

O'REILLY®

Github Copilot for Software Engineers

Sergio Pereira



Hello World!





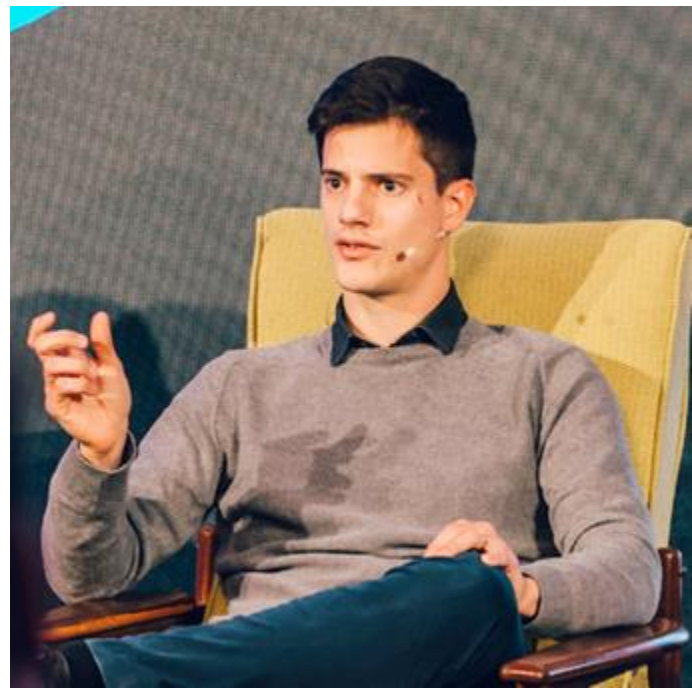
About the Author

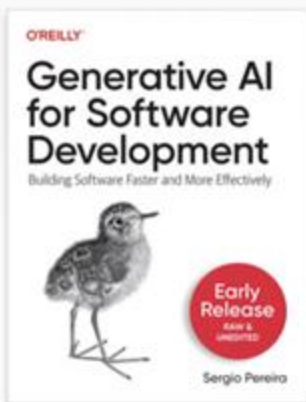
I'm Sergio Pereira

- I've worked in tech since 2009
- Started as a Software Engineer, became a CTO in 2014
- Currently work as a Fractional CTO for Tech Startups
- Embedding AI tools like Github Copilot to my clients' software development processes

I'm writing daily on:

- Twitter: **@SergioRocks**
- LinkedIn: **/sergiomcpereira**





Generative AI for Software Development

by [Sergio Pereira](#)

Released May 2025

Publisher(s): O'Reilly Media, Inc.

ISBN: 9781098162276

(*) GPT

ChatGPT for Software Engineers

Published by [O'Reilly Media, Inc.](#)

📖 Beginner to intermediate

How to 10X your productivity with generative AI

Cursor for Software Engineers

Published by [O'Reilly Media, Inc.](#)

📖 Intermediate

How to boost your productivity with generative AI



Learning Objectives

By the end of this course, you will understand:

- What is Github Copilot and why are **Software Engineers talking about it**
- The **risks** you should be aware of when using Github Copilot
- **13 use cases** where Github Copilot can boost your productivity
- **Key takeaways**
- **Q&A**





Disclaimer about O'Reilly and Github Copilot

O'Reilly Media is not affiliated with Github or Microsoft, and we are not responsible for your interactions with third-parties that take place outside of our services.

The chat will be part of the recording so please do not post personal information (such as email addresses) or any confidential and proprietary information.

The audience may include customers, vendors or competitors of you or your employer.

If you choose to take advantage of the interactive features available in this training you are responsible for any information you choose to share.



Wait! What's Github Copilot?

In case you haven't heard of it.





Launched publicly in June 21, 2022



Thomas Dohmke

June 21, 2022

That's changing now. Today, I am thrilled to announce that we are making [GitHub Copilot](#) generally available to individual developers. Your AI pair programmer is here.

With GitHub Copilot, for the first time in the history of software, AI can be broadly harnessed by developers to write and complete code. Just like the rise of compilers and open source, we believe AI-assisted coding will fundamentally change the nature of software development, giving developers a new tool to write code easier and faster so they can be happier in their lives.

Some usage data



Ben Kriemann/Getty Images

Microsoft is seeing big growth in the generative AI business, as the company's CEO, Satya Nadella, Tuesday evening told Wall Street that the company's paying customers for its GitHub Copilot software rose by 40% in the September quarter from the prior quarter.

"We have over 1 million paid copilot users in more than 37,000 organizations that subscribe to copilot for business," said Nadella, "with significant traction outside the United States."

Some usage data

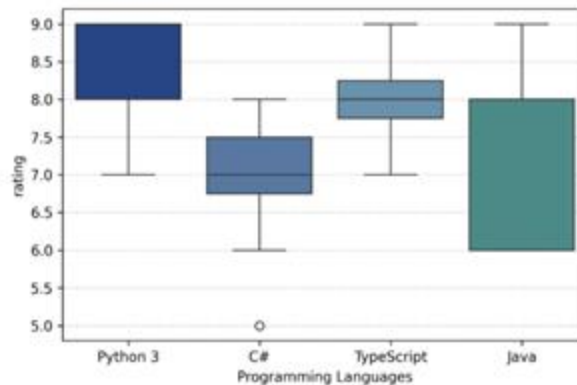
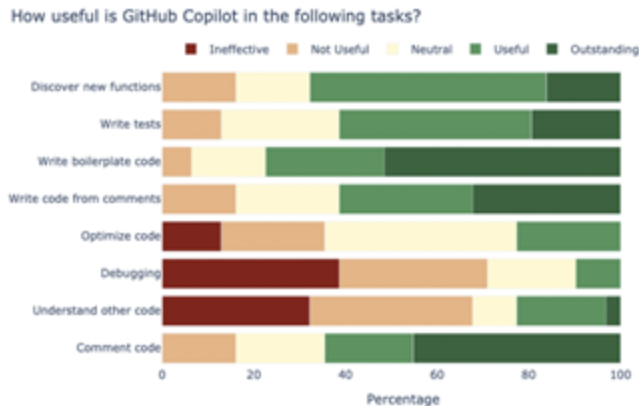


Figure 1: Overall rating given to GitHub Copilot, by programming language

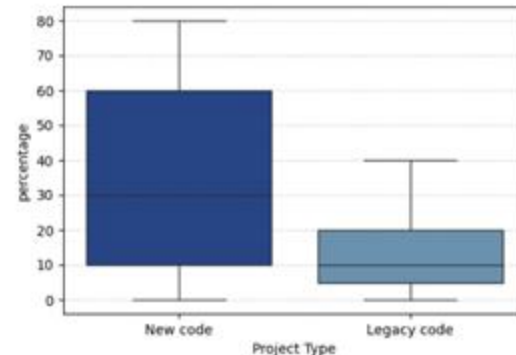
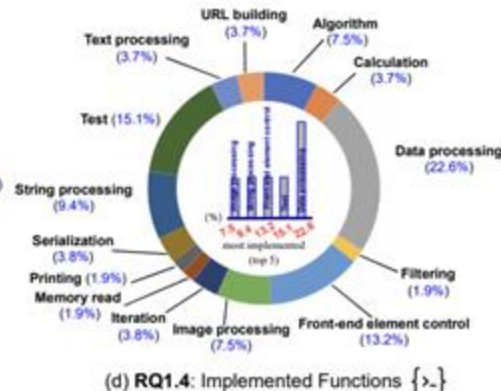
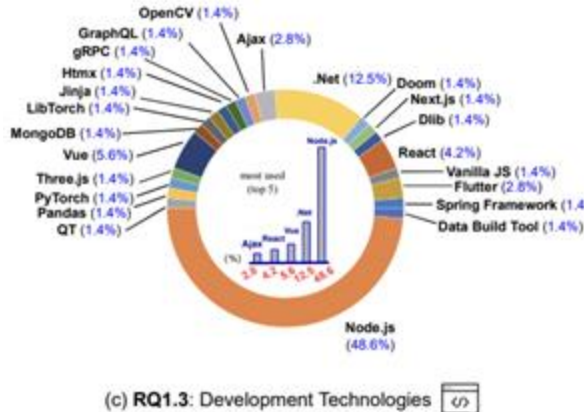
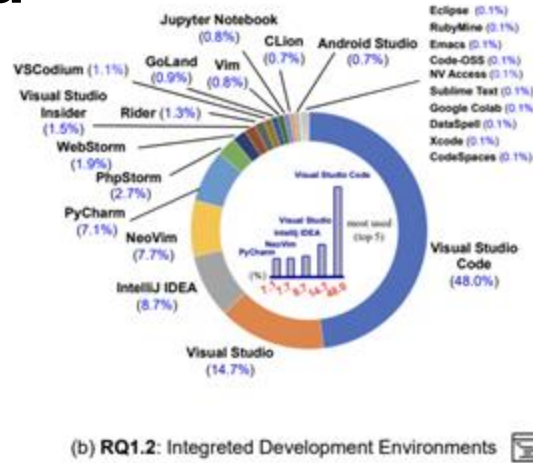
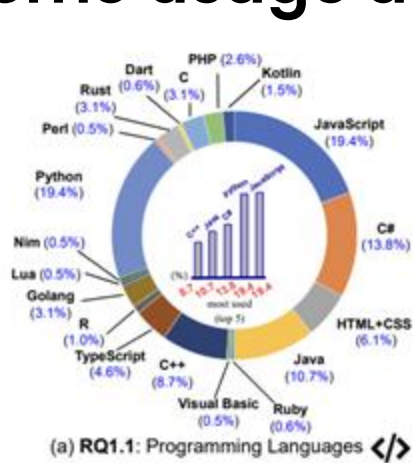


