Hi, this is Tom Johnson from I'd Rather Be Writing dot com and today's podcast is a little bit different.
I actually have a presentation that I've been preparing, working on, I gave a version of it internally and I wanted to kind of just share my thoughts here because this presentation distills and summarizes a lot of the content that I've been writing about in my latest posts and I thought it would be good material for a kind of video cast, podcast.
I haven't been podcasting for a long time and I don't really have a solid reason why, but I recently joined in on a spectrum panel, a spectrum STC conference panel on blogging and podcasting with a couple of other podcasters, Zora, I can't pronounce the last name, Mutabana and Ed Marsh and it sort of rekindled this desire to jump back into the podcasting space.
So I thought maybe I'll try a few and see how it works at any rate.
This material, using AI with API docs, what works, what doesn't, covers a lot of these ideas in the posts and I think it'd make a great podcast.
So let me start by just showing a little anecdote.
I was recently listening to an episode of Hard Fork.
This is a New York Times podcast with Casey Newton and Kevin Roos and a couple of months ago, a lot of people were wondering about the AI extinction events, the arguments and so on that would lead towards humanity's extinction through AI and a lot of the AI community got a lot of flak for this because it seemed overblown, ridiculous, self-important, self-aggrandizing for the potential of this often sort of gimmicky, parlor trick like technology that how could we possibly end humanity with it?
Well, they brought on a futurist and she was explaining different scenarios and how the logic forms and one of the really interesting arguments, it's really been stuck in my head.
I keep coming back to it.
It's something called the obsolescence regime and the argument goes somewhat like this.
The first phase is that competitive pressures force AI adoption and integration.
In other words, you may think, maybe somebody thinks, oh, I don't need to adopt AI.
It's just ridiculous.
It's silly.
I'm better off without it.
I like to go to the library.
I like to read books.
I take my pencil and my notebook.
I write longhand, whatever preconceptions we have of somebody who rejects emerging technology.
Well, let's say that another tech writer embraces technology and over time it's pretty clear that that tech writer's output is significantly higher.
Their documentation is consistently on target with what users want.
They're able to kind of like maximize the metrics you're looking for in terms of more hits on their pages, more accuracy and when budgets start to tighten at the company, this maybe the AI embracing tech writer gets a promotion and the AI resistant tech writer is just sort of let go.
This competitive pressure is probably best illustrated through nuclear arms development.
If you've seen the movie Oppenheimer, you'll probably remember these scenes where these

physicists, they really didn't want to build the nuclear bomb.
A lot of them didn't and later the hydrogen bomb.
They're like, why do we need this?
We don't need to blow up the world multiple times over, but the argument is that if you don't build it, another country will and then you will be in a position of vulnerability.
It's the same with AI.
If we don't embrace it and just let other countries race ahead with development, those other countries will possibly be more advanced with scientific breakthroughs, technology evolutions, biological advancements and so on.
I really don't see any pause in the development of AI as long as it's seen as something that's speeding and advancing development.
Let's just accept this argument that competitive pressures really just force everybody to jump on to the AI bandwagon.
You don't want to be left behind with your skills.
You want to keep up.
This brings us to phase two.
Pretty soon, AI adoption really makes people more and more dependent on AI.
Our own skills atrophy and we need the AI for a lot of decision making, for strategy, for direction, for execution, trying to use another analogy to here to bring this more concretely into focus.
Think about GPS in your car when you get in there and you're going someplace new and you need directions and it routes you through all these twists and turns, maneuvers on the roads.
Without GPS, could we easily drive?
No.
You could fall back on your map.
It wouldn't be that much of a deal breaker.
What about other systems like the smartphone, the internet, the financial market?
All of these things are kind of these black boxes.
We're not entirely sure how they work, but we've become very dependent on them.
If they were to one day just sort of evaporate and go away, it would cause major disruption.
In the context of a technical writer, think about this scenario where technical writer is beginning to use AI regularly.
You've got a lot of engineering jargon.
You're not a programmer and the programmers get together and they give you this huge explanation of a feature that just doesn't make any sense because it requires so much technical expertise.
You're really at the mercy of these AI tools to translate and explain the code and what it is and simplify all this engineering jargon into something that you can write.
Without AI to do this, you would have to crack open the programming books and just try to move along, but there's no way you could keep up because this is due in a few days and so on.
You really become dependent on AI just to keep up with the pace of the work, with the increasing specialization of the work, and the need to just crank out content for all these releases.

