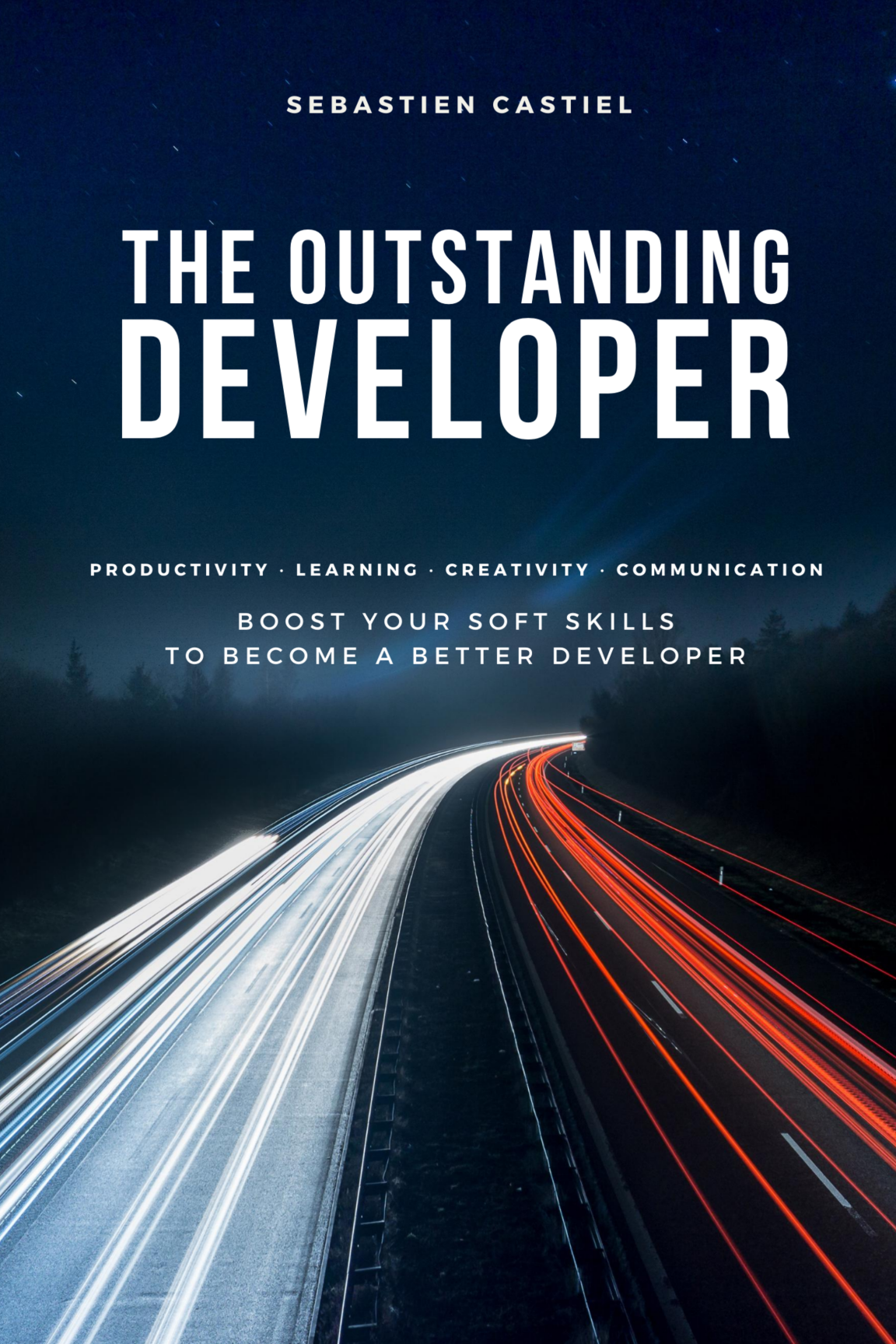
SEBASTIEN CASTIEL

# THE OUTSTANDING DEVELOPER

PRODUCTIVITY · LEARNING · CREATIVITY · COMMUNICATION

BOOST YOUR SOFT SKILLS
TO BECOME A BETTER DEVELOPER



# Contents

# Introduction

A while ago I started reading nonfiction books, mostly about self-improvement. And I read a lot of them. Some talked about productivity, others were about finance, entrepreneurship, creativity, communication... And each time I finished a book, I thought about how I could apply what I found in my day to day life.

