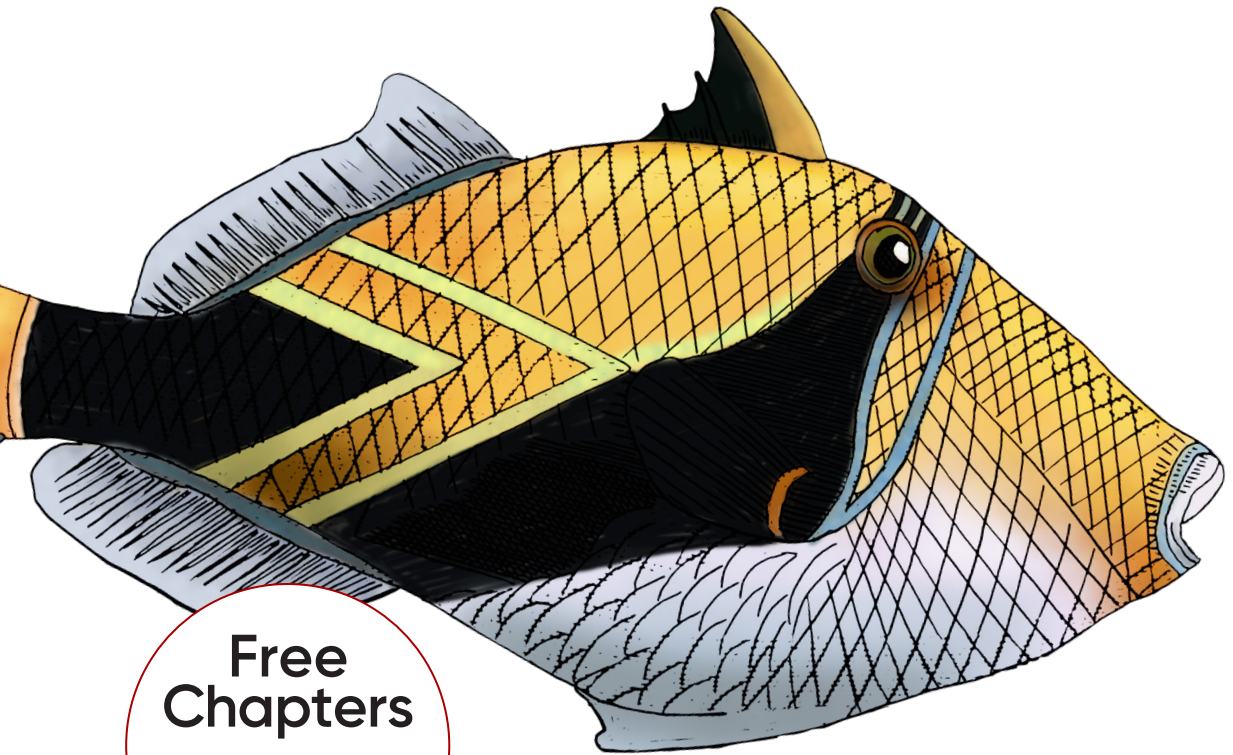# AI-Assisted Programming

Better Planning, Coding, Testing, and Deployment

Free
Chapters

Compliments of

Coder

Tom Taulli

Coder

Safely deploy AI coding agents at enterprise scale

Get started

Add Unit Test for API
10s ago: Identify target API endpoint...
Agent Working ●

Update Dependencies
55s ago: Scanning for outdated packages...
Agent Working ●

Fix Mobile Layout Bug
16m ago: Preview ready...
Needs Feedback ●

Update Deprecated Function Calls
5s ago: Locating preferred replacements...
Agent Working ●

Add Linter and Auto-Formatter
30s ago: Committing changes with context...
Agent Working ●

Fix Typo in User-Facing Text
8m ago: Preview ready...
Needs Feedback ●

AI-Assisted Programming
Better Planning, Coding, Testing, and Deployment

This excerpt contains Chapters 1, 8, and 9. The complete book is available on the O'Reilly Online Learning Platform and through other retailers.

Tom Taulli

O'REILLY®

Table of Contents

New World for Developers

While juggling dense neural network architectures and pixel-wrangling computer vision at Stanford from 2011 to 2016, Andrej Karpathy also moonlighted at Google. Over there, he tinkered around and whipped up a feature-learning system for You-Tube videos. Then he decided to become a founding member of OpenAI and later the senior director of AI at Tesla, where he led a team to create the Autopilot system.

It's safe to say he's one the world's top coders. He is also a skilled wordsmith with a massive Twitter—or X—following of nearly 800,000 followers. When ChatGPT catapulted onto the scene, he tweeted:

> The hottest new programming language is English.

He wasn't kidding. This wasn't just a poetic ode to coding but a nod to a future where typing out natural language prompts could conjure up computer code in seemingly any language. It's like having a bilingual genie in your computer, ready to transcribe your English wishes into code commands.

Then there came a tweet that echoed the sentiments of many developers:

> Copilot has dramatically accelerated my coding, it's hard to imagine going back to "manual coding". Still learning to use it but it already writes ~80% of my code, ~80% accuracy. I don't even really code, I prompt. & edit.

Karpathy was tipping his hat to Microsoft's GitHub Copilot, a fresh brew of AI-assisted programming. But it wouldn't be long until many other tools sprouted up. The pace of innovation was breathtaking.

Now, for all the coders out there, the landscape might look like a dense jungle. What's this brave new world of AI tools? Where do they dazzle, and where do they fizzle? And how do you wade through all this to become a savvy AI-assisted programmer?

Well, this book will be your guide to help answer these questions—and many more. The spotlight will be on harnessing these tools to code not just faster but smarter, and with a sprinkle of fun. So, let's roll up our sleeves and jump into this AI-assisted programming journey.

Evolution and Revolution

A key theme of the evolution of programming languages is *abstraction*. This is a fancy way of describing how systems get easier for developers to use. When the tedious details are handled in the background, developers can focus on what matters most. This has been a driving force of innovation, allowing for breakthroughs like the internet, cloud computing, mobile, and AI.

Figure 1-1 highlights the evolution of abstraction over the decades.
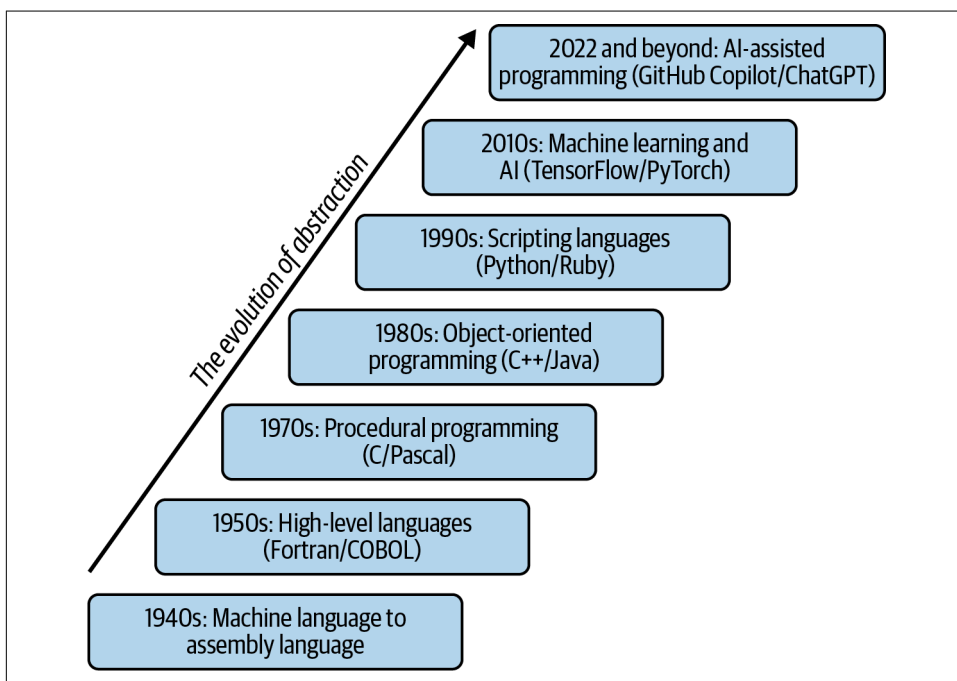
Figure 1-1. The abstraction of programming languages and tools has evolved over the decades

Let's go into more detail, starting from the 1940s:

Machine language to assembly language
At the dawn of the computer age, programmers had to wrestle with 0s and 1s to bend machines to their will. But then, assembly language came onto the scene. It offered alphanumeric instructions, which made coding easier and less error-prone.

High-level languages
The 1950s brought us Fortran and COBOL, languages that let programmers code using somewhat plain English like DISPLAY, READ, WRITE, and IF/THEN/ELSE. A compiler would convert these into the 0s and 1s that a computer could understand. At the same time, people without a technical background could generally read the code well enough to understand the workflow. The emergence of high-level languages would be a huge catalyst for the computer revolution.

Procedural programming
Languages like C and Pascal introduced procedural programming, essentially packing complex tasks into neat little boxes called functions. This abstraction allowed for reusability and maintainability, and it made managing colossal software projects less of a Herculean task.

Object-oriented programming (OOP)
Some of the stars of this type of computer language include C++ and Java. Object-oriented programming brought a whole new level of abstraction, allowing programmers to model real-world entities using classes and objects, encapsulating both data and behavior. This promoted modularity and allowed for more intuitive problem solving.

Scripting languages and web development
Python, Ruby, and JavaScript abstract many of the lower-level tasks associated with programming. They offer extensive libraries and built-in data structures, simplifying common programming tasks and reducing the amount of code needed to accomplish them.

Machine learning and AI
With the rise of AI and machine learning, specialized libraries and frameworks like TensorFlow and PyTorch have abstracted away many intricate mathematical details of programming. This has enabled developers to focus on model architecture and training processes.
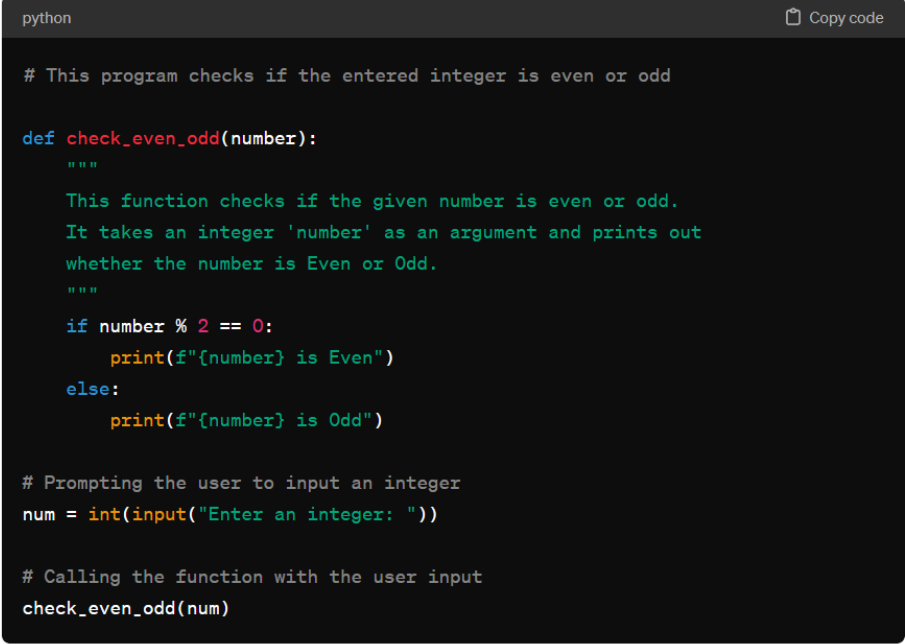
AI-assisted programming

Of course, the latest entrant to this abstraction narrative is AI-assisted programming, á la GPT-4 and other massive large language models (LLMs). These are like your backstage crew, ready to pitch in with code generation at your command.

Let's look at a simple example. For this, we'll use ChatGPT, which has a robust ability to gin up code. We will use a prompt to ask what we want the system to do. Suppose we give it the following prompt:

Prompt: In Python, write a program that checks if a given integer is even or odd and print the result.

Figure 1-2 shows the response from ChatGPT.

```python
# This program checks if the entered integer is even or odd

def check_even_odd(number):
    """
    This function checks if the given number is even or odd.
    It takes an integer 'number' as an argument and prints out
    whether the number is Even or Odd.
    """
    if number % 2 == 0:
        print(f"{number} is Even")
    else:
        print(f"{number} is Odd")

# Prompting the user to input an integer
num = int(input("Enter an integer: "))

# Calling the function with the user input
check_even_odd(num)
```

Description of the Program

1. **Function Definition (`check_even_odd`):**
   - The function `check_even_odd` is defi ↓ to determine if a number is even or odd.
   - It takes one parameter, `number`, which is the integer to be checked.

Figure 1-2. When asked to create code, ChatGPT's response will include not only a listing but an explanation

We get the code listing, which even comes with helpful comments. Then there is also an explanation of how the program works. You can press the Copy code button at the top right to include the code in your IDE and run it.

Generative AI

Before we go deeper into how AI-assisted programming tools work, let's get an overview of generative AI. This is the foundation of these systems.

Generative AI is a branch of artificial intelligence (AI), which allows for the creation of new and unique content. Figure 1-3 provides a visual of how the different parts relate to each other.
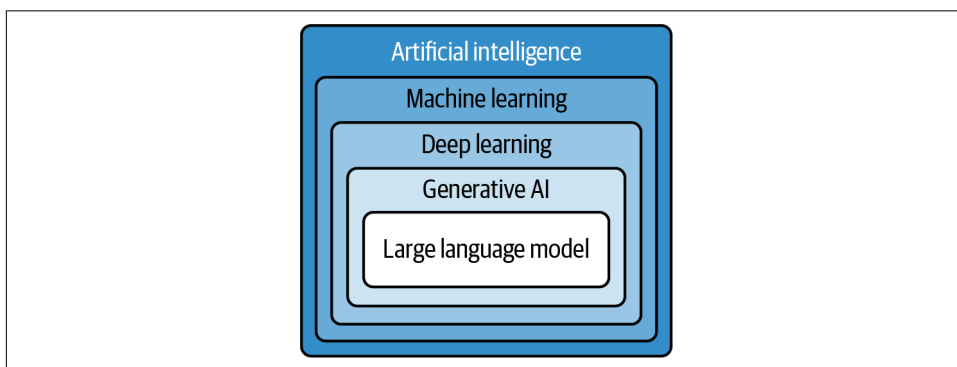
Figure 1-3. There are different types of AI, and they can be represented as nested subsets, with generative AI and finally large language models at the center

AI is the big umbrella: it includes all systems that can pull off tasks with the flair of human intelligence. Tucked within AI is machine learning (ML). Instead of marching to the beat of explicit instructions, ML systems come up with insights based on heaps of data. ML is generally based on complex algorithms, which allow for making predictions or decisions without hardcoding.