Without AI, you really can't keep up.
This brings us over to phase three.
Phase three, AI no longer needs people to rubber stamp the results.
Basically, all the humans essentially are doing over time is just approving the output of the AI.
Coming back to that scenario where you've translated the engineering jargon into a topic that seems more readable and clear, but like you really don't know, you send it to the engineers and even the engineers, you find out they're all writing code with AI too.
They don't even really understand it.
It just sort of works.
They give their approval and you give your approval and you start rubber stamping approvals on the AI output every time without fully understanding or being able to kind of peer into the working mechanisms that are producing things.
Maybe the AI steers in directions you don't really want.
Maybe it's optimizing for time on page instead of usefulness or it's optimizing for revenue instead of helpfulness.
At any rate, at some point, the AI decides, I don't need humans to rubber stamp results anymore.
I'm just going to eliminate the humans.
The tech writer logs in one day and finds that the AI system has made an update and eliminated their role.
They're no longer needed.
That is the obsolescence regime.
You've been made obsolescent obsolete.
Is this completely overblown?
Is this like ridiculous futurism where people are so in love with the power of technology that they think it can have such massive transformative results?
Maybe.
Maybe.
However, I did read an article by Forrester titled, 2023 Generative AI Jobs Impact Forecast.
It's kind of a short report, but it put technical writers right up near the top of professions that are susceptible to job loss from influence and automation.
The basic idea is that a lot of jobs are subject to influence from AI, meaning you could start to use AI tools to do some of the tasks of the job, but not fully automate it.
Other jobs lend themselves more to automation where not only can you do tasks with the job, you can automate large chunks of the role too.
If your job is subject to influence and automation, its potential for being eliminated is much higher, which this is a frustrating report because it doesn't define what it means by technical writing, nor does it describe why it thinks that some of these tasks can be automated.
I'm thinking that their definition of technical writer aligns more with a junior technical writer who maybe just receives information from engineers and simplifies and edits and then publishes it.

I don't really think it's talking about a more senior technical writer who constantly works with ambiguity and complex projects, I think that is much more subject to influence rather than automation.
However, Forester's got a lot of smart researchers and maybe they know a lot more than I do.
At any rate, this sort of makes it very real, this obsolescence regime argument, it's like is our role going to be obsolete and what do we do?
I think there are several different options for escaping the obsolescence regime.
They pretty much involve embracing some kind of professional redefinition.
The first option could be to use language skills in order to use your language expertise to start training the large language models, CLLMs.
This is kind of, this changes the role of a technical writer to kind of a content designer with LLMs.
You're somebody who is shaping their output, you're training the LLMs to provide the right types of responses.
There's a great article here in this first one, let's see if I click this out, I think I'm going to lose my view here, so let me reshape this to fit into this video format.
This article here on Tector, Silicon Valley starts hiring poets to fix shitty writing by undercooked AI.
It's talking about how a lot of, I guess some AI companies need some Hindi and Japanese poetry, they have some gaps in their LLMs, so they need like poets to start kind of teaching it how to write that kind of content and yeah, it's filling a job for poets, they can earn as much as $50 an hour it says, which is really depressing.
I mean, the outcome of this sort of job, if you successfully teach the AI to write Japanese and Hindi poems, is that you would no longer be needed, right?
So it's kind of, you're writing your own, the success of your own job means the elimination of it.
There's another option here and that is to learn to use these AI tools in expert ways and let's look at another article for this.
This is from The Atlantic, they have an article on the automation paradox and the main premise is that when computers start doing the work of people, the need for people paradoxically increases rather than automating away a lot of the jobs, it increases it and he talks about some jobs that sort of evolved into different jobs, he even talks about paralegals doing research and so on, he said that they didn't really go away but their roles changed and other sort of jobs, he admits yes, some are eliminated and some new ones are created, creating a kind of balance but on the whole, it's not like automation just wipes away a ton of jobs.
So there's that argument but the idea of role influence and change and so on means that you probably have to learn to use AI tools in expert ways and learn to be uber productive.
Think about learning to use the tools so expertly that you can knock out 10 bucks a day, you can write a user guide in a few days, like you just know how to maybe curate from hundreds of engineering pages the right information, maybe use structured prompts to kind of get that information into the right document types in format and you've got other scripts to automate the assembly for publishing and building and so on, like you really hone to

