AI-assisted software delivery from beginner to advanced

Powered by GitHub Copilot 

Hong Wu

Everything about GitHub Copilot

AI-assisted software delivery from beginner to advanced

Hong Wu

What is GitHub Copilot

GitHub Copilot is an AI coding assistant that helps you write code faster and with less effort.

GitHub Copilot *vs* GitHub Copilot X

- **On 29 June 2021**, GitHub Copilot was first announced by GitHub and powered by the OpenAI Codex, which is a modified version of the **GPT-3**.
- **In March 2023**, GitHub announced plans for "**Copilot X**", which will incorporate a chatbot based on **GPT-4**.
- **X** stands for coding suggestion, chat, voice, CLI, etc.

Why GitHub Copilot

- The world's **most widely** adopted AI developer tool
- Developers using GitHub Copilot code up to **55%** faster
- **74%** of developers focus on more satisfying work
- **88%** of developers feel more productive
- **96%** of developers are faster with repetitive tasks

See more information here: https://resources.github.com/copilot-for-business/

Feature Set

- **Coding suggestions** as you type
- **Chat**
- **Smart Actions**

What is AI-first Software Delivery?

1. AI-assisted Software Delivery
2. AI-powered Digital Products

AI-assisted Software Delivery

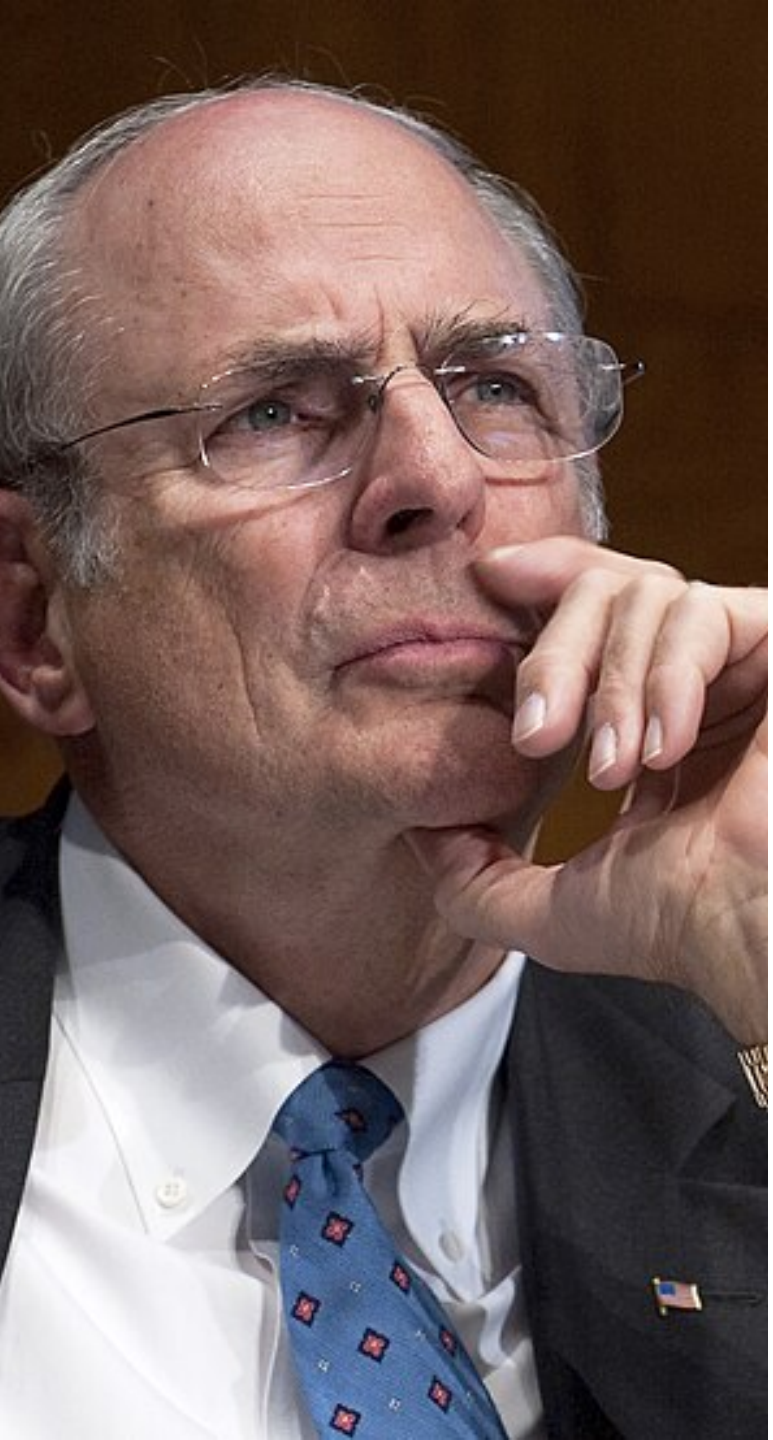
- Feature request sent & accepted
- Analysis
- Design
- Ready for Development
- Development
- Testing
- Deployment