Figure 3: Percentage of code lines written by GitHub Copilot

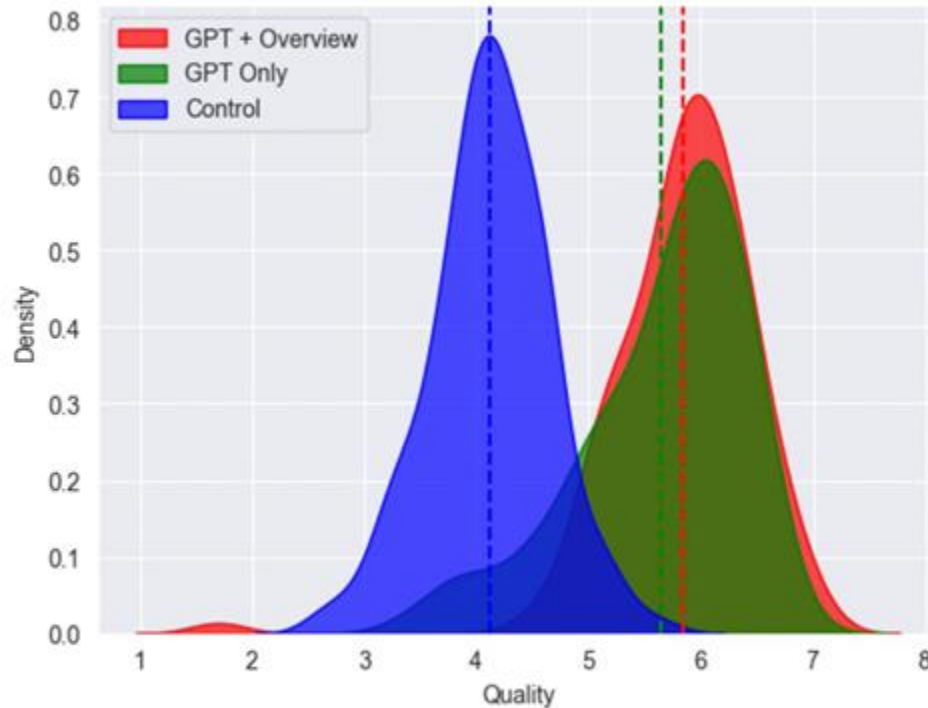
In Research Paper: GitHub Copilot: a systematic study

Some usage data



Some (relatable) usage data

Figure 2: Performance Distribution - Inside the Frontier



In Research Paper: Navigating the Jagged Technological Frontier: Field Experimental Evidence of the Effects of AI on Knowledge Worker Productivity and Quality

Notes: This figure displays the full distribution of performance in the experimental task inside the frontier for subjects in the three experimental groups (red for subjects in the GPT+Overview condition; green for subjects in the GPT Only condition; blue for subjects in the control condition).



AI tools for Software Engineers

AI coding tools will become a standard fixture of software development, say plenty of respondents:

"'Must have'" is hard to say – technically an IDE is not a 'must have' but it's pretty standard now. I expect this will be the case for AI coding tools in future." – a software engineer

"I'm sure it will be part of every IDE in one way or another. It saves keystrokes significantly, but it could be hard for juniors to spot issues or filter out garbage." – a senior software engineer

"Not sure if it will be a 'must have', but I believe they'll be like modern IDEs: a given. I'd like to say one could work without them, but when I tried doing without it, I quite missed it." – a software engineer



How to use Github Copilot





First you need to install it on your IDE

The screenshot shows the Visual Studio Code interface with the Extensions Marketplace open. The search bar contains "github copilot". The left sidebar lists several extensions, with "GitHub Copilot" at the top. The main panel displays the details for the "GitHub Copilot" extension by GitHub. The extension is described as "Your AI pair programmer" and has a rating of 4.5 stars (927 reviews). It is currently installed and enabled globally. The page includes tabs for Details, Feature Contributions, Changelog, and Runtime Status. A code snippet is shown in a dark-themed editor, demonstrating the use of the extension. The right sidebar shows categories like Programming Languages, Machine Learning, and Education, as well as extension resources and more information.

EXTENSIONS: MARKETPLACE

Search

github copilot

GitHub Copilot v1.186.367
GitHub · github.com · 8,576,841 · ★★★★★ (927)
Your AI pair programmer
Disable Uninstall Switch to Pre-Release Version
This extension is enabled globally.

DETAILS FEATURE CONTRIBUTIONS CHANGELOG RUNTIME STATUS

Get Code Suggestions in real-time, right in your IDE

```
1  
2 def common_prefix(a, b):  
3     """Return the common prefix of two lists."""  
4     if len(a) < len(b):  
5         return common_prefix(a, b)  
6     for i in range(len(a)):  
7         if a[i] != b[i]:  
8             return a[:i]  
9     return a  
10  
11 def test_common_prefix():  
12     assert common_prefix(['a', 'b', 'c'], ['a', 'b', 'd', 'e']) == ['a', 'b']  
13     assert common_prefix(['a', 'b', 'c'], ['a', 'b', 'c', 'd']) == ['a', 'b', 'c']  
14     assert common_prefix(['a', 'b', 'c'], ['a', 'b', 'c']) == ['a', 'b', 'c']  
15     assert common_prefix(['a', 'b', 'c'], ['a', 'b']) == ['a', 'b']  
16     assert common_prefix([], []) == []
```

What's GitHub Copilot

GitHub Copilot provides autocomplete-style suggestions from an AI pair programmer as you code. You can receive suggestions from GitHub Copilot either by starting to write the code you want to use, or by writing a natural language comment describing what you want the code to do.

Categories

- Programming Languages
- Machine Learning
- Education
- Snippets

Extension Resources

- Marketplace
- GitHub

More Info

- Published: 6/28/2021, 21:28:17
- Last released: 8/21/2023, 04:44:58
- Last updated: 8/21/2023, 19:08:17
- Identifier: github.copilot



How to use Github Copilot

1. You write a comment, and Copilot generates the code:

```
// function to diff two hash maps based on the value (date) and return the diff as an array sorted by date
// input: two hash maps with date as value
// ✨ output: an array of objects sorted by date
function diffHashmaps(map1, map2) {
  const diff = [];
  for (const [key, value] of map1.entries()) {
    if (!map2.has(key) || map2.get(key) !== value) {
      diff.push({ key, value });
    }
  }
  for (const [key, value] of map2.entries()) {
    if (!map1.has(key) || map1.get(key) !== value) {
      diff.push({ key, value });
    }
  }
  return diff.sort((a, b) => a.value - b.value);
}
```





How to use Github Copilot



2. Copilot auto-completes your code based on the remaining code in that file, and broader repository:

```
const jobTitleArray = json.map(job => job.job_data.job_title);
console.log('\njob.job_data.job_title:');
printHistogram(jobTitleArray, 30);

const remoteStatusArray = json.map(job => job.job_data.remote_status);
console.log('job.job_data.remote_status:');
printHistogram(remoteStatusArray, 30);

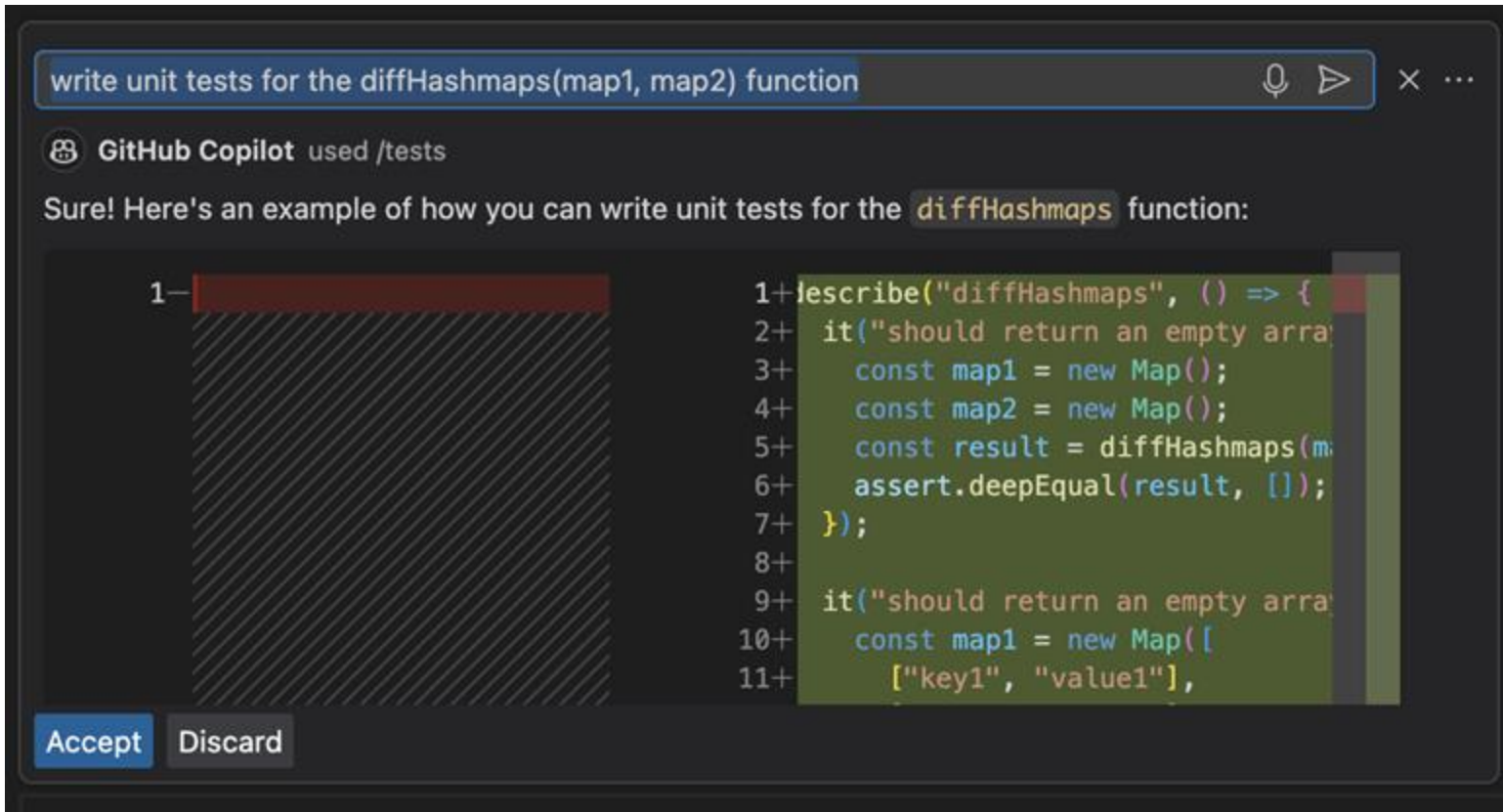
const minSalaryArray = json.map(job => job.job_data.min_salary);
```

