



CUSTOMER REQUIREMENTS

**EVERYTHING PROGRAMMERS
NEED TO KNOW BEFORE
WRITING CODE**



BY MARCO BEHLER

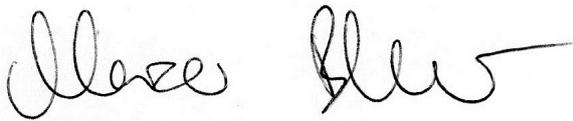
THANKS FOR READING

Thanks for reading *Customer Requirements - Everything Programmers Need To Know Before Writing Code*.

If you have any suggestions, feedback (good or bad) then please do not hesitate to contact me directly at marco@marcobehler.com or leave a comment on our blog at www.marcobehler.com/blog.

(This is a one-man operation, please respect the time and effort that went into this book. If you came by a free copy and find it useful, you can compensate me at <http://www.marcobehler.com/books>)

Thanks!

A handwritten signature in black ink, appearing to read 'Marco Behler', with a long horizontal flourish extending to the right.

- Marco Behler, Author

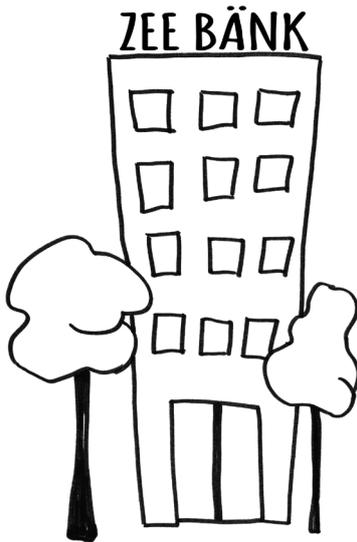
Copyright 2015 Marco Behler GmbH. All Rights Reserved.

Communication

- Concrete: What (not) to talk about. Who. How. When. Why.

Yes, I am suspicious too. Communication is one of those super vague words that can mean *ANYTHING*. Instead, let's make it crispy!

Our guy on the cover, a Brazilian programmer called Neilson, is working for a new start-up. It is called *Zee Bänk*, our fictitious world-changing PayPal competitor, albeit very similar in features and capabilities.



You can register an account with your email-address, have different payment methods (wire transfer, direct debit etc.) and then use that account to pay at your favourite online shop, for example iTunes.

As our start-up is still very young, we are missing credit cards as a valid payment option. Hence, one morning our CEO storms into the room and shouts:

"We need users let their credit card connect to their accounts, immediately! Should not take too much time, I mean there are probably libraries for that, right? Plus you guys are super smart! And we support wire transfer already! We need to be fast....two weeks?! Need a reliable estimate for go live in an hour. Bye!" - Ouch!

This chapter will give you theory and advice on what exactly you should do now. We will start with broader topics like "who should talk" and "meetings" and end the chapter with very specific "what" and "how".

The next chapter will be then be a practical based on this information.

Who should talk now?

Before we talk about *what and how*, let us have a quick look at *who* and why this is so important.

In smaller organisations, start-ups or agencies you will find that developers often are hybrids: they are not only expected to code, but also directly help the founder or sales person come up with concepts and specifications. Same goes for freelancers.

In bigger organisations there is usually a dedicated role like a business analyst or product owner who is supposed to bridge between business and tech people.

No more code monkey

The distribution of roles or the organisation does not matter however. Problems always arise, if honest, two-way street communication between business/client and developers is missing and instead is being replaced by good old top-down communication: on one side your boss/client who comes up with all concepts, deadlines and specifications. On the other side the developers who are only supposed to implement and not ask too many questions anymore.

Assuming you are the programmer, this means you have to stop being an isolated code monkey and actively be involved in shaping requirements or at least fight for that. But more on fighting later.



Heterogeneity

The key to understanding requirements and your problem domain is *heterogeneity*, not homogeneity. Your primary instinct might be to talk to other programmers, how to solve possible scaling problems or what not. But we are in the requirements phase, so forget all that.

You or you and your product owner should sit down with the sales guys - what exactly do they want or did they sell already? Or with the customer support guys - do they have specific credit card workflows, like for chargebacks or stolen cards? Or the marketing guys - any special marketing campaigns planned for credit cards, e.g. bonuses if you refer a friend?

On a bigger scale

If we are talking about bigger companies, the different teams or rather their representatives should talk to each other. Again, here I am not talking about e.g. just the programming teams talking with each other, but about the marketing team talking with the programming team etc. Even if it is just for a fraction of a day each month. That is much better than working in isolated silos.

Too Many Cooks Spoil the Broth

Be warned however, that inviting too many people to clarify a topic is a big mistake. Similarly to inexperienced tug-of-war teams, the more people get together in a room to brainstorm, the less output you get from everyone individually. It has nothing to do with shutting out team members, but all with effectiveness.

Keep it simple, stupid

As a rule of thumb, always start at the low end. Only add people if needed. You will be amazed at how many topics two or three people can actually not only conceptualise, but also implement and ship.

In our example, let us start with just two people driving our credit card topic requirement. A business representative, your CEO, and you, a programmer.

