



Dr. Matthias Liebeck

GitHub Copilot in Visual Studio: AI as a Pair Programmer & Best Practices

GitHub Copilot Dev Days @Global AI Community

19.03.2026



1. Motivation for AI-Driven Development
2. Introduction to GitHub Copilot
3. Integration in Visual Studio
- 4. Workshop: Live Coding**
5. Pitfalls & Responsibility
6. Best Practices
7. Conclusion

- Professional software developer since 2009, C# since 2006
- 2009 – 2015: Studies in Computer Science and Mathematics
- 2018: PhD in Computer Science / Artificial Intelligence, focus on Natural Language Processing
- Since 2019: Senior Solution Architect at anyMOTION GmbH, Düsseldorf, focusing on
 - ASP.NET Core MVC / Web API / C#
 - SQL Server
 - Azure
- Organizer: Azure Düsseldorf Meetup
- Community Speaker: GitHub Copilot, MCP, Azure OpenAI
- Newsletter: ghcp.liebeck.io



GitHub Copilot Newsletter

Motivation for AI-Driven Development

- Phase 1: Print Media & Offline Help

- Books
- Magazines
- F1 Help



- Phase 2: Online Communities

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



- Phase 3: ChatGPT in the browser
 - copy & paste of code, a problem or a question
 - every single time
 - explain the code structure
 - copy out relevant methods
 - explain the context
 - Why this was so tedious
 - context gets lost
 - developer is pulled away from the IDE / mental interruption
 - copying answers back into the code



- Phase 4: GitHub Copilot: Context as a Game Changer

- What is context?

- current code
- project structure
- naming conventions
- dependencies

ChatGPT in Browser	GitHub Copilot in the IDE
<ul style="list-style-type: none">• no project context• code must be copied• structure gets lost• context switch	<ul style="list-style-type: none">• knows project structure• sees current code• uses conventions• no tool switching

👉 Why IDE integration matters:

- no tool switching
- less explaining
- better suggestions



- analyze existing code
- extend existing code with new features
- generate a new project from scratch
- refactor existing code
- debug exceptions
- update NuGet packages / .NET versions
- create tests
- reduce compiler warnings
- ...

Introduction to GitHub Copilot

Definition:

- GitHub Copilot is an AI-powered assistant that supports developers directly in the IDE to be more efficient at understanding, writing, and reviewing code while reducing manual effort.
- Copilot uses the current project context to make situation-specific suggestions and speed up repetitive tasks.

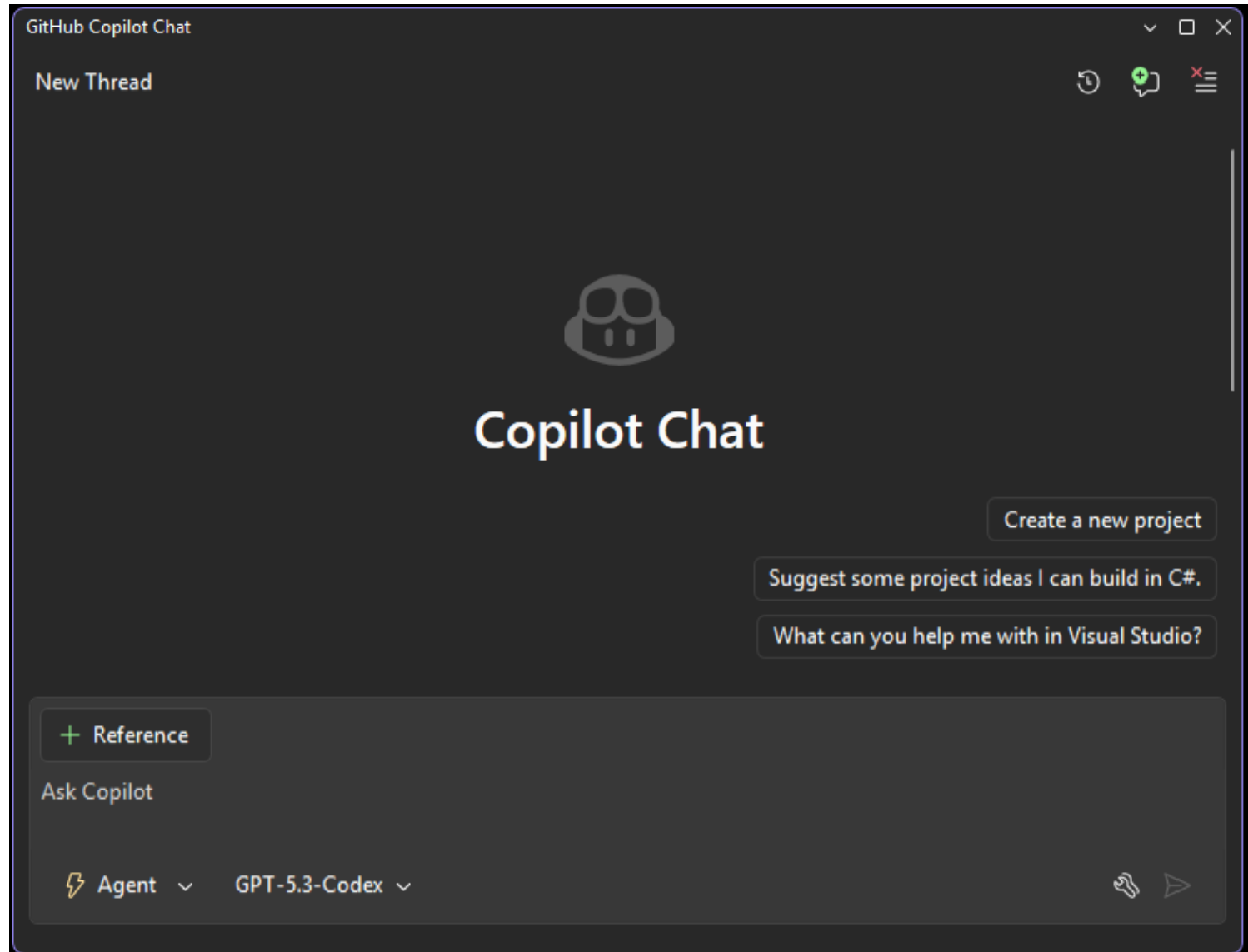
What GitHub Copilot is:

