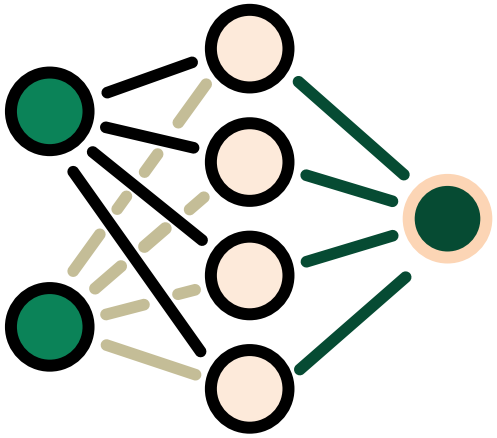
VICTORIA UNIVERSITY OF WELLINGTON
TE HERENGA WAKA

Getting Started with GitHub Copilot



Nabeel Aslam

Ph.D. Student in Computer Science
Evolutionary Computation Research Group
School of Engineering and Computer Science (SECS)

Coding Faster with AI

A Hands-on GitHub Copilot Workshop

Today’s journey

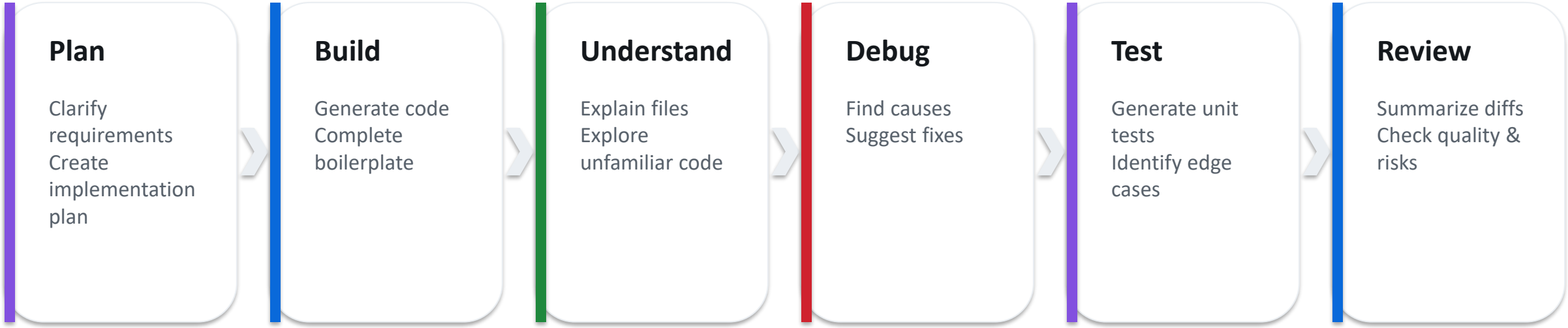
- Idea & requirements → code
- Understand → debug → refactor
- Generate tests → improve → review
- Short demonstrations + hands-on challenges

What is GitHub Copilot?

GitHub Copilot is an AI pair programmer that helps you write code faster. It's designed to help with programming tasks and acts as your assistant while working in your editor and on github.com.
In addition to code completion, GitHub Copilot can help you:

- Write code using natural language prompts
- Describe changes in your pull requests
- Debug and refactor code
- Can help explain code, generate tests, suggest fixes, and support development tasks.