AI-powered Digital Products

- OpenAI's ChatGPT
- Tesla's Full Self-Driving(FSD)
- GitHub Copilot
- Google's AlphaGo

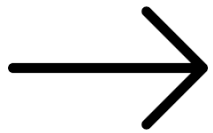
Why AI-assisted Software Delivery?

It could slow down the deterioration of software.



> Software is like entropy. It is difficult to grasp, weighs nothing, and obeys the second law of thermodynamics; i.e. it always increases.

— *Norman Ralph Augustine*



Why do some developers feel that the GitHub Copilot falls short of expectations?

1. Limitations of GitHub Copilot
    - Inaccurate suggestions
    - Limited context

    Solution: Adoption of state-of-the-art LLM

2. Have wrong expectations about it

> Lack of clear division of labor

> Lack of cognition of their role when pair with Copilot

Solution: 3R Cycle

3. Lack of aware of the basic concepts and underlying technologies of the software industry

> *You Don't Know What You Don't Know*, therefore, precise questions couldn't be raised.

4. Lack of fundamental mindset of:

- **Software engineering**
  - Pair Programming
  - Test Driven Development
  - Domain Driven Design
  - Kick Off & Sign Off
  - Clean Code
- **How to collaborate with Copilot**
  - What capabilities can Copilot provide?
  - How to maximize the capabilities of Copilot?

5. Lack of deliberate practice

How to pair with GitHub Copilot productively?

How to maximize the capabilities of Copilot in the daily work?

3**R** Cycle: **Request → Review → Refactor**

3R Cycle - Request: Design Thinking Driven Prompting

Step 1. Identify the real problem so you can do the right thing

Break down the problem into tasks

- **Product**
  - ↳ **MVP**s
    - ↳ **User Stories/Tasks**

Foundation 1: Design MVPs

- Begin With the End in Mind

Foundation 2: Kick Off & Sign Off

Foundation 3: Test Driven Development

- Tasking
- Acceptance Test Driven Development/Outside-in TDD vs Unit Test Driven Development/Inside-out TDD
- `Red - Green - Refactor` Cycle
- Main rules
    i. You must write a failing test before you write any production code. This is what your statement is referring to. You should not implement a function until you have written a test for it that fails.
    ii. You must not write more of a test than is sufficient to fail, or fail to compile.
    iii. You must not write more production code than is sufficient to make the currently failing test pass.

Step 2. Find the right solution to do things right

1. Breadth based Prompting for each task to find the right solution
2. Depth based Prompting for each solution to implement it right

Task → Options of Solution → Right Solution → Implementation

If encounter a new issue in the process, then deal with it in the same way as above.

3R Cycle - Review

On-demand Knowledge Transfer: By learning only what is necessary each time, you can make sure that your knowledge can iterate quickly.

3R Cycle - Refactor

Foundations:

- Domain Driven Design
- Clean Code

Let's try it out