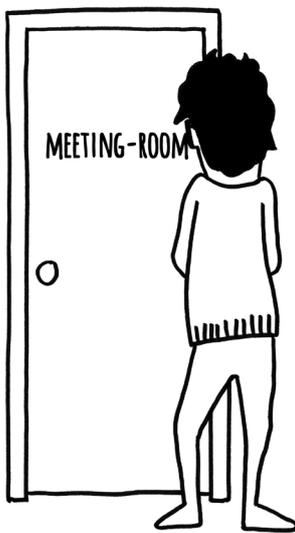
As credit cards are a huge topic however, it is quite likely that at some point you have to “meet” someone, be it a technical advisor from VISA, your internal PCI compliance expert or one of your application engineers. For those moments, you will find the next chapter helpful.

A detour on meetings

I truly believe that you should cut down meetings as much as possible. I would not go so far and say that 100% of meetings are bad. But out of my gut, a fair 90% are simply utterly ineffective time and money wasters, certainly not effective communication. Jason Fried and David Heinemeier Hansson even go as far and call them toxic in their book [Rework](#).

Instead talk to people face-to-face, individually and informally. Relentlessly prepare memos. Make it quick. Share the knowledge afterwards.

(If you however find that many meetings are essential in your company, head to the Objections section later in this chapter.)



The Mind-Set Going Into a Meeting

An endless amount of meetings in a software project is usually an indicator of how bad things are, not how smooth they are running. Do not mix up quantity with quality here.

If you are in a meeting, you want things to be short and effective. The notable exception is customer meetings, where you do not want to give the impression that you are running away. Nevertheless it is always a good habit to get into the mind-set of *“which questions do I have to ask to leave this meeting as soon as possible?”*

Length of Meetings

Be wary of long meetings. It does not matter if you believe in a 20 minute attention span or you read something about exactly 47.521 minutes in your favourite psychology magazine.

Your productivity and output will drop insanely if you try to meet for hours and hours, no matter how much everyone at the table is trying to tough it out and let nobody notice. I also recognise that there is a special type of senior architect or management person who like having Fidel Castro style meetings, but they are ineffective nevertheless.

If your sessions are running longer than 60 minutes, start having mini-breaks. Open the windows, stretch your legs. Let your mind wander for a couple of minutes and do not check your mobile phones immediately for new e-mail.

Stand-Ups and Other Short Meetings

Take this paragraph with a grain of salt.

Do not let a stand-up become a lean-on-the-nearest-wall or sit-on-a-table. It is the first sign that people already need to sit down to endure what is coming up. Stand-Ups also easily give you the impression of communicating, but more often than not it is a couple of people daydreaming while one person talks as much as he can as to look productive.

This Is How You Should Meet

If there is no way around meetings, do it the Amazon way and get in the habit of writing detailed memos. What exactly does that mean?

One person, the meeting holder, prepares a detailed memo on the subject *before* the meeting. That person clears out most issues in person-to-person talks before everyone else sits together in a room.

People who are *not attending* the meeting but are still somewhat affected have to read this memo as *homework*, to know what is going on.

People who *attend* the meeting have to read that memo *before* the meeting or alternatively you can start the meeting with everyone studying the memo for 10-15 minutes. Reading, not talking.

The goal is to get everyone up to speed as soon as possible, level out knowledge differences and use the talking time to reach conclusions or decisions quickly and then leave that room again as soon as possible. Try it out. The practice section will contain an example of a memo and how to spread knowledge after a meeting.

Pushing Back – Handling Your Boss Or Client

The CEO's statement at the beginning of the chapter was a bit of a hyperbole, but pretty much captures every warning flag when you need to immediately push back on a requirement and push for more clarification. Let us have a look at each one in turn.

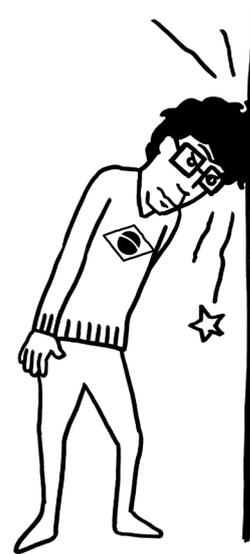
The Appeal To Existing Solutions–“I mean there are probably libraries for that, right?”

This one is actually a double-edged sword. On one hand, an astounding number of developers tends to reinvent the wheel and shy away from off the shelf solutions. Registering and logging in a new user? Let's write a new user management! In a way, this almost seems obvious, looking at the word “developer”.

On the other hand it gets equally counter-productive, when business suddenly decides that they *somehow know* that there are already existing libraries – or that you already implemented something almost completely *similar*. As something is already existing, integration cannot be that hard.

To come back to our example, credit card integration IS hard, no matter how many libraries exist. It also does not matter how many e.g. debit-card payment methods you already implemented, the overlap in functionality and re-use can only be seen *after* proper requirements work, not before.

What you have to say: “I can only tell after careful analysis, if and how useful those existing libraries/solutions are”



The Appeal To Your Ego – “Plus, you guys are super smart!”

This one is tricky and I fell prey to it myself once or twice at the beginning of my career (maybe I still do? 😊).

It is especially dangerous when you are a freelancer/agency and your client tries to pull this on you. Your client simply wants to save money by appealing to your ego and your skill to literally bake that pizza in 30 seconds instead of the usual 10-12 minutes. Should you agree on that estimate and then nevertheless need the usual 10 minutes, I can almost promise you that your ego will not go crawling back to the client and go “sorry, I was not so smart after all, it took me 10x longer”.

What you have to say: “The complexity of this requirement unfortunately has nothing to do with super smartness. Sure, I can promise you that fresh, Italian pizza after one minute. But you probably would like it in the oven a bit longer, wouldn't you?”

-- END OF PREVIEW --