Business Horizons (2024) 67, 649—661

Available online at www.sciencedirect.com

ScienceDirect

KELLEY SCHOOL OF BUSINESS
INDIANA UNIVERSITY
www.journals.elsevier.com/business-horizons

Navigating software development in the ChatGPT and GitHub Copilot era

Stephen L. France

Mississippi State University, Mailstop 9582, Mississippi State, MS 39762, USA

KEYWORDS
Generative AI;
Large language models;
Software developers;
AI prompting;
Prompt engineering;
Capability maturity model

Abstract Generative artificial intelligence (GenAI) technologies using LLMs (large language models), such as ChatGPT and GitHub Copilot, with the ability to create code, have the potential to change the software-development landscape. Will this process be incremental, with software developers learning GenAI skills to supplement their existing skills, or will the process be more destructive, with the loss of large numbers of development jobs and a radical change in the responsibilities of the remaining developers? Given the rapid growth of AI capabilities, it is impossible to provide a crystal ball, but this article aims to give insight into the adoption of GenAI with LLMs in software development. The article gives an overview of the software-development industry and of the job functions of software developers. A literature review, combined with a content analysis of online comments from developers, gives insight into how GenAI implemented with LLMs is changing software development and how developers are responding to these changes. The article ties the academic and developer insights together into recommendations for software developers, and it describes a CMM (capability maturity model) framework for assessing and improving LLM development usage.

1. Generative AI: A coding revolution in the making?

Generative artificial intelligence (GenAI), based on LLMs (large language models), exploded into public consciousness in November 2022, spurred by the release of a trial version of the chatbot ChatGPT. ChatGPT was developed by OpenAI utilizing the GPT-3 neural network, which has a size of 175 billion possible parameters and was trained on 300 billion words (Hughes, 2023). The previous generations of AI often had quite well-defined inputs and outputs. For example, a chess-playing AI would be trained using the rules of chess and many different past game scenarios. The current generation of GenAI using LLMs, powered by billions of neurons, has more breadth and scope for nonstructured tasks. ChatGPT and similar LLMs can generate creative output from simple user prompts and have the potential to create unstructured outputs, such as student essays, news releases, computer code, music, and art (Sommers, 2023).

E-mail address: sfrance@business.msstate.edu

https://doi.org/10.1016/j.bushor.2024.05.009

While the effects of LLMs on education have grabbed headlines and policy interest, software development has the highest rate of LLM adoption and usage, with nearly 25% of software-development companies implementing LLM-based GenAI initiatives (MacRae, 2023). Since the launch of first preview of GitHub Copilot in 2021 (Gershgorn, 2021), coders have been utilizing LLMs in their workflows, and as of July 2023, a survey on Stack Overflow of over 90,000 developers (Vincent, 2023) found that over 70% of developers were using or planned to use LLM-based tools for coding in 2023, though fewer than 50% of developers replied that they "somewhat trust" or "highly trust" AI coding tools.

But what do the rapid changes outlined above mean for developers? In general, there are contrasting views of the effects of GenAI on employment. It may be that AI will replace categories of skilled jobs that were previously not at risk from automation. But it may be that LLMs will be used to supplement workers' existing skills, enabling workers to adapt and to improve their productivity by incorporating AI into their workflows (Chatravorti, 2023). This adaption is a key focus of this article. How can developers and development teams best incorporate LLMs into their workflows? How can they modify their skill sets to thrive in an environment where LLM-based tools are ubiquitous?

This article looks to help software developers and business managers who utilize the services of software developers—for example, a marketing manager who needs custom reports or sales-automation macros—apprise this rapidly changing environment. One could pose the question: Is GenAI with LLMs going to lead to a radical transformation of the software industry with mass job losses among developers, or will the effects be more positive, with developers using LLMs to improve productivity? Given the pace of technological advancement, it is impossible to provide a definite prediction of the overall future of software development; however, it is possible to apprise the current technological situation and initial implementations of LLMs, and to make concrete, forward-looking recommendations. Thus, the major aims of this article are, first, to understand how LLMs are being implemented in the software industry; second, to provide developers with a framework for implementing LLMs; and third, to provide advice to developers on how to adapt to this new environment.

The article starts with a brief overview of the current state of the global software industry, followed by analysis of the working environment for software developers, with a focus on factors related to the productivity of developers. This is followed by a review of the initial findings on the effects of LLMs on software development and a content analysis of developer conversations. While similar analyses have been performed in the software-engineering literature, this analysis has a broader focus than previous analyses, and includes insights into developers' hopes and worries for their careers and into future-looking trends in the software development industry. A capability-maturity model (CMM) framework is introduced for evaluating LLM-based software development, and content analysis is used to help evaluate the maturity of current LLM-based development efforts.

2. Software development: Industry and skills

The scale of the global software-development industry is enormous. As of 2023, globally there are approximately 35 million software developers, with the most common programming languages (in order) being JavaScript, Java, Python, C/C++, and C# (Noll et al., 2023). Most projections show that the demand for programmers will grow. In addition to dedicated software developers, there are many millions more people who perform some coding tasks as part of their job requirements, a segment of developers that may grow with increased use of LLMs that democratize coding and allow more people to easily create code (Davenport et al., 2023). Examples include accountants writing VBA code to automate spreadsheets, business analysts running analyses in R or Python code to find actionable insights from data, and artists writing programming scripts to help animate artwork and add visual effects.

Despite high-profile cutbacks in the tech sector in 2023 by companies such as Amazon, Meta, and Salesforce (Trueman, 2023), long-term growth is expected in software development jobs. In the US, the Bureau of Labor Statistics (2022) has estimated around 15% growth in computer and information technology jobs between 2021 and 2031.[1] In addition, commentators have predicted a global shortage of technology workers, with a potential talent shortage of 85 million software developers by 2030 (Sloyan, 2021).

