sonatype

The Effects of AI on Developers

Generative artificial intelligence (AI) and machine learning (ML) tools have dramatically reshaped the landscape of the working world throughout the past year. These advanced computational systems have been seamlessly integrated into domains of work where their historical proficiency was limited, surpassing previous shortcomings with resounding success and widespread adoption. However, this rapid evolution has taken many people by surprise.

Suddenly, every organization is grappling with the pressing question of how to harness the potential of generative AI and ML to enhance their operations. This question resonates not only from the executive level of the company but also trickles down to junior individual contributors. The question is also being asked of people working in a range of careers. The questions about how to apply generative AI are just as pertinent for marketing and human resources as they are for software developers.

If you're a software developer, there's a good chance you have mixed feelings about generative AI tools. Many developers are naturally curious about how these tools function. That's not surprising — generative AI tools are fascinating technological leaps. When you experiment with these tools, you'll quickly find they regularly yield astonishing results.

But there's another edge to the sword. The ability of generative tools to autonomously generate complex code raises an existential question: Do developers still hold a pivotal role in such a scenario? The enthusiasm for innovative technology and beneficial tools can be overshadowed by concerns about these advancements potentially rendering many developers obsolete.

In this paper, we attempt to forecast the ways generative AI will impact developers over the next few years. While we lack a crystal ball for making definitive predictions, we can draw insights from how these technologies operate along with their historical precedents. To give a little sneak peak: developers have reasons to embrace the opportunities that generative AI presents and alleviate some of their anxieties.

Generative AI tools require a great deal of guidance

Odds are good you've caught an impressive AI demo at some point. A developer types in a small amount of text describing what they want a block of code to do. A computer somewhere runs through billions of decision nodes, and seconds later generates a block of code that does exactly what the developer asked for in their prompt. Such presentations can be awe-inspiring, but they also have a tendency to make developers uneasy. After all, a single descriptive prompt isn't difficult for someone without software developer training to write.

Reality is far from the picture of simplicity portrayed in these eye-catching demos. After you get past the flare and start to work with large language models (LLMs) on a daily basis, you find that your AI's impact is patchy. Sometimes, it rapidly generates useful code and cuts out lots of tedious work. Other times, you find that it generates completely irrelevant comments or lines of code seemingly out of nowhere.

For the day-to-day, leveraging a generative AI tool trained on an LLM for development work requires a substantial degree of guidance to yield satisfactory results. That guidance requires steady, knowledgeable hands of someone well-versed in software development. After all, if an LLM produces code that deviates from your intentions or spawns irrelevant output, the only person who can discern these subtleties is an experienced software developer.

That may not be likely to change

One common misconception about generative AI tools is the assumption that they will "only continue to get better." In reality, making such a prediction with any degree of certainty remains a challenge. Research into LLMs and generative AI tools is research in the purest sense of the word. We're expanding the borders of human knowledge in real-time. It's entirely plausible that choices in developing these tools may lead us down genuine dead ends. Alternatively, we might stumble upon entirely new optimization methods, dramatically enhancing the efficiency of these tools in the near future.

The growth of tools like Generative Pre-trained Transformer (GPT) has not been remotely linear. GPT-1, released in 2018, used about 120 million parameters. GPT-2 used more than 1.5 billion. That's a tenfold increase in parameters. Yet, the first tool that was widely considered sufficiently powerful to do real work was GPT-3. How many parameters did that include? 175 billion. That's more than a hundredfold increase in model size.

Furthermore, the recent emergence of GPT-4 by OpenAI indicates substantial enhancements in efficacy. OpenAI has chosen to keep the inner workings of GPT-4 undisclosed, so we don't know how many more parameters that model uses. It might be that the improvements come from techniques that don't require scaling the model. Or it might involve another substantial increase in size.

The crucial point to recognize here is that even though increasing model size tends to increase performance and accuracy when it comes to simple text completion tasks, improvements in other tasks, like telling jokes, generating analogies, or answering questions, remains unpredictable. The development and refinement of these "emergent" abilities do not always align with model size increments.

AIs are not bound by facts or logic

Undoubtedly, artificial intelligence exhibits extraordinary capabilities. However, what often takes generative AI users by surprise is the realization that these tools operate beyond the constraints of conventional facts and logic. It is quite common for LLMs to occasionally generate what's been termed as "hallucinations." Hallucinations, in and of themselves, aren't a fundamental issue, provided that the individual using the LLM can promptly discern when the model makes a mistake. Often, these errors are glaringly evident to anyone with even a modicum of familiarity in the field.

