday afternoon if convenient, **check your deep work hours tracking system**. Does it look good? You will know if you've been focused or not.

Then, **look at next week's calendar**. Is there a meeting you would like to cut out, because you feel you aren't needed, or it's too long, or it's poor quality? Consider sending a message to the meeting organiser.

Lastly, **create a "big stones" to do list for the week**. Finally, do your usual daily ritual at the end of the working day (the weekly review doesn't replace the daily ritual, as detailed in chapter 4).

Conclusion

Deep work is a process, so don't beat yourself up for failures. Each day is a new day, and may be more or less successful in your attempt at doing deep work. Each week is a new week. The process is what matters: the more deep work you manage to do, the more value you provide as a professional, the more successful you can be (whatever success means to you), the more relaxed you feel about your work.

Reassess how you provide value (see chapter 1) at least once a year, and whenever your job title or description changes.

And remember, the goal isn't deep work 8 hours a day : your brain couldn't cope! **Aim for 3 to 4 hours daily, and you'll deliver a lot of value**. So it's perfectly fine to enjoy long chats with colleagues, or a break in the games room, or leaving the office at 3pm to pick your kids up from school.

==========

Enjoyed this eBook? Feel free to email it directly to someone, or send them to my website where they can download it for free.

Got some feedback? Feel free to email me at natalie@cogitas.net. I read all emails, though I do not reply to all. I will reply if your email contains interesting ideas I haven't heard before, or if your story particularly touches me. But I do appreciate "thank you" emails, they are warmly received even if I fail to reply!

Finally, if you want to **support this eBook**, you can donate.