



# Effiziente Entwicklung mit GitHub Copilot in Visual Studio: KI als Pair Programmer & Best Practices

Referent: Dr. Matthias Liebeck

*technical* **SUMMIT**



LinkedIn



GitHub Copilot Newsletter



1. Motivation für AI-Driven Development
2. Einführung in GitHub Copilot
3. Integration in Visual Studio
- 4. Workshop: Live-Coding**
5. Stolpersteine & Verantwortung
6. Best practices
7. Fazit

- Professioneller Softwareentwickler seit 2009, C# seit 2006
- 2009 – 2015: Studium der Informatik und Mathematik
- 2018: Promotion in Informatik / Künstlicher Intelligenz mit Schwerpunkt Natural Language Processing
- Seit 2019: Tätigkeit bei der anyMOTION GmbH in Düsseldorf / Deutschland als Senior Solution Architect mit den Schwerpunkten
  - ASP.NET Core MVC / Web API / C#
  - SQL Server
  - Azure
- Veranstalter Azure Düsseldorf Meetup



# Motivation für AI-Driven Development

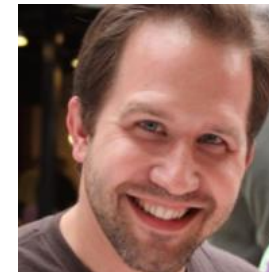
- Phase 1: Print-Medien & Offline-Hilfe

- Bücher
- Zeitschriften
- F1 Hilfe



- Phase 2: Online-Communities

- myCSharp.de 
- Blogposts (Scott Hanselman, Scott Guthrie)
- Stack Overflow



- Phase 3: ChatGPT im Browser
  - Copy & Paste von einem Code, Problem oder Fragestellung
  - jedes Mal
    - Aufbau vom Code erklären
    - relevante Methoden rauskopieren
    - Kontext erklären
  - Warum das alles mühsam war
    - Kontext geht verloren
    - Entwickler wird aus dem Code bzw. IDE gerissen / mentale Unterbrechung
    - Kopieren von Antworten in den Code zurück



- Phase 4: GitHub Copilot: Kontext als Game Changer

- Was ist Kontext?

- aktueller Code
- Projektstruktur
- Namenskonventionen
- Abhängigkeiten

ChatGPT im Browser	GitHub Copilot in der IDE
<ul style="list-style-type: none"><li>• kein Projektkontext</li><li>• Code muss kopiert werden</li><li>• Struktur geht verloren</li><li>• Medienbruch</li></ul>	<ul style="list-style-type: none"><li>• kennt Projektstruktur</li><li>• sieht aktuellen Code</li><li>• nutzt Konventionen</li><li>• kein Toolwechsel</li></ul>

👉 Warum IDE-Integration entscheidend ist:

- keine Toolwechsel
- weniger Erklärungen
- bessere Vorschläge



- Bestehenden Code analysieren lassen
- Bestehenden Code mit neuen Features erweitern
- Neues Projekt generieren lassen
- Bestehenden Code refactorieren
- Bei Exceptions beim Debuggen helfen können
- Update von NuGet packages / .NET Version
- Tests erstellen
- Compiler warnings reduzieren
- ...

# Einführung in GitHub Copilot

## Definition:

- GitHub Copilot ist ein KI-gestützter Assistent, der Entwickler direkt in der IDE unterstützt, um beim Verstehen, Schreiben und Überarbeiten von Code effizienter & schneller zu sein und manuellen Aufwand zu sparen.
- Copilot nutzt den aktuellen Projektkontext, um situationsbezogene Vorschläge zu machen und repetitive Aufgaben zu beschleunigen.