However, for those lacking such experience, recognizing the AI's blunders when it crafts responses becomes a far more intricate task. Sometimes, even experienced users fail to notice hallucinations. There have been more than a couple of cases where trial lawyers have used ChatGPT (the chatbot trained on GPT) to construct their case arguments. Unfortunately, their unwavering trust in LLMs led to the issuance of nonsensical arguments and references to non-existent legal cases. This serves as a cautionary tale, particularly when employing LLMs for generating content like software code.

Beyond factual accuracy, AIs also operate without the confines of logical constraints or rules. YouTube is full of people attempting to play chess against ChatGPT and laughing along as the LLM issues wildly invalid moves.

None of these shortcomings disqualify LLMs from doing valuable work. But they do underscore the necessity of human oversight when striving to ensure the quality of the generated output. While AIs can provide real leverage for an experienced developer to do good work more efficiently, they won't be able to replace developers entirely. The people leveraging these tools must possess a deep understanding of their domain and vigilantly monitor the output to avert the rapid generation of subpar content.

Generative AI benefits developers at every level

Another common concern regarding the impact of generative AI on developers is the belief that it will primarily benefit experienced developers, potentially leaving junior developers to grapple with replacement. For a forward-thinking person, that's a scary prospect. Investing in junior developers forms the bedrock for cultivating productive and capable senior developers. If you're replacing junior developers with AI, chances are you might save money in the short run, but your organizational productivity could suffer in the long run.

In reality, the benefits of generative AI manifest at every level of the experience ladder. Let's explore how generative AI benefits developers at both senior and junior levels.

For senior developers: Enabling a motivated junior

One of the most evident benefits of AI becomes apparent after developers gain access to tools like GitHub Copilot. They can promptly offload repetitive and laborious coding tasks to the AI. What might have consumed ten or fifteen minutes of tedious typing can be accomplished with a brief description of their requirements, followed by some editing to validate the AI's output.

This is where a lot of junior developers get worried. Historically, these tasks were relegated to junior developers. Yet, there is no inherent reason why junior developers should handle this specific work, as it doesn't significantly contribute to their professional development. Such tasks are often perceived as "grunt work," and senior developers prefer to avoid them. Thus, they routinely roll down to the junior developers.

It's imperative to note that just as we vigilantly supervise generative AI to ensure valuable outcomes, we need to extend a watchful eye to work assigned to junior developers, as it might not produce results as promptly as ChatGPT does.

For juniors and seniors: An extensive reference

This aspect of generative AI bridges the gap between junior and senior developers. Well-trained generative AI tools, proficient in popular programming languages, serve as remarkably effective reference sources for developers at all levels. It's essential to remember that not every response generated by an AI is infallible; it requires validation. Nonetheless, these tools prove highly beneficial for investigating common behaviors in programming languages and providing guidance on how to achieve tasks without the need to master the basic syntax.

This concept is especially advantageous for junior developers who may possess an idea of what they want to accomplish but struggle to express the concept effectively. Normally, this is a task that would require the guidance of a senior colleague, which can occasionally become a significant hurdle for junior developers. Senior developers often find themselves preoccupied with their own tasks, even though mentorship is integral to the senior-junior dynamic. By providing junior developers with a resource to brainstorm more efficiently, generative AI tools reduce the time spent teaching routine techniques to juniors.

Furthermore, the benefits of brainstorming extend to senior developers as well. ChatGPT is not limited to helping you differentiate between `**Array.split**` and `**Array.slice**`, although it's pretty helpful on that front too. For senior developers, brainstorming typically involves tackling less-understood or unique problems. Additionally, ChatGPT can serve as a valuable "rubber duck" during debugging sessions, providing a fresh perspective and insights.

For juniors and seniors: A private repositories resource

While somewhat experimental, tools like GitHub Copilot Chat can offer substantial benefits to developers regardless of their level of experience. These tools grant developers with access the ability to query "the repository" itself, with the LLM being trained on the code within the private repository and offering text-based interaction for developers.

This resource is invaluable to any developer, irrespective of their background, for a simple reason: the challenges posed by transitioning into a new role and encountering an existing codebase with limited local knowledge or documentation. Legacy projects often suffer from a chronic lack of documentation, which can hinder their progress and impede future development. In most cases, resources capable of consistently providing in-depth answers are few and far between. Many businesses run on systems that haven't been updated in years.

This is where a tool like Copilot Chat fills in so well. It's important to remember that not every response it provides should be taken at face value — verification of the information is crucial. Yet, even a resource that offers occasional inaccuracies along with accurate insights is more advantageous than not having access to such a resource at all. The true value lies in having access to such a resource, despite its occasional inaccuracies. Allowing GitHub Copilot to learn about your repository and then answer your questions is likely quicker and easier than doing it yourself.