[1] There is a projected increase for the categories of "Web Developers and Digital Designers" and "Software Developers, Quality Assurance Analysts, and Testers", but a slight decrease for the "Computer Programmers" category. This may indicate a change in development focus or perhaps a change in job title nomenclature.

3. The development environment and developer productivity

Thus far, this article has explored how LLMs are increasingly being used by software developers. To fully understand how LLMs impact developers, it is important to understand some of the basics of developer roles and workflows. This section acts as a prerequisite for the remainder of the article by summarizing academic software development work and describing aspects of developer jobs that will be affected by LLMs, along with associated terminology.

Working as a software developer is a multifaced job. Besides coding, job responsibilities can include testing, documentation, systems design and analysis, technical support, and interacting with clients. Software development is a creative field, but it is important to approach coding tasks systematically. For example, research found that when making code changes, developers who methodologically worked through the existing code to understand it had better performance than developers who took a more piecemeal approach (Robillard et al., 2004).

Generally, as developers become more experienced, they take on broader responsibilities, with senior developers taking on both supervisory responsibilities over junior developers and management responsibilities over areas such as coding standards and processes (Ardimento et al., 2022). Developers have both external and intrinsic motivations for good performance. For example, the tech industry is notorious for stack-ranking reviews, where all employees are ranked on a scale, and a certain bottom percentage of employees are marked as having unacceptable performance (e.g., Hess, 2023), which leaves employees with constant performance anxiety. Intrinsic motivation often comes from a sense of pride in one's technical ability. Research has shown that software developers value a sense of agency and control over their work and have a sense of achievement from working productively on coding tasks (Meyer et al., 2021), and they often test their skills with competitive coding challenges (Kumar et al., 2018).

A developer does not exist in a vacuum, and a developer's success depends on a range of factors, including interactions with other team members and the use of tools and search resources to find information. A developer may work on a piece of code individually or with other people, and quality control and code reviews are a big part of the development process. Code reviews can be informal among team members or can be formal processes built into the software development process, with assigned reviewers and scores given to reviewed output, and automated processes checking for common errors and coding-standards violations (Badampudi et al., 2023). Code reviews can differ between companies. For example, pair programming incorporates reviewing into the programming process by having one programmer working as a driver and focusing on the coding and syntax, while another programmer works as a navigator, guiding the coding and design process and reviewing code (Bird et al., 2022). Google employs a lightweight review process, with changes often reviewed by only one reviewer, albeit with high reviewing standards and quick review times (Sadowski et al., 2018).

The knowledge required by software developers can be immense, with developers needing to know programming-language commands, coding standards and syntax, licensing information, tool usage, bug fixing, and how to access both general and organization-specific coding libraries and resources. It is impossible for a developer to have all this required information available on tap from memory. To get additional information and knowledge, practitioners utilize a range of resources, including help searches in integrated development environments (IDEs), text searches on the web, colleagues, product documentation, online forums and bulletin boards, online tutorials, code repositories, and blogs (e.g., Xia et al., 2017). Developers often engage in opportunistic reuse of code found on the web but need to edit and adapt code to fit into their codebases (Ciborowska et al., 2018). This practice saves time but leads to potential security and licensing issues. The Stack Overflow community (including the Stack Exchange coding forums) is particularly important for programmers, allowing them to ask questions and receive answers. The whole archive of questions and answers is searchable. In fact, querying Stack Overflow to find useful code snippets and to help find solutions for debugging code is an important skill (Li et al., 2022).

The environment described above is one in which developers utilize a range of knowledge and skills but frequently need to search for additional resources and knowledge. Developers work with other team members, review code, and manage code in repositories. A developer's responsibilities depend on their level of experience. Many of the facets of development described above are in the process of being altered by LLMs. Despite the recent introduction of LLM-based tools for coding,

with the first preview of GitHub Copilot, released in 2021 (Gershgorn, 2021) and a demo version of ChatGPT, released in 2022 (Marr, 2023), the use of these tools for coding has become widespread.

Along with the rapid commercial implementation of LLMs, researchers have noted the potential for GenAI to transform development, but they have also found that there are limits to what LLMs can achieve. Overall, most studies have found that GenAI using LLMs works well on small, well-defined coding samples. But LLM usage stumbles with larger, more complex problems, and though LLMs can create good-quality code, they are not at the stage of being completely independent from human control. Most of the empirical studies on code correctness (e.g., Nguyen et al., 2022) found that LLMs could only solve a proportion of the posed problems correctly. Additionally, LLMs do not always generate code to high security standards, and since they are reliant on the code base used to train them, it is possible for security risks and malicious code to be propagated through the LLM models (e.g., Pearce et al., 2022). However, there is great potential for LLMs to help speed up code development by offering suggestions and by creating initial versions of code (e.g., Imai et al., 2022; Peng et al., 2023), and LLMs could be an effective alternative to a human pair programmer. The currently deployed LLM models are a long way from artificial general intelligence. The actual performance of LLMs may depend on the application and on how much training code is available. For example, Nguyen et al. (2022) found that LLMs perform differently on different programming languages. Using LLMs with a niche language, one with nonstandard syntax, is likely to give poorer performance than with a more popular language, such as Java or Python.