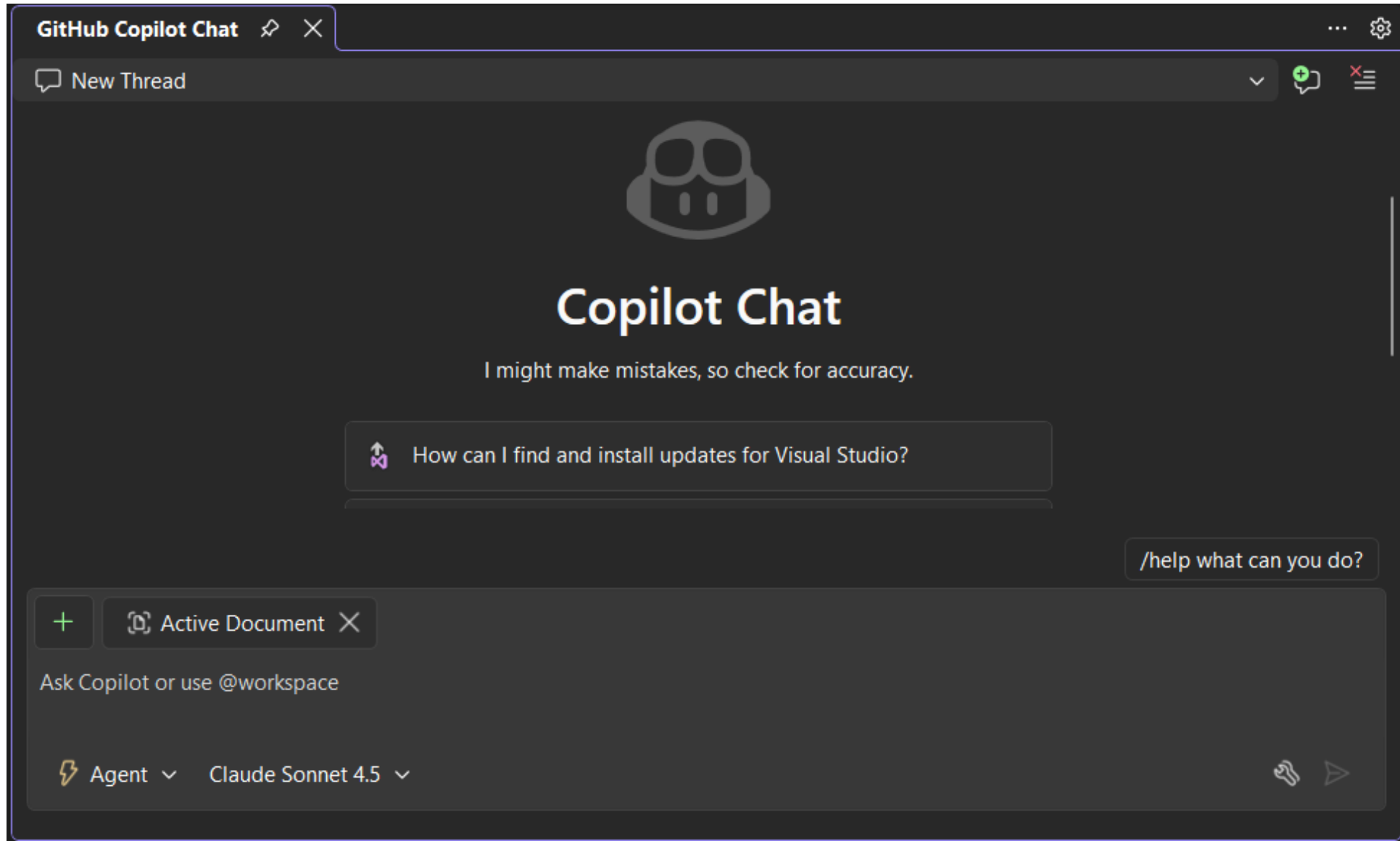
## Was GitHub Copilot ist:

- kontextbewusster Assistent
- Pair Programmer
- Ideengeber

## Was Copilot nicht ist:

- kein Autopilot
- kein Ersatz für Entwickler
- kein Ersatz für Code Reviews
- keine Garantie für korrekten Code



## GitHub Copilot Chat-Window:



- GitHub Account
- GitHub Copilot Subscription
- IDE-Integration:
  - Visual Studio
    - 2022: ab 17.8
    - 2026
  - Visual Studio Code
  - CLI
  - Web Browser
  - JetBrains IDE
  - Eclipse
  - Xcode

	Copilot Free	Copilot Pro	Copilot Pro+	Copilot Business	Copilot Enterprise
Pricing	Not applicable	\$10 USD per month, or \$100 USD per year (free for some users)	\$39 USD per month, or \$390 USD per year	\$19 USD per granted seat per month	\$39 USD per granted seat per month
Premium requests	50 per month	300 per month	1500 per month	300 per user per month	1000 per user per month
Purchase additional premium requests at \$0.04/request	×	✓	✓	✓	✓