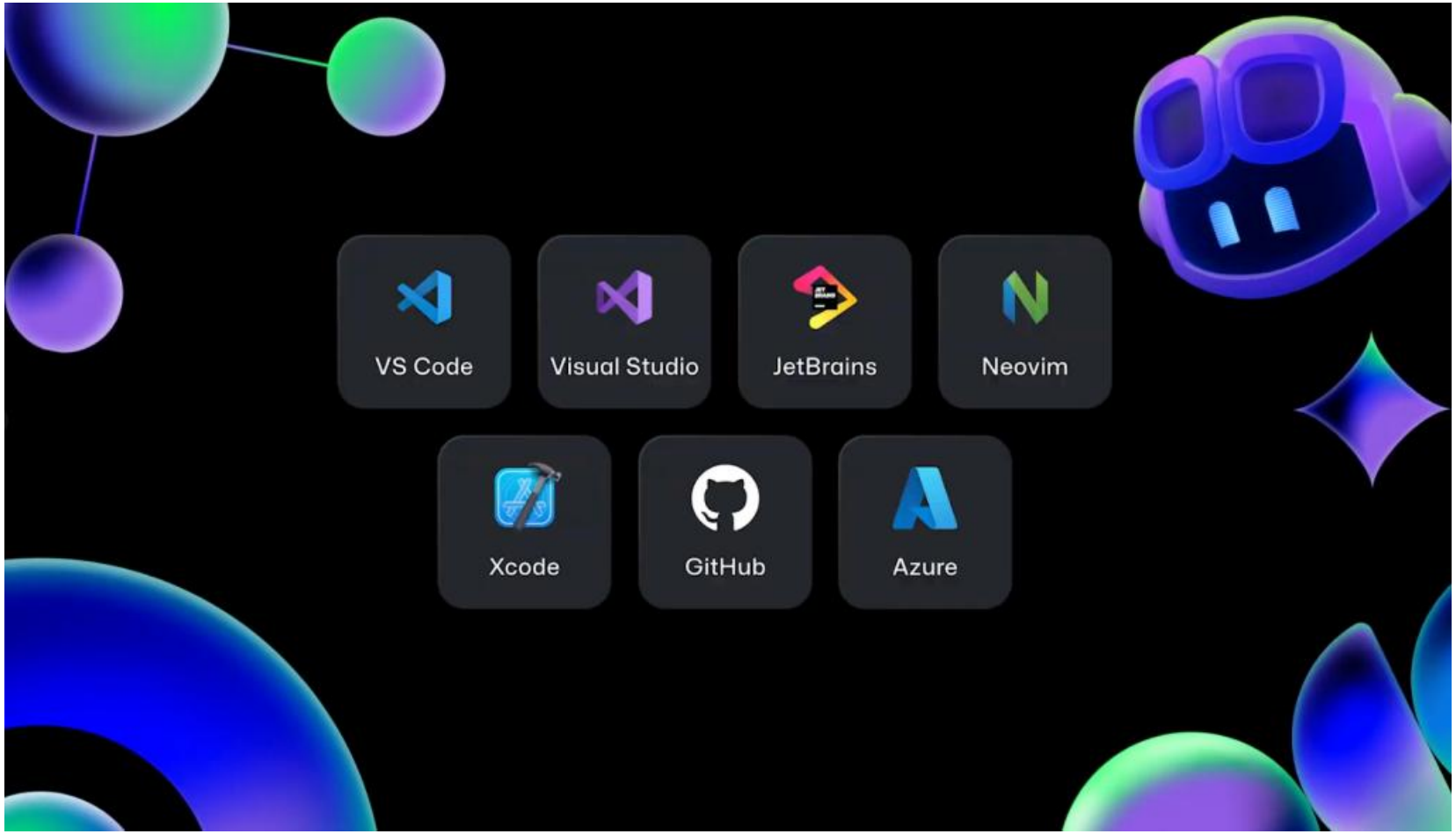
Where Copilot fits in the software development lifecycle



Core principle: Copilot can accelerate each stage, but responsibility for requirements, correctness, security, and final decisions remains with the developer.

Source: GitHub Docs / GitHub Copilot official resources

You can use GitHub Copilot with a variety of programming languages, and it’s available in VS Code, Visual Studio, JetBrains IDEs, Neovim, XCode