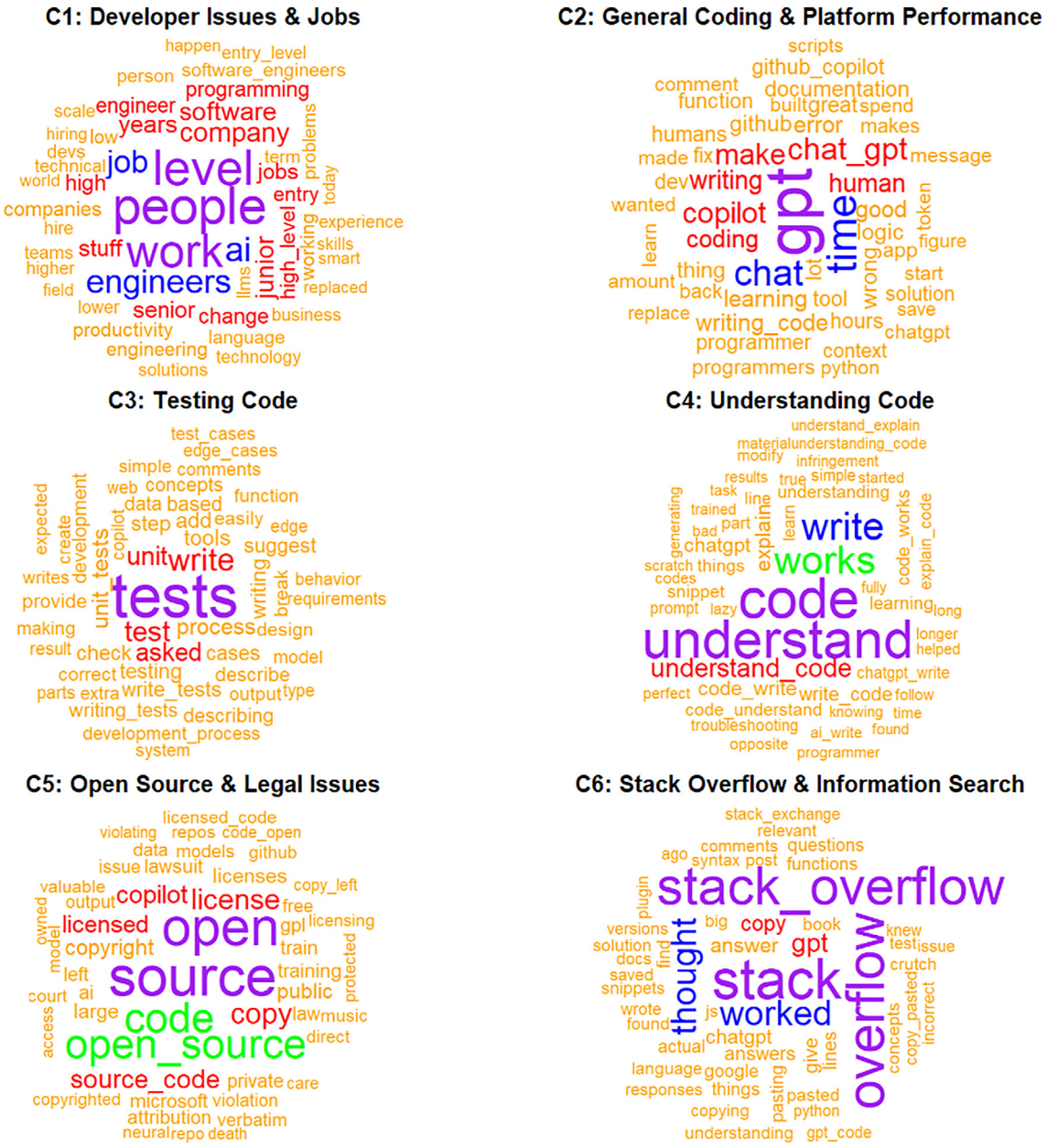
4. Qualitative content analysis of developer views

The previous section gave some insight into the usage of LLMs in software development and into some of the potential benefits and limitations of LLM usage. But how do developers feel about these new technologies? Can we gain any additional insights from the initial implementation of LLMs by developers? I performed a social-media content analysis to gain further insight into how software developers are utilizing GenAI and LLMs and to understand trends in LLM usage, with a view to giving forward-looking recommendations for LLM use. This work follows previous work that analyzed LLMs using social-media and discussion-board posts with both ChatGPT (Feng et al., 2023) and GitHub Copilot (Zhang et al., 2023), which found such benefits as helpfulness in generating code, reduced coding time, and integration with standard editors. The same studies also found some drawbacks, including poor code quality, privacy issues, issues with the pricing or subscription structure, and limitations on the size of the code that could be created. My analysis builds on this previous work but aims to provide a wider understanding of the environment for GenAI and software development, and of the opportunities and anxieties expressed by developers regarding GenAI. Reddit was an important source, as it has a much broader scope than Stack Overflow or GitHub and includes more general discussion of careers.[2] To find relevant posts, I searched for each of the different LLMs along with terms related to coding. The search was not restricted to any specific subreddit, though the majority of posts came from coding- and development-focused subreddits. Posts were chosen that (1) had both LLM and coding content, (2) were written by commenters who had at least a minimal knowledge of coding or development, and (3) had original content (i.e., were not purely a reply to other posts). A saturation-sampling approach was used to ensure a broad range of posts for each of the initial thematic categories, which resulted in 2,567 posts being retrieved. I then used a semiautomated content-analysis procedure (e.g., Horne et al., 2020) to help categorize the posts. Categories were formed by analyzing the topics in the literature review and then matching the categories with those found from a cluster analysis of the text comments.[3] Cluster analysis is a method of grouping items and is used in the business world for grouping or segmenting products or customers. It is also used for finding and extracting topics from text data. In this case, I used cluster analysis to categorize the Reddit comments into homogeneous groups or topics. A word-cloud visualization of the categories is given in Figure 1. The world cloud gives the top 50 terms based on occurrence in the category comments over the baseline rate

[2] I considered using Twitter too, but given its recent removal of academic API access and its ongoing platform instability, I discounted it.

[3] The comments had punctuation and low-information "stop" words removed, and data were converted into text × feature data (with words + bigrams), and k-medoids clustering was used. An eight-cluster solution was chosen using stability analysis, and these eight clusters were mapped to six categories, with clusters for the different LLMs mapped into one platform category.

Figure 1. Category word cloud



across all topics, with the size of each term based on relative occurrence.

To aid further analysis, each comment was assigned both a centrality score (how core the article was to the category)[4] and a positive/negative sentiment score, derived using sentiment analysis. This information was used to find core concepts in the categories and core representative comments. To name each of the categories, the top terms from the word cloud were analyzed, along with the comments in each category with the highest centrality. A summary of the resulting six categories is given in Table 1.[5] I selected 10 representative comments for each category, and I combined some very similar comments. The

[4] This was operationalized as a silhouette score for cluster analysis, with larger values of the score indicating more central comments.