- context-aware assistant
- pair programmer
- idea generator

What Copilot is not:

- not an autopilot
- not a replacement for developers
- not a replacement for code reviews
- no guarantee of correct code

GitHub Copilot Chat Window:



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

Compare features	Free \$0	Pro \$10 per month	Pro+ \$39 per month
Premium requests			
Premium requests			
Chat, agent mode, code review, coding agent, and Copilot CLI use premium requests, with usage varying by feature or model. Learn more	50 per month	300 per month	1,500 per month
Purchase additional premium requests at \$0.04/request ¹	—	✓	✓
	Business \$19 per user / month	Enterprise \$39 per user / month	
Premium requests	300 per user per month	1,000 per user per month	

- The GitHub Copilot Chat Window offers multiple LLMs to choose from.
- New models are added every month:
 - now directly general availability (no preview)
 - new models must be enabled in settings
- Billing is based on **premium requests, not on tokens**
- **No session limit or weekly limit** (unlike some other tools), but a **monthly limit**.
- **However, there are free models!**

Auto	Variable
Premium Models	
Gemini 3.1 Pro	1x
GPT-5.2-Codex	1x
GPT-5.3-Codex	1x
GPT-5.4 mini	0,33x
Grok Code Fast 1	0,25x
GPT-5.1	1x
⚠ GPT-5.1-Codex	1x
⚠ GPT-5.1-Codex-Mini	0,33x
⚠ GPT-5.1-Codex-Max	1x
Claude Haiku 4.5	0,33x
⚠ Gemini 3 Pro (Preview)	1x
Gemini 3 Flash (Preview)	0,33x
Gemini 2.5 Pro	1x
GPT-5.2	1x
Standard Models	
GPT-5 mini	0x
GPT-4.1	0x
GPT-4o	0x
⚙ Manage Models	

Copilot responds to:

- Clarity of the prompt
- Context
- Goal

Poor prompts produce:

- wrong assumptions by GitHub Copilot
- not the desired code

Good prompts produce:

- better-integrated code
- less rework

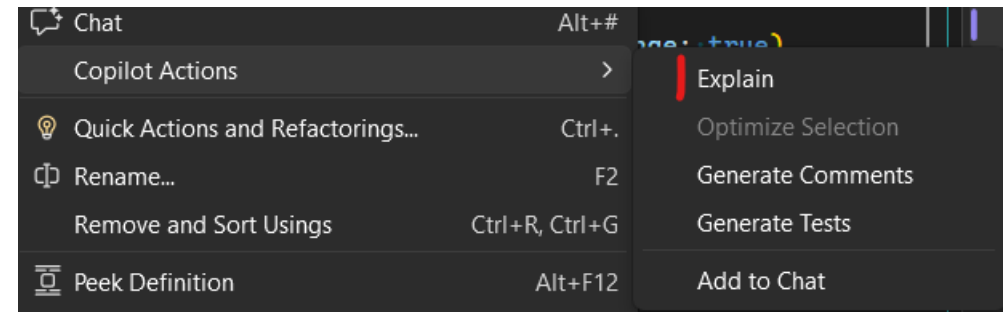
Integration in Visual Studio

Copilot Features in Visual Studio – Overview

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

Definition Ask Mode: GitHub Copilot gets read-only access to the project.