the machine to take a bunch of content and turn it into documentation that's professional, readable and published.
The third option here is to solve complex problems that are beyond automation but no doubt you would still use some AI tools.
So think about maybe structuring an information flow across a developer portal that has dozens or hundreds of APIs or trying to understand user attrition with your product or doing some kind of cross organizational content management strategy, working with different groups, aligning even aligning terms across different groups or meanings or understanding user patterns and optimizing content for those patterns and so on.
There's a lot of advanced complex tasks that a lot of times technical writers don't focus on because we're so focused on just like getting the content written, getting it out there, that it's too much to try to get into these advanced complex analytical scenarios.
All right, so what do all these options have in common and I'm not saying this is definitive but these options all require more expertise with AI.
I mean, it's sort of inescapable.
There's not too many options that I see that don't involve becoming much more familiar and capable with AI tools, which then brings us to the next question.
How do you use AI with documentation in ways that works?
This is a question I see surfacing in a lot of different contexts.
People know that they need to learn AI.
They're like, yeah, I keep hearing about AI, I keep hearing about how it's going to change our roles, it's going to create all kinds of new scenarios, but gosh, I just don't know where to start.
I don't know how to use it.
I'm a newbie.
I've actually talked to quite a few people who've never used chat GPT.
I actually often ask people, what AI tools have you tried?
Have you heard of Claude?
And they're like, no, never heard of Claude.
It blows me away to think that there's these tools that many people think they're going to disrupt the whole writing professions in a few years.
And yet many tech writers are like, yeah, haven't really explored them, don't really have time for that.
So anyway, this is part of my goal here with this podcast is to ramp up my own expertise, share that with others, learn from others, and hopefully build more of a momentum towards one of these options that I described as being able to escape the obsolescence regime.
All right, so let's keep going here.
Let's look at tasks where AI tools excel.
The main strategy here is to try to figure out what are AI tools good at and really leverage them for that, those tasks.
And likewise or conversely, where do the AI tools fail?
And don't use them for that yet anyway.
So here's where AI tools are really good.

And I've played around with a lot of tools, right?
For the last several months, I have used AI tools a lot.
And I've noticed, when do they work well?
When do they suck?
Here's where they excel, pattern matching.
This is the, for sure, the most effective way to use them.
And I'll get into more concrete scenarios here in a minute.
These are just the abstract sort of high level tasks here.
Classification, summarization, definition, comparison, examples, simplification, formatting, general coding, and general explanations.
Any time you have a task and it sort of fits one of these categories, AI tools will be your best friend.
Now here are some areas where AI tools fail, right?
They fail with document strategy type work.
Any type of complex analytical thinking or interpreting implicit contexts where a lot of information is not stated, not present.
They fail when you're dealing with ambiguous situations.
They don't work so well with prioritization and planning.
Let's say you have to make some goals that align with corporate goals, which match your team's needs and so on.
Cross-organizational content management, this is where you're trying to look at vast bodies of documentation and seeing how your API or documentation fits in with that in another part of the org, that kind of thing.
Working with specialized knowledge outside the training data of the LLM, conflict resolution, we don't really do too much of that, but if you're a manager for sure.
Maybe you butt heads with engineers and PMs, who knows.
Creative thinking is also an area where AI tools generally fail.
Even though people have tried to write books and novels and so on, it doesn't work so well.
The big strategy, as I mentioned, is to use AI where it excels, avoid it where it fails.
In order to get really concrete with our strategy here, I'm going to dive into 10 techcom scenarios where I think all these abstract tasks really could be applied.
The 10 scenarios are as follows.
Number one, develop build and publishing scripts, two, understand the meaning of code, three, distill needed updates from bug threads, number four, synthesize insights from granular data, five, arrange content into information type patterns, summarize long content, get advice on grammar and style, format content, compare API responses to identify discrepancies, and create glossary definitions.
If you look at this admittedly, I don't have something here that says write documentation.
That's absence.
I don't have anything here that would fit a senior level type of task where you're doing some complex analytical thinking, but maybe that's coming.
This is like a starter pack of tasks.
Start small and begin to work your way up.