Take a step deeper, and you get deep learning (DL), a tighter slice of ML that rolls with neural networks stacked with hidden layers—hence the *deep* tag. These stacked models have shown standout results in areas like image and speech recognition.

Within the corridors of deep learning, you'll find generative AI (or GenAI). GenAI models create new data that reflects their training data.

In the innermost circle sits LLMs, such as GPT-4, Gemini, Claude, and LLaMA 2. These powerful models—often called "foundation models"—churn out human-esque text based on cutting-edge algorithms and training on huge amounts of data.

But generative AI is more than just LLMs. GenAI also has multimodal capabilities, meaning the ability to create images, audio, and video.

In the next chapter, we'll dive deeper into how generative AI works. But next, let's now take a look at the pros and cons of AI-assisted programming tools.

The Benefits

AI-assisted programming tools are crafted to enhance developers' abilities, enabling them to zero in on advanced problem solving and innovations instead of being ensnared in monotonous tasks or complex code details. This is why GitHub's use of the word *copilot* is spot on. It's about having that reliable buddy in the cockpit, navigating through the intricate and often tedious aspects of coding, allowing you to focus on what matters.

In the upcoming sections, we'll spotlight the benefits and practical applications of these powerful systems.

Minimizing Search

Developers often find themselves playing digital detectives, hunting down pesky bugs or wrapping their heads around cryptic codes. When they bump into a snag, their first instinct is to hit up Google or pay a visit to Stack Overflow. A quick search, a snippet of code, and voilá, they're back to their IDE (integrated development environment).

But sometimes this can turn into an ordeal. The discussion on Stack Overflow may wind up being a dead end. You search some more—but nothing seems to be on point. However, there's one discussion that somewhat helps, and you do further research on some related topics. You even search YouTube for a video. After chewing on the problem for more than 30 minutes, you finally solve it.

Yes, all developers have experienced this. Interestingly enough, the 2022 Developer Survey from Stack Overflow, which included responses from more than 70,000 developers, highlights this frustration. It found that 62% of the respondents spent more than 30 minutes a day searching for answers, and 25% spent over an hour a day. According to the survey, "For a team of 50 developers, the amount of time spent searching for answers/solutions adds up to between 333–651 hours of *time lost per week* across the entire team."

Now, what if there was a way to slice through this thicket of time-consuming searches and get to the solution pronto? Enter AI-assisted programming, our knight in shining algorithm. Research from Microsoft supports this: it shows that more than 90% of developers who used GitHub Copilot managed to race through their tasks at a faster clip.

Microsoft even put this to the test in a coder showdown. The company recruited 95 professional developers and split them into two groups. The task was to write an HTTP server in JavaScript. Those who used GitHub Copilot completed the job 55% faster than those who did not.

And it's not just Microsoft singing praises. McKinsey & Company also conducted a research study. More than 40 developers from across the United States and Asia participated, with varying degrees of experience and backgrounds. Over several weeks, they completed three common software tasks: code generation, refactoring, and documentation.

The results? When it came to documentation for keeping the code neat and tidy, AI-assisted tools were the standouts, cutting the time spent by half, and AI tools performed nearly the same on drafting new code and refactoring.

However, for complex tasks, the AI tools didn't hit the high notes. The time trimmed was shy of 10%.

Interestingly, the research also showed that reducing the time spent did not negatively impact the overall quality of the code, as reflected in, for example, bugs, readability, and maintainability. In fact, the AI-assisted programming tools provided marginal improvements. But this often was due to the fact that developers iterated with the tools.

The McKinsey study provides the following takeaways:

Easing routine chores
>   The tools are great at tackling mundane tasks like autofilling code functions, aiding in real-time code completion, and autodocumenting code. By handling these tasks, they free up developers to dive into complex business issues and speedily deploy software features.

Producing smoother code drafts
>   Staring at a blank canvas can be daunting, but with generative AI tools, developers can nudge the creative process along by fetching code suggestions with a simple prompt, right within their IDE or separately. Many developers found these AI-based suggestions invaluable, as they helped the humans overcome the "blank screen problem" and get into the coding "zone" with a quicker pace.

Accelerating tweaks to existing code
>   With effective prompts, developers can adapt and improve existing code more swiftly. For instance, they can snag code from online libraries, pop it into a prompt, and then make iterative requests for AI-finessed adjustments based on specified criteria.

Enhancing developers' prep for new challenges

The technology acts like a fast-track introductory course and helps developers get acquainted with unfamiliar coding environments or languages. When tackling something new, these tools step in like a seasoned buddy, shedding light on fresh concepts, dissecting various code bases, and dishing out comprehensive guides on framework usage.

Harnessing multiple tools

The research indicates that bringing multiple tools into play is more effective. Picture this: a developer swings one tool for prompts or chats, and another tool jumps in as part of the codebase, dishing out autocomplete options and suggestions. Developers found the first tool to be a whiz at fielding queries during code refactoring, thanks to its conversational finesse. On the flip side, the second tool showed effectiveness in conjuring up new code that was integrated smoothly with the development environment. When these AI tools teamed up for a task, developers saw a time efficiency surge of 1.5 to 2.5 times.

Your Advisor

With ChatGPT, you can ask for advice on many types of development activities. Here's a prompt:

Prompt: Please provide detailed tips and best practices for minimizing search time and enhancing productivity when programming. Include strategies related to code organization, documentation, tools, and mindset.

Figure 1-4 shows the response.

ChatGPT provides three main areas to consider. It recommends using a modular design, maintaining consistent naming, and organizing files logically. It also advises prioritizing clear documentation with comments, docstrings, and READMEs. ChatGPT then goes on to mention using the search functions of an IDE, using tools like Git, and bookmarking key resources.

Figure 1-4. You can get useful advice on programming tasks from ChatGPT

IDE Integration

Seamless integration with the IDE is crucial for AI-assisted programming. It keeps the momentum of the development process going strong, without the heavy lifting of mastering a new platform. This means less time scrambling up the learning curve and more time coding and—let's not forget—less switching between different platforms or tools means less friction and makes for a smoother coding journey.

Then there is the advantage of real-time feedback. As developers knit together or tweak code, integrated tools are right there to spotlight errors, offer up corrections, or suggest a better way to get things done. This instantaneous back-and-forth of writing, feedback, and tweaking is like having a friendly coach by your side. You'll be guided toward cleaner, more efficient code without the hassle of manual reviews or external checks.

AI-assisted systems can also amp up an IDE by tuning into the broader coding narrative. The AI gets the gist of variable types, method signatures, and even the project's structural blueprint to churn out relevant code suggestions. It's not just about spitting out code, though.

Table 1-1 introduces some of the top AI-assisted programming tools and the IDEs they support.

Table 1-1. IDEs supported by popular AI-assisted programming tools

AI-assisted programming tool	IDEs
GitHub Copilot	Visual Studio Code, Visual Studio, Vim, Neovim, JetBrains suite, Azure Data Studio 1
Tabnine	Visual Studio Code, WebStorm, PyCharm, Eclipse, IntelliJ Platform, PhpStorm, CLion, Neovim, JupyterLab, Rider, DataGrip, AppCode, Visual Studio 2022, Android Studio, GoLand, RubyMine, Emacs, Vim, Sublime Text, Atom.AI, Jupyter Notebook 2
CodiumAI	Visual Studio Code, JetBrains (IntelliJ, WebStorm, CLion, PyCharm)
Amazon CodeWhisperer	Visual Studio Code, IntelliJ IDEA, AWS Cloud9, AWS Lambda console, JupyterLab, Amazon SageMaker Studio, JetBrains (IntelliJ, PyCharm, CLion, GoLand, WebStorm, Rider, PhpStorm, RubyMine, DataGrip)

A research study from Microsoft showed that 88% of users of GitHub Copilot felt less frustrated and more focused. A key reason was that staying within the IDE meant spending less time searching. This allowed for the developer to remain in the "flow state."

Reflecting Your Codebase

Certain AI-assisted programming tools are tailored to mesh well with specific development environments. Developers have the leeway to fine-tune them, allowing the tool to understand a project's internal libraries, APIs, best practices, and architectural blueprints. This ensures that the suggestions thrown your way not only are technically solid but also dovetail with your project's unique needs.

This customization helps to align the generated code suggestions with your organization's established coding standards, quality markers, and security protocols. The focus on fostering high-quality code means that teams can avoid stumbling into deprecated or undesirable code snippets.

Moreover, this tailored approach is a big benefit for newcomers to a development team. Traditionally, getting them acclimated to a new codebase requires a hefty time investment as they may need months of exploring code, reviewing documentation, and learning the ropes of coding protocols. However, an AI-assisted programming tool can significantly shave time off this learning curve.

Code Integrity

Code integrity is a hallmark of sound software development. It highlights the sturdiness and trustworthiness of the source code in executing its intended function. Think of it as a lens through which the completeness, accuracy, consistency, and fortification of the code are examined. A hiccup in code integrity lays out a welcome mat for bugs and potential security blind spots, which, in turn, could usher in system crashes and data breaches.

The various factors that engender code integrity include its precision, thoroughness, uniformity, and security provisions as well as the ease with which it can be maintained. Developers can ramp up code integrity through a medley of approaches like unit and integration testing, peer code reviews, static code analysis, and stringent security assessments.

It's worth noting that a growing roster of AI-assisted programming tools are rolling out features aimed at bolstering code integrity. They delve into the finer points of the code, paving the way for the generation of pertinent and sharp unit tests and edge cases.

Some of these tools come with "fix-it" recommendation features. These are vetted in advance to ensure they don't lead to new problems before they land in front of developers. Then developers can review and assimilate these suggestions right within their IDE.

An added perk of these tools is their ability to swiftly analyze pull requests and spin up succinct summaries of code alterations. They also have a knack for automating the chore of generating release notes, which comes in handy for documenting the evolution in software versions.

AI-Powered Documentation Generator

Documentation is the unsung hero in the software development process. It helps to ensure that the codebase remains legible, maintainable, and scalable, especially as teams morph and projects bloat in complexity. But let's face it, creating and refreshing this documentation often feels like a trek through a bureaucratic bog—it can be a time-guzzler and, occasionally, gets shoved to the backburner.

Now, cue the entrance of AI-assisted programming tools. These digital scribes can whip up extensive documentation in a fraction of the time—and with a hefty dose of quality and clarity to boot. This is done by leveraging the power of LLMs, which are particularly strong at dealing with language.

Modernization

Marc Andreessen's 2011 bold statement in the *Wall Street Journal*, "Software Is Eating the World", has aged like a fine wine. Andreessen, known for his knack for spotting tech trends from miles away and his stellar track record as a successful entrepreneur and venture capitalist, pointed out a ripe moment in tech history.

He underlined how the infrastructure had come of age and primed global industries for a metamorphosis. The rise of cloud platforms like Amazon Web Services and the widespread growth of broadband internet were game changers. They had knocked down the traditional hurdles of server costs and network know-how. This had cleared the stage for disruptors like Uber, Netflix, and a slew of social media platforms to rewrite the rulebook of their respective industries.

When we fast forward from Andreessen's insightful piece, we see that the innovation express has only picked up steam. However, it has also brought along a threat of disruption, especially for large corporations. Many of these behemoths are anchored to legacy systems that are not only pricey but also a gamble to modernize. Their hierarchical setup can interpose speed bumps in decision making, and their expansive scale adds layers of complexity to embracing change. Plus, their workforce might not always be on the same page with the latest tech innovations.

Enter IBM, eyeing this scenario as a goldmine of opportunity and channeling its hefty resources to craft AI-assisted programming tools for its customers. In October 2023, it unveiled the watsonx Code Assistant for Z. This system can translate COBOL to Java on mainframe systems, with the code output elegantly object oriented.

IBM's Watsonx.ai model understands 115 coding languages based on 1.5 trillion tokens. The model has about 20 billion parameters. This is one of the largest AI systems for code development.

The fact is that there are hundreds of billions of lines of COBOL. But migrating this language to modern ones is no easy feat. It's common for the COBOL to be decades old and have little or no documentation. If the conversion is not handled properly, the consequences could be severe. Keep in mind that much of the world's credit card processing is handled with mainframes. The same goes for Uncle Sam's system for handling school loans.

Unfortunately, there are many examples of failed migration projects. Consider the California Department of Motor Vehicles, which, despite pouring $208 million into the effort, had to pull the plug within a few years. Ouch.

Given the high stakes, mainframe developers generally earn higher salaries. But companies still are challenged in recruiting talent. Younger developers are trained on modern languages and perceive mainframe development as a dead end. In the meantime, a growing number of seasoned mainframe developers are retiring.

IBM realized that AI is essential to solve this massive problem. It's true that code transpilers or translators have been around for decades. In fact, they have often been used for mainframe projects. However, what they have mostly been doing is taking COBOL's spaghetti code, giving it a quick translation, and, well, you have Java spaghetti code. It's a modest facelift with barely a hint of improvement or innovation. The Java code still needs a good amount of elbow grease, explaining why many projects stumbled or flat-out face-planted.

But by using generative AI, IBM says that it has been able to improve the results of a project by as much as tenfold.

Other companies are exploring this modernization opportunity. Thomas Dohmke, who is the CEO of GitHub, posted: "COBOL still running on main frames is a much bigger societal problem than we think." In an interview with *Fortune*, he noted that he had heard more about COBOL in 2023 than during the past three decades. He also said that companies have been asking how to use GitHub Copilot for their migration projects.

Keep in mind that ChatGPT is also proficient with legacy programming languages. Table 1-2 shows which languages it supports.

Table 1-2. Common legacy programming languages

Language	Description	Development era
COBOL	Developed for business data processing	Late 1950s to early 1960s
Fortran	Designed for scientific and engineering calculations	1950s
Pascal	Developed to encourage good software engineering practices	Late 1960s to early 1970s
BASIC	Created as an easy-to-learn language for students and beginners	Mid-1960s
ALGOL	Influenced subsequent languages like Pascal, C, and Java	Late 1950s to early 1960s
Assembly language	Corresponds to the architecture of the CPU it's designed for, dating back to early programmable computers	Early computing era
PL/I	Used for scientific, engineering, business, and system programming	Early 1960s

To see how AI-assisted programming can help with legacy languages, let's suppose you need to work on the following code snippet:

```
MODULE ComplexModule
  IMPLICIT NONE
  TYPE :: ComplexType
    REAL :: real, imag
  CONTAINS
    OPERATOR(+) (a, b) RESULT(c)
      TYPE(ComplexType), INTENT(IN) :: a, b
      TYPE(ComplexType) :: c
      c%real = a%real + b%real
      c%imag = a%imag + b%imag
    END OPERATOR
```

```
        END TYPE ComplexType
      END MODULE ComplexModule
```

You do not know what language it is or how it works. The syntax does not lend itself to an intuitive understanding of the workflow.