[5] A full dataset with the comments and cluster analysis is available from the author.

Table 1. Summary content for thematic categories

Category 1: Developer issues and jobs
Why do businesses hire developers when they can build their apps with no code solutions?
Most programmers are not writing difficult code at all, I think AI today can compete with entry level positions.
A lot of folks are going to become prompt engineers. Using ChatGPT these days, I think of myself as a solution engineer or solutions architect.
I never got into coding and now I'm doing stuff I usually had to ask our engineers to do.
I see a lot of lower/middle people getting a bite taken out of their salaries in the coming years, and there being increased requirements/competition.
It is to modern programming what high-level languages were to assembly designers, or what digital programming was to card-punchers.
This will just speed up and change the workflow. Mechanical engineers aren't obsolete because CAD is faster than drawings.
Freezing entry level hiring because of AI is incredibly dumb and short-sighted. You don't hire juniors to do simple, menial tasks. You hire juniors to turn them into seniors.
It can help entry level people scale their work faster, help them write test cases, help them think through problems, and so on.
AI probably does kill several jobs, but it also is going to be best when paired with skilled workers to enhance what they can do, or help train lower skill workers to improve themselves more quickly.
I have never met a client who could explain what they needed well enough to get an AI to code it.
I think this is my problem with inexperienced analysts/coders using the tool. If they use a LLM to write their code and get a result, they have no idea how to validate that result. Learning the technology is still a good idea IMO. A job interviewer for an analyst position may expect you to know a few things.
Category 2: General coding issues and platform performance
The GitHub copilot is the best coding tool I've ever seen in 25 years of coding. Increased my speed by 50% by doing all the boring repetitions and quickly suggesting whole functions based on comments.
Just spend most of the time writing good documentation and have copilot fill in the rest.
I do more complex algorithms and generation in ChatGPT4 and use Copilot just to speed things up in my editing.
Copilot consistently saves me about 1-4 hours a week of writing code manually.
GitHub Copilot is very good at what it does, suggestions and autocomplete, but is not fully functional by itself. It needs good prompts and guidance from a human developer.
I used GitHub Copilot—it saves my fingers from keystrokes for boiler plate code. But it also generates utter garbage where I stare at it.
Unless it knows my entire codebase, APIs, and database, it really doesn't do much more than give me a template for functions.
It generates patterns…But it doesn't spend time thinking: "Okay, so this is easy, but that is best practice."
Copilot code cleanup is horrible. It's good for writing boiler-plate code, and that's about it.
He's right in saying that writing code goes quicker, except now you're spending 10x time fixing bugs.
Every coder will tell you ChatGPT produced horrendous code and makes $%# up as it goes as along.
Category 3: Testing code
I find it useful for writing generic unit tests and to document the tests with comments.
The largest benefit we've found is how adept it is at extrapolating from one unit test and autocompleting 5—6 additional test cases for a function.
It honestly generates an entire 30-line test exactly how I would have.
It is useful for regexp optimizing, unit test auto generation, and static analysis.
Once a test is done, it autocompletes my other tests, which follow the same pattern.
My belief is we will be the ones writing the tests. What better way to verify the output?
I could easily fall into the trap of expecting ChatGPT to write all the tests for me and then just have it enter a feedback loop where it's only looking at tests it has generated meaning the tests become less reliable over time.
Tests are the check to make sure the AI is in check. If AI is writing tests, you have to double check the tests.
I love that people would rather have AI write tests for them than admit that our testing practices are rudimentary and could use substantial improvement.
I find best use of ChatGPT is to run a check on your own code to see if it needs improvements, and for writing unit tests.
It seems a little risky to rely on it for tests, since it can still hallucinate sometimes.

Table 1 (*continued*)

Category 4: Understanding code
I can now understand the code BETTER than I ever did—through prompting questions and getting really good explanations.
Point to a line number of a code snippet and it'll tell you exactly what, why, and how it works.
I ask it to explain how each piece of code works then save the snippet of code for future use after gaining full understanding.
I always understand the code first, then rewrite it to my coding standards.
I'm good enough of a programmer to be able to use and modify it as I need it, but I don't fully understand the code.
I would never use ChatGPT to write code. I need to understand my code.
I think I have never committed a piece code that I did not understand. I am responsible for it, so I need to know what it is doing.
I always understand the code first, then rewrite it to my coding standards. If you don't, your codebase will turn into an unmaintainable heap of $%#^.
Having to fix things forces me to learn and understand the code, so a win-win all around.
Sometimes it can make weird mistakes or miss details that require you to understand the code after all.
It doesn't understand code. It's predicting what should come next. For simpler things that have a ton of examples it is easy. For others it is hard.
ChatGPT has been teaching me to be a better programmer because I review the code to make sure it'll actually work. Some of which I don't understand why and I ask it to explain.
Category 5: Open-source and legal issues
The copilot AI frequently copypastes code directly from its training data. That would be fine, since that's what a lot of developers do anyway, were it not for open source/free software licenses.
Open source licenses will need to be updated to include specific verbiage regarding usage in training sets since this is a grey area
This could be like sampling music, but it still has to be differentiated enough and code has less leeway to claim artistic expression or parody.
Things like the Quake code snippet that included the comments of the license and the original comments on the implementation make using large snippets of code from copilot concerning legally.
If an engineer memorizes and replicates a copyrighted section of code, it will still violate copyright. Fair use for AI is a gray area.
They need to include the relevant license and attribution if the output is an exact match from something within the training set.
If it's using code with copy left license to train, it must also be licensed properly.
If this is regularly direct copying unlicensed code, I would expect it to be banned by large corporations purely out of risk management.
xxxxxx shouldn't be able to destroy open source licensing by putting an AI in front of code copying.
Category 6: Stack Overflow and information searches
It is like Stack Overflow, but you can ask follow-up questions to help diagnose code you struggle with.
It is like having someone on Stack Exchange answer my question instantly.
It is preferable to Stack Overflow as you do not need to deal with "stuck up $%#^&," sarcasm, or condescension.
It's just a more efficient Stack Overflow.
What's crazy is that it did 99% of its training On Stack Overflow, and somehow got the technical knowledge WITHOUT becoming an $%#^&%.
It saves you a Google search and a couple clicks through outdated stack overflow posts.
It is good for small, trivial, or minor questions that impede your progress but aren't important enough for Stack Overflow.
If you feel like after some time ChatGPT is taking you in a loop, then you have to ask aka Slack Overflow.
It doesn't explain as well as Stack Overflow and it misses lots of nuances/gives incorrect explanations.
I'd much rather use an article where the author validated and explained the code, like Stack Overflow.
It tries to guess the answer you want, not to deliver the real answer. You are better off Googling for code on GitHub or Stack Overflow.
One of the barriers I faced when starting to code (and probably most people do too) is dealing with small, trivial, or minor questions that impede your progress but aren't considered important enough to ask on forums like Stack Overflow or would take up to an hour of Googling.