- Das GitHub Copilot Chat Window bietet mehrere LLMs zur Auswahl an
- Neue Modelle kommen monatlich dazu.
  - Inzwischen direkt (ohne preview) in general availability
  - Neue LLMs müssen auf github.com in den settings freigeschaltet werden.
- Abrechnung erfolgt anhand von **premium requests**, **nicht anhand von tokens**
- **kein session limit / weekly limit** wie z.B. bei Claude

Auto	Variable
Premium Models	
GPT-5	1x
GPT-5.1	1x
GPT-5.1-Codex	1x
GPT-5.1-Codex-Mini	0,33x
Claude Sonnet 4	1x
✓ Claude Sonnet 4.5	1x
Claude Opus 4.5 (Preview)	3x
Claude Haiku 4.5	0,33x
Gemini 3 Pro (Preview)	1x
Gemini 2.5 Pro	1x
Standard Models	
GPT-5 mini	0x
Grok Code Fast 1 	0x
GPT-4.1	0x
GPT-4o	0x
 Manage Models	

## **Copilot reagiert auf:**

- Klarheit des Prompts
- Kontext
- Ziel

## **Schlechte Prompts erzeugen:**

- falsche Annahmen durch GitHub Copilot
- nicht den gewünschten Code

## **Gute Prompts erzeugen:**

- besser integrierten Code
- weniger Nacharbeit

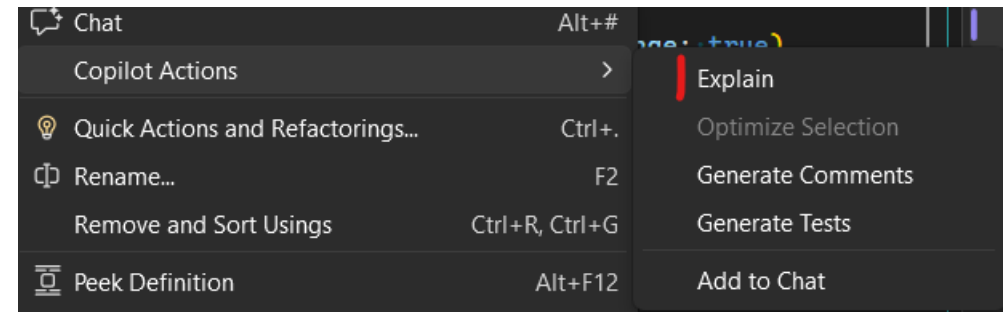
## Integration in Visual Studio

## **Copilot Funktionen in Visual Studio - Übersicht**

- Ask Mode
- Agent Mode
- Debug Mode
- Inline Suggestions

**Definition Ask Mode:** GitHub Copilot bekommt einen Schreibschutz und darf nur lesend auf das Projekt zugreifen.

- Beispiel Prompts:
  - „Wie funktioniert die Methode XX?“
  - „Wird irgendwo im Code XX verwendet?“
  - „Wie kann ich XX verbessern?“
- Copilot nutzt Projektkontext in Echtzeit



### Warum sinnvoll:

- Kein Copy/Paste in ChatGPT mehr notwendig. Copilot löst Referenzen im Projekt auf und kann im ganzen Projekt nachschlagen
- Fragen direkt „im Code stellen“

**Definition Agent Mode:** Im Agent Mode kann Copilot Änderungen selbstständig über mehrere Dateien hinweg umsetzen und neuen Code erzeugen 👉 **für mich das Herzstück in GitHub Copilot**

- Arbeitet mit prompts, instruction Files oder internen Regeln
- Beispiele:
  - **Neue Features implementieren**
  - **Bugfixes**
  - Dokumentationsgenerierung
  - Testfall-Generierung
  - Code Cleanup über mehrere Dateien
  - ...