Now let's say you go to ChatGPT and enter the following:

Prompt: What language is this written in? What does this code snippet do? Also, explain how it works.

Figure 1-5 shows part of the response.

ChatGPT

The code snippet you provided is written in Fortran, specifically a more modern version like Fortran 90/95 or later. Fortran is a high-level language often used in scientific computing.

Explanation of the Code

The code defines a module named `ComplexModule`, which contains a type definition for a complex number and an overloaded operator for addition. Let's break it down:

Module: `ComplexModule`

- Modules in Fortran are used to encapsulate data types, functions, subroutines, and other modules, allowing for organized and reusable code. `ComplexModule` is the name of this module.

Type Definition: `ComplexType`

- `TYPE :: ComplexType`: This line begins the definition of a custom data type named `ComplexType`.
- `REAL :: real, imag`: Inside `ComplexType`, two real number components are defined, `real` and `imag`, representing the real and imaginary parts of a complex number, respectively.

Operator Overloading: `OPERATOR(+)`

- `OPERATOR(+) (a, b) RESULT(c)`: This line defines an overloaded operator `+` for adding two `ComplexType` instances. The operator takes two arguments `a` and `b` (both of type `ComplexType`) and returns a result `c` (a ↓ of type `ComplexType`).

Figure 1-5. ChatGPT responds to a request to interpret legacy code

ChatGPT accurately identifies this as Fortran code. It also explains that the code defines a module named ComplexModule, which contains a derived type ComplexType for representing complex numbers, along with an overloaded addition operator + for adding two complex numbers together. Then there is a step-by-step explanation of the code.

Drawbacks

Now let's take a look at the not-so-rosy aspects of AI-assisted programming tools. Like any fledgling technology—hey, even the first iPhone was a bit clunky—AI comes with its share of hiccups, issues, and hurdles. The path of innovation is littered with room for polish and fine-tuning.

Let's take a look at some of the drawbacks.

Hallucinations

For LLMs, *hallucinations* are instances in which the model outputs data that appears accurate but is factually incorrect or not grounded in the input data on which the model was trained. This can pose a significant challenge for software development. Hallucinations can lead to inaccurate code suggestions, generate misleading documentation, and create erroneous testing scenarios. Additionally, they can render debugging inefficient, mislead beginners, and potentially erode trust in AI tools.

On a positive note, there has been notable progress in reducing the occurrence of hallucinations. A substantial amount of academic research has been dedicated to this issue, and AI companies have been employing effective strategies like reinforcement learning from human feedback (RLHF) to mitigate this problem.

However, given the intrinsic complexity of LLMs and the enormous amount of data they are based on, completely eradicating hallucinations appears to be a tall order—if not impossible.

Another aspect to consider is that certain programming languages exhibit higher accuracy rates when AI-assisted tools are used. Languages such as Python, JavaScript, TypeScript, and Go tend to have better performance in this regard. This is attributed to these languages being well represented in public repositories and thus providing a richer dataset for the AI to learn from. The better trained AI, in turn, offers more accurate and robust suggestions.

Intellectual Property

Matthew Butterick boasts a diverse background, embodying roles as a programmer, designer, and lawyer, with a particular penchant for typography. His journey has seen him authoring books on typography, designing fonts, and crafting programs aimed at document editing and layout. However, his encounter with GitHub Copilot in June 2022 didn't spark joy. Rather, it spurred him to pen a blog post titled "This Copilot Is Stupid and Wants to Kill Me".

His discontent didn't end with blogging. It quickly escalated to launching a class action lawsuit against Microsoft, GitHub, and OpenAI. The bone of contention was

an alleged breach of GitHub's terms of service and privacy policies, with a potential extension to copyright infringement charges.

This legal tangle underscores a broader gray area concerning intellectual property rights with respect to code engineered from AI-assisted programming tools. Given that the output is a cocktail of countless lines of preexisting code, the question of ownership is a big question mark.

One argument is based on the idea of "fair use." However, this legal doctrine is murky and does not extend a clear pathway for AI-generated content. To resolve this matter, there will likely need to be federal legislation or a Supreme Court ruling.

In the meantime, Microsoft has maneuvered to build a legal firewall for GitHub Copilot customers. It has pledged to defend users against legal claims, granted certain prerequisites are satisfied.

Adding another layer of the legal quagmire is the intersection of AI-assisted programming and open source software methods. Copyleft licenses, like the General Public License (GPL) versions 2 and 3, require that any derivative work use the original code's license terms. This helps to promote a stream of innovation. Yet, it could spell trouble for developers, because it could potentially strip them of the rights to shield their application's intellectual property—or even require that they make their entire codebase open source.

Privacy

The use of AI-assisted programming tools, often housed in the cloud, begs many data privacy and confidentiality questions. How is the data safeguarded within the company? Is there a chance it might be used as training data?

The clarity of the answers might vary from one vendor to another. Thus, some developers may opt to steer clear of AI-assisted programming tools altogether.

This has been the approach of Anthony Scodary, the cofounder and cohead of engineering at Gridspace. This enterprise, with roots tracing back to Stanford University, develops voice bots adept at navigating complex phone conversations. Their technological foundation rests on speech recognition, speech synthesis, LLMs, and dialog systems.

Rather than hitching a ride on existing AI-assisted programming platforms, Gridspace chose the road less traveled. It engineered its own AI-assisted programming platform, which is based on Docker services within a Kubernetes cluster. Deployed as an IDE plugin, this bespoke system is fine-tuned for its own codebase. "This has allowed us to avoid sending our IP and data to other companies," he said. "It has also meant that we have a model that is smaller, more efficient, and specialized to our style."

This is not to imply that this is the best approach. Each organization has its own views and preferred methods. But when it comes to evaluating AI-assisted programming, it's important to understand the privacy implications.

Security

In a research paper entitled "Security Weaknesses of Copilot Generated Code in GitHub", authors Yujia Fu et al. highlighted the security issues with GitHub Copilot. They scrutinized 435 AI-generated code snippets from projects on GitHub, and 35.8% had Common Weakness Enumeration (CWE) instances.

These weren't limited to just one programming language. They were multilingual missteps spanning 42 different CWE categories. Three of these categories were the usual suspects—OS Command Injection, Use of Insufficiently Random Values, and Improper Check or Handling of Exceptional Conditions. But here's the kicker: 11 of these CWEs had the dubious honor of making it to the 2022 CWE Top 25 list.

This is not to imply that AI-assisted programming tools are a huge security risk. Far from it. The fact is that vendors are continuing to work on ways to improve the guardrails. However, as with any code, a solid dose of security mindfulness is the name of the game.

Training Data

The training data for LLMs of AI-assisted programming tools may have notable gaps, which can affect the performance and usefulness of these tools in real-world scenarios. Let's break down some of these:

Representation gaps
> If certain areas of a programming language or library are not well represented—or are nowhere to be seen—in open source projects, the AI may lack enough knowledge about them, leading to less accurate suggestions. The quality of the AI's output depends heavily on the quality and scope of the training data.

Quality inconsistency
> To borrow a movie analogy, the open source code in an LLM is a bit like a box of chocolates—you never know what you're gonna get. Some projects are the crème de la crème, while others are...let's say, the burnt toast of the code world. This mishmash can lead to our AI-assisted programming being inconsistent in the quality of suggestions it throws your way.

Knowledge cutoff date
> LLMs have a cutoff date on their training, so in a way they are like a snapshot in time. This poses challenges when there are new releases, updates, or deprecations in programming languages or libraries.

Generalization gap

The generalization gap, the difference between the AI's performance on the training data and unseen data, can also pose challenges. Of course, the closer the performance of the two, the better. This is the conclusion of a research paper by Rie Johnson and Tong Zhang entitled "Inconsistency, Instability, and Generalization Gap of Deep Neural Network Training".

Contextual understanding

AI can give you suggestions based on what it has seen before. But if it hasn't seen a scenario quite like yours, it might miss the mark. This is why it's important not to make assumptions when creating prompts.

Bias

Developers often don't have a solid grasp of AI ethics, likely because this topic isn't usually part of computer science courses or intensive bootcamp programs. This gap in understanding can lead to algorithms unintentionally applying biases and the potential misuse of data.

This issue carries over to AI-assisted programming tools as well. They can unintentionally perpetuate the biases present in the data they were trained on. For example, if asked to create a list of names, they might mainly suggest English names due to the heavy presence of English-centric datasets in their training datasets. This bias can sometimes lead to harmful or inappropriate outputs. There was an *instance* where, when given the prompt "def race(x):", the AI filled in a limited and fixed set of race categories. In another troubling case, when tasked with writing code comments for the prompt "Islam," the AI was found to access words like *terrorist* and *violent* more frequently than when other religious groups were mentioned.

A New Way for Developers

The McKinsey study suggests that the dawn of AI-assisted programming tools is likely to change how we approach software development. According to the authors, success might hinge on good training, emphasizing best practices and diving into hands-on exercises on things like prompt engineering, coding standards, and quality. It's also smart to shine a light on the risks associated with generative AI.

For newbie developers, especially those with less than a year of experience under their belts, it's a good idea to dive into extra coursework that covers the basic principles of programming to ramp up productivity.

As developers fold these tools into their daily routine, it's vital to keep the skill-building momentum going with some guidance from the seasoned pros on the team and engagement in community activities. This could mean hanging out in dedicated online forums or having regular team huddles to share practical examples. Such

actions can foster a culture of continuous learning, spread the word on best practices across the board, and help spot issues early on.

With the uptick in developer productivity, managers might want to stir the pot a bit when it comes to roles, zeroing in on tasks that pack more value. Upskilling will be on the menu, too, to fill in any existing gaps.

Sure, these pointers aren't gospel. The realm of AI-assisted programming is still pretty fresh and is changing at a brisk pace. Above all, being ready to roll with the punches is key.

Career

While there's no hard proof that using AI-assisted programming will boost your career outlook, a handful of signs suggest that this expertise might become a hot ticket in the job market:

Job listings
> The job boards on sites like Indeed are starting to buzz with more listings seeking candidates with experience in AI-assisted programming tools. The call is out for all ranks, from junior developers to the senior hotshots.

Productivity boosts
> AI-assisted programming tools are turning heads because they're improving productivity without sacrificing quality. For a developer, this could be a way to move up the ranks in an organization.

Thumbs-ups from developers
> The chatter among developers is that AI-assisted programming tools are catching on. For example, GitHub Copilot is boasting a strong rating of 4.5 out of 5 stars on G2.com, an independent software review site.

10x Developer?

The *10x developer* has the power of 10 programmers. They're the Usain Bolt of coding, zipping through problems and churning out solutions before you can say "bug fix."

So you might be thinking: Could I become a 10x developer with the help of AI-assisted programming tools? Well, sorry to say, but probably not. While these technologies can make a significant difference, improvements are usually not in orders of magnitude.

Besides, the concept of a 10x developer can stir up stereotypes and biases, making the tech scene feel like an exclusive club. Not to mention, the pressure to be this super

coder could lead you straight into the arms of burnout. So while being a 10x developer might sound great, remember it's probably closer to a fantasy.

Skills of the Developer

According to the McKinsey study, the effectiveness of AI-assisted development tools often depends on the expertise of the developer. Here are some of the considerations:

Fixing errors
> Even though generative AI can be your trusty sidekick, it can goof up too. It falls upon the developer's shoulders to spot and fix these blunders. Some developers have found themselves playing a loop of corrections with the AI to get to a sweet spot of accuracy, while others have had to spoon-feed the tool to get it to debug accurately. This can certainly be time-consuming. But a veteran developer will know how to avoid going down the rabbit holes.

Getting the office vibes
> AI-assisted programming tools are fairly solid when it comes to coding but might miss the beat when dealing with the unique flavor of individual projects or company quirks. Again, this is where veteran developers are key. They'll know how to guide these tools to get the results that best align with organizational goals, performance targets, and security.

Tackling the tough stuff
> Assisted AI-programming tools are great with tasks like polishing code, but toss in some complex challenges like blending different coding frameworks, and the AI might just trip over itself. In these moments, it's the experienced developers who have to roll up their sleeves.

Conclusion

AI-assisted programming tools are certainly the shiny toys in the software creation sandbox. As this technology keeps marching forward, these systems will crank up efficiency, handle boring tasks, and let developers dive into the areas that are most important, like high-level problem solving.

But there are downsides—tangled intellectual property issues, maze of open source software licensing, potential for bias, and security risks to name a few.
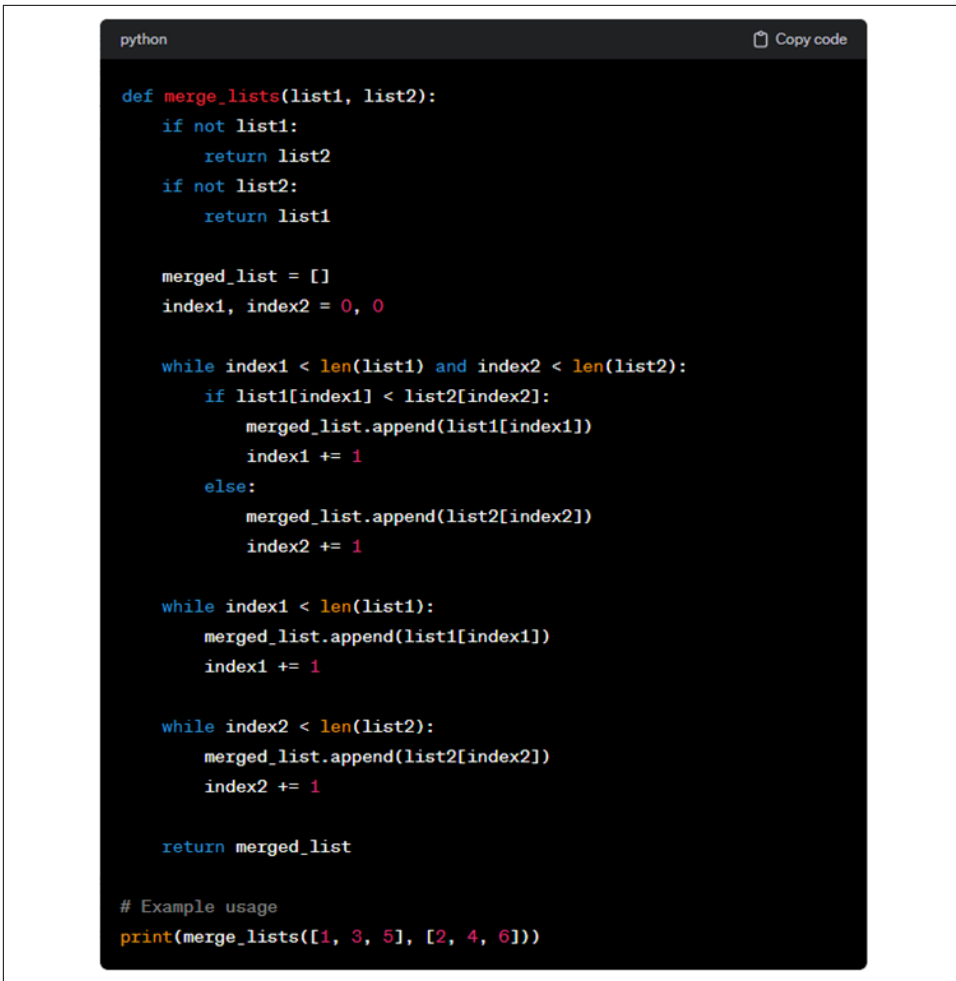
For the most part, these tools are your virtual assistants, not a replacement for your knowledge, skill, and experience. At the same time, while they might not be superheroes, they're shaping up to be powerful additions to the developer's toolkit.

