Putting the Technical in Technical Writer

Hi, it’s Shivank! Recently our newer writers have been asking me about how they can get more technical. It’s a good question!

Writers come from many backgrounds, but we all started somewhere and we’re all continuing to learn and grow. Keep in mind: if you’re here, you’ve already gotten started. Technical information can be intimidating at first, but software quickly rewards understanding. You’ll soon be surprised by how much you’ve learned.

I’m putting this note together as a general guide to building technical depth as a writer. I’m going to talk about:

1. Asking good questions
2. Using the product
3. Becoming an expert

Asking good questions

I always tell new writers to ask more questions, because at first I didn’t ask enough. I wanted to be considerate of people’s time, and even though I’d said the words “there’s no such thing as a dumb question” many times in my life, I suspected I might have thought of some.

Often people hesitate to ask questions because when you’re new, every question is really two questions:

1. The question
2. Should I be able to answer this on my own?

It’s easy to feel like you should minimize the number of questions you have to ask. But the more questions you ask early on, the faster you actually learn what you don’t need to ask about. So instead of being considerate by aiming to reduce your questions, be considerate by aiming to improve your questions.

You can turn every question into these two questions:

1. The question
2. What’s a better version of this question?

Here are some strategies for improving questions:

Start with what you know.
When I have to think about what I know, I often think of a more complete and useful way to describe what I don't know.

Make your best guess.
This helps me get an answer that corrects any misconceptions I might have and matches my level of understanding.

Ask the right person.
"Who is the right person?" is sometimes its own mystery, but I find it very useful to think about who I'm talking to and why I think they can answer my question. Being attentively wrong about this is how I built up my network of SMEs.

Do the reading.
A little research goes a long way. Checking the legacy docs, looking at comparable features in other products, or doing a Google search can really help me ask more specific questions and get more understanding out of the answers.

Asking questions is an important part of our job. It never goes away, so we might as well get better at it. Pay attention to good questions, helpful people, and useful answers, and you'll start to develop a knack for technical learning.

There is one more strategy I often use to generate good questions, and it's important enough to be discussed on its own.

Using the product

Use what you write about! Is it practical for me to test every feature I document? No. But every minute I spend using DataStage myself has two major benefits:

1. It gives me direct access to the user's experience.
2. It gives me a point of reference to understand what our developers are talking about.

When I became a technical writer, my software engineering background gave me some practical experience with the tasks our users carry out and made it easier for me to tell whether something was specific to my product. It was a good starting point, but actually using the product still made a big difference.

Getting hands-on will help you understand how the parts of our product fit together, how people interact with them, and what mental models our users are working with when they accomplish their tasks.

Sometimes I find people hesitate to work with the product because they're not sure they'll be able to use it right. But I actually think that's the best time to give it a try! You can learn a lot by using something wrong. My understanding of any tool tends to increase the most when I have to troubleshoot it.

Figuring out how to make it work doesn't just teach you about the tool you're using; it also builds technical problem-solving skills that carry over from tool to tool. The time I spent in college figuring out merge conflicts, segfaults, and other errors now helps me approach new problems with a sense of confidence and curiosity.

So don't be afraid of encountering errors. Plan on it!

Becoming an expert

If you do all of the above, you'll accrue technical knowledge just by doing your work well. But if you want to go beyond that and build expertise, you can take a proactive approach. Here are some suggestions.

Learn fundamentals
It's good to keep track of concepts that keep coming back. Identifying and learning important ideas is a great way of building deeper understanding, and it'll help you have better conversations. I know some of us have already picked out Kubernetes and REST APIs as topics of interest.

Keep experimenting
I try to keep an eye out for things I haven't done before. Volunteering to test something or try out a new tool helps me maintain my confidence and interest in software. Sharing my knowledge with the rest of the team really pushes me to deepen my understanding.

Do hard things
We often gravitate towards projects we know we can do easily. We all want to do a good job, and if you're less confident about a particular project it's easy to feel like you should leave it to someone else.

In reality, though, there aren't enough experts to go around. When I'm intimidated by a project, I try to keep in mind that my biggest asset is my ability to learn. When I've tried difficult things in the past, I've struggled, learned, and gained confidence in direct proportion to difficulty. Challenging myself in the present is probably going to have the same outcome.

An important thing to remember is that expertise takes time. The people I look up to have worked at getting better for decades, but at some point they were approximately where I am. I have a lot to learn, and that’s how I like it! If you feel the same way, we’re in the right place.