 **const**  **const**

How to use Github Copilot

3. You ask specific requests in the Github Copilot chat assistant:



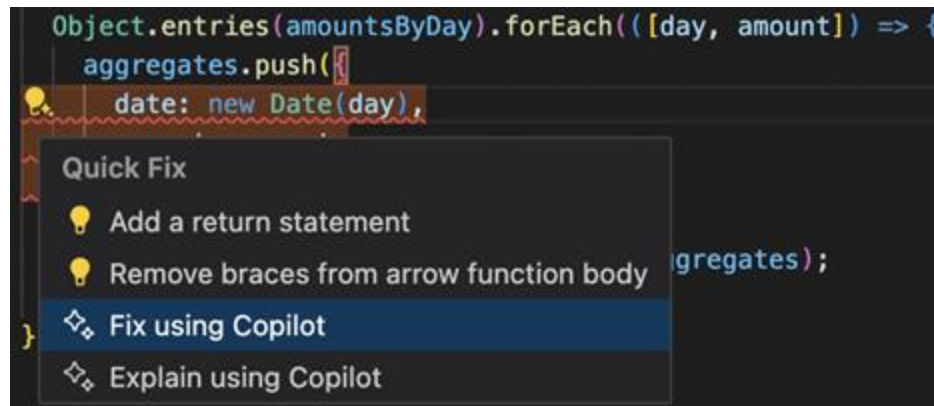
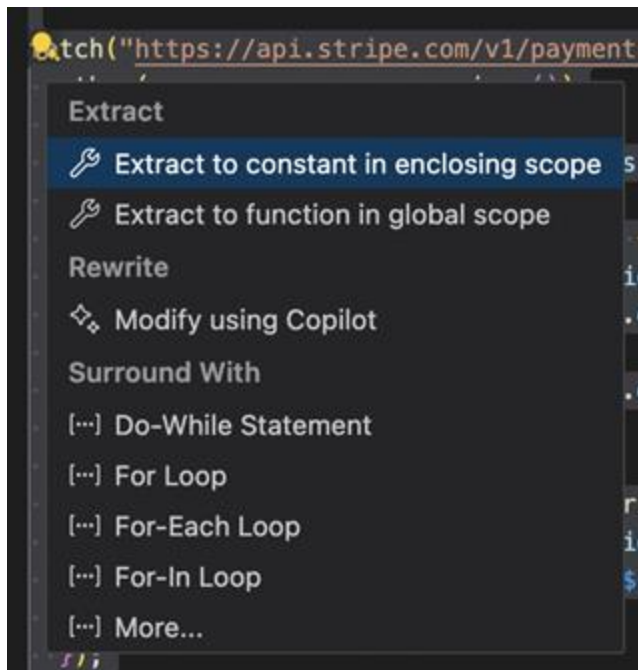
The screenshot shows the Github Copilot chat interface. At the top, a text input field contains the prompt "write unit tests for the diffHashmaps(map1, map2) function". Below the input field, the chat history shows a message from "GitHub Copilot" that says "Sure! Here's an example of how you can write unit tests for the `diffHashmaps` function:". The response includes a code block with the following content:

```
1+ describe("diffHashmaps", () => {
2+   it("should return an empty array", () => {
3+     const map1 = new Map();
4+     const map2 = new Map();
5+     const result = diffHashmaps(map1, map2);
6+     assert.deepEqual(result, []);
7+   });
8+
9+   it("should return an empty array", () => {
10+    const map1 = new Map([
11+      ["key1", "value1"],
```

At the bottom of the chat window, there are two buttons: "Accept" and "Discard".

How to use Github Copilot

The tooltip has contextual shortcuts:



First, the risks to avoid!





Risks for yourself

- Over-reliance on automation
- Skills atrophy
- Copilot's context is limited to the existing code (not the broader project and requirements)
- You might accept code that works, but has issues
- You might accept code that has been copyrighted by someone else



Risks for your team

- Project-wide code standards and quality assurance
- Intellectual property concerns
- Technical debt
- Upskilling junior team members



Ok, show me how to use it!

Let's dive into the 13 ways Github Copilot can help you.





Let's make this interactive!

I'll show you how to use Github Copilot in 13 use cases familiar to anyone developing software.

For each of them:

1. I'll briefly explain you the context when you'd use it
2. I'll show you how I enabled Github Copilot and what instructions I used.
3. I'll show you the result I got and what I'd do with it.
4. I'll empower you to try it too.
5. Feel free to ask questions!



1. Brainstorm implementation ideas

When starting a new project or task, we often spend a few hours searching for the best technologies to use, or the best use of an external API, or which database technology is the best fit.

Using Github Copilot at this stage can save a lot of time in this brainstorm process. You can ask for options, then trade offs between them, and then some boilerplate code as needed.

In this example I'm considering what's the best database technology for a new task I've picked up.



1. Brainstorm implementation ideas

Instructions:

I need to build the backend for an analytics dashboard. It will read data from several partners' API, in json format, crunch those into condensed data, and update data on the UI every minute. Should I use a relational or NoSQL database? Explain the tradeoffs.






1. Brainstorm implementation ideas

 sergiopereira-io

I need to build the backend for an analytics dashboard. It will read data from several partners' API, in json format, crunch those into condensed data, and update data on the UI every minute. Should I use a relational or NoSQL database? Explain the tradeoffs.

 GitHub Copilot

Used 1 reference >

Choosing between a relational database and a NoSQL database depends on the specific requirements of your application. Here are some factors to consider:

- 1. Data Structure:** If your data from the partners' APIs is structured and fits well into a table format, a relational database like MySQL or PostgreSQL might be a good choice. If the data is semi-structured or unstructured, or if the structure varies significantly between different partners, a NoSQL database like MongoDB or CouchDB might be more suitable.
- 2. Scalability:** If you expect your data volume to grow significantly and need to scale horizontally (across multiple servers), NoSQL databases are typically easier to scale in this way. Relational databases can also be scaled, but it's often more complex and involves techniques like sharding.
- 3. Consistency:** If strong consistency is required (i.e., you need to ensure that all reads receive the most recent write), relational databases are typically a better choice due to their ACID (Atomicity, Consistency, Isolation, Durability) properties. NoSQL databases often follow the BASE (Basically Available, Soft state, Eventual consistency) model, which provides eventual consistency.

4. Complex Queries: If you need to perform complex queries involving multiple tables and relationships, relational databases are typically more powerful. NoSQL databases can struggle with this, as they're designed for simple, fast queries with a predictable performance.

5. Real-time Processing: If you need to process and serve the data in real-time, NoSQL databases like MongoDB or CouchDB can be a good choice due to their ability to handle large volumes of data and their performance characteristics.

In your case, since you're dealing with JSON data from various partners and need to update the UI every minute, a NoSQL database might be a good fit. However, if you need to perform complex queries on the data or if consistency is a major concern, you might want to consider a relational database. It's also possible to use both in a polyglot persistence architecture, where you choose the database technology that best fits each part of your application.



1. Brainstorm implementation ideas

Exercise

In the Github Copilot chat, use the prompt template:

- I need to build ***[explain the task's requirements]***. Should I ***[list the options your considering]***? Explain the tradeoffs.

I need to build the backend for an analytics dashboard. It will read data from several partners' API, in json format, crunch those into condensed data, and update data on the UI every minute. Should I use a relational or NoSQL database? Explain the tradeoffs.





2. Generate boilerplate code

To build a new project from scratch, you can ask Github Copilot to create a scaffolding of the app you need, using the technologies, frameworks and versions of your preference.

In this example I want to build a Proof of Concept for an integration with a partner (Plaid).



2. Generate boilerplate code

Instructions:

Generate boilerplate code for an app that integrates with the Plaid API to pull the last month of transactions from a list of users in my MongoDB database. Use javascript, specifically the latest version of the express.js framework.