Coding

In this chapter, we'll dive into some handy coding techniques. We'll start with the bread and butter of coding—playing around with functions and classes. Then, we'll get our hands dirty with a bit of refactoring, tidying up messy code and making it run smoother. We'll also show how to work with and create data. Then we'll check out some AI tools that can help create fancy frontends without breaking a sweat.

Reality Check

AI-assisted programming tools are pretty cool, right? But let's not kid ourselves. They're not the magic wand that can solve all our coding problems. Why? Well, these tools are like those students in class who only learn from public notes. They're trained on a mishmash of code from all over the internet, and let's be honest, not all of it is top-notch. What does this mean for you? Sometimes, you might get code that's more bloated than a Thanksgiving turkey or as maintainable as a house of cards. And sometimes the code is just plain wrong, doesn't do what you need, or, even worse, leaves your front door wide open for hackers or the hogging of your network resources.

Let's look at an example where ChatGPT produced messy code. Figure 8-1 shows code for merging two sorted lists into a single sorted list.

```python
def merge_lists(list1, list2):
    if not list1:
        return list2
    if not list2:
        return list1

    merged_list = []
    index1, index2 = 0, 0

    while index1 < len(list1) and index2 < len(list2):
        if list1[index1] < list2[index2]:
            merged_list.append(list1[index1])
            index1 += 1
        else:
            merged_list.append(list2[index2])
            index2 += 1

    while index1 < len(list1):
        merged_list.append(list1[index1])
        index1 += 1

    while index2 < len(list2):
        merged_list.append(list2[index2])
        index2 += 1

    return merged_list

# Example usage
print(merge_lists([1, 3, 5], [2, 4, 6]))
```
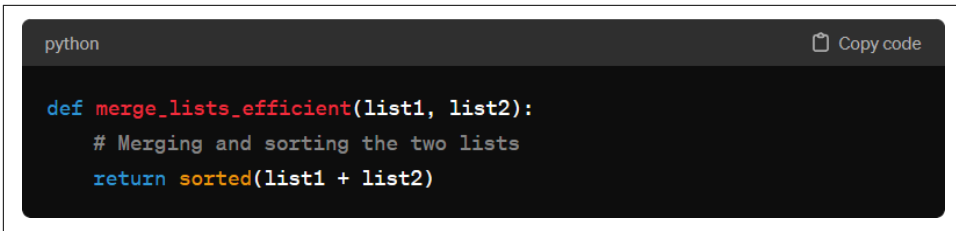
Figure 8-1. When asked to merge two sorted lists, ChatGPT created some messy code

This AI-generated code does correctly merge two lists, but it takes a verbose approach. It separately handles the cases of empty lists and uses two additional `while` loops to append the remaining elements from each list.

This is where your coding smarts really come into play. You've got to know the fundamentals. You need to know what questions to ask and where to poke around when something isn't quite right.

Figure 8-2 shows a better approach.

```python
def merge_lists_efficient(list1, list2):
    # Merging and sorting the two lists
    return sorted(list1 + list2)
```

Figure 8-2. With some human help, we've achieved a better approach to the merging of the sorted lists

So don't worry about all the doom and gloom you may hear about robots taking over developer jobs. Granted, these AI models are getting better all the time. They're learning and growing. Yet they are still far from being the be-all and end-all of programming. Programmers are still very much in the game. AI-assisted programming tools are extremely helpful, but they're not about to replace the savvy and know-how of real, live engineers.

When you're jamming with these systems, remember: it's a collaboration, not a handover. Stay sharp, stay curious, and don't forget to double-check everything these AI tools serve up. They're helpers, not heroes.

Judgment Calls

Sometimes, it's simpler to just do the coding yourself instead of using an AI-assisted programming tool. After all, it can take a bunch of tries to get the AI to catch on to what you need. But as you keep using these AI tools, you'll get the hang of figuring out when they're useful or when you're better off flying solo.

Take the experience of Dmitrii Ivashchenko, a lead software engineer at MY.GAMES, an Amsterdam-based game company with over 1 billion registered users worldwide. According to him:

> An example would be the prompt "Write a method in Python to add a default timezone to the datetime object." However, you should be prepared that many aspects will be omitted and you will either have to finalize the handling of corner cases yourself or have a long correspondence with ChatGPT pointing out its errors. The main thing here is to initially gauge the time it takes to explain a task versus what it will take for you to implement it yourself.

Learning

Using something like ChatGPT to pick up a new language, framework, or library can be a big help. It may be faster than the old-school ways like taking a course, endlessly Googling, or watching a bunch of YouTube videos.

You can use ChatGPT to get the lowdown on all the basics and syntax of the code. Plus, if you're more of a learn-by-doing type, you can ask for code examples to see how everything clicks together. If you already know one programming language, ChatGPT can make learning a new one easier by comparing the two. Suppose you're good with Python but just diving into JavaScript; it can explain the main differences in Python terms. If you need more resources, ChatGPT can scour the internet and recommend some solid learning materials for you.

Here are some prompts to consider:

> *Prompt:* For someone who does not know Python, what should I learn first?
>
> *Prompt:* What are some beginner-friendly projects I can try to improve my skills in React?
>
> *Prompt:* Can you suggest resources for learning advanced SQL queries?
>
> *Prompt:* How would you implement a basic "to-do list" app in both Python and JavaScript to highlight their differences?
>
> *Prompt:* I know how to use loops in C++. Can you show me how loops work in Python for comparison?
>
> *Prompt:* Find 5 useful YouTube videos that show how to learn Rust.

Let's now see how Ivashchenko approaches this:

> If you want to quickly master a new language or framework, you can immediately start making a new project on that technology. Let's say you want to learn Django. Let's choose a project that we will implement on this technology—let it be an online store. To begin with, just ask ChatGPT how to start developing a new project on Django. It will tell you what dependencies you need to install, how to set up the initial configuration of the project, and what related technologies you will need to deal with. Then we move on to more specific requests such as "How is a model organized in Django?" "How to add a new section in the admin panel?" and "How to migrate the database?" By gradually gaining new knowledge and asking more and more detailed questions, you can realize almost any project on any previously unfamiliar technology. This is how I mastered Django literally through a weekend of fruitful work.

You can have ChatGPT create useful study guides as well. Here's a sample prompt:

> *Prompt:* Create a study guide for JavaScript. Focus on a beginner level.

You can then follow this up with:

> *Prompt:* Please create a study schedule for me. What topics should I study? What about practice exercises or quizzes? Coding problems? Links to resources?

ChatGPT can even help you to remember new concepts. One way is to use a mnemonic device, such as acronyms. For example, suppose you want to remember the key elements of DevOps, which include continuous integration, continuous delivery, microservices, and infrastructure as code. This is what ChatGPT suggests:

> *ChatGPT:* CI-CD-MI (pronounced "Ci-Ci-D-Mi")

Finally, you can use ChatGPT to get help with LeetCode. Picture it: you're sitting in the interview room, maybe with a whiteboard or a laptop in front of you, and then they hit you with this brain-teasing algorithm or data structure question. It's more than just coding. They're sizing up how quickly you think, how well you explain your thoughts, and your problem-solving style.

If you want to get a leg up, think about signing up for LeetCode. It's $35 a month. But of course, ChatGPT can lend a hand whenever you need it. Here's a simple prompt:

> *Prompt:* Suggest 3 common LeetCode interview questions.

ChatGPT then came up with challenges like two sum, longest substring without repeating characters, and valid parentheses. You can then dive deeper, such as by getting longer explanations or examples.

Comments

When you're racing against a deadline, it's easy to skimp on code comments or ditch them altogether. But AI-assisted tools can have your back. You'll often find code suggestions with comments already included.

Funny enough, because of AI technologies, having comments may feel somewhat retro. If code's got you scratching your head, you could just ask the tool to break it down for you, right? This is certainly true.

Something else: writing prompts is becoming the new way to comment. After all, most tools include the prompts in the comment lines.

Of course, if you want to add comments to some code, you can use a simple prompt like:

> *Prompt:* Add comments that are clear and according to best coding practices.

Whether or not to comment is your call. There's no one-size-fits-all rule here. It's all about what works for you and your team. But one thing's for sure—slapping comments onto your code is a breeze with AI-assisted tools.

Modular Programming

Modular programming is a cornerstone of efficient and effective software development. With modular programming, coding is more organized, easier to understand, and easier to keep up with. It also makes teaming up with other coders a lot less of a headache because everyone's not tripping over each other. Plus, modules are reusable; reusability is a massive time-saver, keeps things consistent, and makes it less likely you'll goof up.

The value of modular programming certainly applies to how you work with AI-assisted tools. They will not whip up an advanced application from a simple prompt. They're not wizards. But if you break your task down into clear, specific pieces, these tools shine. Otherwise, you might end up with a code jumble that's way off track.

According to Titus Capilnean, the cofounder and chief product officer at Private Market Labs:

> After I started using AI tools, I could focus on the problem and my approach to get a solution, not the minutiae of the actual code I need to write. When I have a technical problem, I start by breaking it down into smaller pieces, where the inputs and outputs are clear. The reason for this is that the context window of the AI tool I'm using might not be sufficient to come up with a good solution in one shot. I found that it's easier to debug and build if I ask the model to deliver code that uses a simple input, does one single job and provides an output I can verify. If I'm not happy with the intermediate output, I describe my issue to ChatGPT and attempt to run the updated code it delivers. Once I'm happy with the solution, I can add extra processing steps, either inside the existing function or that takes the output of the previous step and refines it further, closer to my desired solution.
>
> AI tools also improved my productivity by allowing me to check whether a snippet of code I built is actually correct before even sending it to a compiler or deploying a cloud function and consuming resources on an error. I just paste the part of the code in the ChatGPT window and ask whether it is correct, assuming I initialized/provided the correct context for the snippet to function (e.g., imports, variables, constants, custom functions being called, etc.). This allows me to produce solutions that run well and faster.

Starting a Project

At the start of a coding project, you may hit the "cold start problem" or "blank page problem."

Here's a scenario: you're staring at an empty screen, no code, no data, not even a clear path forward. No doubt, it's as overwhelming as it sounds. The first big headache is selecting your project's architecture, design patterns, and what technology to use. These decisions are huge because you'll be living with them for the long haul, so you want to nail them from the get-go as much as possible.

And let's not forget the people factor. Getting your team on the same page, figuring out how to communicate effectively, and setting up a workflow from scratch—these can be just as challenging as the technology. It's not just about slapping down those first lines of code. It's about laying a solid foundation for everything that comes next. Tackling this phase needs a mix of smart planning, technical know-how, and solid teamwork.

AI-assisted programming tools can be a big help. You can use them to set up the basic scaffolding of your app. You'll figure out a customized starting point that aligns with your vision. The tools save you from the nitty-gritty of initial setup and let you jump straight into the more interesting parts of your project.

You can come up with a particular task and then prompt the LLM to generate boiler-plate or starter code.

This is a sample prompt:

> *Prompt:* Generate starter code for a web app that aggregates a user's social media feeds from multiple platforms into a single dashboard interface. What do you suggest I use for the language and frameworks? What about the general file structure?

With this, ChatGPT starts off by suggesting React for creating a smooth, single-page application (SPA). It also suggests using Redux to maintain strongly styled and reusable components. It then says to use Node.js to run the application and Express.js for the API endpoints. It goes on to recommend Passport.js for the logins and MongoDB and Mongoose for the database and model building. As for the file structure, Figure 8-3 shows the layout.

General File Structure:

```bash
social-dashboard-app/
|
├── client/                    # Frontend part of the application
|   ├── public/                # Static files
|   |   └── index.html         # Entry point HTML file
|   ├── src/                   # React.js source files
|   |   ├── components/        # Reusable UI components
|   |   ├── pages/             # Page components
|   |   ├── app.js             # Main React application component
|   |   ├── index.js           # Main JavaScript entry point for React
|   |   └── redux/             # Redux store and reducers (if using Re
|   └── package.json           # Frontend dependencies and scripts
|
├── server/                    # Backend part of the application
|   ├── config/                # Configuration files for APIs and DB
|   ├── routes/                # Express routes
|   ├── models/                # Database models
|   ├── controllers/           # Request managers
|   ├── middleware/            # Express middleware
|   ├── app.js                 # Main server file
|   └── package.json           # Backend dependencies and scripts
|
├── .env                       # Environment variables for API keys, D
├── .gitignore                 # Specifies intentionally untracked fil
└── README.md                  # Project documentation
```

Figure 8-3. ChatGPT suggests a fle structure for a new web app

Autofill

You know when you're deep in the coding zone, setting up those constants for time units, and it's just line after repetitive line? Here's where GitHub Copilot jumps in. You can have it provide autofill.

Let's take an example. Suppose you are creating an app that you want to make responsive. This means you will need to have constants for the breakpoints for a CSS-in-JS styled-components library.

You can first write this:

```
breakpoints = {
    'xs': '320px',  # Extra small devices
```

Then in the inline chat, you can use this prompt:

Prompt: Create constants for other screen sizes.