I adopted new habits, made some adjustments to the way I worked, always to find a new way to improve productivity, learn new things, be a better colleague, or a better team leader. Sometimes the changes I made didn't produce the expected result, but most of the time they did.

Now I can say that these books (and a lot of articles or blog posts) helped me to improve as a developer. Not that I was a bad developer (at least I hope not), but in a developer job I strongly think that you hit a plateau if you're not careful to improve regularly, and not only technically.

Of course, keeping up to date in your technical fields is more than recommended: what you know today about your favorite language or framework may be obsolete in a couple of years. But there is some space for improvement in soft skills too: become more productive, learn new things more easily, or communicate in a more efficient way.

These soft skills are what I wanted to talk about when I decided

to write this book. I took the challenge not only to put on paper what I learned, but also to dig deeper, exploring what other books and articles could bring, what research studies concluded, and how I could make the link with our developer job.

## Who is this book for?

Although this book is written for developers, you might find interesting content if you work in IT in general: if you're a QA analyst, a UX designer, a project manager, etc.

The advice you'll find will apply whether you work for a big company, for a start-up, or if you're self-employed, and whether you're a senior developer or you just graduated.

Finally, I suppose in this book that you work as a developer, but everything can apply if programming is not your professional activity: if you're contributing to open-source projects, if you program as a hobby, etc.

## What this book is not

This book is focused on improving your soft skills in your developer job. Some other aspects of improvement could have fulfilled other books written by someone better at it than me. Therefore, this book is not:

**A guide on how to code better**: a lot of books exist on that subject, most of them depending on the language you use, or at least the "type" of development you do: web, software, games, embedded... Here you won't find any technical advice, nor any line of code.

**A career guide**: advising on someone's career is a full-time job, requiring a lot of expertise. Although you may find here some clues about my career and the way I decided to evolve as a developer, this shouldn't be considered as advice for your career. However, I'm confident that with the advice found in this book, it's going to be easier for you to reach your career goals.

**A guide on how to create your start-up**: creating a business as a developer (a start-up or a freelance activity) is a very exciting goal, but it's not the only one. And since it hasn't been mine, and still isn't at least for the near future, I wouldn't be a good person to tell you how to do it. But again, the advice found here can make things easier for you if you pursue this goal.

## What to expect from this book?

In this book, I try to give an overview of the challenges that developers are facing today. I identified four big axes to group these challenges: productivity, learning, creativity, and communication. For each of these axes, we'll explore what problems they relate to in our developer job and try to find advice on how to overcome these problems. Sometimes there won't be any actual problem; just a space to become even better.

As much as possible, all the content of this book is based on two things: what I found in the literature (books, research articles, blog posts, etc.), and personal experiences (mine or not). Linking these is the most interesting, since most of the time what you find in books or articles is not specific to a developer job.

# A note about this preliminary version:

You are currently reading a non-final and incomplete version of the book. I'm releasing each chapter as soon as it is finished, to get feedback as quickly as possible. So please keep in mind that some content may change in the final version, or even in the next preliminary version.

Especially, this introduction is very temporary, and will probably be completely rewritten for the final version.

If you have any feedback about the book, I'd be more than happy to hear about it. Send me an e-mail at sebastien@castiel.me[1].

I wish you a pleasant read,

—Sebastien

[1] mailto:sebastien@castiel.me

# Chapter 1 — Productivity

Thinking about how to improve at a developer job, it's very likely you think instantly about improving your productivity. It may even be true for most of the jobs.

Let's say you're good at coding (and I'll assume you are in the rest of this book). If you happen to just code for five minutes a day, or if you write a great piece of code that will never go to production for any reason, you may not feel very productive, therefore there might be space for improvement.