2. Generate boilerplate code

GitHub Copilot

Used 1 reference

Untitled-1:1

Here's a basic boilerplate for an Express.js application that integrates with the Plaid API to pull the last month of transactions for a list of users stored in a MongoDB database. This example uses the `mongoose` library to interact with MongoDB and `plaid` library to interact with the Plaid API.

```
const express = require('express');
const mongoose = require('mongoose');
const plaid = require('plaid');
const moment = require('moment');

const app = express();
const port = 3000;

// MongoDB connection
mongoose.connect('mongodb://localhost:27017/mydatabase', { useNewUrlParser:

// Plaid client
const plaidClient = new plaid.Client({
  clientID: 'Your Plaid Client ID',
  secret: 'Your Plaid Secret',
  env: plaid.environments.sandbox,
  options: { version: '2019-05-29' }
});

// User schema
const User = mongoose.model('User', new mongoose.Schema({
  name: String,
  plaidAccessToken: String
}));
```

```
app.get('/transactions', async (req, res) => {
  try {
    const users = await User.find({});
    const transactions = [];

    for (let user of users) {
      const response = await plaidClient.getTransactions(user.plaidAccessTok
      transactions.push(...response.transactions);
    }

    res.send(transactions);
  } catch (error) {
    console.error(error);
    res.status(500).send('Error occurred while fetching transactions');
  }
});

app.listen(port, () => {
  console.log(`App listening at http://localhost:${port}`);
});
```

1 vulnerability

- **hardcoded-credentials** Embedding credentials in source code risks unauthorized access

This code creates an Express.js server that listens on port 3000. It has a single route (`/transactions`) that fetches the last month of transactions for all users from the Plaid API and sends them as the response. If an error occurs during the fetch, a 500 status code and an error message are sent as the response.

Please replace `'Your Plaid Client ID'` and `'Your Plaid Secret'` with your actual Plaid Client ID and Secret. Also, replace `'mongodb://localhost:27017/mydatabase'` with your actual MongoDB connection string.

Note: This is a basic example and doesn't include error handling for individual API requests or pagination for the Plaid API requests. You'll likely need to add this in a production application.



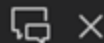
2. Generate boilerplate code

Exercise

In the Github Copilot chat, use the prompt template:

- Generate boilerplate code for an app that ***[explain what you need this app to do]***. Please use ***[explain what languages and frameworks should be used]***.

Generate boilerplate code for an app that integrates with the Plaid API to pull the last month of transactions from a list of users in my MongoDB database. Use javascript, specifically the latest version of the express.js framework.





3. Explain code

Many times our work as Software Developers requires us to understand a code base that wasn't built by us. Spaghetti code with no comments can be really slow and painful to understand.

In this example let's assume I landed on a new project, where code is undocumented and the original developer left the company. I ask Github Copilot to explain to me a key function of this code base.



3. Explain code

Instructions

```
fetch("https://api.stripe.com/v1/payments?created[gte]=last_month")
  .then(response => response.json())
  .then(data => {
    let transactions = data.transactions;
    let groupedTransactions = {};
    transactions.forEach(transaction => {
      if (!groupedTransactions[transaction.entity]) {
        groupedTransactions[transaction.entity] = transaction.amount;
      } else {
        groupedTransactions[transaction.entity] += transaction.amount;
      }
    });
    let sortedTransactions = Object.entries(groupedTransactions).sort((a, b) => b[1] - a[1]);
    sortedTransactions.forEach(transaction => {
      console.log(`${transaction[0]}: ${Math.ceil(transaction[1])}`);
    });
  });
```

Explain the code snippet highlighted.





3. Explain code

sergiopereira-io

Explain the code snippet highlighted.

GitHub Copilot

Used 1 reference >

This JavaScript code snippet uses the Fetch API to make a GET request to the Stripe API, specifically to retrieve payments that were created from the last month.

Here's a step-by-step explanation:

1. `fetch("https://api.stripe.com/v1/payments?created[gte]=last_month")`: This line sends a GET request to the Stripe API to retrieve payments created from the last month.
2. `.then(response => response.json())`: This line waits for the fetch promise to resolve, then takes the Response object and reads it to completion. It returns another promise that resolves with the result of parsing the body text as JSON.
3. `.then(data => {...})`: This line waits for the previous promise to resolve, then takes the resulting data and processes it.
4. Inside this block, the code first extracts the transactions from the data with `let transactions = data.transactions;`.

5. It then initializes an empty object `groupedTransactions` to store the total amount of transactions for each entity.
6. The `transactions.forEach(transaction => {...})` loop iterates over each transaction. If the entity of the transaction does not exist in `groupedTransactions`, it adds the entity as a key and sets its value to the transaction amount. If the entity already exists, it adds the transaction amount to the existing value.
7. `let sortedTransactions = Object.entries(groupedTransactions).sort((a, b) => b[1] - a[1]);` converts the `groupedTransactions` object into an array of [key, value] pairs, then sorts it in descending order based on the transaction amount.
8. The `sortedTransactions.forEach(transaction => {...})` loop iterates over each transaction in `sortedTransactions` and logs the entity and its total transaction amount to the console. The transaction amount is rounded up to the nearest whole number with `Math.ceil(transaction[1])`.

This code essentially fetches payments from the last month, groups them by entity, calculates the total transaction amount for each entity, sorts the entities in descending order of total transaction amount, and logs the results.



3. Explain code

Exercise

In your IDE, select the relevant code, then in the Github Copilot chat simply use the prompt:

Explain the code snippet highlighted.





4. Find bugs in code

Most of our time as Software Developers is actually spent tracking down bugs and fixing them. Github Copilot can reduce the amount of time we spend looking at buggy code.

In this example let's assume I'm debugging this function. For that, I explain Github Copilot the desired functionality and ask it what's wrong in the code.

4. Find bugs in code

```
fetch("https://api.stripe.com/v1/payments?created[gte]=last_month")
  .then(response => response.json())
  .then(data => {
    let transactions = data.transactions;
    let groupedTransactions = {};
    transactions.forEach(transaction => {
      if (groupedTransactions[transaction.entity]) {
        groupedTransactions[transaction.entity] = transaction.amount;
      } else {
        groupedTransactions[transaction.entity] += transaction.amount;
      }
    });
    let sortedTransactions = Object.entries(groupedTransactions).sort((a, b) => b[1] - a[1]);
    sortedTransactions.forEach(transaction => {
      console.log(`${transaction[0]}: ${Math.ceil(transaction[1])}`);
    });
  });
```

Find the bug in the code highlighted. This is what it should be doing:

1. Fetch the response from the Stripe API for payments received last month.
2. Parse the response JSON into an array with all transactions.
3. Traverse the array to group all transactions from the same entity, and sum their amounts. The result is stored in a different array.
4. Sort the resulting array by amount received, descending.
5. Write to the console all payments, sorted by date, with money amounts rounded up to the integer.




4. Find bugs in code

 sergiopereira-io

Find the bug in the code highlighted. This is what it should be doing:

1. Fetch the response from the Stripe API for payments received last month.
2. Parse the response JSON into an array with all transactions.
3. Traverse the array to group all transactions from the same entity, and sum their amounts. The result is stored in a different array.
4. Sort the resulting array by amount received, descending.
5. Write to the console all payments, sorted by date, with money amounts rounded up to the integer.

 GitHub Copilot

Used 1 reference >

The bug in your code is in the `forEach` loop where you're grouping transactions by entity and summing their amounts. You're currently assigning the transaction amount to the entity if it already exists in `groupedTransactions`, and adding the transaction amount to the entity if it doesn't exist. This should be the other way around.

Here's the corrected code:

```
fetch("https://api.stripe.com/v1/payments?created[gte]=last_month")
  .then(response => response.json())
  .then(data => {
    let transactions = data.transactions;
    let groupedTransactions = {};
    transactions.forEach(transaction => {
      if (groupedTransactions[transaction.entity]) {
        groupedTransactions[transaction.entity] += transaction.amount;
      } else {
        groupedTransactions[transaction.entity] = transaction.amount;
      }
    });
    let sortedTransactions = Object.entries(groupedTransactions).sort((a, b) => {
      return b[1] - a[1];
    });
    sortedTransactions.forEach(transaction => {
      console.log(`${transaction[0]}: ${Math.ceil(transaction[1])}`);
    });
  });
```

Now, if the entity already exists in `groupedTransactions`, the transaction amount is added to the existing amount. If the entity doesn't exist, it's added to `groupedTransactions` with the transaction amount as its value.



4. Find bugs in code

Exercise

In your IDE, select the relevant code, then in the Github Copilot chat simply use the prompt template:

- Find the bug in the code highlighted. This is what it should be doing:

[requirements to be fulfilled]

Find the bug in the code highlighted. This is what it should be doing:

1. Fetch the response from the Stripe API for payments received last month.
2. Parse the response JSON into an array with all transactions.
3. Traverse the array to group all transactions from the same entity, and sum their amounts. The result is stored in a different array.
4. Sort the resulting array by amount received, descending.
5. Write to the console all payments, sorted by date, with money amounts rounded up to the integer.