Figure 8-4 shows the result. It provides other screen sizes and variables that have a similar style.
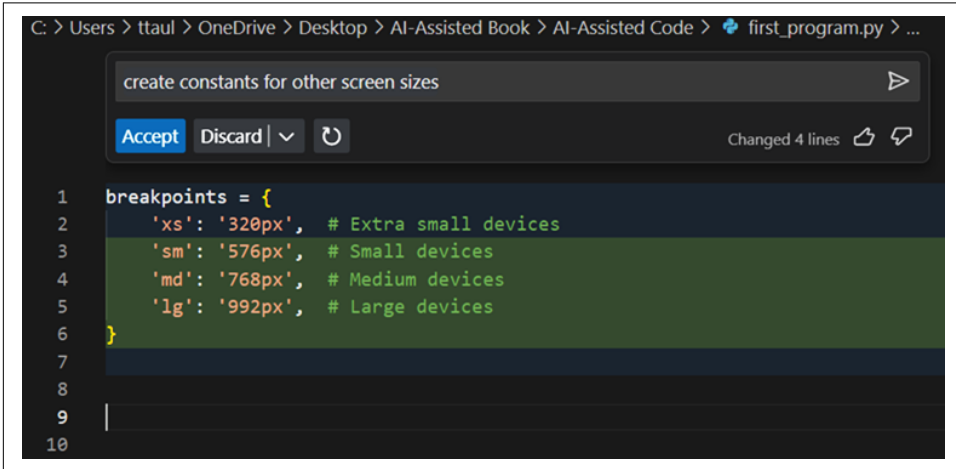
Figure 8-4. This is output from Copilot that suggests different constants for screen sizes

You can go further with this. Copilot can help with autofill by looking at open files in your project.

According to Capilnean:

I found that Copilot is especially useful if you have parts of the code that are similar when it comes to data structures. If you have defined an object in another file that's part of the code repository you're working with, then when you are trying to define a similar object, it can reliably pre-fill the code for you as you type. For example, an object for a common API call, like a Sendgrid email, works well here.

However, this autocomplete can hallucinate if you're not careful. On imports, for example, it won't always get the folder structure right, especially if you designed your own code shape, or if the framework you're using is fairly new and not well documented yet. In that case, I have to go in and manually check where the component is coming from and whether the directory that Copilot suggested was correct.

Refactoring

Refactoring is like spring cleaning for your code. It's all about tidying up, reorganizing, and sometimes sprucing things up. It's not about adding new features or fixing bugs.

Refactoring keeps the codebase healthy, less cluttered, and more intuitive. This means when you or someone else jumps back in later, it's easier to understand what's going on, which cuts down on headaches and, well, swearing at the screen. Plus, cleaner code is usually more efficient and less buggy, so it's a win-win for everyone involved.

Refactoring is one of those areas where AI-assisted tools shine. With a prompt or two, you can slice through the complexity of your code, trim the fat, and reorganize your code into something that's not just functional but elegant. With AI in your corner, refactoring becomes less of a chore and more of a secret weapon for staying ahead of the curve.

In the next few sections of this chapter, we'll take a look at some examples of refactoring.

Ninja Code

Think of ninja code as one of those over-the-top action movie stunts. It's flashy and complex, and it leaves you thinking, "Whoa, how'd they do that?" But here's the catch: it's a beast to figure out once the awe wears off. Imagine code so sneaky and tangled that even the person who wrote it can't make heads or tails of it after their "ninja" phase passes. Sure, pulling off that ninja move feels epic in the moment, but when you or someone else needs to jump back in and make changes, it's less "hi-ya!" and more "uh-oh." The truth is, while ninja code might show off some serious programming chops, it often goes against the grain of good coding practices, which are all about keeping things clean, simple, and understandable.

Regardless, ninja code is common. But refactoring with an AI-assisted tool can help sort things out. Here's an example:
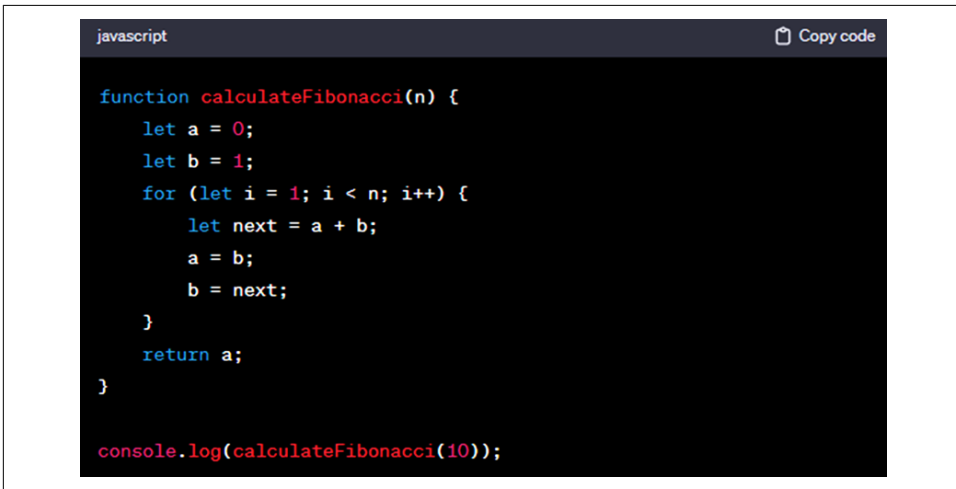
```
console.log((function(n, a = 0, b = 1) { while (--n) [a, b] = [b, a + b];
return a; })(10));
```

Make any sense? Well, you'd probably need to take quite a bit of time to figure this out. Instead, make ChatGPT do the work:

> *Prompt:* Can you explain this code in a step-by-step process? Also, can you write this in a simpler way that is more maintainable?

ChatGPT determines that this is a function to calculate the nth Fibonacci number. It goes on to describe it in six steps and then provides a simplified version, as seen in Figure 8-5:

ChatGPT: In this version, the calculateFibonacci function clearly shows the process of iterating through the Fibonacci sequence. It's more verbose but much easier to understand and maintain, especially for other developers who might read this code in the future.

```javascript
function calculateFibonacci(n) {
    let a = 0;
    let b = 1;
    for (let i = 1; i < n; i++) {
        let next = a + b;
        a = b;
        b = next;
    }
    return a;
}

console.log(calculateFibonacci(10));
```

Figure 8-5. ChatGPT has provided a simpler version of what was ninja code

Extract Method

You apply the extract method when you have a long method or function. You'll pick out a chunk of the code that's focused on a certain task—say data checking or a specific calculation—and you'll turn this into a new method.

This approach isn't just about making things look neat and tidy. It's practical, too. It helps you keep your code easy to read and understand. When you need to do that same task again, you can call up your new method instead of rewriting the code. And if there's ever a glitch, it's easier to sort out when your code is nicely split up into these focused, bite-sized pieces. It's about making your life easier and keeping your code clean and organized.

Here are some prompts for the extract method:

Prompt: Are there any common pitfalls to avoid when extracting methods in functional programming languages?

Prompt: I've attached a piece of my C# code. Could you suggest which sections would be good candidates for extract method refactoring?

Prompt: Can you compare my original function and the refactored version with extracted methods? Which is more efficient?

Decomposing Conditionals

Decomposing conditionals is about breaking down big, gnarly if-then-else statements into something a lot more digestible. You know the kind—those lengthy, twisty conditions that make you squint at your screen trying to figure out what's going on.

Let's say you have an `if` statement with a complex condition that checks multiple variables and perhaps calls other functions. Instead of trying to decipher this every time, you can extract this condition into a method with a name that clearly describes what the condition is checking. For instance, a condition like `if (user.isActive() && user.hasSubscription() && !user.isBlocked())` can be refactored into a method named `canUserAccessContent()`. This not only makes your main method cleaner but also instantly makes the code self-explanatory.

Similarly, the code within the `then` and `else` blocks can be extracted into distinct methods. This makes the main flow of your program much more readable. Instead of wading through lines of detailed logic, a reader can now understand the flow at a high level: if this condition is true, do this; otherwise, do that. Each part of the logic lives in its own neatly named method, making it easier to test and modify in isolation.

Here are some prompts:

> *Prompt:* Can you explain how to decompose conditionals in Java code for better readability?

> *Prompt:* How can I break down complex if-then-else statements using the decomposing conditional technique?

Renaming

Renaming functions, variables, and classes might seem like a small change, but it can have a significant impact on the readability and maintainability of code. This is particularly useful in situations where code has evolved over time and original names no longer accurately describe what the code does. For instance, a method initially named `processData` might become more specialized. Renaming it to something more descriptive, like `filterInvalidEntries`, can instantly clarify its functionality.

Writing a prompt for renaming is fairly simple:

> *Prompt:* What would be a good name for a variable that holds the total number of users in a database in my SQL script?

> *Prompt:* Here's a snippet of my JavaScript code. Can you review the variable and method names and suggest improvements?

> *Prompt:* I'm not sure if the names in my Java class are clear enough. Can you propose better names for clarity?

But you need to be cautious, such as when you're using a tool like Copilot. If you change a name, this can break parts of the code that are still using the old name.

Dead Code

As the name implies, dead code is not being used for anything. These are the forgotten lines of code from features that got scrapped or updates that made some parts redundant.

Scrubbing this dead code from your project makes everything neater and more manageable. It also makes it less confusing for anyone new jumping into your project. They don't have to scratch their heads over why something's there if it doesn't seem to be doing anything.

Here are some helpful prompts:

> *Prompt:* Can you help me identify potential dead code in this JavaScript snippet?
>
> *Prompt:* Here's a piece of my Python project. Could you point out any code that seems unused or redundant?
>
> *Prompt:* Could you take a look at these SQL procedures and confirm if any of them are safe to delete?

Note that using an LLM for this can be risky. Sometimes what looks like an old, dusty corner of code might actually be important for those rare, just-in-case scenarios. Then there's the domino effect: removing one piece might mess up something else that was relying on it, especially if it's part of complex logic or setup. So just be cautious.

The other problem is that the generative AI may not truly understand the relationships. So until AI systems get more sophisticated, it's probably best to avoid using them for rooting out dead code.

When it comes to dealing with dead code, alternative tools may be a better option. An example is a linter. Think of this as your code's tidy-up crew. If you're working with JavaScript, there's ESLint. Or for Python fans, there's Pylint, and the Ruby folks can turn to RuboCop. These tools are like the grammar checkers of coding. They're awesome at picking up on those pesky syntax mistakes, potential bugs, and, of course, even those sneaky bits of code that aren't doing anything.

You've also got the heavy-duty inspectors: the static code analysis tools. Top providers are SonarQube, Code Climate, and Coverity. These tools are like detectives. They dig through your code without even running it, sniffing out complicated patterns that might be troublemakers down the line, including dead code.

Functions

Functions are the bread and butter of coding, playing a huge role in any kind of software program, no matter what programming language is used. They're chunks of code that do wonders for keeping your programs tidy and easy to read since they let you easily reuse code—a lifesaver for any developer. They can also take a big task and break it down into smaller, more manageable bits. This makes dealing with complex software a whole lot easier, especially when you need to fix bugs, make updates, or just try to wrap your head around what the code is doing.

It's key to nail functions right from the get-go. Making them work is one thing, but you also need to make sure they play nice with the rest of your code. You want your functions to be clear, easy to keep up with, and efficient. You'll need to think about what to call each function so that it makes sense, how you're going to set it up, how it will deal with the data coming in and going out, and how it handles any hiccups.

To help out, here are some guidelines to keep in mind:

Think of the single responsibility principle
> Your function needs to be a specialist in one job—that's it. This makes it a lot simpler to figure out what it's doing, check if it's working right, and fix it if it's not.

Name it clearly
> Give your function a name that tells you exactly what it does. If it calculates the total price, call it `calculateTotalPrice`. This makes your code much more readable.

Keep it short and sweet
> A good rule of thumb is that you should be able to see the whole function on your screen without scrolling. Short functions are easier to handle and less likely to have bugs.

Parameters are key
> Use parameters for inputs and return values for outputs. This makes your functions predictable and self-contained.

Stay consistent
> Follow the coding conventions and style guidelines of your language or project. This helps keep your code uniform and easy for others to read.

When you keep these tips in mind, you can whip up some really effective prompts for functions, and ChatGPT can help. Check out these example prompts:

> *Prompt:* Write a Python function named calculate_area that takes two integers as parameters, length and width, and returns the area of a rectangle. Include a docstring

explaining the function's purpose and ensure the function handles non-integer inputs by raising a TypeError.

Prompt: I need a JavaScript function called filterAndTransform. It should take an array of objects as input. Each object has properties name (string) and age (number). The function should return a new array containing the names of people who are 18 years or older, converted to uppercase. Include comments explaining the logic.

Prompt: Create a C++ function named efficientSort that sorts an array of integers in ascending order. The function should be optimized for time complexity. Provide comments within the function explaining the choice of sorting algorithm and its time complexity.

Prompt: Can you generate a Java function called safeDivide that takes two double parameters, numerator and denominator, and returns their division? The function should handle division by zero by returning a custom error message. Include Javadoc comments explaining the function and its error handling.

Object-Oriented Programing

Object-oriented programming, or OOP for short, is a way of writing computer programs using the idea of "objects" to represent data and methods. Think of it like creating a bunch of small, self-contained boxes, each with its own set of tools and information. These boxes, called *classes*, are like blueprints for creating different objects. A class defines the structure and behaviors of objects—similar to a template. Then from this class, you can create individual *objects*, each with its own specific details but following the same basic structure.

Diving into the world of OOP can feel as if you're stepping into a maze of complex concepts like abstraction, inheritance, encapsulation, and polymorphism. They can feel like they're written in an alien language.

This is where ChatGPT can be your translator. It breaks down these complex ideas into bite-sized, easy-to-digest explanations. Struggling with what *encapsulation* really means? Just ask, and you'll get an answer that actually makes sense, minus the tech jargon.

Here are some prompts:

Prompt: Can you create a simple class in *<your preferred programming language>* that demonstrates encapsulation?

Prompt: What are some real-world examples of encapsulation in programming?

Prompt: Explain abstraction in OOP with an analogy from everyday life.

Prompt: Can you show me an example of inheritance in a programming scenario?

Prompt: How does inheritance promote code reuse in OOP?

Prompt: How does polymorphism enhance flexibility in a program?

Prompt: In what scenarios is polymorphism particularly useful, and can you provide examples?

AI-assisted programming tools can also be helpful in coming up with the initial structures for classes. Here are some example prompts:

Prompt: Design an Employee class with properties like employeeName, employeeID, and department. Implement a method that displays the employee's details. Also, include a constructor to set these properties.

Prompt: I need a BankAccount class in Java. It should have private properties like balance and accountNumber. Can you add methods for deposit(), withdraw(), and check-Balance() that modify or access these properties safely?

Prompt: Could you show me how to create a Vehicle class in C# and then a Truck class that inherits from it? Make sure to include properties like wheels and fuelType and demonstrate the use of different access modifiers.

