# 10 Expert Prompt Templates I Use as a Developer and Writer

*Not just hacks - these are the prompts that actually get the job done.*

CodeWithYog • 9 min read • 2025-07-25
https://generativeai.pub/10-expert-prompt-templates-i-use-as-a-developer-and-writer-d6e97275edff


*Photo by mingxi liu on Unsplash*

Non Medium member, click here.

Forget everything you think you know about “prompt engineering.” The secret isn’t writing a smarter prompt at all. In my case, it was writing a dumber one - just in a totally different way.

When I first started using ChatGPT, I thought I had it all figured out. I’m a software engineer by trade and a writer by passion, so how hard could crafting a few AI prompts be? I’d pop in a question or task, get a quick answer, and pat myself on the back. For a while, that felt *okay*. But then the novelty wore off and I noticed something: the responses were **bland**. Too safe. Too “default.” They gave me the right facts or a decent outline, sure, but they lacked the spark I was looking for.

I kept wondering, “Why isn’t ChatGPT giving me what I *really* want?” I tweaked wording, added fancy vocabulary, even tried overly complex instructions hoping to sound “smart.” Spoiler: that wasn’t the solution. In fact, after *hundreds* of failed prompts (think *1,200+ misfires*), it finally hit me: The AI was doing exactly what I asked - the problem was me. I wasn’t asking the right way.

## I didn’t start out as a “prompt engineer.”

I’m an AI and C# developer. I write technical posts on Medium. I test APIs. I debug deadlocks. I fight with nullable reference types.

But over time, I realized that how I talk to AI tools - ChatGPT, Claude, even Copilot - can save me hours or waste my day.

What started as casual “help me write this” moments evolved into a serious part of my workflow. Especially when I found myself getting better results, faster answers, and smarter code. Not because the model got smarter - but because I started asking better.

In this post, I’m sharing the **10 exact prompts** that I’ve shaped and reused across projects. Some were born out of frustration. Some came from late-night Stack Overflow spirals. All of them are real.
And more importantly: **they work.**

Let’s get into them.

## 1. Explain This Code Like I’m New

I use this when reviewing someone else’s code or revisiting mine after a few months. Sometimes, you just need a clear, conversational breakdown - like a teammate walking you through it on a whiteboard.

> ***Prompt:***
> “You’re a senior C# developer. Explain the following method like you’re mentoring a junior dev. Go section by section. Use plain English and assume basic C# knowledge. End with a summary of what the function does.”

```
// Paste your method here
```

## Why it works:

• You’re giving the model a role: a senior dev explaining something.

• You’re defining the audience: not a beginner, but not an expert.

• The output is usually human, not robotic - and you can often reuse it in documentation or comments.

I don’t just use this to understand code. I use it to teach others what the code is doing, without rewriting it by hand.

## 2. Generate Tests (The One That Saved My Week)

Let me tell you exactly when I built this.

We were prepping for a release. I had just finished writing a price calculation module. Business logic was complex, edge cases everywhere. And guess what I hadn’t written yet? Unit tests.

It was Friday. My brain was mush.

I dropped the main method into GPT-4 and used this prompt:

> ***Prompt:***
> “I’m using xUnit to test this C# method. Generate 5-6 test methods that cover both normal usage and edge cases. Use meaningful test names. Add comments explaining what each test is verifying.”

```
// Your method here
```

It gave me four solid test cases and reminded me of a case I'd missed: what happens when the `customerLoyaltyYears` is negative?

## Why it works:

• You name the testing framework. That matters.

• You specify what to test: happy path + edge cases.

• You ask for comments, which turn it into learning material.

This one prompt saved me at least 90 minutes that day. It's now part of my coding routine.

## 3. Clean Up This Ugly Code

Sometimes you finish writing a method and immediately feel gross about it.

It works. But it's ugly. Bloated. Repetitive. Maybe it mixes too many concerns.

This is when I turn the AI into a refactoring buddy:

> ***Prompt:***
> "Refactor this C# method to be easier to read and maintain. If needed, split it into smaller functions. Improve variable names. Keep the logic intact. After refactoring, explain what changed and why it's better."

```
// Your ugly function
```

## Why it works:

• You're not just asking for new code. You're asking for improvement.

• The "explain what changed" part helps you actually learn, not just copy-paste.

• The result is almost always cleaner and closer to SOLID principles.

I've used this on everything from API controllers to batch processors. It won't make your code perfect - but it gets you 80% of the way there.

## 4. Review This Before I Ship It

You don't always have a team to review your PR. But that doesn't mean you should commit unreviewed code.

When I want feedback on structure, naming, performance - or just a sanity check - I use:

> ***Prompt:***
> "Act like a senior developer reviewing this code. Give bullet-point feedback on correctness, efficiency, naming, readability, and best practices. If anything could cause bugs or be simplified, point it out."

## Why it works:

• Clear roles: senior dev doing a code review.

• Clear categories: not just “does this work?” but “is this good?”

• Output is usually fast, blunt, and useful.

This prompt has caught:

• A variable that could be null (and wasn’t being checked)

• A loop that could’ve been a LINQ query

• A bad method name that I was too tired to notice

Use it before every commit. It’s faster than waiting on a teammate - and almost always right.

# 5. Write Documentation from a Code Comment

We all write ugly, short comments like this:

```
// Gets all users with active subscriptions
```

But when I need actual docs for an endpoint or function, I use this:

> ***Prompt:***
> “Turn this comment into full documentation. Include purpose, parameters, return type, and one example usage. Assume the audience is a developer new to the project.”

```
// Gets all users with active subscriptions
public List<User> GetActiveUsers() { … }
```

## Why it works:

• It expands your shorthand into something maintainable.