Before exploring how to be more productive, I suggest we first ask ourselves what we hear when talking about productivity. It may be obvious, it's a very common word. Yet in the specific context of a developer job, you may get different answers depending on who you ask.

## Defining and measuring productivity

Imagine yourself in the following situation: you are in an office, in the IT department of a company, or a start-up. It's the middle of the morning, everybody seems to be working, it's quiet and you can just hear some people talking about last weekend at the coffee machine behind you.

In front of you, two people are working at their desk. You guess by looking at their screen that both of them are developers. Let's call them Alice and Bob.

A third developer comes to ask a question to Bob. Still typing code on his keyboard, Bob tells him that he's very sorry, but he has an urgent task to finish before the end of the day, and since he has his afternoon full of scheduled meetings, he needs to advance as much as possible in the morning.

The developer then comes to Alice, who takes off her earphones and stops typing. She tells him that she's in a middle of an intense problem-solving session and prefers staying focused for now, but if it can wait twenty minutes, she'll just grab a cup of coffee then she'll be glad to come to him afterward.

With so little information, if you were asked which one of Alice or Bob is more productive, what would you answer?

For most people, Bob would be considered more productive. He seems to be working a lot; he doesn't even take time to answer a question and has a full agenda. Alice, on the other hand, can stop working almost instantly, has time to help other people, and even for a cup of coffee.

In the chapter *Answering an easier question* of his famous book *Thinking, Fast and Slow*,[2] Daniel Kahneman details how sometimes our brain tricks us when trying to answer a question without having all the information we need. This mechanism is called a *heuristic*, and you probably heard of it in the context of algorithmics and optimization.

Think of heuristics as optimizations that your fast-thinking brain (System 1, as Kahneman calls it) does to save you of the painful task of gathering all the data, analyzing them, and interpreting them. Is the president doing a good job? Just ask yourself if he's

[2]Kahneman (2011)

popular. Are you living a happy life? Ask yourself if you're in a good mood right now. Is this developer productive? Well, do they look busy? Am *I* productive at my job?

This heuristic, consisting of estimating your productivity by looking at how busy you are, has been described by Cal Newport in his book *Deep Work*[3]. He talks about *Busyness as a Proxy for Productivity*: since it may be difficult to measure how productive you are, it's tempting to rely on how busy you look, meaning how busy people around you think you are.

A research team studied in 2017[4] how software developers perceived their productivity. 400 developers at Microsoft answered a survey asking them about what made them feel productive or not, and what indicators they used to estimate their productivity.

This study used the answers to categorize the developers into six groups: the social developers, the lone developers, the focused developers, the balanced developers, the leading developers, and the goal-oriented developers. In each category, developers perceived their productivity differently.

For instance, lone developers feel productive when they spend most of the day coding, with as fewer interruptions and social interactions as possible. Social developers, on the other hand, feel more productive in helping and interacting with others or doing code review, but still, find focus time by coming early to work or working late. On the opposite side, leading developers prefer spending time designing things, and are not afraid of meetings or emails.

This study also brings us very interesting indicators to measure productivity, from a developer's point of view. They change depending on which developer category you observe, but the two most important ones according to developers seem to be:

[3]Newport (2016)

[4]André N Meyer, Zimmermann, and Fritz (2017)

- The time spent on coding, and
- The longest period focused on a task without interruption.

In the end, maybe the productivity of a developer would be best estimated not by how busy they are, but by looking at how much time they spend developing things, whether it is coding or contributing with others, with as few distractions as possible.

Having this idea of how productivity may be estimated, the next step is finding how to improve it. Of course, my goal here is not to help you to estimate your coworkers' productivity, even less to improve it, but to improve your own.

In the next section, we'll see some blockers for productivity in most of today's workplaces and how to overcome them. Then we'll explore how setting goals for yourself can help you to become better in your job.