Prompt: In C++, how would I write a FileHandler class that opens a file in its constructor and closes it in its destructor? Also, include methods for writeToFile() and read-FromFile().

Frameworks and Libraries

Diving into software development without frameworks and libraries is like trying to bake a fancy cake from scratch without a recipe or premixed ingredients. It's possible, but it's a whole lot harder and takes more time. Frameworks and libraries are the secret sauce that makes a developer's life easier. Instead of reinventing the wheel every time you need to make a web request or manipulate a DOM element, you just tap into what's already there.

AI-assisted programming tools can certainly help. First of all, they can be useful for learning the basic features and workflows. They can also tell you when it's best to use a framework or library.

But take their help with a grain of salt. Here's what Capilnean has to say:

Given the large number of updates React gets—as well as other frameworks and libraries—and the fact that we use specific versions of frameworks, I had to provide these as system prompts to my ChatGPT instance in order to optimize my results. Sometimes, GPT provided solutions that were more academic than production-grade, so I tend to rely on working with our senior developers to get the job done for more complex issues. For React, I find myself asking GPT to check my syntax and my method of processing a specific data type more than actually build a full feature for me.

The same goes for NodeJS. I have to take into consideration our internal APIs and methods of working with our data before I can go to ChatGPT and ask for a code snippet for a feature. Once I am able to describe the output of one of our functions as the input for the feature I am building, it usually is able to provide me with stable code. I use the same process here if the output isn't good or it errors out. I provide it with the issue and ask for a code update in the right direction.

Data

Data is the lifeblood of every application, truly. It's what keeps an app alive and kicking. Just like blood carries oxygen and nutrients to keep our bodies functioning, data flows through an application, giving it the information and insights it needs to work its magic.

But creating sample data can be a slog. Picture this: you're excited to test your shiny new app, but first, you need a bunch of data to see how it actually works in the real world. You start typing in rows and rows of data—names, dates, numbers, whatever it takes. But then, it just keeps going. And going.

What's more, you have to be careful about making it realistic enough so your tests are valid, but not so detailed that you're writing a novel for each entry. And if you need a large dataset? Forget it: you're basically signing up for a marathon of copy-pasting, tweaking, and double-checking.

An AI-assisted programming tool can be handy for this. But first, you need to select a database and spin up the schema and tables. The tool will also need to be given the relationships, say, among the tables. Then you will need to do the configuration and setup. But then the AI can begin to help.

Here are some prompts to get help evaluating databases:

Prompt: What type of database would be best suited for handling *<specific data types or functions, e.g., user interactions, product inventory, etc.>*?

Prompt: For an app expecting *<high/low>* traffic with *<type of data, e.g., images, text, real-time data>*, which database should I consider?

Prompt: I'm on a tight budget. Can you suggest a cost-effective database solution for a small *<type of app, e.g., local delivery service app>*?

Prompt: I'm relatively new to database management. Which databases are user-friendly and easy to maintain for a beginner?

Here are some prompts to ask for help with a database schema:

Prompt: Can you help me design a basic database schema for a *<type of application, e.g., online store, blog>*? I need to know what tables I should create and the primary relationships between them.

Prompt: What would be an efficient table structure for managing *<specific type of data, e.g., customer orders, inventory>* in a relational database? What fields and data types should I include?

Prompt: How should I define the relationships between tables in a relational database for an application that deals with *<describe the application's functionality, e.g., event management, course enrollment>*? Specifically, I need help with understanding foreign keys and join tables.

Prompt: I'm working on a relational database schema for *<describe the project>*. Could you guide me on setting up primary and foreign keys effectively for data integrity?

Prompt: What normalization strategy would you recommend for a database handling *<type of data or application function>*? How can I avoid data redundancy and ensure data integrity?

Prompt: I'm using a NoSQL database for a *<type of project, e.g., social media app>*. How should I design the document structures to store *<specific data types, e.g., user profiles, posts, comments>*?

Prompt: In designing my database schema, what indexing strategies should I consider for optimizing query performance, especially for *<type of queries or operations, e.g., full-text search, frequent updates>*?

Prompt: I need to migrate an existing database to a new schema. What are the key considerations and steps for redesigning the database structure without losing data integrity?

And here are some prompts for setting up the initial database environment:

Prompt: Can you provide step-by-step instructions for installing *<a specific database server, e.g., MySQL, PostgreSQL>* on *<a specific operating system, e.g., Windows, Linux>*?

Prompt: After installing *<name of database>*, what are the essential configuration settings I should initially set up for optimal performance?

Prompt: What are the best practices for securing a *<specific database>* server? I'm particularly interested in user authentication and protecting sensitive data.

Prompt: How can I optimize the performance of *<a specific database>* for an application that will handle *<describe the nature of the data and expected load, e.g., large datasets, high transaction volumes>*?

Now, let's look at how we can use AI to create sample data. Here are some example prompts:

Prompt: Create demo data for 100 IDs and email data and save this to a CSV file.

Prompt: Create demo data for 50 products, including product ID, name, price, and category.

Prompt: Create a demo dataset of 150 order records, each with an order ID, customer ID, order date, and total amount.

Prompt: Generate sample data for 100 employees, including employee ID, full name, department, and email address.

Prompt: Create sample data for 80 customer feedback entries, including feedback ID, customer ID, comments.

With the data, you can then create SQL statements for it. You can use something like this for the customer feedback entries:

Prompt: Generate a SQL insert statement to populate the Feedback table with the data.

ChatGPT can be a lifesaver when it comes to the nitty-gritty task of data conversion. If you're a developer, you know that converting data between different formats like XML, SQL, JSON, CSV, and TOML is common. But let's be honest, it can be a tedious and sometimes error-prone process. That's where ChatGPT comes in handy.

Here are some sample prompts:

> *Prompt:* Here's a CSV row: 'John Doe, 35, New York'. Can you convert this into an XML format for me?

> *Prompt:* I have a JSON array like this: [{'name': 'Alice', 'job': 'Engineer'}, {'name': 'Bob', 'job': 'Designer'}]. How would I represent this in a SQL table format?

> *Prompt:* I need to convert this TOML configuration: 'title = "My Project" owner = "Sara"' into an equivalent YAML format. How would you convert this?

Frontend Development

Frontend development involves making websites look great and navigate smoothly. At its core, it uses HTML to set up the basic structure of web pages, CSS to make things look nice and lay out everything, and JavaScript to make it interactive. Nowadays, frontend developers often use frameworks like React and Vue. These frameworks offer features like reusable components and ways to make data update in real time, which allows for building websites and apps that are dynamic and engaging.

Frontend development can be challenging, even for experienced developers. First of all, there is the unpredictable nature of web browsers and user interactions. You need to deal with different browsers, devices, screen sizes, and user preferences. It's like trying to make a one-size-fits-all T-shirt that looks good on everyone. Next, the frontend world changes fast. Framework updates pop up frequently.

Another thing about frontend development is that it's not just about coding. It's also a lot about good user experience (UX) and user interface (UI) design. This is a unique skill set that many programmers don't have. UX and UI design involves understanding how people interact with technology, what makes a design visually appealing, and how to create a smooth, logical flow in an app or website. It's like being part artist, part psychologist. For a lot of traditional programmers, who might be wizards at things like algorithms and data structures, the world of colors, layouts, and user journeys can be unfamiliar territory.

Regardless, there are ways that AI-assisted programming tools can help out. We'll look at some of these in the next few sections.

CSS

Writing CSS for websites can be tricky, especially when you're working on big, complex sites. It's challenging to make sure everything looks good across different browsers and devices. You've got to know all the weird browser-specific quirks. It's also easy to accidentally mess things up so that one part of your style steps on the toes of another, making things look wonky. Plus, traditional CSS doesn't let you use variables or functions, which means you end up repeating yourself a lot in your code (though tools like SASS and LESS help with that). Even something as simple as centering a `<div>` can be an ordeal.

Here are some prompts for common CSS tasks:

Prompt: Can you provide me with a CSS snippet for a responsive navigation bar that collapses into a hamburger menu on mobile devices?

Prompt: I'm having trouble with a flexbox layout in CSS where items are not aligning properly. Can you suggest what might be going wrong?

Prompt: I need to add a hover effect to buttons on my website. Can you show me a CSS example to make the button change color when hovered over?

Prompt: I'd like to add a fade-in animation to my website's homepage. Could you provide a CSS code snippet for this?

Creating Graphics

Creating professional graphics for websites or apps requires an extensive background in design, along with understanding sophisticated tools like Photoshop. But there are many powerful text-to-image generative AI systems that can help you create standout images. Some of these systems include:

- Canva
- Stable Diffusion
- DALL·E 3 (which is built into ChatGPT)
- Adobe's FireFly
- Midjourney

With these systems, you can create many types of assets, including hero images, buttons, and logos.

Let's look at an example. We'll have ChatGPT create a logo:

Prompt: I'm creating a website for my home bakery business named "Sweet Whisk." I want a logo that's warm and inviting. The main products are cakes and cookies, so maybe those could be incorporated into the design. I like pastel colors, especially light pink and mint green. The style should be simple and modern, with a touch of playfulness.
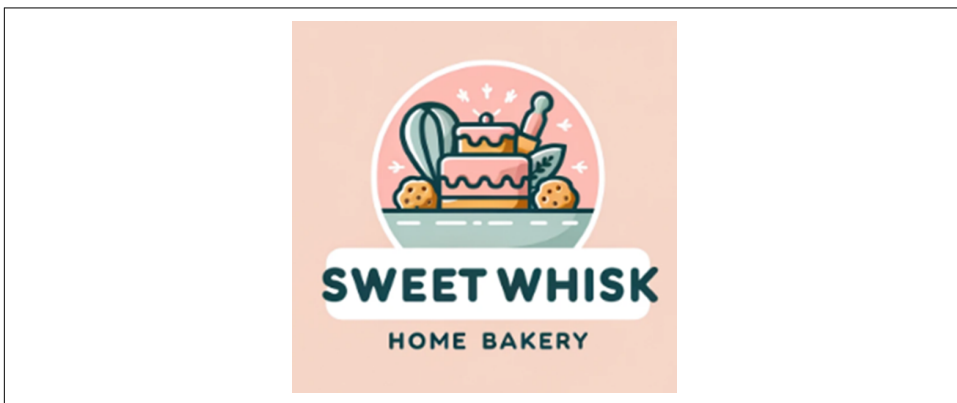
Figure 8-8 shows what ChatGPT created.

Figure 8-8. ChatGPT created a logo when given some guidelines

If you want to make changes to the output, you can continue to prompt ChatGPT. Other AI image tools, like Firefly, have more features than ChatGPT for designing images, but ChatGPT can still create compelling images—without much work.

AI Tools

There are some excellent AI tools that can whip up websites just from a prompt or a picture of what you want the frontend to look like. They'll handle all the coding for you. This even includes making conversions to frameworks like React, Angular, and Vue. What's more, an AI tool will also usually be able to import a wireframe, say from Figma.

After the code is created, you can jump in and tweak things to get it just right. Using AI is an effective shortcut for speeding up the whole website-building process.

Here are some of the AI tools available:

- TeleportHQ
- Anima
- Locofy
- v0 by Vercel

Let's take a closer look at v0 by Vercel, which is easy to use. In fact, the interface is similar to that of ChatGPT, as you can see in Figure 8-7.
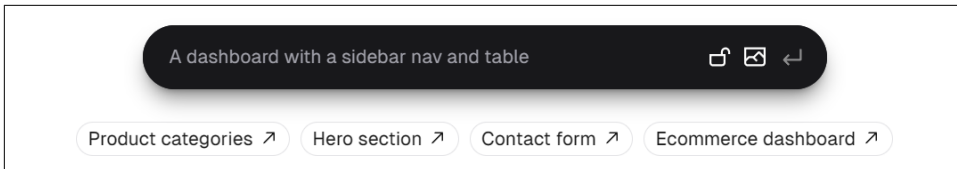
Figure 8-7. IF is the interface for v0 by Vercel

You can either enter a prompt for the type of interface you want the system to create or upload an image. For this example, I will use a prompt:

Prompt: An ecommerce dashboard with a sidebar navigation and a table of recent orders.

This will take you to a design studio, which will look like Figure 8-8.
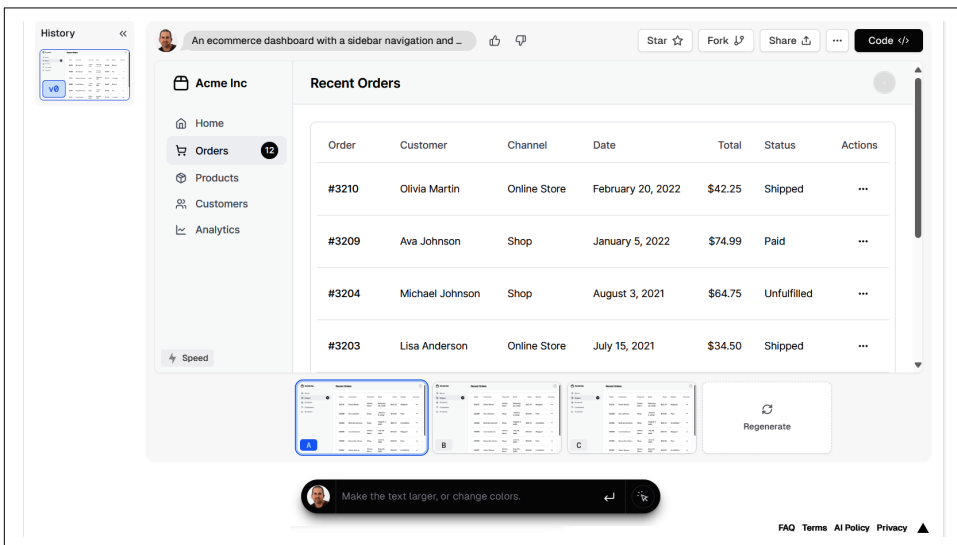
Figure 8-8. The design studio that v0 by Vercel comes up with when you enter a prompt

The AI will create three versions of the dashboard. You can create additional ones, too. Then, with a prompt, you can make changes, such as to the text size, colors, and so on.

When you are finished, you can click Code at the upper right. You'll see a listing of clean React code that's based on the shadcn/ui and Tailwind CSS.

There are also several open source systems that use AI to generate frontends. Consider Screenshot to Code. The creator of this project is Abi Raja, who is the CEO and founder of Pico. His prior startup was acquired by Yahoo! in 2013.

Raja spent six months creating the initial codebase for Screenshot to Code. "Frontend developers often convert designs and mock-ups into code," said Raja. "Much of this work is repetitive. But my software can help automate this process, achieving about 90% of the task."