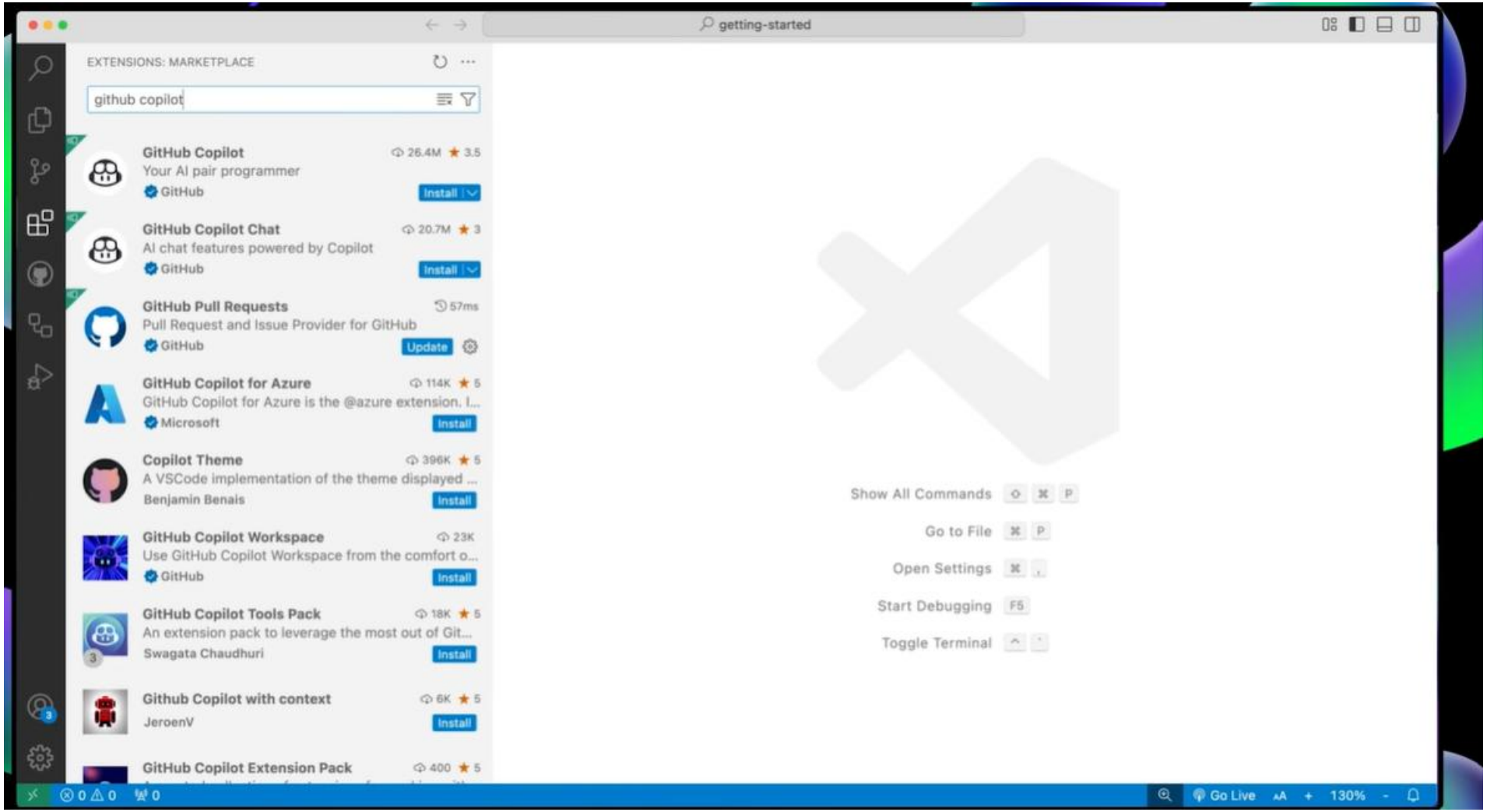


How to install GitHub Copilot in VS Code

1. Navigate to the Extensions Marketplace and search for “GitHub Copilot.” Click on the “GitHub Copilot” extension, authored by GitHub and then click Install. This will install two extensions: GitHub Copilot and GitHub Copilot Chat.

2. Once installed, sign into VS Code with your GitHub account that has access to GitHub Copilot. (If you didn’t previously authorize VS Code in your GitHub account, you’ll need to sign in to GitHub in VS Code.)

How to install GitHub Copilot in VS Code



Getting started in VS Code

Current setup flow

Install or update Visual Studio Code.

Sign in to GitHub and enable Copilot / Copilot Free or use an available paid plan.

Open the Copilot interface from the VS Code UI.

Start with a small repository or practice project.

Use /init when appropriate to help Copilot understand the project and create custom instructions.