## Obstacles to productivity

Think about a normal day of work. You arrive at the office, take a coffee, start coding, do some code review, have lunch, some meetings, code again, and that's it. But is it really?

I could bet that looking more precisely at your day, it looks more like: start coding, answer a question on Slack, help a colleague, have a short meeting, code again, ask a question to a colleague, fix the build pipeline, answer an e-mail, fill your timesheet... and it's not even noon.

When twenty developers from the US, Canada, and Switzerland were asked to keep a precise diary of what tasks they worked on during the day minute by minute[5], the results were surprising.

[5] Andre N Meyer et al. (2017)

First, code-related tasks (coding, debugging, code review, etc.) take only two hours a day. This is 25% of the classic day of work. You can add almost one hour of "work-related browsing" (I would consider this as part of the coding activity, as long as it's not to procrastinate), then it's e-mail (one hour), planning (30 minutes), documentation (30 minutes), planned or informal meetings (1.5 hours), etc.

Maybe you'll tell me that you spend more than two hours a day coding, but I can't recommend you enough to try during a day or two to keep a diary of everything you do. And I mean everything: when you leave your computer for a minute to get a glass of water, when you stop coding for thirty seconds to answer a question on Slack, or when you open your inbox to get some information you need.

If you do this, you may realize another interesting and disturbing thing: the average time you spend on coding before switching to something else: an interruption, a meeting...

The same study highlighted that on average, a developer writes code for 36 seconds before switching. Not even a minute! Of course, sometimes you code during much more time, but it's compensated by the several other times you code for just a few seconds before being interrupted.

The other tasks follow the same pattern, except the planned meetings (it's more difficult to leave a meeting after one minute I guess). So, looking at the global picture, a developer's day is composed of a lot of small periods of a couple of minutes, when they can either code, answer e-mails, write documentation, or participate in an informal meeting.

But I can imagine that if you try to plan your day, you don't expect such a fragmentation of work. This is a big obstacle to productivity. So, what causes this fragmentation and how can we overcome it?

A study made in 2011[6] found that most interruptions encountered by developers during a day are in-person interruptions. On average, there can be between 12 of them for an entry-level developer, and up to 40 for a senior developer or technical lead. Other kinds of interruptions are provoked by instant messaging (4 to 9), e-mail (8 to 52), and phone (0 to 20).

Let's focus on colleague interruptions. Most of them are legitimate: it's part of a developer job to ask questions to colleagues, and you rely on your coworkers to answer yours, especially for junior developers. Therefore, removing these interactions is out of the question.

Still, there must be a way to keep a reasonable level of interaction with coworkers without causing that many interruptions.

In 2017, a team[7] developed a tool to help other teams in several IT companies (developers, testers, project managers, etc.), reducing in-person interruptions. It was composed of a LED light above each desk and an application installed on developers' computers to automatically update the LED color. The LED was green if the developer was considered *available*, red for *busy*, and blinking red for *do not disturb*. The software used indicators such as keyboard typing during a given time to determine the developer's status.

This tool showed good improvement in the teams where it was tested. The number of interruptions was divided by two, and around 60% of participants estimated they had been less interrupted when busy (20% disagreed with that statement).

More importantly, almost 60% of participants felt that their productivity increased during the study. Not all interruptions were removed, but they tend to happen less when the developer was deeply focused on a task. Plus, the LED helped people being aware of the cost of disturbing someone.

[6] Sykes (2011)
[7] Züger et al. (2017)

This system may not be a revolution to improve developers' productivity, but we can draw inspiration from it. What it demonstrates is that people tend to disturb people less often when aware of how busy they are.

In the different teams I worked with, we almost always had a system to indicate others that we were busy and preferred not to be disturbed. Sometimes, it was having headphones on. Other times, we used the status of our instant messaging application, setting it to *busy*.

But one of the most efficient systems we found was not a specific sign to indicate we were busy. I still try to initiate it with the teams I work in today. We simply consider everyone as busy. It means that if you ask a question to someone on instant messaging, you shouldn't expect an immediate answer. More importantly, if you need to ask a question in person, first send a message on IM, such as: "I need your help, tell me when you have a moment". Or just: "Got a second?"

