Version 1, July 2023

Getting started with GitHub Copilot

/thoughtworks
Strategy. Design. Engineering.

Ground rule: You're still responsible for your code. Copilot is your assistant.

How to prompt?

Comments are part of the prompt. Describe what you want the code to do, it's not like prompting a chatbot.

```
group the customer list by the "city" field
```

Descriptive **function and variable names** are part of the prompt.

```
groupListOfCustomers ByCity()
```

Test descriptions, and test code, are part of the prompt. Have your test file open while you're implementing.

```
describe("should group a customer list by city")
```

The more **surrounding context**, the better: **Open files** are part of the prompt. Whatever code you are typing is part of the prompt.

Iterate on your prompt - provide examples, make the comment or function name more specific, reduce the scope of what you are trying to do.

For changes to existing functions with good test coverage, consider deleting and regenerating the whole function.

When it works better

These are some of the parameters under which it is expected to work better than in other situations.

Relatively **straightforward or standard problems**- repetitive, boilerplate, common logic like list processing, a repeating pattern, things you'd usually copy & paste.

Commonplace languages and frameworks.

Typed languages provide a shorter detection loop for syntactically incorrect suggestions.

Tasks you **know how to do in theory**, but you need assistance with the **details**. You should know enough to judge if Copilot's suggestions makes sense.

For small chunks: About 10-15 lines of code, maximum, so that you can follow along, and don't increase your cognitive load with lots of code review.

Beware: Copilot can amplify "bad" local code

Getting stuck in older patterns
After refactoring, or introducing new patterns into the codebase, you might find that Copilot is still stuck in "the old ways".

Proliferating bad practices
Copilot uses our existing code to make more suitable suggestions for our context, but depending on what it chooses as its example, it creates either the good or the bad kind of consistency.

It's a balance: Flow boost vs flow disruption

Move on	Don't spend too much time fixing things if the suggestions are obviously not useful. Two strikes and move on!
Review fatigue	The larger the piece of code generated, the more code review you will have to do! Consider the balance between the "typing" boost and the energy needed for review.
Prompt abstraction	If prompting takes more time than coding itself, consider if it's a good use case for Copilot, or if you should discard the assistance and write the code yourself.

Don't become complacent

TLDR; Don't become the person who drives into the lake because their navigation system tells them to.

Automation bias
Once you have had some success with Copilot, be careful not to start over-trusting it.

Anchoring bias
Be aware that once you have seen Copilot's suggestions, you might have a harder time thinking about other solutions.

Sunk cost fallacy
Even though with Copilot, we did not invest time to write a suggestion ourself, we might still get over-attached to the supposed time saved of a multi-line suggestion.

Security vigilance
Copilot does some pre-filtering for vulnerable patterns, but you cannot rely on the code to be secure for your context. Especially watch out for hallucinated dependency names.

Are you super excited about this?

Awesome, this is an important new area for developers to understand! But be careful not to overinvest into making it work all the time because you want it to work.

Are you skeptical?

That's a healthy attitude, there is a lot of hype at the moment and the tool is not perfect! Give it a chance, and try some things even when you think "that won't work anyway".

Read more about GitHub Copilot here