• You get back XML-style summaries or markdown-ready content.

• If you mention “example usage,” it adds a code sample too.

I use this in tandem with prompt #1 to write docs from real code, especially when prepping for handoffs or onboarding.

# 6. Summarize This File for Me

Large files are exhausting. Especially when you’re trying to make a quick change.

Instead of scanning for 20 minutes, I use:

> ***Prompt:***
> “Give me a summary of what this C# file does. List each class and method with a short description of its purpose. Keep it under 150 words.”

```
// Paste the file or large part of it
```

## Why it works:

• Great for onboarding (especially on legacy codebases).

• Helps me know “what does this even do?” before editing.

• Clean enough to copy into a README or wiki.

You can run this on each service layer file to build lightweight internal docs without doing it all by hand.

## 7. Help Me Fix This Error

You could Google that stack trace. Or paste it into an AI that’s read 10 million of them.

> ***Prompt:***
> “I’m getting this C# error. Can you explain what it means and what usually causes it? Also suggest 1-2 ways to fix it.”

```
Error: Object reference not set to an instance of an object
// Paste the related code here
```

## Why it works:

• You get more than just a fix - you get reasoning.

• Often includes variations of the fix: guard clauses, null propagation, logging suggestions.

I use this during production fire drills when I need clarity, fast.

## 8. Brainstorm Blog Post Titles That Don’t Suck

I’ve written enough Medium posts to know: titles matter. A lot.

If I have the post written but the title feels flat, I use:

> ***Prompt:***
> “I’m writing a blog post for Medium about async/await in C#. Suggest 5 strong titles - 2 that are punchy and clear, 1 that’s funny, 1 list-style, and 1 curiosity-driven.”

## Why it works:

• Forces range: different types of titles instead of five boring ones.

• The curiosity-driven ones often spark *my own* ideas.

• The funny one? Even if I don’t use it, it makes the editing process more fun.

Use it. Test multiple. Pick the one that makes you click.

## 9. Structure This Article for Me

Some days, you have an idea - but the structure isn’t there yet.

This helps me go from a topic to a real outline:

> ***Prompt:***
> “I want to write a blog post titled ‘Why C# Developers Should Care About Dependency Injection’. Give me an outline with: an intro, 3 main sections (with subpoints), and a brief conclusion. The tone should be clear and helpful, not salesy.”

## Why it works:

• You’re setting the topic, the title, the tone, and the structure.

• What comes back is usually a usable skeleton I can start filling in.

• Often includes transitions and a suggested call to action.

This has saved me from staring at blank Google Docs more times than I can count.

## 10. Make This Paragraph Sound Human

Last but not least - the polish.

Sometimes, I write a good paragraph. It’s clear. Technically correct. But it sounds... stiff.

Here’s what I send:

> ***Prompt:***
> “Rewrite this to sound more natural and human. Use contractions. Vary sentence length. Keep it professional, but less robotic. Don’t add or remove info - just rewrite tone.”

```
In conclusion, using async and await improves code efficiency and readability by reducing blocking operations.
```

## Why it works:

• Adds warmth, rhythm, and flow.

• Keeps your original idea, just dresses it in better clothes.

• Especially useful for intros, conclusions, or summaries.

This is often the final step before I hit “publish.”

You don’t need to be a prompt engineer to get better answers from AI.
You just need **better prompts**.

| Technique | Example Prompt | Why It Works |
|---|---|---|
| Assign a Role | Act as a veteran software engineer. Explain blockchain to a junior developer. | Sets tone and depth; makes answers sound expert and confident. |
| Provide Context | I'm a college student struggling with time. Why do I procrastinate, and how can I fix it? | Makes the response personal and relevant to your situation. |
| Specify Format or Style | Give me three YouTube ideas in a bulleted list with a one-sentence description each. | Shapes the output format to match how you want to use the info. |
| Simplify the Complex | Explain inflation in plain English, maybe with a simple analogy. | Breaks down complicated ideas into something you can actually grasp. |
| Let ChatGPT Ask Questions | I have an idea for a novel but it feels undeveloped. Ask me 5 questions before suggesting anything. | Turns the AI into a thoughtful coach or brainstorming partner. |
| Iterate Your Prompt | Make that more casual and mention that I love outdoor photography. | Small tweaks refine the AI's answer until it fits perfectly. |
| Show an Example | Here's my format: Q: What's your favorite movie? A: I absolutely love Inception... Now answer 3 questions in this style. | Gives the AI a clear pattern to mimic, reducing confusion. |
| State What to Avoid | Explain quantum computing in simple terms, and avoid any equations or heavy math jargon. | Prevents the AI from including content or tone you don't want. |
| Ask for Multiple Options | Give me 5 blog post titles about AI trends in 2025. | Boosts variety and helps you explore more creative choices. |
| Prompt to Create Better Prompts | Help me write the best prompt for getting a friendly yet professional rejection email. | Uses AI's own knowledge of prompt design to craft better queries. |

*P.C Author*

These ten are my daily drivers - in real development, real writing, and real deadlines.

They save time. They improve quality. And most importantly, they help me **think faster**.

Start with one. Tweak it. Make it yours.

And when you hit that moment where the prompt gives back something better than you expected?

That's when you know you've stopped using AI - and started collaborating with it.

Generative AI

This story is published on Generative AI. Connect with us on LinkedIn and follow Zeniteq to stay in the loop with the latest AI stories.

Subscribe to our newsletter and YouTubechannel to stay updated with the latest news and updates on generative AI. Let's shape the future of AI together!

## Other mentions by Author

• generativeai.pub | Published in Generative AI

• medium.com | Written by CodeWithYog