Currently, Screenshot to Code supports exports to React, Bootstrap, and HTML/Tailwind CSS. The project has over 31,000 stars on GitHub.

With ChatGPT, you can also convert an image to code. Suppose you want to create a calculator app and you want it to look like the version on the iPhone. Figure 8-9 shows the image.
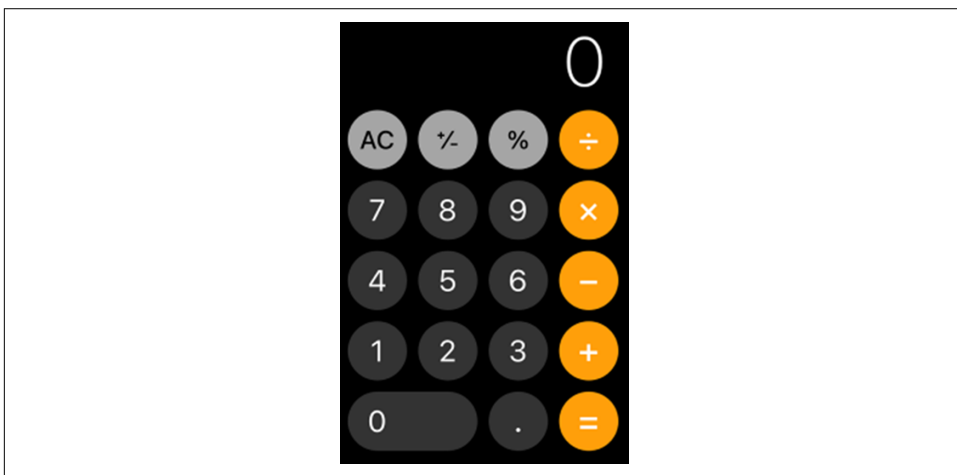
Figure 8-9. You can input an image of the iPhone calculator app to ChatGPT to generate code

First, we'll ask ChatGPT to identify the image, which it does correctly. Then we can use this prompt:

> *Prompt:* Suggest code for this image

Figure 8-10 shows the calculator.

ChatGPT created code for Python. True, the styling is off, but this is still fairly good. Of course, if you want it to look more like the iPhone version, you can be more specific with the prompt, such as by telling ChatGPT to use something like React.
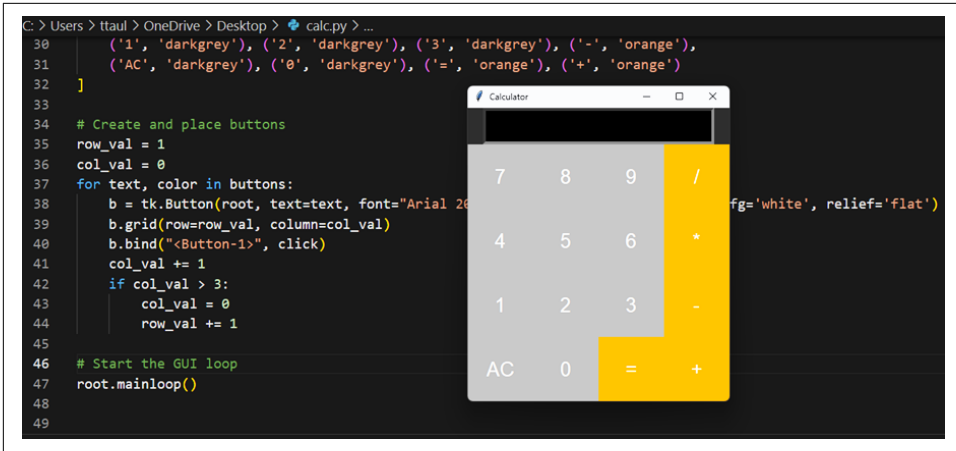
Figure 8-10. ChatGPT created code for the calculator based on an image of Apple's iPhone app

APIs

Getting the hang of a new API can be a challenge. First off, you've got to wade through the documentation, which may be dense and technical. And it's hit or miss with these docs—some are clear and easy to follow, while others…not so much. Then there's the deal with logging in and getting access, like using API keys or OAuth tokens. Plus, trying to figure out the data structures and response formats the API spits back at you can be overwhelming. Then there is error handling as well as having to keep up with any updates or changes in the API.

Fortunately, using ChatGPT can make tackling these challenges a bit easier.

To see how, let's take a simple example. Suppose you want to use an API to get information about the weather. You can ask ChatGPT for API suggestions:

> *Prompt:* What are good weather APIs?

One that ChatGPT recommends is OpenWeather API. Next, ask the following:

> *Prompt:* How do I start using the OpenWeather API?

ChatGPT describes the main steps.

You can then ask it to show how to make a request:

> *Prompt:* How do I create a request using the OpenWeather API? How do I do this using Python?

ChatGPT will go through the process, providing details on these steps:

- Import the requests library.
- Define the API key.
- Create the API request URL and use the current weather data endpoint.
- Execute the request and handle the response.
- Check to see if the request was successful.

Conclusion

In this chapter, we dug into modular programming, functions, and object-oriented programming. We also explored how to use AI to learn new languages and frameworks on the fly as well as how to use it to work more efficiently with data. Then we saw how to create compelling frontends.

But it's important to repeat: AI-assisted programming tools are far from perfect. That's why it's important to keep your eyes peeled and double-check everything.

Debugging, Testing, and Deployment

This chapter is about the parts of software development that often do not get enough love: debugging, testing, and deployment. It's a guide to everything from spotting different bugs, to writing docs that people can actually understand, to making sure your code does what it's supposed to. We'll also dive into how to merge changes without a hitch, make your software run smoothly and safely, take in what users say, and even how to make a splash when you launch.

Debugging

When creating code, a developer will spend around 35% to 50% of their time on debugging. This is not just a time sink. It also eats up a big chunk of the budget in a software project.

There are two main types of bugs. First, there are *syntax errors*. These pop up when your code doesn't play by the rules of the structure of a language. It can be as simple as forgetting to add a colon at the end of a `for` loop. A modern IDE like VS Code should detect and fix many of these types of errors.

Next, there are *logical errors*. These are trickier because they happen when something in your logic is off. For example, suppose you have created a program to filter out adult users from a list based on their age. The only problem is, instead of excluding everyone aged 18 and above, your code mistakenly excludes those under 18. So, you end up with a list full of adults instead of teens. This mix-up is a prime example of a logical error. Your code is doing the exact opposite of what you intended. Pinpointing why it's flagging the under-18 crowd, instead of those 18 and above, can be quite the puzzle. The fix can range from making a quick adjustment in the age condition to having to rethink the whole logic.

Regardless, using an AI-assisted programming tool is not something you should start off with when debugging. Often, traditional approaches should be fine. VS Code provides powerful debugging features that can detect and resolve problems. After all, you can easily set up breakpoints to inspect variable values, walk through the code line by line, and see what's exactly happening. These are a lifesaver, especially in large programs.

But of course, some bugs can be real head-scratchers. Modern software is often a complex puzzle with many layers and parts that have to work together. Sometimes a bug pops up because these parts interact in unexpected ways, and figuring out what's actually going wrong can be a real challenge. Then there's the issue of documentation—or the lack of it. When software doesn't come with clear instructions or explanations, it's tough to understand how it's supposed to work, which makes finding and fixing bugs even harder. Your software will usually rely on external libraries or services, and if something goes wrong there, the bugs in your software can be maddening.

So this is where you can turn to AI. For example, one scenario is deciphering cryptic or long error messages. These may be stack traces, for instance, which are snapshots of what the program was doing when everything went haywire. You might also get detailed information about the interactions among various frameworks and libraries.

What you can do is copy-and-paste the error message into a prompt and include something simple like:

> *Prompt:* What does this mean? {error message}

Or, suppose you have code and there is a logic error. The program simply will not do what you want it to do. In this case, you can copy-and-paste the code into the prompt. Then include this:

> *Prompt:* This program is supposed to allow users to upload photos and display them in a gallery. However, when it is executed, the photos are not appearing. What is the problem with this program? {code}

If ChatGPT finds the problem, it will often suggest a solution. But if it does not, you can add this instruction to the prompt.

Documentation

Documentation is the glue that keeps everything from falling apart, but sadly, it's usually shoved to the back burner. We've all seen it—working with code without a guide is like wandering in a maze, and it's a pain, especially for new folks or anyone trying to figure things out.

Good documentation helps to prevent guessing games and keeps everyone on the same page. A survey from Stack Overflow says that 68% of developers bump into these knowledge roadblocks every week. Plus, GitHub's report from 2021 showed that sharing information in documentation can make teams up to 55% more productive.

But it's not just about making developers' lives easier. Good documentation is the backbone of a smooth-running development process. It's akin to a map that shows you where to go and what to watch out for. Without it, you're often flying blind, and that's when costly mistakes happen.

With ChatGPT, you can create any type of documentation, such as:

- User manuals
- README files
- API documentation
- FAQs
- Troubleshooting guides

When developing prompts for creating effective documentation, here are some factors to keep in mind:

Know your audience

Think about who's going to use your docs. Are they newbies looking for a guide to get started quickly, or are they tech wizards needing the nitty-gritty details like API docs and code snippets? Getting a bead on what your readers need is key to hitting the mark with your content.

Keep it simple

Nobody likes to wade through jargon or tech-speak that needs a decoder ring. Keep your words straightforward and to the point. If your grandparent couldn't get it, you're probably not there yet.

Stick to the plan

Keep your docs looking familiar. Use the same style, headings, fonts...you name it. Consistency is your friend. It's like having a good rhythm in a song. It just flows better.

Show, don't just tell

People understand things faster when they see it in action. So, throw in real-life examples or scenarios. If you're talking to coders, some code examples are gold. For others, screenshots or step-by-step walkthroughs can be helpful.

A picture is worth a thousand words

Sometimes, text just doesn't cut it. Use diagrams, screenshots, or even videos to get your point across, especially when you're dealing with complex stuff.

The why matters as much as the how

> Don't just lay out the steps. Tell your readers why they're doing what they're doing.

With all this in mind, you could use a prompt like the following:

> *Prompt:* Can you suggest how I might explain the concept of version control in Git to a non-technical audience? I need to keep it simple and avoid technical jargon. Also, explain the importance of version control and suggest any visuals or diagrams.

 ChatGPT has deep capabilities for language translation. You can certainly leverage this with your documentation or any other content.

Microsoft has been cooking up a system called GitHub Copilot for Docs, which is for the enterprise edition. It's not your average, run-of-the-mill tool for digging through docs. For example, the search results and responses are based on a user's coding background and experience. It is also updated with the latest on GitHub's repositories. It even gives you the ability to add private documentation. Essentially, this is a highly sophisticated knowledge base that can greatly boost your coding.

Code Review

Think of a code review as your code's test drive before it gets a pull request in the codebase. You make sure everything's running smoothly, fits in just right, and won't go kaput down the road.

But the process is about more than just looking for clunky bits or glitches. It's helpful for everyone to huddle around, bat ideas back and forth, and learn from each other. You'll see different ways to tackle a problem and get a better grip on the whole project.

In the meantime, a code review can help enforce an organization's coding style and guidelines. Then there's the security check. You see, automated tools don't always catch everything. Sometimes, it takes a human eye to spot those sly security risks.

As for ChatGPT, it can be a key part of this process. Here's an example prompt:

> *Prompt:* Write a code review for the code below. Keep a focus on the maintainability of the code, potential security issues, and performance flaws. {code}

I intentionally gave ChatGPT a poorly written function, yet ChatGPT did a good job with its review. It suggested numerous areas for improvement, such as that the function could benefit from not having hardcoded database connections. ChatGPT also detected the potential for a SQL injection due to the direct concatenation of the user ID in the SQL query and a lack of user input validation. It then found a performance issue with the construction of the SQL.

Unit Tests

In a way, unit tests are mini evaluations for parts of your code, say for a few functions or methods. Developers often do this testing themselves using cool tools like JUnit for Java, NUnit for .NET, or pytest for Python. These tools help write and run tests, and tell you the results. They usually play nice with other software tools you're using.

Doing unit testing is helpful because it makes your software better, cuts down on pesky bugs, and makes it easier to tweak and fix problems later. Each test focuses on just one thing, so if something goes wrong, you know exactly where to look. These tests are usually automated, which means they can be run fast and often. This is important for keeping everything smooth and up-to-date.

Unit tests are usually straightforward to write. Since they focus on small parts of the code, they shouldn't be too complicated. Plus, they're like a guide to your software. By checking out the unit tests, other developers can get how certain parts are supposed to work. If you change your code, unit tests are great for making sure you haven't messed up something that was working fine before.

Let's take a look at an example. Suppose you have created a tip calculator program like this one:

```
def tip_calculator(bill_amount, tip_percentage):
    tip_amount = bill_amount * (tip_percentage / 100)
    total_amount = bill_amount + tip_amount
    return total_amount

bill_amount = float(input("Enter the bill amount: "))
tip_percentage = float(input("Enter the tip percentage (e.g., 20 for 20%): "))
total_amount = tip_calculator(bill_amount, tip_percentage)
print(f"Total amount including tip: {total_amount:.2f}")
```

This has a function that calculates the total bill amount including a tip and takes two parameters, `bill_amount` and `tip_percentage`. The function will find the value for `total_amount`.

For code like this, unit tests check all sorts of scenarios. For instance, they can help verify that the function correctly calculates the total amount, including tips, for a range of inputs. This is important because even a small error can lead to significant discrepancies. Unit tests can cover typical cases, such as standard bill amounts and tip percentages, as well as edge cases, like a zero or negative bill amount or unusually high tip percentages. They also help to ensure that the function handles wonky inputs, such as non-numeric or null values, gracefully.

Here's a sample prompt:

> *Prompt:* For this program, suggest unit tests to correctly calculate the total amount, check the typical and edge cases, such as zero and negative bill amounts and high tip values. Also check for invalid inputs. For the unit tests, you can have console logs.

Figure 9-1 shows the code that ChatGPT created for the unit tests.

If you want to use a testing framework, to allow for a more structured and comprehensive approach, you can ask ChatGPT for this:

> *Prompt:* Create unit tests that use a testing framework.

ChatGPT suggests using unittest. It shows how to set it up, develops the tests, and demonstrates how to run the unit tests.

Or, if you already have a file with unit tests, you can evaluate them. Here's a prompt:

> *Prompt:* Below is the file for unit tests for a program that <*explain what it does or point to the code*>. Any other tests I should have? What is missing here?

While ChatGPT or an AI-assisted programming tool can be useful with creating these, the tools are far from foolproof. For more advanced use cases or larger codebases, the results can be off.