- Example Prompts:
 - "How does method XX work?"
 - "Is XX used anywhere in the code?"
 - "How can I improve XX?"
- Copilot uses project context in real time



Why this matters:

- No more copy/paste to ChatGPT. Copilot resolves references and searches across the entire codebase.
- Ask questions directly "in the code"

Definition Agent Mode: In Agent Mode, Copilot implements changes autonomously across multiple files and generates new code 🙌 **the core feature of GitHub Copilot for me**

- works with prompts, instruction files or internal rules
- Examples:
 - **implement new features**
 - **bugfixes**
 - generate documentation
 - generate test cases
 - code cleanup across multiple files
 - ...

Definition Debug Mode: GitHub Copilot accesses variables and call stack during a debug session to find errors more effectively at runtime

Example: Analyze logic errors

```
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;
}
```

```
return Math.Abs(temperature - baseline) < threshold;
} ← true
<<Main>>g_IsAnomaly|0_0 will return true
Analyze with Copilot
```

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 ×

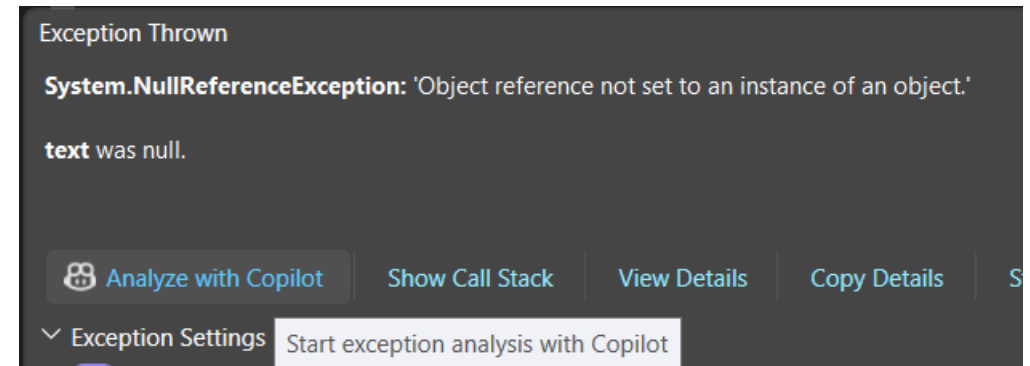
Ask Copilot or use @workspace

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 are Copilot's support while writing code. We don't need to explicitly write a prompt.

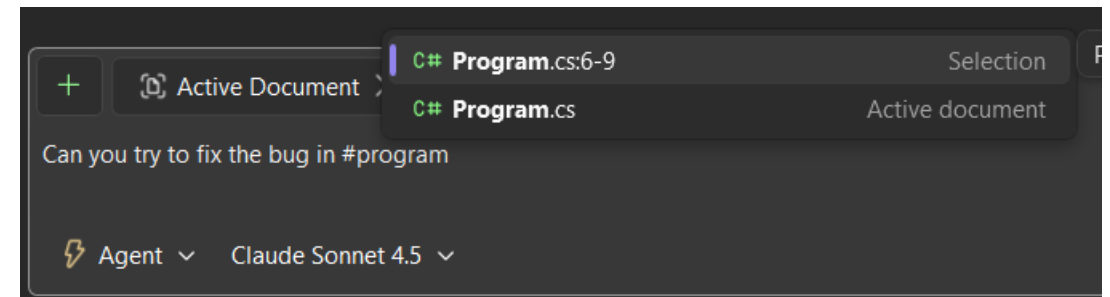
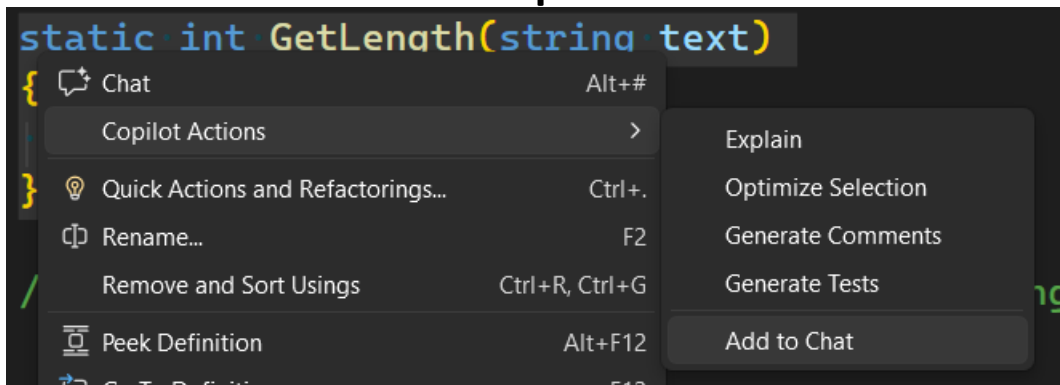
```
//·Todo:·Write·a·method·that·reverses·a·string
```

```
Tab Accept Esc Dismiss hod·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);  
}
```

- suggestions must always be reviewed
- sometimes spot-on, sometimes way off
- Copilot can make wrong assumptions

GitHub Copilot uses various context sources:

- open tabs
- referenced code in other files
- independently looks up mentioned class and method names in source code
- can reference specific code blocks



Workshop: Live Coding

What did we just see?