comments were standardized for grammar and consistency.

The results show a large degree of concurrence with the literature summarized in the previous section, but there are some interesting additional insights, particularly concerning the developer job market and practical limitations of LLM models.

4.1. Category 1: Developer issues and jobs

This category focuses on the broad effects of LLMs on developer job security. More pessimistic commenters thought that junior developers, who typically undertake simpler coding tasks, would become automated or at least face tougher competition. Other commenters noted that companies still needed a pipeline for senior developers and that automating basic coding functions could risk this. LLMs can be used to enhance the skills of more senior developers and help train more junior developers. Some commenters noted that development has gradually moved from low-level development (assembly language) through to high-level development and no- or low-code environments, and they observed that the use of prompts is just an additional level of abstraction for developers.

- Key takeaways: While GenAI tools may improve efficiency, trying to completely replace junior developers will harm the development pipeline, so there will still be opportunities for junior developers. But all developers must learn new workflows and new skills, such as prompting LLMs.

4.2. Category 2: General coding issues and platform performance

This category contains discussions on the actual utility of the new LLM-based tools. Positive comments noted time savings of 50%, removing repetitive tasks, and usefulness in simple coding tasks and generating documentation. Negative comments noted poor-quality code, a lack of knowledge of proprietary software and application programming interfaces (APIs), time lost in additional testing, and the need for skilled, knowledgeable developers to provide meaningful LLM prompts.

- Key takeaways: Developers should understand how best to utilize LLMs to improve efficiency without compromising quality, and they should learn how to write effective prompts for LLMs. The exact balance will depend on the development environment, developer experience, and the goals of the project (e.g., business reports or mission critical software). Senior developers should mentor junior developers on how to make use of LLMs while maintaining coding quality.

4.3. Category 3: Testing code

This category focuses on the usage of LLM tools for testing. Positive commenters noted that LLMs are good at writing small unit tests (i.e., for individual functions) from documentation; once given some initial tests, they can then autocomplete a comprehensive range of tests. More cautious commenters worried that automating testing and coding could lead to a lack of controls, or that LLMs can get stuck in a feedback loop by basing tests only on previous tests, while others maintained that since LLMs can hallucinate, it is too risky to entrust testing to LLMs.

- Key takeaways: At the current stage of maturity, LLMs should not be entrusted both to create and also test the same code. LLMs can be used to increase the efficiency of testing and documentation (particularly on unit tests), but strong quality controls should be introduced to ensure proper test coverage and accuracy.

4.4. Category 4: Understanding code

This category focuses on the broad topic of understanding code. Some commentators note that when used with appropriate prompts, LLMs can improve developers' understanding of code. There is a general view that when using code generated by LLMs, developers should ensure that they fully understand the code before using it, as a developer committing code is responsible for it, including errors generated by LLMs. Some commentators admit that they do not always fully understand the code that they utilize, whether from an existing code base or generated by LLMs.

- Key takeaways: Understanding code is key to quality software implementations. Code taken piecemeal from LLMs can lead to unexpected functionality and potential security and privacy risks. Any code generated by LLMs should be read through line by line and thoroughly documented. LLMs have strong potential for helping developers understand existing code and can be

utilized to help the learning curve of new developers assigned to existing projects.

4.5. Category 5: Open-source and legal issues

This category focuses on issues with licensing. Commenters note that code used to train LLMs may have licenses that allow free use, but often, the licenses do not allow commercial distribution or using the code without attribution. There have been instances where LLMs have replicated copyrighted code exactly, such as code from the game Quake. This poses a challenge from a risk-management standpoint. Commenters note that the legal concept of fair use is currently untested for coding applications, and it may take lawsuits and court rulings to develop precedence.

- Key takeaways: Any company engaging in software development needs a clear policy on code reuse and copyright. This is true for code snippets taken from Stack Overflow as well as code taken from LLMs. In particular, there should be rules to prevent large chunks of code being used verbatim with comments. The legal situation and any precedent-setting court cases should be monitored carefully.

4.6. Category 6: Stack Overflow and information searches

This category puts the use of LLMs into context with existing methods of searching for information. Multiple comments (over 20 in the full dataset) focused on the general unfriendliness of Stack Overflow participants and the fact that novice programmers were put off from asking questions by potential sarcasm or condescension. Commenters also noted that using LLMs is faster than waiting for answers on Stack Overflow. But some commenters noted that Stack Overflow provides high-quality explanations of the rationale for coding decisions, which were sometimes superior to answers created by LLMs.

- Key takeaways: Developers should be encouraged to utilize a full range of search resources, including internal documentation, IDE help functionality, coworkers, Stack Overflow, and LLMs. All have advantages and disadvantages. Developers could utilize LLMs for syntax and library queries, rely on colleagues for questions about company coding standards, and utilize Stack Overflow for more complex coding issues and queries.