This is particularly beneficial for junior developers who are relatively new to the workforce, typically lacking the experience and expertise needed to swiftly acclimate to a new codebase. For many developers, honing the skill of rapidly comprehending a new codebase is a process that takes time and experience to develop fully. It's not uncommon to hear businesses express that they don't anticipate new hires making substantial contributions for six months or longer. A tool like Copilot Chat could reasonably cut that time down to a couple of weeks of onboarding and regular questions to the LLM.

For junior developers: An entry-level mentor

Junior developers have a lot of questions. This isn't a critique of their level of experience; rather, it's a natural and expected part of their professional journey. Asking those questions plays a pivotal role in their career growth. But it's often challenging for a mentor to be available around the clock to address these inquiries. Moreover, many of the questions posed by junior developers are questions that senior developers have answered numerous times before.

That's where a generative AI tool can provide real value for those developers. ChatGPT doesn't take days off, and it doesn't have "working hours." It stands ready to swiftly respond to a junior developer's questions, aiding them in grasping key concepts and terminology that might have been previously unfamiliar. While the information it provides may not always be flawless, it often serves as a reliable stepping stone toward understanding.

ChatGPT excels in addressing common queries, enabling senior developers to direct their efforts toward tackling more intricate and value-driven questions. This not only maximizes their productivity but also eliminates the need for juniors to grasp common terminology and jargon that might be unfamiliar to them. By streamlining this process, it frees up time for senior developers to focus on more substantial development work.

LLM tools have the capacity to shed light on the rationale behind selecting specific tools or architectures to accomplish technical tasks. These inquiries are not strictly reference-based and often lack a definitive correct answer. Instead, they delve into comprehending the inherent trade-offs associated with particular choices. Such questions are commonly posed by junior developers, and senior developers may find themselves frequently reiterating the same explanations. Turning to ChatGPT to understand why you would use a NoSQL over a relational database is an optimal use of an LLM's capabilities.

Jevons Paradox: We're going to need more coal

The final reason why developers should be excited about the AI revolution involves a compelling historical analogy from the Industrial Revolution. Developers often hold concerns that AI will replace large parts of software development work, leading to reduced demand for developers themselves.

In reality, the opposite could be true. Instead of reducing the demand for developers, the increased cost efficiency in software creation is poised to intensify the demand for their expertise.

Increasing efficiency increases demand

In the mid-1800s, England underwent an Industrial Revolution, spurred on by the invention of the steam engine. Widespread utilization of steam engines necessitated a great deal of coal supply to power the engines. Over time, new steam engines were invented, improving their fuel efficiency. This meant that each individual engine burned less coal.

Naturally, one would expect that increased efficiency spread out across a fixed amount of work would lead to less demand for coal. This is exactly what developers fear will happen with software development, with AI driving redundancy. However, history tells a different story. Economist William Stanley Jevons observed that rising efficiency didn't lead to lower demand. The increased efficiency meant that demand rose, because jobs that used to be cheaper using manual labor now benefitted from mechanical automation.

This tracks with existing trends in ease of development. AI is not the first wave of innovation to simplify software development processes. Developers previously harbored concerns regarding the emergence of dynamic languages, improved compilers, or web-based code deployment tools. Much like with AI, these advances empowered developers to write more complex applications, fueling overall innovation and drastically increasing the demand for developers worldwide. The demand for software developers has reached an all-time high, making it the most opportune time for professionals in this field to thrive.

Many companies already grapple with backlogs extending years into the future, perpetually expanding, and ultimately leading to the disbanding of teams responsible for managing these backlogs. The prospect of a dearth of work for software developers is far from realistic. The cost-efficient and more effective nature of software development is only going to increase the amount of work that companies are willing to move to the realm of custom software.

Would your company benefit from writing more software?

If you're a developer and apprehensive about AI/ML, you shouldn't be asking when the robots are coming for your job. Instead of pondering when automation might encroach on your role, ask if your company could reap benefits from an amplified output of software. If the answer to that is yes, then the AI wave should be a source of excitement. Integrating generative AI tools into your workflow not only expands your capacity but also affords you the opportunity to address long-standing issues and elevate the quality of your codebase.

The vast majority of companies would benefit from the ability to produce a greater volume of software. This is especially encouraging news for developers currently active in the job market. It signifies a promising future ahead, one marked by engagement with intriguing challenges and the ability to effect change more rapidly than ever before.

Rather than dreading the AI revolution, now is the time to learn how to use these tools and leverage them to become better at the jobs we do every day. It's important to recognize that these tools are not infallible, and it's improbable that they will attain perfection in the foreseeable future. But by increasing our efficiency today, we stand to improve our own circumstances. By understanding how to use these tools, we'll be well-prepared to utilize forthcoming iterations adeptly. Skills may evolve, but for a developer, the ability to learn and adapt remains paramount. Learning how to harness generative AI tools might feel like a unique challenge, but it's a skill like any other — acquirable and advantageous.