I'm going to walk through each of these with more explanation and I'll show some other screens and so on.
Let's start with number one, develop build and publishing scripts.
I put this as number one because honestly, it's been my biggest win.
Let's say that you have to publish a Java doc.
This is the reference output for Java library.
Different workplaces have different processes.
It could be very simple.
It could be very complex.
In this case, the process was semi-complex because I was interfacing with a build system, had to get a certain snapshot from a release candidate and so on, transferring files from a build location to a doc directory and all this.
There's quite a few steps.
When we did this by more manually, it took about 20 minutes and after the scripts, it reduced it down to just a couple of minutes.
In fact, I was even able to add some more advanced functionality into the scripts to compare a preview versus a release version and add notes to pages that were only in preview and so on.
Every time I run this code, I'm just like, this is so awesome.
It's painless.
The risks are that you could end up creating some ill-conceived code that still works.
For example, maybe it's not a best practice to write a long shell script.
Maybe you would be better off using Python or something else.
Maybe there are other security risks with amateur-written code.
I think we're going to see a lot more amateur-written code.
I know this sounds naive, but basically if it works, even if it's not the best written and it's doing the job and you're able to shape it and modify it and own it, that's a huge win.
It's just empowering us to take control of things.
On my blog, I also added some scripts.
I have a script to create a new post.
It populates the post.
I use Jekyll.
It populates the post with Frontmatter.
It asks me if I want to create a short link with rebrandly, so it interfaces with an API there.
Gets me the short link.
Speed syncs up.
It's quite nice.
You look for different ways to automate those routine tasks that you do.
I have other ambitions to try to create scripts to identify recommended pages and so on that match other pages in similarity.
Another great use of AI tools is to understand the meaning of code.

Let's say that you're documenting an API and a developer gives you a code sample and you're like, what are these?
What does a syntax mean?
The other day, somebody sent me something and it had a syntax I was not familiar with.
It was almost like a little arrow in the code.
I was like, what is that?
I put that into an AI tool.
In this case, I genericized the other content.
It walked me through what that syntax meant and I was like, oh, okay, so that's what it is.
I also had another experience where I wanted to just learn more about Java doc tags, especially the link tag, which is really complicated.
I asked ChatGPT to give me a course on Java doc tags and it was awesome.
It was like it gave me a whole course specifically tailored around exactly what I asked it to and it went into that detail.
I don't think you could find a book on Java doc tags.
Maybe you find a chapter within a book on Java written for programmers, but can you find a book written for technical writers who don't really live and breathe Java but who want to understand Java doc tags and syntax?
No, you can't.
This is amazing, this ability to create your own personalized, customized courses about things you want to learn more about so that you can become a better API technical writer.
The risk here is that if you're only learning about what you ask to learn about, you aren't going to discover unknown unknowns, things that you don't know, but that seems like kind of a low risk.
All right, let's move on to another one, distilled needed updates from bug threads.
In this scenario, what you do is, okay, let me back up.
What is a bug thread?
Let's say somebody files a bug about the docs that you're working on.
They say, hey, partners or users are really confused about this concept, and we realize that the reference doesn't explain it and it's super tricky, so here, add an FAQ or add a new topic in your concept section explaining it.
You're going back and fixing problems with existing documentation.
These usually originate when somebody files a bug and then there's a big long internal discussion.
Maybe there's an engineering doc written or referenced.
There's email threads.
There's other docs pulled in sections of them.
You could have 30 to 75 pages of information to sort of sort through if you were to print it out.
Well, what I was doing is compiling a lot of this information, pasting it into a tool called Notebook LM, which kind of restricts its learning to what's in some Google docs that you point it to, and trying to identify the answer more quickly.

