DOCS BY DESIGN

Bob Watson ponders technical writing, API documentation, and the world in general

MAY 28, 2018 BY BOBW

Yes, YOU can write docs as cool as Twilio's

After attending Write the Docs 2018 (Portland, edition) and watching a YouTube video, I've got it figured out: How you (i.e. *anyone*) can create documentation as cool as Twilio's–a consistent entry in lists of the best API documentation examples.

All you need to do is follow these six steps:

1. Think, plan, and work iteratively.
2. Infuse user research and analytics into your entire writing process.
3. Treat your documentation as a product.
4. Do what's best for your documentation customer.
5. Create and support a writing process that enables contributions from outside of the writing team.
6. Build the CMS that supports all of the above.

That's it!

You'll need to understand that it took them a few years to get to where they are, today. And, it sounds like they'll keep moving the bar.

But, don't be discouraged. The sooner you start, the sooner those three to five years will be behind you.

A little background

Twilio's Kat King opened Write the Docs 2018 in Portland this May, starting the conference off with a strong sense of user testing and research. That always gets my attention at a tech-writing conference. The next session by Jen Lambourne continued by describing how she applied user research on documentation. I've been going to tech writing conferences for quite a while and I can't recall going to one with such a user-research focus.

I'll make references to these videos, in what follows in case you want to go straight to the source (something I always encourage). The practices they describe, however, are straight out of user-centered-design. You can also see some of my favorite books on the topic for more background.

- APIS & CODING TRACK | How Twilio Writes Documentation – Jarod Reyes (Twilio)

- Building Empathy-Driven Developer Documentation – Kat King – Write the Docs Portland 2018

What's noteworthy here is that the Twilio team has applied them successfully to their documentation–proving, if nothing else, that documentation works like any other product (and perhaps works best when treated as one).

So, here's how the steps work:

Think, plan, and work iteratively

The Twilio documentation talks focused on iteration. In both videos, it's assumed that whatever they do, they'll do again only differently.

Try, test, analyze, change, and repeat.

This is to what I was referring when I said it follows a user-centered design pattern. To change, you must iterate. If you don't apply an iterative method to your documentation, just stop here and start working on that.

The underlying assumption at work is that you don't know what you don't know and you won't know until you find out. Then, after you find out, all the analysis in the world won't help you if you can't change things based on your analysis. Finally, you need to find out if those changes actually helped.

But, first things first. If you think of your documentation as an evolving entity and your work processes support evolution and change on a regular basis, you're ready to proceed. If your writing still has a notion of "finished," you know what you need to work on.

Infuse user research and analytics into your writing process

Just to be clear, by "infuse," I don't mean to add the Google Analytics script to your content. While your infusion might include some Google Analytics, the process the Twilio talks described was much more involved. It included A/B testing, actual sit-down usability testing, and multiple iterations.

But, it's more than just bringing in a user every now and then, it's working the sense of research and analysis into and throughout the entire writing process. This means having goals, metrics that track to those goals, and the ability to test and adjust (see preceding point).

Where I've seen this step go astray is in counting things that don't count (i.e. that have no material impact on your business goals), not being able to change the process or the content based on what you learned, and not being able to run a controlled test to find out if the change actually helps or not. These are solvable problems, but it helps to recognize them as far in advance as possible, because we're still in the easy part.

Treat your documentation as a product

When you treat your documentation as a product, it takes on a different perspective in the eyes of the writers and every other stakeholder. As a product, it helps drive the business goals. It has performance goals to meet.

An important consideration that isn't really stated explicitly in the videos, yet still quite present, is that the documentation has its own customers to understand and please, or, more to the point, **the documentation has a different set of customer tasks than the actual product does**.

While it's possible that the documentation and the actual product are used by the same purpose, they are not used to accomplish the same tasks. You need to separate them and treating the documentation as you would any other product is one way to keep the distinction clear.

Do what's best for your documentation customer

With the documentation defined as a separate product, you can now describe its customers–who are they and what do they want to accomplish? Good questions and important ones to answer in order to please YOUR customers. YOUR customers want to BUILD something with the help of the documentation, while the product's customers want to DO something with the product. Even if they are the same people, they want to accomplish very different goals and that's an important distinction to keep in mind.

In the earlier video, the presenters talked repeatedly about concepts such as, "*getting out of the way*," "*reducing friction*," and *'not breaking the flow."* The goal of the documentation product, as they described it, was to be as informative as possible while being as un-intrusive as possible. They were quite candid about some of the struggles they had reconciling this.

It's not easy. It requires all the preceding steps to do—and, even then, it's still not easy. The way they describe it:

The goal of the documentation is to make the developer and the product it documents the star.

Your role as a writer is not to make you or the documentation the star of the moment. They describe how it took several iterations to get over that idea and break some habits.

Your [documentation] customer wants your documentation to coach them when necessary and then get out of the way (and stay out of the way) so they can succeed.

Create and support a writing process that enables contributions from outside of the writing team

These steps are a lot of work, and generally (I almost feel comfortable saying, *'Always"*) take more effort than your writing team will ever have. So, your process must enable (if not invite and encourage) help from other people.

If you're using a proprietary, "writers-only" tool, you can pretty much abandon the idea of anyone wanting to help you out with your content production and just enjoy your never-ending backlog of unfinished tasks. If you see that as a good thing because they'll just mess up all your good writing, well, you're probably not reading this post.

Both videos describe some of the ways the Twilio team facilitates and invites this support and it is worth taking down some notes.

Build the CMS that supports all of the above

Twilio's presenters described adopting and adapting a CMS (Wagtail) to do all they needed. Now, every technical writer knows that the last thing the world needs is yet another CMS, but they make a good point. If you want it to do all the things you need it to do, in your environment, with your environment's tools, and your team of contributor candidates, creating a custom CMS might be the only way to get there. It could be a valid case of, *'if you want something done right, you need to do it yourself."*

Building a CMS can be quite a challenge, not to mention, the maintenance it will require (and that your writing team will be the ones who are stuck with maintaining it). So, I would not encourage this to be approached without some serious forethought. However, if you take a team approach and can manage incrementally improving the CMS along with the documentation, if the Twilio experience is any indication, it sounds like it might work.

Must you create your own CMS? From what I've seen, if you want to check all the boxes, I'm afraid so (but, I'd be happy to be convinced otherwise). If Twilio's writers are cooking with the secret sauce, this could be one of those ingredients that ruins the recipe if you substitute it. Why? I've not seen a tool or CMS that provides all the features they described in the video AND still integrated into the existing developer tool chain and workflows.

Why not just copy them?

Why go through all this trouble of changing writing processes and tools? They have found the answer, so why not just copy/paste it?

Well, it depends on what you want to copy and paste.

What I saw in both of these descriptions of their process is that they iterated to address their product, in their market, and for their audience. **I strongly encourage writers to employ a similar *(if not identical)* PROCESS.** I would just as strongly discourage you from copying their content, style sheet, page layout and considering yourself done. While, after following their journey, you might end up with similar style sheet, content, and examples, you might not. The problem is you can't really know until you complete the journey.

What's best about all this is they are giving their process away for free! You can watch the videos. You can read their developer docs. You can see their GitHub repos TwilioDevEd, Twilio. It's all hiding in plain sight. What might not be obvious, unless you know where or how to look, is the process they use to get there, but watch the videos with an ear for "how did they do that?" and "how did they know that?" and you'll start to see it.

Don't make your documentation customers happy that they read your docs, make them happy that your documentation helped them be as successful as possible.

AGILE DOCUMENTATION, TECHNICAL WRITING, USER TESTING, WRITE THE DOCS