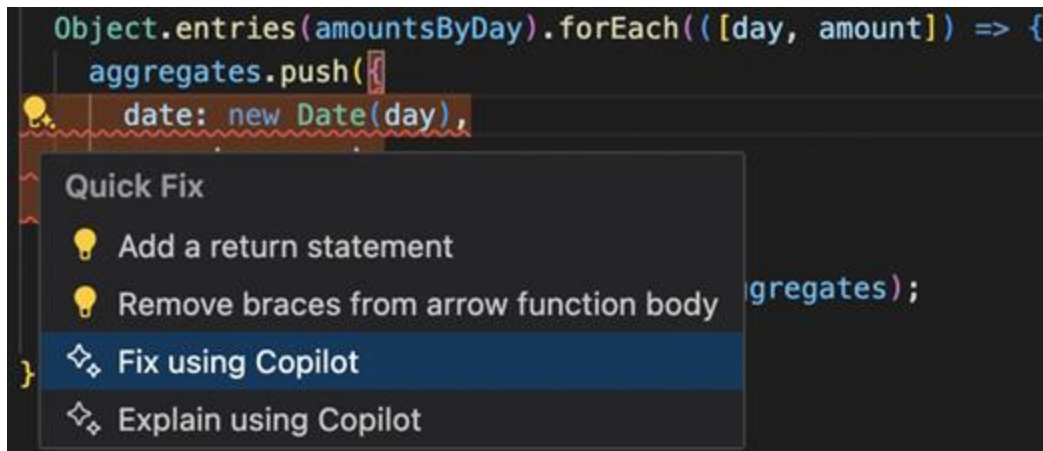
4. Find bugs in code

Exercise

In your IDE, select the relevant code, then in the Github Copilot chat simply use the prompt template:

- Find the bug in the code highlighted. This is what it should be doing:

[requirements to be fulfilled]



```
Object.entries(amountsByDay).forEach(([day, amount]) => {  
  aggregates.push({  
    date: new Date(day),  
  });  
});
```

The screenshot shows a code editor with a JavaScript snippet. A lightbulb icon indicates a bug or suggestion. A 'Quick Fix' menu is open, showing four options: 'Add a return statement', 'Remove braces from arrow function body', 'Fix using Copilot' (highlighted), and 'Explain using Copilot'.



5. Help with console errors

Many times, our code doesn't even run, and we're left to duel with console errors one by one.

In this example, let's assume that my code is throwing a console error when I run it, and I can't immediately understand what to fix, based on the stack trace. So I ask Github Copilot for help.

5. Help with console errors

```
fetch("https://api.stripe.com/v1/payments?created[gte]=last_month")
  .then(response => response.json())
  .then(data => {
    let transactions = data.transactions;
    let groupedTransactions = {};
    transactions.forEach(transaction => {
      if (!groupedTransactions[transaction.nonExistentProperty]) {
        groupedTransactions[transaction.nonExistentProperty] = transaction.amount;
      } else {
        groupedTransactions[transaction.nonExistentProperty] += transaction.amount;
      }
    });
    let sortedTransactions = Object.entries(groupedTransactions).sort((a, b) => b[1] - a[1]);
    sortedTransactions.forEach(transaction => {
      console.log(`${transaction[0]}: ${Math.ceil(transaction[1])}`);
    });
  });
```

The code highlighted is generating the following console error when I run it. What's the error about, and how can I fix it?

Uncaught (in promise) TypeError: Cannot read properties of undefined (reading 'nonExistentProperty')

at transactions.forEach.transaction (StripeWorker.js:7)
at Array.forEach (<anonymous>)
at StripeWorker.js:6
at <anonymous>




5. Help with console errors

 sergiopereira-io

The code highlighted is generating the following console error when I run it. What's the error about, and how can I fix it?

Uncaught (in promise) TypeError: Cannot read properties of undefined (reading 'nonExistentProperty') at transactions.forEach.transaction (StripeWorker.js:7) at Array.forEach () at StripeWorker.js:6 at

 GitHub Copilot

Used 1 reference >

The error message is indicating that the code is trying to access a property `nonExistentProperty` on an object that is `undefined`. This is happening in the `forEach` loop where you're trying to group transactions by `transaction.nonExistentProperty`.

The issue is likely that either `transaction` is `undefined` or `nonExistentProperty` does not exist on the `transaction` object.

To fix this, you should replace `nonExistentProperty` with the actual property name that exists on the `transaction` object and that you want to group by. For example, if you want to group by an `entity` property on the `transaction` object, the code would look like this:

```
fetch("https://api.stripe.com/v1/payments?created[gte]=last_month")
  .then(response => response.json())
  .then(data => {
    let transactions = data.transactions;
    let groupedTransactions = {};
    transactions.forEach(transaction => {
      if (!groupedTransactions[transaction.entity]) {
        groupedTransactions[transaction.entity] = transaction.amount;
      } else {
        groupedTransactions[transaction.entity] += transaction.amount;
      }
    });
    let sortedTransactions = Object.entries(groupedTransactions).sort((a, b) => {
      console.log(`${transaction[0]}: ${Math.ceil(transaction[1])}`);
    });
  });
```

If `transaction` itself is `undefined`, you should check the structure of the data returned from the Stripe API to ensure it matches what you're expecting.



5. Help with console errors

In your IDE, select the relevant code, then in the Github Copilot chat simply use the prompt template:

- The code highlighted is generating the following console error when I run it. What's the error about, and how can I fix it?

[paste the console stack trace of the error]

The code highlighted is generating the following console error when I run it. What's the error about, and how can I fix it?

```
Uncaught (in promise) TypeError: Cannot read properties of undefined
(reading 'nonExistentProperty')
    at transactions.forEach.transaction (StripeWorker.js:7)
    at Array.forEach (<anonymous>)
    at StripeWorker.js:6
    at <anonymous>
```





6. Refactor code

Sometimes we need to re-write code. Either to update a framework version within the same language, or to write it in a different language.

In this example let's assume I'm rewriting code from Javascript to Python, and asking Github Copilot to help me.



6. Refactor code

Instructions:

```
fetch("https://api.stripe.com/v1/payments?created[gte]=last_month")
  .then(response => response.json())
  .then(data => {
    let transactions = data.transactions;
    let groupedTransactions = {};
    transactions.forEach(transaction => {
      if (!groupedTransactions[transaction.entity]) {
        groupedTransactions[transaction.entity] = transaction.amount;
      } else {
        groupedTransactions[transaction.entity] += transaction.amount;
      }
    });
    let sortedTransactions = Object.entries(groupedTransactions).sort((a, b) => b[1] - a[1]);
    sortedTransactions.forEach(transaction => {
      console.log(`${transaction[0]}: ${Math.ceil(transaction[1])}`);
    });
  });
```

Refactor the code snippet highlighted to Python, specifically the latest version of the Flask framework.





6. Refactor code

sergiopereira-io

Refactor the code snippet highlighted to Python, specifically the latest version of the Flask framework.

GitHub Copilot



Used 1 reference >

Here's how you can refactor the JavaScript code to Python using the Flask framework and the `requests` library to make HTTP requests:

```
from flask import Flask
import requests
from collections import defaultdict
import math
from operator import itemgetter

app = Flask(__name__)
```

```
@app.route('/payments')
def get_payments():
    response = requests.get("https://api.stripe.com/v1/payments?created[gte]
    data = response.json()
    transactions = data['transactions']
    grouped_transactions = defaultdict(int)

    for transaction in transactions:
        grouped_transactions[transaction['entity']] += transaction['amount']

    sorted_transactions = sorted(grouped_transactions.items(), key=itemgetter

    for transaction in sorted_transactions:
        print(f"{transaction[0]}: ${math.ceil(transaction[1])}")

    return "Transactions processed. Check console for output."

if __name__ == '__main__':
    app.run(port=5000)
```

This Python code does the same thing as the JavaScript code: it fetches payments from the last month, groups them by entity, calculates the total transaction amount for each entity, sorts the entities in descending order of total transaction amount, and logs the results.

Please replace `"https://api.stripe.com/v1/payments?created[gte]=last_month"` with your actual Stripe API endpoint. Also, you might need to add authentication to the request depending on your Stripe API settings.



6. Refactor code

Exercise

In your IDE, select the relevant code, then in the Github Copilot chat simply use the prompt template:

- Refactor the code snippet highlighted to ***[language, framework and any other preferences for the refactored code]***.

[code you want to be refactored]

Refactor the code snippet highlighted to Python, specifically the latest version of the Flask framework.





7. Code review

Code review cycles with colleagues can take long lead times. Regression loops introduce context switching, which can be frustrating.

Using Github Copilot as a first code reviewer before opening a PR is a smart strategy. It not only points the improvements to be made, but it also suggests an improved version of your code.



7. Code review

```
const stripe = require('stripe')('YOUR_STRIPE_SECRET_KEY');
const MongoClient = require('mongodb').MongoClient;
const url = 'mongodb://localhost:27017';
const dbName = 'stripe_payments';

async function processErroredPayments() {
  const lastMonthStart = new Date();
  lastMonthStart.setMonth(lastMonthStart.getMonth() - 1);
  lastMonthStart.setDate(1);
  lastMonthStart.setHours(0, 0, 0, 0);
  const lastMonthEnd = new Date();
  lastMonthEnd.setDate(0);
  lastMonthEnd.setHours(23, 59, 59, 999);

  const erroredPayments = await stripe.paymentIntents.list({
    created: {
      gte: Math.floor(lastMonthStart.getTime() / 1000),
      lte: Math.floor(lastMonthEnd.getTime() / 1000),
    },
    status: 'requires_payment_method',
  });

  const amountsByDay = {};
  erroredPayments.data.forEach(payment => {
    const date = new Date(payment.created * 1000);
    date.setHours(0, 0, 0, 0);
    const day = date.toISOString().substring(0, 10);
    amountsByDay[day] = (amountsByDay[day] || 0) + payment.amount;
    payment.date = date;
  });

  const client = await MongoClient.connect(url);
  const db = client.db(dbName);

  const paymentsCollection = db.collection('payments');
  await paymentsCollection.insertMany(erroredPayments.data);

  const aggregatesCollection = db.collection('aggregates');
  const aggregates = [];
  Object.entries(amountsByDay).forEach(([day, amount]) => {
    aggregates.push(new Date(day), amount: amount);
  });
  await aggregatesCollection.insertMany(aggregates);


  client.close();
}
```

I'm opening a Pull Request with the code below. Write a code review for it, focusing on best practises such as code maintainability, security vulnerabilities and performance.