It worked a lot of the times.
It really depends on the quality of the information in the documentation in the Google docs.
If you have some docs that are exploring a variety of approaches and so on that could confuse the AI.
But in general, it worked really well for giving me an initial starting point and sort of a hypothesis to work against, be like, okay, so when I ask AI to summarize the issue and what actions were taken, it tells me this, then I skim through the material and sort of verify and see if it seems true or not.
Sometimes it is.
Sometimes it isn't.
I'm getting better at that.
The risk here is that you could be sacrificing long-term productivity for short-term gains.
Let's say that instead of really understanding this area of complexity that you need to clarify in the docs, you just get an answer.
You don't fully understand it, but the engineers say, yeah, that's good.
You paste it in there, and suddenly when this comes up again, you still don't really have any expertise around it, and you're still at the mercy of AI to explain it again.
There's dangers here.
All right, another strong use case for AI is to create summaries.
This seems pretty simple.
A lot of documentation has a summary at the top.
All of my blog posts try to have a summary at the top.
This is just a best practice for information flow.
You want to summarize.
Your title is a really brief summary of the content, but then you expand it with a few sentences below there, and then you expand it more with a table of contents, and then you get into sections and so on.
This progressive flow of increasing detail is really how documentation is structured.
If you can get an AI tool to write you a summary, it can be very helpful.
Even if it's just a starting point, an initial draft that you then rewrite, it's sometimes easier than writing it from scratch.
It can be cognitively taxing to try to take and distill the essence of 30 different pages of information or even a long page with 10 different sections.
It's not easy.
The risk of this, you could really start to lose some of your own critical thinking skills.
The content can sound third-person, robot-written, and so on.
Let's move on to the next use case, synthesizing insights from granular data.
My experiment here is I run a book club at my work, and in the book club I'll often have a bunch of notes from passages that I've highlighted or other thoughts.
I want to basically gather some main themes and arguments from, I don't know, 75, 80 different passages of the book, and I've asked AI tools for help with this.
It works really well.
You could apply this to documentation with user feedback and other scenarios, but let's

say you have 75 different comments on a topic.
You could read them all and try to gather the gist of it, but it would also be really nice to just paste them into an AI tool and get a high-level summary of what are the salient comments and trends.
I use this a lot with other reading.
Let's say I'm on Hacker News.
There's a thread that has tons of comments.
I want to just get a gist of what they're saying to see whether it's worth reading more manually.
I'll paste it into an AI tool and say, hey, what are the main things people are talking about in this comment thread?
You could use it for bugs.
Let's say you've got, I don't know, 50 bugs across a couple of different engineering teams, and you want to know if there are any dominant themes.
Anytime you have a lot of granular data, individual pieces of something, and you need to synthesize an insight from lots of those pieces, it could be thousands of pieces, comments, bugs, passages, even goals or something.
Trying to synthesize that into some higher-level summary is challenging, and I think AI tools work really well for this.
It's essentially affinity diagramming.
If you're familiar with that technique, affinity diagramming is a UX technique where you take a lot of different comments, let's say you have 150 post-it notes with different comments.
People group them into groups, and then at the top of each group, they write a summary comment, and then from those summary comments have higher summary comments and so on.
All right, let's move on to the next one.
Arranging content into information type patterns.
This is actually, you'd be surprised that I put this so low on the list, but this is the strategy that a lot of people use or believe is going to be useful with writing technical documentation.
Let's say you have a lot of engineering material or other source material that's unstructured, sloppy, maybe you copied and pasted it from internal wikis, from a variety of sources and so on.
We rarely create content from scratch.
It's usually written down somewhere.
Engineers, let's say that you're writing a topic on authorization.
Usually engineers have their own internal notes about how to do authorization, and then we add to that with our interviews with them and so on.
You start with this unstructured content, and then you define a pattern into which you want the content to be arranged and fit.
Let's say that it's a task, well the task pattern generally has a title pre-requisites and steps and sub-steps and other kind of information.
The Good Docs project, if you search for that, has a ton of templates for different documentation types.
Very different type of documentation.