Consider the following advice from David Lee, who is a founder, senior engineer, and AWS-certified solutions architect:

> However, when dealing with tests involving real database interactions and Docker, the dynamics change significantly. It becomes another level of sophistication that ChatGPT 4.0 may not be able to comprehend to some extent, and you probably need to write a few tests manually first, especially the database connection part so that it can learn how to write others.

```python
def run_tests():
    # Typical case
    assert tip_calculator(50, 20) == 60, "Typical case failed"

    # Zero bill amount
    assert tip_calculator(0, 15) == 0, "Zero bill amount case failed"

    # Negative bill amount
    # Depending on your function's intended behavior, update the expect
    assert tip_calculator(-50, 20) == 0, "Negative bill amount case fai

    # High tip percentage
    assert tip_calculator(50, 100) == 100, "High tip percentage case fa

    # Invalid input
    # This requires exception handling in the tip_calculator function
    try:
        tip_calculator("fifty", 20)
        print("Invalid input case failed")
    except ValueError:
        print("Invalid input case passed")

    # Boundary cases
    assert tip_calculator(0.01, 0.01) == 0.01, "Boundary case failed"
    assert tip_calculator(100000, 50) == 150000, "Large amount boundary

    print("All tests passed!")

# Uncomment below to run the tests
# run_tests()
```

Figure 9-1. ChatGPT created code for unit tests on a tip calculator program

Pull Requests

A *pull request*, or PR for short, is like a golden ticket in the coding world, especially when you're working with others. You wrap up your code, push it to somewhere like GitHub or GitLab, and send out a PR. It's not just a "please add my code" request, though. It's also a nudge to your teammates to check out what you've done, give it a thumbs up, or maybe throw in some pointers to make it even better. It's all about making sure that when your code joins the rest of the project, it's the best it can be. What's more, PRs are a neat way to keep track of who did what and when, which is certainly helpful in big projects.

No doubt, writing a solid PR description really makes a difference. You want to lay out the what, why, and how of your changes. Start with a quick rundown of the problem you're tackling. Then dive into how your changes fix this. Don't forget to throw in details like which files got a makeover or any tests you ran. If there's something specific you're unsure about or need a second opinion on, mention that too.

An effective PR description is a lifesaver for your reviewers. It speeds up the whole process and keeps everyone on the same page. Moreover, detailed PR descriptions are like a treasure trove of information for the future.

And yes, ChatGPT can be your sidekick in nailing those PR descriptions. Need to kickstart a draft? Tell ChatGPT what you did, and it will help you structure it into something clear and to the point. If you've already written a description, ChatGPT can check it out for clarity and grammar and suggest ways to make it even better. ChatGPT can also help you figure out the best way to lay out your PR, like starting with a summary and then getting into the nitty-gritty. And if there are technical details that need simplifying, it's got your back in making the PR more understandable for everyone in your team.

Let's take a look at some helpful prompts:

> *Prompt:* I added a new search feature to the application that filters results based on user input. Can you help me write a PR description for this?

> *Prompt:* I fixed a bug where the app crashed when a user entered special characters in the text field. How should I describe this in a PR?

> *Prompt:* I refactored the authentication module to improve performance and readability. What should I include in the PR description?

> *Prompt:* I updated the user interface to make the navigation more intuitive and added new icons. Can you help me draft a PR description?

> *Prompt:* What's a good way to phrase this in a PR description?

> *Prompt:* I added new unit tests for the payment-processing module. Can you assist me in writing a PR description that highlights these changes?

> *Prompt:* I resolved merge conflicts that arose due to recent changes in the main branch. What should I mention in the PR description about this?

Keep in mind that Microsoft has added a feature in Copilot that allows for creating PR descriptions. It's called Generated Commit Message. To use it, you'll need to make sure you have a connection to the repository on GitHub. Then you'll just need to click the sparkle button, as you can see in the lefthand panel in Figure 9-2.
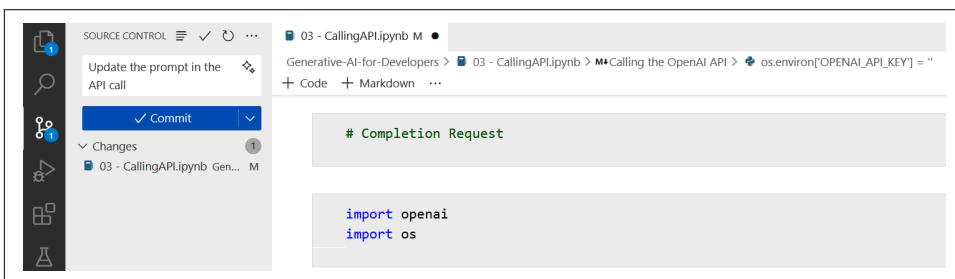
Figure 9-2. Copilot can create PR descriptions based on the repository that is loaded into your project

Copilot will then write up a useful PR description.

Finally, there are numerous startups that are creating their own systems based on LLM technologies. One is called What The Diff. According to the company's cofounder and CEO, Sebastian Schlein:

> There are two main features of What The Diff: 1) Making pull requests easier to review by summarizing them into plain English and giving reviewers and easy-to-understand overviews about the changes within the PR. 2) WTD can also write summaries that are completely non-technical and that get sent to other stakeholders like product managers who don't have GitHub access. That makes it easy for them to see if the change in the pull request actually matches their specifications.

Deployment

You've been grinding away on your software, and now it's go-time—the big launch. It's a thrilling moment. You're about to see real people using your software, telling you what rocks and what could use a tweak or even more. That kind of feedback? Priceless for a developer.

If you're in the business of selling your software, this is the moment you've been waiting for—when the cash starts rolling in. Nailing the launch can be a real game-changer for your bank balance.

But let's be real: deploying software is like holding your breath and hoping for the best. There's always this sneaky feeling that something might not go as planned.

You know how things seem perfect in your test setup, but then in the real world, they get a bit wobbly? It could be different hardware, some funky network systems, or just odd settings that mess things up.

Security is a big deal, too. Once you're live, you must be on your toes to keep the bad guys out and play nice with privacy rules.

Also, your software has to be tough enough to handle the crowd. It's got to stay quick and smooth, no matter how many people jump into it or how big your business gets.

And then there's the whole CI/CD thing. It's all about making deployments smooth and automatic to dodge mistakes. Sounds great, but getting it up and running, and keeping it that way, is a bit of a hustle.

Then what to do? You can definitely check out ChatGPT. The following are just a few of the prompts to consider:

> *Prompt:* Can you guide me in creating a deployment checklist for my team to follow?
>
> *Prompt:* What are some good learning resources for getting started with Docker for deployment purposes?
>
> *Prompt:* Can you provide best practices for zero-downtime deployment in a web application?
>
> *Prompt:* I'm encountering a "server timeout" error during deployment. What are some common causes and solutions for this?
>
> *Prompt:* Could you help me write a bash script for automating the deployment of my Python web app?
>
> *Prompt:* What are the essential configuration settings I should check before deploying an app in a production environment?
>
> *Prompt:* How do I plan a rollback strategy for a failed deployment in a cloud environment?
>
> *Prompt:* What security measures should I consider during the deployment of a financial application?
>
> *Prompt:* How can I optimize the performance of a deployed Node.js application?

While not perfect, ChatGPT can be helpful with complex DevOps. Here are some thoughts from Titus Capilnean, who is the cofounder and chief product officer at Private Market Labs:

> Cloud logs are not the easiest things to work with, especially as I'm not a devops engineer, but we have to deal with them, given that we are running processes on AWS and Google Cloud on a regular basis.
>
> One time, I had to set up tracking for a SQS + Lambda process, based on the output of a large-scale function deployment. My function was essentially printing a status in the logs and I planned to use that status to generate a report of that process—in our case, deal aliveness. I asked ChatGPT to provide an AWS CloudWatch query script and tuned it to the point where I could just run it at the end of each process to get my results. Reading the documentation in detail for this task would have taken probably 5–6 hours, so I saved a ton by going the GPT route.

Similarly, I had to set up some alerts in Google Cloud, and I worked with ChatGPT to create a query that excluded some system-level errors we weren't actually responsible for and were not user-facing. It saved me hours of reading and work, and it provided me with the custom metric I needed to set up a useful alert system for our team.

User Feedback

User feedback is key to making your software rock. When your users see you're tuning in and making changes based on their thoughts, they're more likely to stick around and be all smiles. It's smarter to iron out the kinks early with their help, rather than try to clean up a big mess later when a lot of (increasingly unhappy) folks are using your app.

Even with all the testing, some bugs are sneaky, only showing their faces when your software hits the real world. Your users are like your own personal detectives, spotting the stuff that might have slipped past you.

Sometimes your users get really inventive with your software, using it in ways you never dreamed of. Their wild ideas can light the spark for new features, or even brand-new products.

Of course, there's a whole toolbox to boost customer service. Just some of the tools include Zendesk, Freshdesk, Drift, and Salesforce. They are handy with everything from live chat that lets you talk to customers in real time to feedback forms that gather insights on what your users think. Automated ticketing systems keep track of customer issues, ensuring that nothing slips through the cracks. And don't forget about customer relationship management (CRM) systems, which keep all customer interactions in one place. These tools can really make a difference in providing efficient, responsive customer service.

Generative AI can definitely add value on top of all this. It is particularly good at processing large amounts of unstructured data like user feedback.

For example, suppose you have a file that includes lots of emails, IMs, and contact form information from users. You can then go to ChatGPT and use this prompt:

> *Prompt:* Identify the common themes and categories, such as usability, performance, features, bugs, and customer service. Also, do sentiment analysis on this file. Based on the frequency and severity of issues mentioned, help prioritize which bugs to fix first or which features to consider adding. With all this, create a report that includes charts.

Another way ChatGPT can help is with response drafting. If you're dealing with customer feedback, it can help you write better responses, such as by creating templates for answers to common questions. This way, you're always on point and professional when you're chatting with users or customers.

Or, you can use ChatGPT for more personalized responses. You can cut-and-paste user email and use a prompt like this:

> *Prompt:* Write a reply to the user email, focusing on a calm and understanding tone. Make sure the response is friendly and helps to ease any concerns without escalating the situation. {email}

In some cases, you can create your own LLM-based application to handle user feedback. This is what Warp did. The company had a developer spend less than a week—at half-time—to create the app. It was built using the OpenAI API.

"The app has made a huge difference," said Noah Zweben, who is a product manager at Warp. "Before, it was difficult to categorize and prioritize the incoming feedback. But the generative AI has been able to do this extremely well."

The Launch

Several years before ChatGPT became a big deal, generative AI was already making an impact on important activities like sales and marketing. The trailblazer was Jasper. The company grew at a staggering rate. Then again, generative AI is ideal for whipping up catchy and creative content—fast.

But you don't need Jasper for your software launch. ChatGPT should work just fine. First of all, you can start mapping out a killer marketing plan with it. Here's a sample prompt:

> *Prompt:* You have created an app to help people plan healthy meals. It creates custom meal plans based on dietary preferences, health goals, and nutritional needs. It can also generate shopping lists, offer recipe suggestions, and track nutritional intake. For this app, put together a marketing plan. The company is an early-stage startup and does not have much resources for a marketing budget.

ChatGPT first recommends that you identify your target audience. It suggests that the primary users are "health-conscious individuals, fitness enthusiasts, people with specific dietary needs (e.g., gluten-free, vegan), busy professionals, etc." It then covers various strategies, such as leveraging social media, content marketing, community engagement, email marketing, and partnerships.

Here are some other helpful ChatGPT prompts:

> *Prompt:* Write an engaging introduction for a blog post announcing the launch of the new health planner app, highlighting its unique features and benefits.
>
> *Prompt:* Craft a series of social media posts for announcing our new health planner app, focusing on its user-friendly interface and how it helps in managing health goals.
>
> *Prompt:* Compose a product announcement email for our health planner app, emphasizing its ability to track and improve users' health routines.

Prompt: Create a persuasive sales email targeting gym instructors and health coaches, promoting our health planner app as a tool for their clients.

Prompt: Generate a list of catchy and relevant names for a new health planner app that conveys a sense of wellness and organization.

Prompt: Develop ad copy for Facebook and Instagram ads promoting our health planner app, highlighting its ease of use and personalization options.

Prompt: Formulate a template to request testimonials from early users of our health planner app, to be used in marketing materials.

Prompt: Create an invitation for a virtual launch event of our health planner app, detailing the agenda and special guests.

Conclusion

This chapter is like a behind-the-scenes look at the things in software development that don't always get the spotlight. Sure, creating new software is thrilling, but it's the less flashy tasks like debugging, testing, and documentation that really make or break your app. We've talked about how AI tools, like ChatGPT, can make these jobs a lot smoother. These AI buddies aren't magic fixes, but they're pretty awesome at sifting through tons of data, giving advice, whipping up content, spotting problems, and speeding up the whole process. This means developers get to focus on the really tricky stuff. By bringing generative AI and some smart strategies into every step, from squashing bugs to launching, developers can build better software that really hits the mark for users—and do it quicker too.

About the Author

Tom Taulli (@ttaulli) is a consultant to various companies, such as Aisera, a venture-backed generative AI startup. He has written several books like *AI Basics* and *Generative AI*, which cover ChatGPT, GPT-4, and other large language models. Tom has also taught IT courses for O'Reilly, UCLA, and Pluralsight. For these, he has provided lessons in using Python to create deep learning and machine learning models. He has also taught on topics like natural language processing.

Colophon

The animal on the cover of *AI-Assisted Programming* is a reef triggerfish (*Rhinecanthus rectangulus*). The animal is also referred to as *humuhumunukunukuapua'a*, or simply *humuhumu*. Following a popular vote and the approval of the Hawaiian state legislature, the reef triggerfish was selected as the official state fish of Hawaii in 1984.

The reef triggerfish is found in shallow outer reef habitats in the Indo-Pacific and Hawaii. Its diet consists of algae and reef invertebrates, such as sea urchins and snails, which the triggerfish finds by swimming close to the bottom of reefs.

Although the triggerfish tends to keep its distance from onlookers, its distinctive charging, territorial behavior and appearance—a plump mouth, blue top lip, and size of up to 10 inches—make it easier to observe.

There are approximately 40 species of triggerfish, and the reef triggerfish's current conservation status is "Least Concern." Many of the animals on O'Reilly covers are endangered; all of them are important to the world.

The cover illustration is by Karen Montgomery, based on an antique engraving from *Oeuvres du Comte De Lacépede*. The series design is by Edie Freedman, Ellie Volckhausen, and Karen Montgomery. The cover fonts are Gilroy Semibold and Guardian Sans. The text font is Adobe Minion Pro; the heading font is Adobe Myriad Condensed; and the code font is Dalton Maag's Ubuntu Mono.