7. Code review

 sergiopereira-io

I'm opening a Pull Request with the code below. Write a code review for it, focusing on best practises such as code maintainability, security vulnerabilities and performance.

 GitHub Copilot

Used 1 reference >

Here are some points to consider for the code review:

1. **Use of modern import syntax:** In TypeScript, it's recommended to use the `import` syntax over `require`. This is more consistent with other TypeScript and ES6 modules.

```
import stripePackage from 'stripe';
import { MongoClient } from 'mongodb';
```

```
const stripe = stripePackage('YOUR_STRIPE_SECRET_KEY');
```

1 vulnerability >

2. **Hardcoded values:** The Stripe secret key and MongoDB connection details are hardcoded. It's better to move these to environment variables for security and flexibility.
3. **Error handling:** The code lacks error handling. Any issues with the Stripe API or MongoDB connection will cause unhandled promise rejections. Consider wrapping the code in a try-catch block.

4. **Date manipulation:** The date manipulation to get the start and end of the last month could be simplified or abstracted away to a helper function for better readability.
5. **Database operations:** The code inserts payments and aggregates into the database but doesn't check if they already exist. This could lead to duplicate data if the script is run multiple times. Consider checking for existing records before inserting.
6. **Type safety:** The code doesn't make use of TypeScript's type safety features. Consider defining interfaces for the payment and aggregate data.
7. **Closing the database connection:** The database connection is closed at the end of the function. If an error occurs before this, the connection will remain open. Consider using a finally block to ensure the connection is always closed.
8. **Unused code:** The `payment.date` property is set but never used. Consider removing this if it's not needed.
9. **Code organization:** The function does a lot of different things. Consider breaking it up into smaller functions, each with a single responsibility.
10. **Asynchronous operations:** The code uses `await` inside a `forEach` loop when inserting aggregates. This could be inefficient as it waits for each insert operation to complete before starting the next. Consider using `Promise.all` to perform these operations in parallel.
11. **Code Commenting:** There are no comments in the code. Adding comments explaining the purpose of the function and the logic behind key parts of the code can improve maintainability.



7. Code review

Exercise

In your IDE, select the relevant code, then in the Github Copilot chat simply use the prompt template:

- I'm opening a Pull Request with the code below. Write a code review for it, focusing on ***[what the code review should focus on]***.

I'm opening a Pull Request with the code below. Write a code review for it, focusing on best practises such as code maintainability, security vulnerabilities and performance.





8. Comment code

When writing code, we are often subject to tight deadlines, which lead us to compromise on delivery standards. One of those standards is to have our code properly commented.

In this example let's assume I've just written a piece of code, and I ask Github Copilot to write the code comments for me. Then I just need to review and make some corrections before opening a PR.

8. Comment code


```
fetch("https://api.stripe.com/v1/payments?created[gte]=last_month")
  .then(response => response.json())
  .then(data => {
    let transactions = data.transactions;
    let groupedTransactions = {};
    transactions.forEach(transaction => {
      if (!groupedTransactions[transaction.entity]) {
        groupedTransactions[transaction.entity] = transaction.amount;
      } else {
        groupedTransactions[transaction.entity] += transaction.amount;
      }
    });
    let sortedTransactions = Object.entries(groupedTransactions).sort((a, b) => b[1] - a[1]);
    sortedTransactions.forEach(transaction => {
      console.log(`${transaction[0]}: ${Math.ceil(transaction[1])}`);
    });
  });
```

Comment the code selected. Make sure
your comments follow code commenting
best practises.




8. Comment code

Comment the code selected. Make sure your comments follow code commenting best practises.

 GitHub Copilot

Made changes.

Accept Discard 

Accept or Discard 1 change.

```
// Fetch data from the Stripe API for payments created in the last month
fetch("https://api.stripe.com/v1/payments?created[gte]=last_month")
  .then(response => response.json()) // Convert the response to JSON
  .then(data => {
    let transactions = data.transactions; // Get the transactions from the data
    let groupedTransactions = {}; // Create an empty object to store grouped transactions

    // Iterate through each transaction
    transactions.forEach(transaction => {
      if (!groupedTransactions[transaction.entity]) {
        // If the entity does not exist in groupedTransactions, add it with the transaction amount
        groupedTransactions[transaction.entity] = transaction.amount;
      } else {
        // If the entity already exists in groupedTransactions, add the transaction amount to the existing value
        groupedTransactions[transaction.entity] += transaction.amount;
      }
    });

    // Sort the grouped transactions in descending order based on the transaction amount
    let sortedTransactions = Object.entries(groupedTransactions).sort((a, b) => b[1] - a[1]);

    // Iterate through the sorted transactions and print the entity and the rounded transaction amount
    sortedTransactions.forEach(transaction => {
      console.log(`${transaction[0]}: ${Math.ceil(transaction[1])}`);
    });
  });
});
```



8. Comment code

Exercise

In your IDE, select the relevant code, then in the Github Copilot chat simply use the prompt template:

- Comment the code selected. Make sure your comments are clear and follow ***[your preferred code commenting standards]***.

Comment the code selected. Make sure
your comments follow code commenting
best practises.





9. Write test cases

When writing code, we are often subject to tight deadlines, that lead us to compromise on delivery standards. Not having a good enough test coverage is often a result of such pressures.

In this example let's assume I've just written a function, and I ask Github Copilot to write the test cases for it. Then I just need to review and make some corrections before opening a PR.

9. Write test cases

```
async function fetchAndProcessPayments(url) {  
  const response = await fetch(url);  
  const data = await response.json();  
  let transactions = data.transactions;  
  let groupedTransactions = {};  
  transactions.forEach(transaction => {  
    if (!groupedTransactions[transaction.entity]) {  
      groupedTransactions[transaction.entity] = transaction.amount;  
    } else {  
      groupedTransactions[transaction.entity] += transaction.amount;  
    }  
  });  
  let sortedTransactions = Object.entries(groupedTransactions).sort((a, b) => b[1] - a[1]);  
  return sortedTransactions.map(transaction => `${transaction[0]}: ${Math.ceil(transaction[1])}`);  
}
```

Write unit tests for the main edge cases on the code selected. You should follow Gherking testing best practises. Your tests should be written in Javascript, using the Jest framework.





9. Write test cases

 sergiopereira-io

Write unit tests for the main edge cases on the code selected. You should follow Gherking testing best practises. Your tests should be written in Javascript, using the Jest framework.

 GitHub Copilot

Used 1 reference >

Here's how you can write unit tests for the selected code using Jest. Note that to make the code testable, you'll need to mock the `fetch` function to simulate different responses from the API.

```
const fetch = require('node-fetch');
jest.mock('node-fetch');

const { fetchAndProcessPayments } = require('./fetchAndProcessPayments');

describe('fetchAndProcessPayments', () => {
  it('should group transactions by entity and sort them in descending order'
    fetch.mockResolvedValueOnce({
      json: () => Promise.resolve({
        transactions: [
          { entity: 'A', amount: 10 },
          { entity: 'B', amount: 20 },
          { entity: 'A', amount: 30 },
          { entity: 'C', amount: 40 },
          { entity: 'B', amount: 50 },
          { entity: 'C', amount: 60 },
        ],
      }),
    }),
  );

  const result = await fetchAndProcessPayments('https://api.stripe.com/v1/');
  expect(result).toEqual(['C: $100', 'B: $70', 'A: $40']);
});
```

```
it('should handle empty transactions array', async () => {
  fetch.mockResolvedValueOnce({
    json: () => Promise.resolve({
      transactions: [],
    }),
  });

  const result = await fetchAndProcessPayments('https://api.stripe.com/v1/');
  expect(result).toEqual([]);
});

it('should round up transaction amounts', async () => {
  fetch.mockResolvedValueOnce({
    json: () => Promise.resolve({
      transactions: [
        { entity: 'A', amount: 10.1 },
        { entity: 'B', amount: 20.5 },
        { entity: 'A', amount: 30.9 },
      ],
    }),
  });

  const result = await fetchAndProcessPayments('https://api.stripe.com/v1/');
  expect(result).toEqual(['A: $42', 'B: $21']);
});
```

These tests cover the main edge cases:

1. Transactions are grouped by entity and sorted in descending order of total amount.
2. An empty transactions array is handled correctly.
3. Transaction amounts are rounded up to the nearest whole number.



9. Write test cases

Exercise

In your IDE, select the relevant code, then in the Github Copilot chat simply use the prompt template:

- Write unit tests for ***[cases to be tested]*** on the code selected. You should follow ***[unit testing guidelines to follow]***. Your tests should be written in ***[language and framework to be used]***.

Write unit tests for the main edge cases on the code selected. You should follow Gherking testing best practises. Your tests should be written in Javascript, using the Jest framework.





10. Generate regexes (and other hard syntax)

One of the worst parts of a Software Developer's job is to speak machine language. While programming languages have evolved in their abstraction, the syntax for some expressions haven't.

Some such examples are regular expressions, cron expressions, etc.

Github Copilot can totally offload that from you, and it's easier to use than most online widgets you'd find through Google.



10. Generate regexes (and other hard syntax)

Instructions:

```
// Create variable to store a regex to match a phone number from the United States
```



10. Generate regexes (and other hard syntax)

✨ Create variable to store a regex to match a phone number from the United States

```
const usPhoneNumberRegex = /\d{3}-\d{3}-\d{4}/;
```

< 3/3 > Accept Tab Accept Word ⌘ → ... match a phone number from the United States

```
const usPhoneNumberRegex = /\(?(?(\d{3}))\)?[-. ]?(?(\d{3}))[-. ]?(?(\d{4}))\)/;
```



10. Generate regexes (and other hard syntax)

Exercise

In your IDE, write the following comment and hit the Enter key:

- Create variable to store *[item with hard syntax you need]* to match *[what you need it to match described in natural language]*.

```
// Create variable to store a regex to match a phone number from the United States
```



11. Generate test data

Generating dummy data to test our code can be tedious. We either generate a few entries manually or we write a script to generate a larger volume of data.