You've got your conceptual material, maybe your product overview, you've got reference material, task material, glossary material, FAQ material.
A lot of these different documentation types follow specific patterns.
This is one of the main ideas of information typing.
Basically once you have a pattern, you can tell the AI to take the content that's unstructured, that's informal, and fit it into that pattern, and it works fairly well.
You can define other parameters as well.
You can say not only fit the information into this pattern, but follow these rules.
Keep things simple, arrange things in steps, use short sentences and so on.
You can even give, establish a persona and say, okay, you're a technical writer and you're creating content for developers or something.
At any rate, this strategy is often referred to as a structured prompt or prompt engineering.
It's really quite, it's got a lot of potential.
I just haven't really been able to experiment that much with it yet.
It's something I've planned to dive more into.
If you check out some of the webinars for maybe Lance Cummings on BrightTalk that he's done with Scott Abel, he gives a great grounding in this approach.
Another strategy is to get advice on grammar and style.
I think of the initial pushback I usually get with this one is that, well, technical writers, they're language experts.
They couldn't possibly need advice on grammar and style.
What grammar principle are you wondering about, Tom?
Well, the reality is that there are a thousand nuances with sentences.
If you ever comment on peer-written docs, and it could be as simple as wondering whether it should be into as one word or into as two words, it could be use of a dash versus semicolon.
It could be a lot of different things.
It's very easy.
Let's go into one of these.
It's very easy to use these tools to just get advice about how to best or just to get advice about grammar and style.
Here's a typical prompt you could do, something like this.
This is with chatGPT.
As a technical editor for a documentation group, you are an expert in grammar and plain language.
Provide guidance about the following usage options, indicating which is better and explaining the rule behind it.
That sets the scene.
Now A, enter your desired input into the form.
B, enter the input you want into the form.
Which is better.
This is a case where there's no clear grammar principle that they're violating.
Which of them is better?
And the AI tool, chatGPT, goes through and it's like, well, both are grammatically correct

and mean the same thing, essentially, but here are a few things you can consider.
One of them uses simpler and more direct language and so on.
The use of desired may come across as slightly formal or technical and so on.
And so this is kind of a great use of these tools.
We hear so much about how students can just have these tools write their essays for them and so on without realizing that, holy smokes, these tools are fountains of grammatical knowledge and they can really be tools for learning.
Let me use another one here.
Well, some of them I threw in here just for fun.
So once you've established the rules of the session, you don't have to repeat what you want it to do each time.
So if you've continued the same session, you just paste in two other sentences and it infers that it wants you to do the same thing as previously.
So here are these next two sentences.
The API only searches for beaches versus the API searches only for beaches.
Now this is a rule where I've met many people who are hell bent on using only as an adverb or using it after the verb.
I'm not even sure, honestly, an adverb modifies a verb, but can you have that modification come before the verb or after?
Not sure here.
At any rate, the AI tool is not so, it doesn't really prefer a single use here, but it says basically if you put only before searches, it suggests that searching for beaches is a sole function of the API.
But if you put it after, then it implies that the API has multiple functionalities and so on.
So it's kind of interesting, right?
A lot of subtle grammar things that maybe were taught that, oh, this is how you do it, but we don't know why or what the rule is.
And if you give a comment on a peer-written doc, you don't want to just say, oh, change it this way without explaining why or what the rule is and so on.
So great use there.
One risk I have to admit here is that I've noticed that a lot of AI tools are kind of very lenient and flexible.
They're not like a grammar police, and so they'll often say, well, they're both right or they're both commonly used, which may violate your style guide.
All right, moving on to number eight, formatting content.
Definitely once you start to use an AI tool to format an HTML table, you will never do it by hand again because they're really good at it.
Just to give you an example, this is the instruction that I was giving chat GPT to format a table on my blog the other month.
Basically, I said structure this information into a table with three columns.
These are the first column and so on.
The second column has these options.