5. A roadmap for developers

The content analysis provides key takeaways for the implementation of LLMs and builds on the summaries of research on the best practices for the initial introduction of LLMs to the development environment. But how can developer LLM usage be evaluated effectively? A commonly used methodology for appraising an organization's current implementation status and maturity level with a development technology is the capability maturity model (CMM). The CMM was originally developed as a methodology for process improvement in software development/engineering at IBM in the 1980s (Paulk, 2009), but models of similar maturity have been deployed in a range of business areas—for example, sourcing innovation (Legenvre & Gualandris, 2018) and human resources (Wademan et al., 2007)—and have already been implemented for general AI technologies (e.g., Sadiq et al., 2021).

The general idea of a maturity model is that in the initial stage, when a new technology or business practice is introduced, this is often done in an ad-hoc manner with little coherent documentation or process. In the repeatable stage, there is some degree of documentation and process control at the project level, but processes are still quite chaotic. At the defined stage, processes are properly documented at the organizational level, and process standards are developed. In the managed stage, process metrics are gathered at an organizational level to guide evaluation, and in the optimized stage, these metrics are used to optimize processes incrementally. This use of metrics and statistics for process improvement is key to the top two maturity stages of most CMM implementations. Different variants of the CMM use different terminology for the stages, but there is some commonality in that, over time, processes go from unmanaged and reactive to managed and proactive. An organization can analyze its processes on different CMM dimensions and then develop a plan for process improvement to reach a higher level of the CMM.

The CMM implemented for this article is given in Table 2. The dimensions chosen were Tools, Coding, Copyright & Security, Testing, and Mentorship. This is not an exact one-to-one mapping to the content analysis, but the dimensions include the main elements of the development process found in the content analysis and literature review. Each of the first five rows of the table contains a description of one of the CMM stages. The last row contains summaries of category discussions

Table 2. A capability maturity model (CMM) for LLM-based software development

	Tools	Coding	Copyright & security	Testing	Mentorship
Initial	Individual developers utilize GenAI tools in an ad-hoc fashion.	There are no rules or policies on the use of LLMs for generating code.	There is no process for checking the copyright of generated code or code that introduces privacy or security issues.	There are no policies for the usage of LLMs in testing.	There is no mentorship for junior developers on appropriate use of LLMs.
Repeatable	All developers have access to specified LLM tools (e.g., ChatGPT, GitHub Copilot).	There is awareness of best practices for the use of LLMs, which is relayed to developers.	There are basic rules for code usage, but potential copyright violations and security breaches are not tracked.	There is some usage of LLMs for testing, and LLMs are also in creating tests.	There is ad-hoc knowledge transfer between junior and senior developers on best practices for using LLMs.
Defined	Developers have access to standardized tools, along with documentation on how to utilize the LLMs for different tasks.	There are rules governing when to use LLMs for coding and procedures for ensuring that rules are enforced.	There are documentation and rules for avoiding potential copyright violations and privacy or security issues.	There is documentation on how to utilize LLMs for unit testing and quality standards to ensure full test coverage.	There is organized mentoring for junior developers to ensure appropriate use of LLMs.
Managed	Developers have access to LLM tools, along with usage statistics, which can be used to help adapt these tools and the IDE for optimal usage.	Use of LLMs in coding, along with developer statistics, (time taken for tasks, errors, etc.) is tracked. Coders can utilize these statistics to improve efficiency.	There are documentation and audits of LLM code usage, to ensure that large chunks of code or potentially copyrighted algorithms are avoided, and that privacy and security issues do not occur.	LLMs are fully incorporated into the testing process. Tests and results are fully documented, with statistics available on the efficacy of LLM coding and LLM testing.	Statistics are made available on developer productivity and coding quality. Mentors use these statistics to help advise junior developers.
Optimized	Using statistics and developer feedback, LLM tools and IDE features are optimized to improve coding quality and productivity.	Developer metrics of LLM usage are used to understand optimal LLM usage and to improve company-wide processes for usage of LLMs.	LLM usage statistics, along with records of potential copyright, privacy, and security breaches from LLM use, are used to proactively alter processes.	Statistics on developer productivity and coding quality are made available. Mentors use these statistics to help advise junior developers.	Metrics should be used to help individual junior developers improve their development processes and training procedures for the use of LLMs.

Table 2 (*continued*)

	Tools	Coding	Copyright & security	Testing	Mentorship
Content-analysis insights	Comments in C2 and C4 indicate that many developers have access to LLM tools. However, most developers utilize LLMs in an ad-hoc manner to support specific coding tasks.	Discussions in C4 on how developers adapt code to coding standards and use LLMs to understand code. C2 contains usage scenarios and limitations.	The comments in C5 show that developers understand licensing issues and that LLMs can regurgitate copyrighted code. But there is little discussion of specific corporate rules, policies, and processes.	Discussion in C3 gives ideas on how to test with LLMs in different testing scenarios (e.g., unit tests), along with limitations and risks (e.g., completely automated coding and testing).	Discussions in C1 on the importance of training new developers and on the dangers of using LLMs without knowing how to evaluate code.

relating to the maturity levels of current LLM-based development efforts. Unsurprisingly, given the recent introduction of LLM-based coding tools, much of the implementation of these tools is still very much ad-hoc. Many of the content-analysis discussions focused on trying out LLMs for some aspect (e.g., coding, testing, or documentation) of a developer's workflow and then evaluating the performance of LLM-based development as opposed to traditional development. In these discussions, amid concerns about best practices for coding, testing, and training, one can see the emergence of more repeatable and defined processes.

While this CMM framework should provide developers with a roadmap to measure and improve LLM processes, it is not intended to be inflexible or overly prescriptive. As described previously, the development world is a broad church. An organization developing mission-critical software for a power station or an airline may initially want to restrict LLM usage to purely informational usage—for example, to help developers understand an existing code base. An organization developing custom macros for business or sales automation software, on the other hand, may take a more aggressive approach to LLM usage, as this area has lower barriers to entry than mission-critical software.

