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!

- Marco Behler, Author

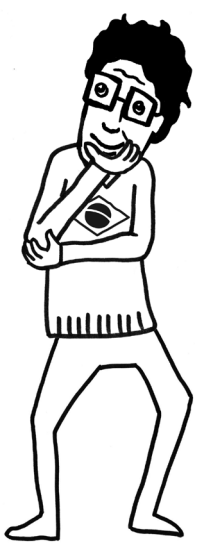
The Problem – Let's get ready to rumble: Customer vs Programmer!

Are you ready for some bad news? You might even call it blasphemy? Here we go: Your customers do not care about your code, frameworks or software architecture. At all.

They only care about a solution to a specific problem they are having. It is all about meeting their *requirements.*

Why is that such a problem?

As a software developer you primarily think of code, not so much requirements. Coding is, after all, what you learned and what you like to do. Often, when a software project is not running as expected the first look is towards tools, frameworks or processes. This is the stage where it is easy to get sucked into framework wars or debating endlessly about tenderly crafted software architecture.



Not even does your customer not care about any of that, the choice of tools or frameworks for most projects is completely insignificant - provided you have programmers knowing how to use them properly. Java and Spring/.NET and ASP/ Ruby and Rails/PHP, for most projects it simply does not matter.

What is important instead?

Projects do not fail because too many developers copy-and-pasted too much code from Stackoverflow or because they used SQL-Server instead of PostgreSQL.

They fail because of bad requirements: Too vague, too bloated. Hidden and then popping up in the last minute. Endless change. Completely unrealistic deadlines for those requirements. Budget and hence profit miscalculations. Hard to test requirements. Nonsense requirements. The list goes on.

Simply put: If you do not work your customer requirements properly, no amount of code or technology can fix your project.

Are you too smart?

The irony is: you might well be too smart to *really* care for the customer's requirements. Your education, your peers and your job title constantly steer you into the direction that code, frameworks and algorithms are the primary things that count.

The customer might after all just want to upload an Excel file to a shared hard drive and get some data processing done. How boring and unsexy is that? You on the other hand almost *deserve* a challenge for all the hard work you put into your education: *"How can we use NodeJS for this problem?"*

Put your ego aside

Do not forget: At the end of the day, someone trades in money in exchange for getting a solution to one of his *business* problems, which *might* be your clever code solution, but often is not. It is not about you. It is all about the customer.

It does not matter if you work in a small shop and interact with your customers on a daily basis. Or if you are in an organisation so big, that you direct boss is the only one remotely resembling a customer.

Think like a chef. Without a solid recipe (requirements), good ingredients and proper preparation, no outstanding meal, no happy customers – completely independent from your technical cooking skills.

Requirements are our recipe

Let us not be picky for the moment: Requirements or needs or wants or specification - names do not matter. When I say requirements, I mean "what exactly are we supposed to build and how does that ultimately lead to $$$ for the company".

Properly defined requirements not only let you figure out what you should build (obvious, huh?), when you are done and if you made the customer happy or not. They play a major part in every phase of software development.

Estimation: Do you want to continue to *guesstimate* your requirements or get a proper feel for how long certain tasks take?

If you are a freelancer, agency or in general directly paid for writing code, do you want to learn how to bill properly? Instead of getting underpaid? How do you know what you should get paid?

Do you want to learn how to handle customers correctly and not always feel one-down? You better know those requirements and the problem domain better than your customer.

Planning, building and testing software: So many books are written on low-level details on how to build or test stuff. But how are we supposed to get the *how* right, if we are still struggling with the ever changing *what*?

Last but not least: What does that look like in practice? How do you write up and manage requirements in the long term? How do you deal with team members and different domain knowledge levels?

Our way to the perfect meal

But programmers can almost be blind to the importance of properly clarified requirements. We want to fire up our IDE. Code, code, code. Try out our latest fancy agile methodology. Meet that next deadline. *"When? Tomorrow!"*

Here is what we should do, instead. Before we think about any code at all, we should start with just one question:

"What is it that the customer actually wants to accomplish?"

Sounds too easy?

Unfortunately, the way to get to that "what" is not so easy. Some villains are trying to get us off that path. Let us meet them:

Mr. Vagueness: "We need a new dunning & collection module for our E-Commerce shop. It should be able to do *everything*."

Mr. Constant Change: "Oh, well, we don't have that many dunning cases anyway, erm, let's focus on friendly reminder emails first, right?"

Mr. Hidden: "Oh, I completely forgot, our VP also wants to see some reporting on those overdue customers. But now that the rest is already implemented it wil just be a minor effort, right?"

Mr. Perfect: "We have to get it perfect, the first time around! Plus, we have to be first to market! Deadline is in x weeks. Beat our competitors! Be the best!"

Fear not

What a bunch of unhappy campers. But here is the trick: We are not going to be able to slay them all, ever. That would be wishful thinking.

What we have to learn instead, is to feel comfortable around them. Not only comfortable, but we have to learn how to *control* them, push back on them and make their impact as little as possible.

But Beware

It is easy to fall into the trap of ignoring all this and start thinking code. Someone else has the job to work out those requirements, right? Some business analyst or product owner! Isn't the customer supposed to know what he wants? I want to setup my MongoDB instead!

If you go down that line of thinking, you will forever *start first, finish last*. It is time to stop think-

ing like an isolated code monkey and being fed bananas, erm..., requirements day-in-day out.

If you really want to make your boss, your customer and yourself happy *in addition* to impressing everyone with your coding skills, then you have to look at requirements as a two way street. You have to work them properly. That is what you will learn in this book.

What exactly you will learn

You will learn what exactly a proper requirement looks like. You will learn how to ask the right questions to clarify vague requirements and make them real crispy.

You will learn how to control our villains (Mr. Vagueness and his friends). Pushing back on vagueness and nonsense. Dealing with change.

You will learn how to stop guesstimating and start estimating. You will see what billing and customer handling has to do with proper requirements and how to become really effective at both.

You will learn when a requirement is "safe" to be implemented and tested, without rushing headless to your IDE and banging out code.

Best of all, this will not just be a lot of dry blah-blah. Sure, as always there is some mild theory to be learned, but every chapter comes with a full on practical, taking you step-by-step from vagueness to clarity.

Buckle up, we are in for a ride! Let's go!