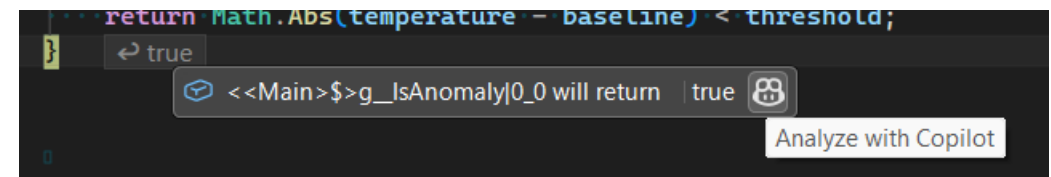
**Definition Debug Mode:** GitHub Copilot kann während einer Debug-Session auf die Inhalte von Variablen & Callstack zugreifen und somit zur Laufzeit noch besser nach Fehlern suchen

## Beispiel: Logikfehler analysieren

```
double temperature = 22.0;
double baseline = 20.0;

Console.WriteLine(IsAnomaly(temperature, baseline));

1 reference
static bool IsAnomaly(double temperature, double baseline)
{
    double threshold = 5.0;
    return Math.Abs(temperature - baseline) < threshold;
}
```



Looking at the code:

- **IsAnomaly** checks if **Math.Abs(temperature - baseline) < threshold**
- Variables: **temperature = 22.0, baseline = 20.0, threshold = 5.0**
- Calculation: **Math.Abs(22.0 - 20.0) = Math.Abs(2.0) = 2.0**
- Check: **2.0 < 5.0 → true**

## Analysis

**The return value true means the function determined that the temperature (22°C) is NOT an anomaly** relative to the baseline (20°C).

However, there appears to be a **logic error**. The function name suggests it should return **true** when an anomaly is detected, but the current implementation returns **true** when the temperature is **WITHIN** the threshold (i.e., normal). The condition should likely use **>=** instead of **<** to correctly identify anomalies.

**Current behavior:** Returns **true** when temperature is normal (within 5 degrees) **Expected behavior:** Should return **true** when temperature is anomalous (outside threshold)



**Would you like me to suggest fixing the comparison operator from < to >=?**



Claude Sonnet 4.5

+ Active Document ×

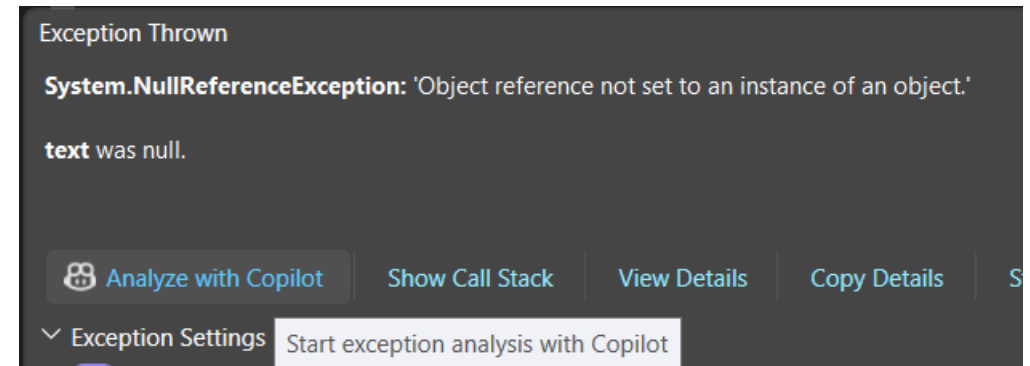
Ask Copilot or use @workspace

Debug ▼ Claude Sonnet 4.5 ▼  

## Beispiel: Exceptions debuggen

```
string? input = null;
Console.WriteLine(GetLength(input));

1 reference
static int GetLength(string text)
{
    return text.Length;
}
```



### Root Cause

The root cause is that the **input** variable is explicitly initialized to **null** in the code (**string? input = null;**), and then directly passed to the **GetLength** method without any validation or assignment of an actual value. The **GetLength** method does not handle null inputs and attempts to access the **Length** property on a null reference, causing the **NullReferenceException**.

### Why This Error Occurs

The code declares **input** as a nullable string (**string?**) and assigns it **null**, but never assigns it any actual value before use. When **GetLength(input)** is called, it receives **null** as the **text** parameter. Inside **GetLength**, the code tries to access **text.Length**, which fails because you cannot access properties or methods on a null reference.

The issue has two components:

1. **No input validation:** The **GetLength** method doesn't check if **text** is null before accessing its properties
2. **No proper initialization:** The **input** variable is never assigned a meaningful value before being used

### Possible Fix

Add a null check in the **GetLength** method to handle null inputs gracefully, or ensure **input** has a valid value before calling **GetLength**.

Would you like me to add null handling to GetLength?