- GitHub Copilot understood the existing code and project structure
- Copilot performed refactorings across multiple files
- Copilot followed explicit instructions and constraints
- The result was working code, but not automatically perfect code

Speed vs. Responsibility

- GitHub Copilot massively accelerates development
- GitHub Copilot does not understand business intent
- GitHub Copilot bears no responsibility for production issues
- Code reviews, tests, and architecture decisions remain human responsibilities 🙌 but GitHub Copilot can help!

🙌 Responsibility always stays with the developer

Why GitHub Copilot works so well in the IDE

- full project and solution context
- existing naming conventions and patterns
- access to complete code instead of isolated snippets
- no context loss by IDE switching, no copy & paste, no mental tool-switching

Pitfalls & Responsibility

What I've learned from all the examples:

- Copilot massively speeds things up
- Copilot makes mistakes
- Copilot does not understand context beyond the code
- responsibility stays with the developer

Typical Copilot mistakes:

- wrong assumptions
- hallucinations
- overly optimistic statements

Everyday Dangers

- blind trust
- missing code reviews
- missing tests
- false sense of security

Responsibility stays with the developer:

- architecture decisions
- error analysis
- operational responsibility
- maintainability

- Best Practice #1: Use Visual Studio Insiders version instead of normal release channel
- Best Practice #2: Add a copilot-instructions.md file
- Best Practice #3: Don't let Agent Mode run blindly! Clarify directions first 🚗
 - > *If you have any questions, ask them before implementing!*
 - 👉 In Visual Studio Code and CLI, we already have Plan mode.
- Best Practice #4: 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*
 - 👉 However, a recent study by [Gloaguen et. al. \(2026\)](#) recommends hand-written instructions

- **Best Practice #5: Break big tasks into small steps**
 - When a larger feature is coming up, describe the scope and break it down into smaller parts.
 - Tell Copilot to start with part 1, then part 2, part 3, and so on.
- **Best Practice #6: Use the chat window for code reviews**
 - While Copilot produces results, note any issues in the chat window or a notepad
👉 Message queueing already in VS Code
- **Best Practice #7: Rotate between Visual Studio, Visual Studio Code and CLI for development to test new features early.**

- Best Practice #8: Not sure if Copilot can handle your specific case? Just give it a try!
 - Just try it! In the worst case, you've already written an implementation plan just by writing the prompt, which you can implement manually if needed.
- Best Practice #9: Unstuck GitHub Copilot
 - If Copilot gets stuck: type "please continue" or "yes" into the chat window
- Best Practice #10: Ask Copilot to write documentation for implemented features
 - Have Copilot document new features directly in a Markdown file as developer documentation and to resume the feature in a new chat session.
 - 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.*

Where Copilot works especially well

- building or extending CRUD applications
- standard patterns
- repetitive tasks
- refactoring

Where Copilot (currently) hits its limits

- complex business logic
- legacy code without structure
- implicit domain knowledge

Does AI replace the developer?

- **No**
- responsibility stays with the human
- architecture remains a human task
- AI amplifies good developers
- bad processes stay bad (garbage in 🙌 garbage out)

My personal takeaway:

- fascinating topic with a rapid pace of change
- Unfortunately, too many new exciting things every week so it's difficult to „not fall behind“
- Reignited my motivation for evening hobby projects
- GitHub Copilot is a powerful tool that can take care of a lot of work
- GitHub Copilot makes mistakes 👉 QA is non-negotiable
- Prompts matter a lot!
- **The responsibility for the co-pilot's work lies with the pilot**

Available since March 2026 for GitHub Copilot Pro & Pro+ users

- Copilot remembers your preferences, patterns, and context across conversations
- No more re-explaining your tech stack, coding style, or project constraints
- Memory evolves with your workflow. Copilot gets better the more you use it
- 👉 Reduces prompt overhead and makes Copilot feel more like a real pair programmer

💡 Check out my online event “Agentic Coding Summit #1: Why Does My AI Forget Everything?” on 27.05.2026



Model Context Protocol – connect external tools directly to GitHub Copilot

- MCP servers integrate into the Copilot Chat window as callable tools
- Copilot decides autonomously when to call which MCP tool based on your prompt
- 🙌 Extends Copilot far beyond code files towards a true development agent

Thank You!

LinkedIn



GitHub Copilot Newsletter