Why sending a message on IM instead of just asking directly, you may ask? It turns out that just asking "do you have a minute?" in person is often enough to make the person lose focus on what they were doing and ruin the whole strategy.

A past colleague of mine, when someone (sometimes it was me) asked him "can I disturb you for a minute?", used to answer, "you just did". Always in a humorous way, but it was enough to make people realize that asking a question as short and easy-to-answer it is ("do you have a moment?"), is enough to make the person lose a precious focus.

When I'm working on a task requiring a lot of focus, I usually turn off IM notifications. My coworkers know that I won't answer them immediately, but that I'll find time to answer them eventually and spend the necessary amount of time to help them. And when I'm the one having a question, as much as possible I use IM to ask

them to ping me when they have a moment.

These good practices revealed themselves to be very efficient to improve developers' focus and limit interruptions. A lot of articles or blog posts describe similar practices in different teams. Yet, it's not perfect. Another source of interruptions is nowadays very common.

## The open-plan office hell

Imagine a big warehouse. From where you stand you can barely see the opposite wall. Steel beams go from the carpet floor to the ceiling's vent pipes —maybe fifty feet high. People are working at desks; some sitting, others standing. The desks are aligned, and between two desk rows, barely the necessary space for someone to walk without touching any chair.

No visible conference room, but you can observe some groups of two to four people talking to each other at one's desk. For some of these meetings, they're using one of the whiteboards available in the alleys.

In total, you can estimate that at least one or two hundred people are working here, maybe more since you can't see far enough.

You may be familiar with this open plan office situation. The one I just described is at Facebook headquarters, in what Mark Zuckerberg described as "the largest open floor plan in the world", as Cal Newport reports in *Deep Work*[8]. Your office may be smaller, but if it is like most of today's companie' offices, it should be a little bit like the one I just described.

Most developers today work very close to their coworkers. Close enough to talk or show something on the screen without requiring

[8]Newport (2016)

moving. If different teams are working in the same office, they are usually separated virtually only, sitting on different desk rows.

Why this tendency in office planning? As Cal Newport points out, Square and Twitter CEO Jack Dorsey declared "We encourage people to stay out in the open because we believe in serendipity —and people walking by each other teaching new things".

I can understand this point of view, and it seems difficult to argue against open-plan offices when trying to improve collaboration. Yet, we established that interruptions are a big obstacle to productivity. Aren't these offices encouraging interruptions instead of preventing them?

*Did you like this preview? Discover more about productivity in the second part of this chapter by purchasing the book at https://leanpub.com/outstanding-developer/*

# Bibliography

Kahneman, Daniel. 2011. *Thinking, Fast and Slow*. Macmillan.

Meyer, Andre N, Laura E Barton, Gail C Murphy, Thomas Zimmermann, and Thomas Fritz. 2017. "The Work Life of Developers: Activities, Switches and Perceived Productivity." *IEEE Transactions on Software Engineering* 43 (12): 1178–93.

Meyer, André N, Thomas Zimmermann, and Thomas Fritz. 2017. "Characterizing Software Developers by Perceptions of Productivity." In *2017 Acm/Ieee International Symposium on Empirical Software Engineering and Measurement (Esem)*, 105–10. IEEE.

Newport, Cal. 2016. *Deep Work: Rules for Focused Success in a Distracted World*. Hachette UK.

Sykes, Edward R. 2011. "Interruptions in the Workplace: A Case Study to Reduce Their Effects." *International Journal of Information Management* 31 (4): 385–94.

Züger, Manuela, Christopher Corley, André N Meyer, Boyang Li, Thomas Fritz, David Shepherd, Vinay Augustine, Patrick Francis, Nicholas Kraft, and Will Snipes. 2017. "Reducing Interruptions at Work: A Large-Scale Field Study of Flowlight." In *Proceedings of the 2017 Chi Conference on Human Factors in Computing Systems*, 61–72.