**Definition Inline Suggestions:** Inline Suggestions sind Copilots Unterstützung direkt beim Schreiben, ohne dass bewusst ein Prompt formuliert werden muss.

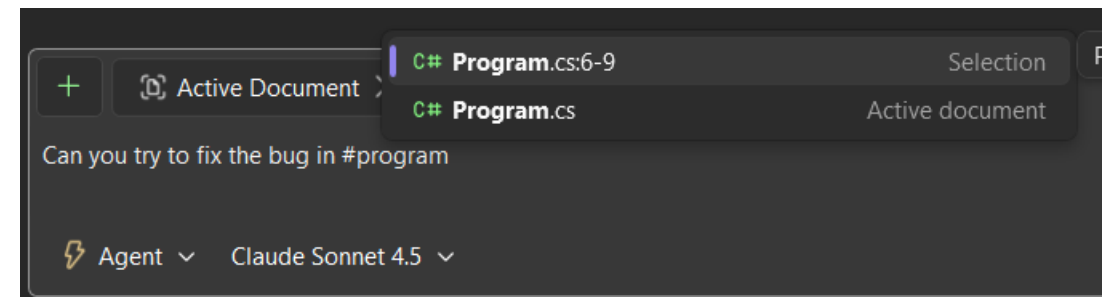
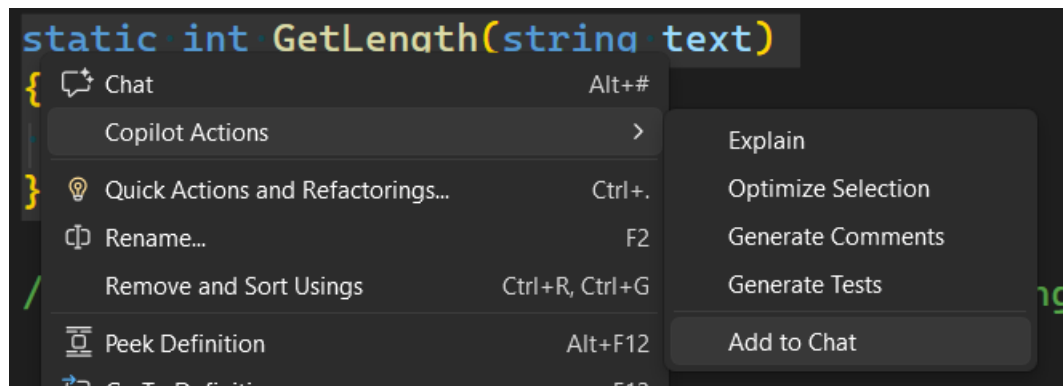
```
//·Todo:·Write·a·method·that·reverses·a·string
```

```
Tab Accept Esc Dismiss  
method that reverses a string  
static string ReverseString(string text)  
{  
    if (text == null) throw new ArgumentNullException(nameof(text));  
    char[] charArray = text.ToCharArray();  
    Array.Reverse(charArray);  
    return new string(charArray);  
}
```

- Vorschläge müssen geprüft werden
- Manchmal sehr nah dran, manchmal sehr weit weg
- Copilot kann falsche Annahmen treffen

GitHub Copilot nutzt verschiedenen Kontext:

- offene Tabs
- referenzierten Code in anderen Dateien
- sucht genannte Klassen- und Methodennamen eigenständig im Quellcode
- kann auf Codeblöcke referenzieren



## Workshop: Live-Coding

## **Was haben wir gerade gesehen?**

- GitHub Copilot hat bestehenden Code und Projektstruktur verstanden
- Copilot hat Refactorings über mehrere Dateien hinweg durchgeführt
- Copilot hat explizite Anweisungen und Einschränkungen beachtet
- Das Ergebnis war funktionierender Code, aber nicht automatisch perfekter Code

## **Geschwindigkeit vs. Verantwortung**

- GitHub Copilot beschleunigt die Entwicklung massiv
- GitHub Copilot versteht keine fachliche Intention
- GitHub Copilot trägt keine Verantwortung für Probleme in der Produktiv-Umgebung
- Code Reviews, Tests und Architekturentscheidungen bleiben menschliche Aufgaben