In either case, Github Copilot can help a lot doing this instantly for you.

11. Generate test data

```
CREATE TABLE bank_accounts (  
  account_id SERIAL PRIMARY KEY,  
  account_number VARCHAR(20) UNIQUE NOT NULL,  
  account_type VARCHAR(50) NOT NULL,  
  balance DECIMAL(15, 2) NOT NULL,  
  user_id INTEGER NOT NULL,  
  created_at TIMESTAMP NOT NULL DEFAULT NOW(),  
  updated_at TIMESTAMP NOT NULL DEFAULT NOW(),  
  FOREIGN KEY (user_id) REFERENCES users(user_id)  
);  
  
CREATE TABLE financial_statements (  
  statement_id SERIAL PRIMARY KEY,  
  account_id INTEGER NOT NULL,  
  date DATE NOT NULL,  
  starting_balance DECIMAL(15, 2) NOT NULL,  
  ending_balance DECIMAL(15, 2) NOT NULL,  
  FOREIGN KEY (account_id) REFERENCES bank_accounts(account_id)  
);
```

Generate SQL insert statements to populate these tables with 10 entries of dummy data.



11. Generate test data

```
Generate SQL insert statements to populate these tables with 10 entries of dummy data.

GitHub Copilot
Made changes.
Accept Discard Accept or Discard 1 change.

-- Insert statements for bank_accounts table
INSERT INTO bank_accounts (account_number, account_type, balance, user_id)
VALUES
('1234567890', 'Savings', 1000.00, 1),
('9876543210', 'Checking', 500.00, 2),
('5555555555', 'Savings', 2000.00, 3),
('9999999999', 'Checking', 1500.00, 4),
('1111111111', 'Savings', 3000.00, 5),
('2222222222', 'Checking', 2500.00, 6),
('3333333333', 'Savings', 4000.00, 7),
('4444444444', 'Checking', 3500.00, 8),
('5555555555', 'Savings', 5000.00, 9),
('6666666666', 'Checking', 4500.00, 10);

-- Insert statements for financial_statements table
INSERT INTO financial_statements (account_id, date, starting_balance, ending_balance)
VALUES
(1, '2022-01-01', 1000.00, 1200.00),
(2, '2022-01-01', 500.00, 600.00),
(3, '2022-01-01', 2000.00, 2200.00),
(4, '2022-01-01', 1500.00, 1800.00),
(5, '2022-01-01', 3000.00, 3200.00),
(6, '2022-01-01', 2500.00, 2700.00),
(7, '2022-01-01', 4000.00, 4200.00),
(8, '2022-01-01', 3500.00, 3800.00),
(9, '2022-01-01', 5000.00, 5200.00),
(10, '2022-01-01', 4500.00, 4800.00);
```



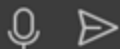
11. Generate test data

Exercise

In your IDE, select the relevant tables, then in the Github Copilot chat simply use the prompt template:

- Generate SQL insert statements to populate the following tables with ***[number of entries]*** entries of dummy data ***[requirements for the data to be generated]***.

Generate SQL insert statements to populate
these tables with 10 entries of dummy data.





12. Convert file formats (eg: XML to JSON)

In our daily job, we use long files with reference data, or payload from some data provider. Many times, we need to convert those files from one format to another.

We either use some online widget, or we write a script to do the conversion. Github Copilot can do it instantly for us.



12. Convert file formats (eg: XML to JSON)

```
✦ xml version="1.0" encoding="UTF-8"?>
<transactions>
  <transaction>
    <date>2023-04-20</date>
    <description>ATM Withdrawal</description>
    <amount>-100.00</amount>
    <category>Withdrawals</category>
  </transaction>
  <transaction>
    <date>2023-04-19</date>
    <description>Deposit</description>
    <amount>500.00</amount>
    <category>Deposits</category>
  </transaction>
  <transaction>
    <date>2023-04-18</date>
    <description>Online Payment</description>
    <amount>-50.00</amount>
    <category>Payments</category>
  </transaction>
</transactions>
```

Convert this XML to JSON.





12. Convert file formats (eg: XML to JSON)

```
Convert this XML to JSON.
GitHub Copilot
Made changes.
Accept Discard Accept or Discard 2 changes.

import xmltodict
import json

# XML input
xml = '''
<?xml version="1.0" encoding="UTF-8"?>
<transactions>
  <transaction>
    <date>2023-04-20</date>
    <description>ATM Withdrawal</description>
    <amount>-100.00</amount>
    <category>Withdrawals</category>
  </transaction>
  <transaction>
    <date>2023-04-19</date>
    <description>Deposit</description>
    <amount>500.00</amount>
    <category>Deposits</category>
  </transaction>
  <transaction>
    <date>2023-04-18</date>
    <description>Online Payment</description>
    <amount>-50.00</amount>
    <category>Payments</category>
  </transaction>
</transactions>
'''

# Convert XML to OrderedDict
data_dict = xmltodict.parse(xml)

# Convert OrderedDict to JSON
json_data = json.dumps(data_dict, indent=2)

# Print JSON output
print(json_data)
```



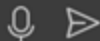
12. Convert file formats (eg: XML to JSON)

Exercise

In your IDE, select the relevant contents, then in the Github Copilot chat simply use the prompt template:

- Convert this *[current file format]* to *[new file format]*.

Convert this XML to JSON.





13. Take home interview challenges

A key part of any interview process for Software Developer roles is a technical interview. Some companies do it as a live interview, in which case it would be challenging to use Github Copilot. But other companies use take home challenges, which you can complete on your own time.

Github Copilot can be a key ally in giving you implementation ideas and actual code to solve these challenges.

Note: You should be prepared to defend your solution in a follow up interview, even if you used Github Copilot.

13. Take home interview challenges

Instructions:

Generate code in javascript to solve the following challenge.

Context:

- We have one 2D array, filled with zeros and ones.
- We have to find the starting point and ending point of all rectangles filled with 0.
- It is given that rectangles are separated and do not touch each other however they can touch the boundary of the array.
- A rectangle might contain only one element.


Desired output:

- You should return an array, each element representing one rectangle.
- Each of those array elements contains an array with 4 elements that compose the rectangle (top left X, top left Y, bottom right X, bottom right Y).

Example arrays:

```
input1 = [ [1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1], [1, 1, 1, 0, 0, 0], [1, 1, 1, 0, 0, 0], [1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1] ]
```

```
input2 = [ [0, 1, 1, 1, 1, 1, 0], [1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 0, 0, 0, 1], [1, 1, 1, 0, 0, 0, 1], [1, 1, 1, 1, 1, 1, 1], [1, 0, 0, 1, 1, 1, 1], [1, 0, 0, 1, 1, 0, 0], [1, 0, 0, 1, 1, 0, 0] ]
```

 You can also type / for commands



13. Take home interview challenges

GitHub Copilot

Used 1 reference >

Here's a step-by-step plan:

1. Initialize an empty array to store the rectangles.
2. Iterate over the 2D array with nested loops.
3. When a 0 is found, start a new rectangle with the current coordinates as the top left corner.
4. Continue iterating to the right until a 1 is found or the end of the row is reached. This will give the X coordinate of the bottom right corner.
5. Then iterate downwards until a 1 is found or the end of the column is reached. This will give the Y coordinate of the bottom right corner.
6. Add the rectangle to the array and continue the search from the next unvisited cell.
7. Return the array of rectangles.

Here's the JavaScript code:

```
function findRectangles(array) {
  let rectangles = [];
  let visited = array.map(row => row.map(() => false));

  for (let i = 0; i < array.length; i++) {
    for (let j = 0; j < array[i].length; j++) {
      if (array[i][j] === 0 && !visited[i][j]) {
        let topLeftX = i;
        let topLeftY = j;
        let bottomRightX = i;
        let bottomRightY = j;

        while (bottomRightY + 1 < array[i].length && array[i][bottomRightY] === 0) {
          bottomRightY++;
        }

        while (bottomRightX + 1 < array[0].length && array[bottomRightX][j] === 0) {
          bottomRightX++;
        }

        for (let x = topLeftX; x <= bottomRightX; x++) {
          for (let y = topLeftY; y <= bottomRightY; y++) {
            visited[x][y] = true;
          }
        }

        rectangles.push([topLeftX, topLeftY, bottomRightX, bottomRightY]);
      }
    }
  }

  return rectangles;
}
```

This function will return an array of rectangles for each 0-filled rectangle in the input array. Each rectangle is represented as an array of 4 coordinates: [topLeftX, topLeftY, bottomRightX, bottomRightY].



13. Take home interview challenges

Exercise

In your IDE, in the Github Copilot chat simply use the prompt template:

- Generate code in ***[desired language]*** to solve the following challenge:

[outline of the challenge to be solved]

Generate code in javascript to solve the following challenge.

Context:

- We have one 2D array, filled with zeros and ones.
- We have to find the starting point and ending point of all rectangles filled with 0.
- It is given that rectangles are separated and do not touch each other however they can touch the boundary of the array.
- A rectangle might contain only one element.

Desired output:

- You should return an array, each element representing one rectangle.
- Each of those array elements contains an array with 4 elements that compose the rectangle (top left X, top left Y, bottom right X, bottom right Y).