6. Remarks for developers

Software developers have many different job functions. Besides coding, job functions include working with clients to build requirements, testing software (both unit and system testing), and integrating systems with other systems and data sources. Application development requires strong domain knowledge that must be built over time. Having rapid access to a large amount of knowledge and being able to query this knowledge efficiently is of great use to developers. To use a computing analogy, developers only have a small portion of the knowledge required to complete their jobs stored in accessible memory, and LLMs provide a rapid interface to the remaining knowledge in long-term memory (i.e., the internet, code repositories, documentation). Software development has gone through a process of abstraction over the last 50 years, moving from assembly language to compiled languages, such as Fortran or C/C++, through interpreted languages, such as Python and Java, to templated, no- or low-code solutions (e.g., Sundberg & Holmström, 2023). The use of prompting with LLMs is another step on this path, and developers need to adapt to this environment to prosper.

LLMs are here to stay. With LLMs, productivity requirements for developers may increase, so developers need to learn how to integrate LLMs into their workflows to improve efficiency and keep up with other developers. This may be achieved by benchmarking different strategies—for example, by using LLMs just for quickly finding syntax (and not for more complex tasks) or by supplementing them with human testing and commenting, and by testing these strategies in the types of coding challenges discussed in the literature review. LLM skills are often complementary to non-LLM skills. For example, there are cases where Stack Overflow queries may receive insightful answers from

expert coders, but for simple syntax queries, LLMs provide quicker answers and avoid some of the problematic aspects of human interaction noted by commenters in the content analysis.

Many commenters in the content analysis stated that LLMs will sometimes hallucinate code, so it is a good idea to understand how well different types of prompts perform for different aspects of the specific programming language being used. For example, commenters noted that LLMs work well with Python and Java, so it may be that complex queries in these languages have a high rate of success, which may not be true for all languages. But developers need to be able to work without LLMs. Security issues in certain fields (e.g., defense), or well-publicized copyright issues or lawsuits (such as the ongoing lawsuit against OpenAI discussed in the content-analysis comments), may cause risk-averse corporations to bar LLMs, an event mentioned by several commenters in the content analysis, so developers wishing to maximize opportunities must still work on improving their traditional coding and non-LLM search skills.

Prompting skills are important. Prompting can be used for both generating code and for queries to understand code. Multiple commentators in the content analysis noted the importance of good prompts, and several speculated that prompt engineer will soon become a developer job title. In fact, courses and documentation are now available for prompt engineering. A coder's exact prompting style will depend on the particular LLM and the language and domain of the coding problem, but it is likely that skills can be built through trying queries at different levels of abstraction and carefully examining results. In a similar fashion to the Google algorithm for search-engine optimization, it is likely that the optimal method of prompting will change with updates to the LLM algorithms, so developers will need to conduct constant testing and stay abreast of updated training materials to keep their prompting skills honed.

7. Concluding comments

Overall, GenAI and LLMs have already had a large effect on development practices and workflows. While commenters in the content analysis displayed some unease about the future of development as LLMs become more widespread, it is unlikely that software development will be destroyed as a profession. However, developers do need to adapt to this new environment. They should understand the potential of LLMs and be able to implement them to improve coding efficiency while preserving quality and accounting for potential copyright and security issues. They can achieve this by implementing and improving processes and standards for LLM usage, using frameworks such as the CMM.

Developers need to supplement traditional coding skills with skills in prompting LLMs to aid in the creation of code, documentation, and tests. As several commenters in the content analysis noted, software developers will still be needed until the advent of completely sentient AI, a hypothesized event that will transform society, and which is beyond the scope of this analysis.

References

Ardimento, P., Bernardi, M. L., Cimitile, M., Redavid, D., & Ferilli, S. (2022). Understanding coding behavior: An incremental process mining approach. *Electronics, 11*(3), 389.

Badampudi, D., Unterkalmsteiner, M., & Britto, R. (2023). Modern code reviews—survey of literature and practice. *ACM Transactions on Software Engineering and Methodology, 32*(4), 1–61.

Bird, C., Ford, D., Zimmermann, T., Forsgren, N., Kalliamvakou, E., Lowdermilk, T., & Gazit, I. (2022). Taking flight with copilot: Early insights and opportunities of AI-powered pair-programming tools. *Queue, 20*(6), 35–57.

Bureau of Labor Statistics. (2022, September 8). *Occupational outlook handbook: Computer and information technology occupations*. Available at https://www.bls.gov/ooh/computer-and-information-technology/home.htm

Chatravorti, B. (2023, June 25). How will AI change work? A look back at the 'productivity paradox' of the computer age shows it won't be so simple. *Fortune*. Available at https://fortune.com/2023/06/25/ai-effect-jobs-remote-work-productivity-paradox-computers-iphone-chatgpt/

Ciborowska, A., Kraft, N. A., & Damevski, K. (2018, May). Detecting and characterizing developer behavior following opportunistic reuse of code snippets from the web. In *Proceedings of the 15th International Conference on Mining Software Repositories* (pp. 94–97). New York, NY: Association for Computing Machinery.

Davenport, T. H., Barkin, I., & Tomak, K. (2023). We're all programmers now. *Harvard Business Review, 101*(5), 98–107.

Feng, Y., Vanam, S., Cherukupally, M., Zheng, W., Qiu, M., & Chen, H. (2023). Investigating code generation performance of Chat-GPT with crowdsourcing social data. In *Proceedings of the 47th IEEE Computer Software and Applications Conference* (pp. 876–885). Piscataway, NJ: Institute of Electrical and Electronics Engineers.