 **Die Verantwortung bleibt immer beim Entwickler**

## **Warum GitHub Copilot in der IDE so gut funktioniert?**

- Vollständiger Projekt- und Lösungskontext
- Bestehende Namenskonventionen und Patterns
- Zugriff auf zusammenhängenden Code statt isolierter Snippets
- Kein Kontextverlust, kein Copy & Paste, kein mentaler Toolwechsel

# Stolpersteine & Verantwortung

## **Was ich aus allen Beispielen gelernt habe:**

- Copilot beschleunigt massiv
- Copilot macht Fehler
- Copilot versteht keinen Kontext außerhalb des Codes
- Verantwortung bleibt beim Entwickler

## **Typische Copilot-Fehler:**

- falsche Annahmen
- Halluzinationen
- zu optimistische Aussagen
- scheinbar „sichere“ Lösungen

## **Gefahren im Alltag**



- blindes Vertrauen
- fehlende Reviews
- fehlende Tests
- falsches Sicherheitsgefühl

## **Verantwortung bleibt beim Entwickler:**

- Architekturentscheidungen
- Fehleranalyse
- Betriebsverantwortung
- Wartbarkeit



- Best Practice #1: Add a copilot-instructions.md file
- Best Practice #2: Don't let Agent Mode run blindly! Clarify directions first 🚗
  - Der 10x game changer für mich persönlich!
  - > *If you have any questions, ask them before implementing!*
- Best Practice #3: Ask Copilot to extend your copilot-instructions.md
  - > *I have a copilot-instructions.md file to help you with instructions in the agent mode. Please read the file and tell me what you think of the current instructions. Should I add anything? Please extend the file to make your life easier*

- Best Practice #4: Break big tasks into small steps
  - Wenn ein größeres feature ansteht, dann den scope beschreiben und in kleinere Teile zerlegen.
  - Copilot dann sagen, er soll erst mit Teil 1 anfangen, danach Teil 2, Teil 3 usw.
- Best Practice #5: Use the chat window for code reviews
  - Während Copilot bereits Ergebnisse produziert und Fehler / Verbesserungsmöglichkeiten auffallen, parallel schon im chat window / Notepad++ schreiben, was geändert werden soll.
- Best Practice #6: Work on branches and commit often
  - In branches arbeiten und Zwischenstände als “ WIP” commits sichern
  -  So sind Änderungen in Git leichter zu sehen und parallel zum agentic coding kann bereits ein code review gemacht werden.

- **Best Practice #7: Not sure if Copilot can handle your specific case? Just give it a try!**
  - Ausprobieren geht über studieren! Im worst case hat man vorher einen Implementierungsplan durch das Schreiben des prompts gemacht, der manuell implementiert wird.
- **Best Practice #8: Unstuck GitHub Copilot**
  - Falls Copilot hängt: „please continue“ oder “yes” ins chat window
- **Best Practice #9: Ask Copilot to write documentation for implemented features**
  - Copilot soll neue Features direkt in einer Markdown Datei kommentieren, als Doku für Entwickler und auch um später bei dem Feature in einem neuen Chat ansetzen zu können
  - copilot-instructions.md:
    - > *After finishing an implementation, write a Markdown file into the /docs directory. In that file, describe important information I would need if I asked you to continue this feature in a new conversation.*



## **Wo Copilot besonders gut funktioniert**

- CRUD Anwendungen erstellen oder erweitern
- Standard Patterns
- Wiederholende Aufgaben
- Refactoring

## **Wo Copilot (aktuell) an Grenzen stößt**

- komplexe Business-Logik
- Legacy Code ohne Struktur
- implizites Fachwissen

## Ersetzt KI den Entwickler?

- **Nein**
- Verantwortung bleibt beim Menschen
- Architektur bleibt menschlich
- KI verstärkt gute Entwickler
- Schlechte Prozesse bleiben schlecht (garbage in 🙌 garbage out)

## Persönliches Fazit:

- sehr spannendes Thema und rasante Entwicklung
- „leider“ zu schnell viel Neues um am Ball zu bleiben
- Hat wieder Motivation für „nach Feierabend am Hobbyprojekt programmieren“ entfacht
- GitHub Copilot ist ein starkes Tool und kann viel Arbeit abnehmen
- GitHub Copilot macht Fehler und daher ist QA unerlässlich
- Prompts sind sehr wichtig!
- **Die Verantwortung der Arbeit des Copiloten liegt beim Piloten**

Danke!

LinkedIn



GitHub Copilot Newsletter