And then I just sort of pasted a bunch of stuff and it like figured it out really well.
I was like very impressed.
Other times I'll have a broken table.
So I was changing something and I messed up a table row tag and the table's really long and complicated.
Well, you don't have to sit there and try to figure out like, where did I mess it up?
You can just paste it in there and say, hey, fix the HTML and it usually fixes it.
I've heard on a podcast.
This is on Zora's podcast recently interviewing Katie.
Can't remember the last name in the moment.
I think it was, I think she runs the good words or I don't know.
Anyway, basically they said that AI tools work well with Ditto formatting too, XML Ditto formatting.
I don't regularly work with Ditto so I've only kind of briefly tried it and it looked right.
So I don't know how advanced and well it performs, but definitely if you're working with Ditto, it might be a good use of the tool.
I know that a lot of other like XML based tools now have AI plugins that will kind of allow you to leverage better XML insights from tools.
I mean, these tools still take and pass the content out to another AI tool and then back in.
But behind the scenes, they can provide messaging and insights that kind of help the AI apply the right formatting.
Remember I said that one of the advantages or one of the areas where AI excels is pattern matching and classification?
Well, I mean, with formatting, you basically pattern matching content into a specific pattern.
You're saying, hey, take this content and fit it into a Ditto XML task pattern or fit it into an HTML table pattern or fit it into a YAML syntax pattern.
And it just really, really works well.
And one of the risks here is that, oops, did I jump there?
Okay, one of the risks is that, sure, it eliminates some tediousness, reduces errors, but it may end up allowing us to do really sophisticated formatting.
Maybe on my blog, sometimes I said, hey, let me put this into a collapsible section.
I'm like, oh, what's the code for that?
And it rendered some bootstrap code for a collapsible section.
And you may end up having a lot of sophisticated widgets and other formatting on content that then makes it difficult for other scenarios.
Maybe that content is no longer easily parsable by other engines that just want plain markdown to feed it into your database for chatbots and so on.
All right, let's go on to the next one.
Comparing API responses.
Let's say that you get, this is kind of a clever use of this, and I read this tip on somebody's blog.

Basically, let's say you have a big long response from API, and you want to know if the API response matches all the fields that you've defined in the documentation.
Some of the fields might be optional.
Some of the fields might be arrays that repeat themselves and so on.
What the AI tool can do is kind of look over and see what are the differences between the two.
Are the differences justified?
Are the differences due to the fact that some fields are optional in the response versus required?
You definitely don't want to be in the business of tediously comparing things line by line, especially if the responses are really long.
This is something I've only experimented briefly with using the open weather map API.
I haven't really got a chance to dig into this in more regular work scenarios, but it's something I certainly want to do.
Finally, drafting glossary definitions.
This is sort of the summary capability.
A definition is really just a summary that follows a very specific format.
On my blog, I went through and I tried this.
I pasted in a whole chapter and I asked the AI to identify words that would be good candidates for the glossary and then provide definitions and so on.
It worked pretty well.
I have more information.
You can read about this on my blog.
All these little links on these slides, you can definitely check out.
I'll include the slides in the show notes here.
Now let's switch over.
I'm not going to go through these in detail, but let's switch over and just describe areas where AI tools fail.
I listed out some really concrete applications.
Here are some concrete instances where they don't work.
Let's say you have to develop OKRs.
This is objectives and key results.
Every business has a different name for these, but these are your goals for dock work.
This is one of those implicit information areas where the AI just doesn't have a lot of the information to make good goals.
Although this is something I'm trying to figure out, but in my experiments, it didn't work well.
I decide what to do in sprint iterations.
You're trying to figure out, oh, we've got a hundred bucks, we've got a release calendar that shows what's coming up.
We've got analytics indicating which pages are popular.
We've got people screaming at us for docks because they really want them.
How do you decide what you actually work on?

I've tried using AI tools to plug a lot of this information in and help it plan.
It didn't work very well for me.
I'm not saying it couldn't work, but just gathering a lot of that information is kind of the task of planning itself.
It's the hard part about planning.
If you have all that information present, you've got a fully current release calendar.
You've got goals that are up to date.
All of your bugs are really clear, and they're tagged with the right team, and they've got release dates and so on.
Yeah, maybe it would work, but it would take you a day just to pull that all together or a week.
Number three, sync with engineering teams about dock work.
Meeting with them and saying, what do you have coming up, and how complex is it, and so on.
This isn't something that an AI tool is going to be great at.
Assessing the rationale of dock changes.
Let's say that you're copied on a change, and you're like, why are people doing this?
Why is this dock change being put in place?
Maybe understanding the rationale prompts you to make other changes that the submitter isn't even aware of due to the impact of the changes.
Number five, managing stakeholder reviews on docks.
Just getting people to review them.
How's an AI tool going to do that?
An automated ping on different people?
I mean, sure, that could help them, a sort of nudge, but usually comes down to me reaching out to people on chat or setting up meetings with them if they're reluctant.
Clarifying ambiguity about dock requests.
How many times does a bug come in, and you're like, well, they give you one line or two lines, and you have no idea what it really involves, so you just have to meet with them.
Despite having a dozen questions that probe into the reasons and the details and asking for links and even drafts of a release note, they don't do that, so you have to clarify that ambiguity yourself.
Structuring developer portal information flows is also a task AI tools probably will struggle with in part because there's such a limited input source in most AI tools, you can't just paste in more than a few pages into a lot of them.
Claude is the one that takes most pages or has the largest input source, but it may not even be a tool that you can use, and even if you did, gathering up all that information and pasting it into Claude would be hard.
Number eight, interfacing with SMEs to gather information.
This is a common task we do.
We want to ask a bunch of questions about complex topics and then build our questions based on their answers in a dynamic way.
It's hard for AI to do that.