Source: VS Code Copilot setup documentation (current 2026 guidance)

The developer + AI workflow

- 1. Define the problem
- 2. Give Copilot useful context
- 3. Generate or modify code
- 4. Review the output
- 5. Run and test it
- 6. Refine the solution

In VS Code:

Open the folder: Ctrl+K, then Ctrl+O.
Open Copilot Chat.
Click **Add Context** (paperclip or +).
Choose **Files & Folders**, then select the folder.

Prompting: give Copilot context

- State the goal: what should the program do?
- Specify the context: language, framework, files, constraints.
- Define the expected output: function, class, tests, explanation, or plan.
- Add quality criteria: readable, secure, maintainable, tested.

Prompt pattern

Context + Task + Constraints + Output

Instead of: “Build a task app.”
Try: “Create a Python task manager. Use functions for add, complete, delete and list. Validate empty task names and return clear error messages. Keep the code modular and easy to test.”

Prompt: “Create a Python task manager with add, complete, delete and list functions.”

Hands-on 1 — Idea → Code

Build the first version

Create a small Task Manager. Start with the requirement, ask Copilot for an implementation, run it, and make one change yourself.

Prompt: “Create a simple Python task manager with add, delete, complete and list functionality.”

Use Copilot to understand code

- Select unfamiliar code and ask for an explanation.
- Ask what each function does and what assumptions it makes.
- Ask Copilot to identify edge cases or potential failure points.
- Use follow-up questions rather than accepting the first explanation.

Hands-on 2 — Understand & Debug

Break the code on purpose

Take the working application. Introduce a small bug, run the program, then use Copilot to diagnose the problem. Compare its explanation with your own reasoning.

Prompt: “Find the bug, explain why it occurs, and propose the smallest safe fix.”

Refactoring with AI

- Ask Copilot to identify duplication, long functions, unclear names, or unnecessary complexity.
- Request a refactor without changing the intended behaviour.
- Review the diff carefully.
- Run tests after refactoring.

Hands-on 3 — Refactor & improve

Make the code better, not just longer

Ask Copilot for a refactoring plan first. Choose one improvement, implement it, then verify that existing behaviour still works.

Prompt: “Review this code for readability, duplication and maintainability. Suggest a refactoring plan before changing it.”

Testing: make AI prove the code

- Ask for unit tests for each core function.
- Include normal cases, boundary cases and invalid inputs.
- Ask: “What important case is missing?”
- Run the tests yourself—generated tests can also contain mistakes.

Hands-on 4 — Generate tests

Build a test suite

Generate tests for the Task Manager. Run them, inspect failures, and ask Copilot to improve the tests or implementation.

Prompt: “Generate unit tests for the task manager, including normal, boundary and invalid-input cases.”

Final challenge — Add one feature

Choose one extension

Add one feature to your application: priorities, due dates, search, persistence, filtering, or another useful feature. Use Copilot to plan first, then implement and test.

Prompt: “First create an implementation plan for adding task priorities. Then implement it and generate tests.”

Review AI-generated code critically

- Correctness — Does it actually solve the requirement?
- Security — Could it introduce unsafe behaviour?
- Quality — Is the code readable and maintainable?
- Testing — Is important behaviour covered?
- Context — Does the solution fit the project?
- Ownership — Can you explain and defend the code?

Copilot is an assistant—not the developer

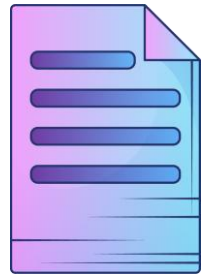
- AI can accelerate implementation, but it does not remove engineering responsibility.
- Treat generated code as a proposal that requires review and testing.
- Use version control and small changes so you can inspect what changed.
- Keep sensitive information and credentials out of prompts and source files.
- Human judgement remains central to design, security, quality and deployment.

Key takeaways

- Start with the problem, not the prompt.
- Give Copilot context and constraints.
- Use iterative prompts rather than one giant request.
- Ask AI to explain, debug, refactor and test—not only generate.
- Review, run, test and understand every important change.
- Use AI to increase developer capability, not replace developer judgement.

Build faster. Review smarter. Learn continuously

Any questions or discussion are highly appreciated.



Nabeel.aslam@ecs.vuw.ac.nz