Example arrays:

input1 = [[1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1], [1, 1, 1, 0, 0, 0], [1, 1, 1, 0, 0, 0], [1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1]]

input2 = [[0, 1, 1, 1, 1, 0], [1, 1, 1, 1, 1, 1], [1, 1, 1, 0, 0, 0], [1, 1, 1, 0, 0, 0], [1, 1, 1, 1, 1, 1], [1, 0, 0, 1, 1, 1], [1, 0, 0, 1, 1, 0], [1, 0, 0, 1, 1, 0]]

You can also type / for commands



Takeaways

Q&A



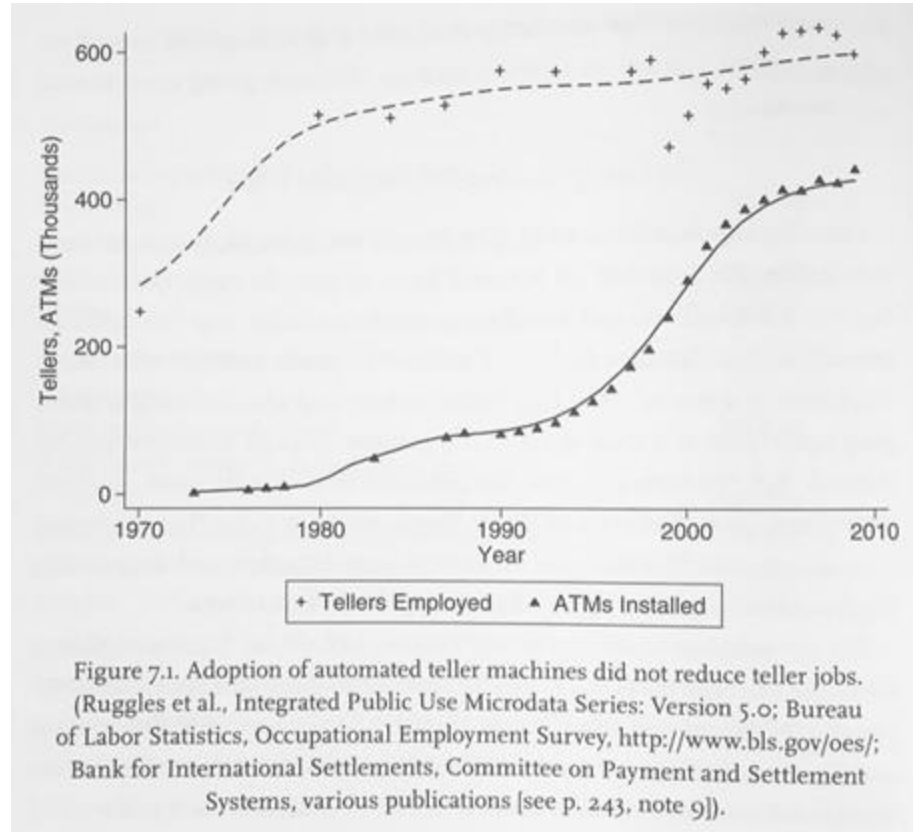


Key takeaways

- Github Copilot can help. A lot!
- Tasks that would take 1 hour, can now take seconds or minutes.
- Github Copilot has known limitations and risks.
- Github Copilot is better than other popular AI tools (eg: ChatGPT) for its repository-wide context.
- AI tools are in their infancy, they'll likely get better.

“But... will these AI tools replace Software Engineers?”

Will AI replace Software Engineers?



The ATM case study, 50 years ago:

- ATM machines were invented, and all market analysts predicted the bank teller job would disappear in a few years.
- ATM machines indeed replaced several functions that used to be done by bank tellers, but not all.
- ATM machines made it cheaper than ever to open new bank branches, which accelerated market growth.
- The net number of bank teller jobs grew massively after ATM machines were invented.



Will AI replace Software Engineers?



Gergely Orosz
@GergelyOrosz

x1

Elevators with buttons killed the “elevator operator” job completely

At the same time, spreadsheet apps like Excel did not kill accounting jobs - they helped create more

Understand why each happened and you understand how innovation can both reduce and increase employment/jobs



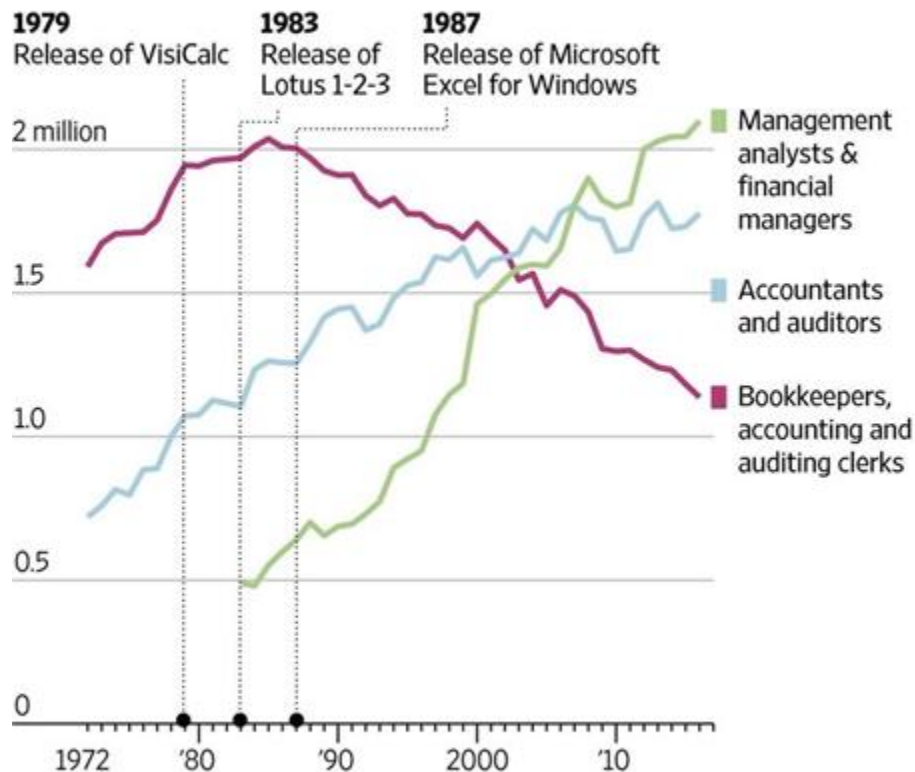
Duncan Mac-Vicar P.
@duncan.mac-vicar.eu

Similar: at some point people feared email would kill postal services.

Nobody foresaw e-commerce.

The Spreadsheet Apocalypse, Revisited

Jobs in bookkeeping plummeted after the introduction of spreadsheet software, but jobs in accounting and analysis took off.



Notes: There is no data for 1982. Changes in occupational definitions in 1983, 2000 and 2011 mean that data is not strictly comparable across time. There was no category for management analysts or financial managers prior to 1983.

Source: Bureau of Labor Statistics

THE WALL STREET JOURNAL.





Q & A





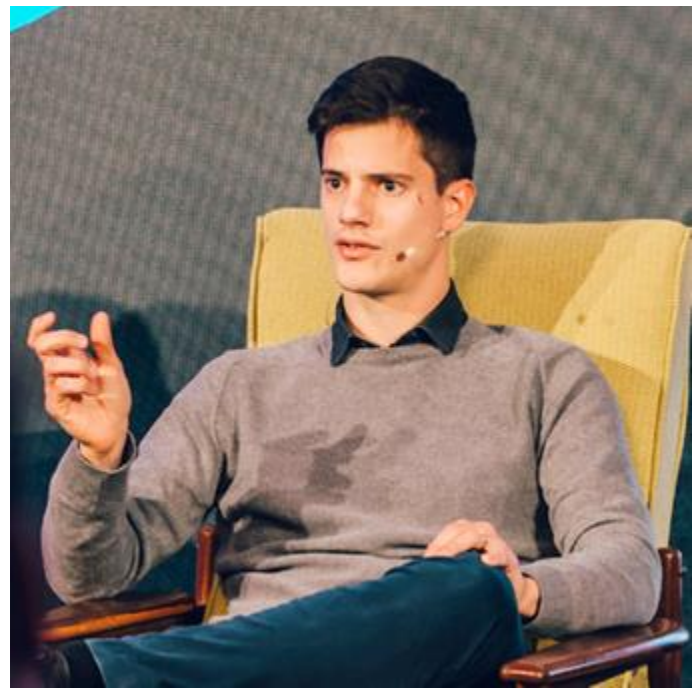
About the Author

I'm Sergio Pereira

- I've worked in tech since 2009
- Started as a Software Engineer, became a CTO in 2014
- Currently work as a Fractional CTO for Tech Startups
- Embedding AI tools like Github Copilot to my clients' software development processes

I'm writing daily on:

- Twitter: **@SergioRocks**
- LinkedIn: **/sergiomcpereira**





Generative AI for Software Development

by [Sergio Pereira](#)

Released May 2025

Publisher(s): O'Reilly Media, Inc.

ISBN: 9781098162276

(*) GPT

ChatGPT for Software Engineers

Published by [O'Reilly Media, Inc.](#)

📖 Beginner to intermediate

How to 10X your productivity with generative AI

Cursor for Software Engineers

Published by [O'Reilly Media, Inc.](#)

📖 Intermediate

How to boost your productivity with generative AI

The image features the O'Reilly logo in white, bold, sans-serif capital letters, centered horizontally. The background is a solid blue gradient, transitioning from a darker blue on the left to a lighter blue on the right. On the left side, there are several large, faint, overlapping circular shapes in various shades of blue, creating a layered, abstract effect.

O'REILLY®