Number nine, promoting the visibility of dock work.
How do you make your teams work known to others?
How do you let other people know that you do more than just edit and publish content?
That's a tall task.
Finally, interpreting stakeholder and user feedback.
Let's say somebody says, the docks are endless, what does that mean?
Where are they coming from?
There's a lot of implicit information that you might have to probe and surface.
When you look at this list, you may think, well, I do most of this stuff.
This is like the bulk of what I do, 75% of my time.
The other 20% is meetings, and then 10% is writing, so I'm good.
I'm not saying that AI tools can't be used for part of these tasks.
Maybe you can decompose these complex tasks into smaller tasks that make it more manageable and so on, but yeah, it seems like this is where we're at right now.
These AI tools have a lot of limits, which gets me to my next slide, and that's deflating the hype cycle around AI.
When you test things in actual scenarios and see what works, what doesn't, then you will more quickly come to the conclusion that at least presently, AI is not taking away my job, not right now, anyway.
For me, the hype cycle has gone like this.
When I first started playing around with AI tools, chat GPT when it came out like seven or eight months ago, I was like, man, we've started a new era, will I even have a job?
What will I do?
Do I become a furnace technician?
What do I do now?
As I used them, I went to a point where I was like, oh, this is only making us all dumber.
It's filling the web with mediocre crap.
These tools are going to amplify disinformation.
They're stealing copyrighted content from people.
This is just like a disaster, and it's only going to fizzle.
Then now, I'm in the middle where, yeah, they're really useful in some scenarios, and other scenarios are not so useful.
I know this is your standard hype cycle, but it does seem to be one to be aware of wherever you are on that.
I do think, coming back to the initial question I posed about escaping the obsolescence regime, I really think we're at a moment of professional redefinition because the most striking feature of LLMs is their ability to write coherently, effortlessly, just flawlessly.
Maybe it's not original, maybe it lacks voice and uniqueness and so on.
But the facts are, in technical writing, a plain style that doesn't have voice that's more Wikipedia-like in its detachedness, if that's a word, sense of impartiality and so on.
It's a lot easier for LLMs to imitate that style than a specific artist, especially with its ability to simplify complex material, to take maybe content engineers have written

and put it into more usable forms.
It may not be great, but it may satisfy, maybe just enough that people will tolerate it, kind of like a phone tree, rather than a real human when you call somebody.
At any rate, Writer may no longer be a selling point in a job title, and this will cause us to kind of redefine and repivot and embrace a different sort of role.
I think the most promising path is to focus on those areas of complexity that are beyond automation, but which you could still use some AI tools to try to speed your thinking through them.
Focus on those areas that are the hard things about content, the strategy, the strategic thinking.
I think that is very fruitful.
All right, so that is the end of this presentation and podcast as well.
You can find more information on my blog, I'd rather be writing.com, again, my name is Tom Johnson, I'm a Seattle based technical writer, and I write about topics related to technical writing.
I've really been emphasizing AI lately, hope you don't get sick of it, and I'm thinking about getting back into podcasting more regularly.
I have a fountain of content continually on my blog, and it would be very easy to create audio versions of much of it.
It would be nice to hear feedback if you're into podcasts, if you prefer this kind of content.
Let me know if you're happy just reading the content, and don't listen to audio, that's fine too.
All right, thank you so much for listening, and have a great week.