Gershgorn, D. (2021, June 29). GitHub and OpenAI launch a new AI tool that generates its own code. *Verge*. Available at https://www.theverge.com/2021/6/29/22555777/github-openai-ai-tool-autocomplete-code

Hess, A. J. (2023, February 16). Ranking workers can hurt morale and productivity. Tech companies are doing it anyway. *Fast Company*. Available at https://www.fastcompany.com/90850190/stack-ranking-workers-hurt-morale-productivity-tech-companies

Horne, J., Recker, M., Michelfelder, I., Jay, J., & Kratzer, J. (2020). Exploring entrepreneurship related to the sustainable development goals-mapping new venture activities

with semi-automated content analysis. *Journal of Cleaner Production, 242*, 118052.

Hughes, A. (2023, September 25). ChatGPT: Everything you need to know about OpenAI's GPT-4 tool. *BBC*. Available at https://www.sciencefocus.com/future-technology/gpt-3/

Imai, S. (2022, May). Is GitHub Copilot a substitute for human pair-programming? An empirical study. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering* (pp. 319—321). New York, NY: Association for Computing Machinery.

Kumar, R., Hasteer, N., & Van Belle, J. P. (2018, January). Evaluating factors influencing contestant behavior in competitive software development. In *2018 8th International Conference on Cloud Computing, Data Science, and Engineering* (pp. 20—25). Piscataway, NJ: Institute of Electrical and Electronics Engineers.

Legenvre, H., & Gualandris, J. (2018). Innovation sourcing excellence: Three purchasing capabilities for success. *Business Horizons, 61*(1), 95—106.

Li, A., Endres, M., & Weimer, W. (2022, May). Debugging with Stack Overflow: Web search behavior in novice and expert programmers. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering* (pp. 69—81). New York, NY: Association for Computing Machinery.

MacRae, D. (2023, May 16). A quarter of tech firms use generative AI for software development. *Developer*. Available at https://www.developer-tech.com/news/2023/may/16/a-quarter-of-tech-firms-use-generative-ai-for-software-development/

Marr, B. (2023, May 19). A short history of ChatGPT: How we got to where we are today. *Forbes*. Available at https://www.forbes.com/sites/bernardmarr/2023/05/19/a-short-history-of-chatgpt-how-we-got-to-where-we-are-today/?sh=299ae6a4674f

Meyer, A. N., Murphy, G. C., Zimmermann, T., & Fritz, T. (2021). Enabling good work habits in software developers through reflective goal-setting. *IEEE Transactions on Software Engineering, 47*(9), 1872—1885.

Nguyen, N., & Nadi, S. (2022, May). An empirical evaluation of GitHub Copilot's code suggestions. In *Proceedings of the 19th International Conference on Mining Software Repositories* (pp. 1—5). New York, NY: Association for Computing Machinery.

Noll, B., Korakitis, K., Solodkov, N., Dodd, L., & Muir, D. (2023, May 1). *State of the Developer Nation 24th Edition - Q1 2023*. Available at https://www.developernation.net/resources/reports/state-of-the-developer-nation-24th-edition-q1-2023

Paulk, M. C. (2009). A history of the capability maturity model for software. *ASQ Software Quality Professional, 12*(1), 5—19.

Pearce, H., Ahmad, B., Tan, B., Dolan-Gavitt, B., & Karri, R. (2022, May). Asleep at the keyboard? Assessing the security of GitHub Copilot's code contributions. In *2022 IEEE Symposium on Security and Privacy* (pp. 754—768). Piscataway, NJ: Institute of Electrical and Electronics Engineers.

Peng, S., Kalliamvakou, E., Cihon, P., & Demirer, M. (2023). The impact of AI on developer productivity: Evidence from GitHub Copilot. *arXiv*. Available at https://arxiv.org/abs/2302.06590

Robillard, M. P., Coelho, W., & Murphy, G. C. (2004). How effective developers investigate source code: An exploratory study. *IEEE Transactions on Software Engineering, 30*(12), 889—903.

Sadiq, R. B., Safie, N., Abd Rahman, A. H., & Goudarzi, S. (2021). Artificial intelligence maturity model: A systematic literature review. *PeerJ Computer Science, 7*, e661.

Sadowski, C., Söderberg, E., Church, L., Sipko, M., & Bacchelli, A. (2018, May). Modern code review: A case study at Google. In *Proceedings of the 40th International Conference on Software Engineering* (pp. 181—190). New York, NY: Association for Computing Machinery.

Sloyan, T. (2021, June 8). Is there a developer shortage? Yes, but the problem is more complicated than it looks. *Forbes*. Available at https://www.forbes.com/sites/forbestechcouncil/2021/06/08/is-there-a-developer-shortage-yes-but-the-problem-is-more-complicated-than-it-looks

Sommers, J. (2023, June 13). How to create with AI, including art, music, and writing, according to people who've written songs, stories, and letters. *Business Insider*. Available at https://www.businessinsider.com/how-to-create-with-ai-art-music-writing-chatgpt-dall-e-2-2023-6

Sundberg, L., & Holmström, J. (2023). Democratizing artificial intelligence: How no-code AI can leverage machine learning operations. *Business Horizons, 66*(6), 777—788.

Trueman, C. (2023, June 19). Tech layoffs in 2023: A timeline. *Computerworld*. Available at https://www.computerworld.com/article/3685936/tech-layoffs-in-2023-a-timeline.html

Vincent, J. (2023, June 13). Stack Overflow survey finds developers are ready to use AI tools — Even if they don't fully trust them. *Verge*. Available at https://www.theverge.com/2023/6/13/23759101/stack-overflow-developers-survey-ai-coding-tools-moderators-strike

Wademan, M. R., Spuches, C. M., & Doughty, P. L. (2007). The people capability maturity model. *Performance Improvement Quarterly, 20*(1), 97—123.

Xia, X., Bao, L., Lo, D., Kochhar, P. S., Hassan, A. E., & Xing, Z. (2017). What do developers search for on the web? *Empirical Software Engineering, 22*, 3149—3185.

Zhang, B., Liang, P., Zhou, X., Ahmad, A., & Waseem, M. (2023). Practices and challenges of using GitHub Copilot: An empirical study. *arXiv*. Available at https://arxiv.org/abs/2303.08733