

Beyond Coding: How Large Language Models Are Redefining the Role of Software Developer

Swati Akhare*¹, Aditi Gawande*², Faiza Patel *³, Nandini Kamble*⁴, Om Shende*⁵, Pavan Shinde*⁶

¹Ast. Prof, ^{2,3,4,5,6} Students, Department of Computer Science and Engineering,

R. V. Parankar College of Engineering and Technology.

Abstract:

Large Language Models (LLMs) are changing the way software is designed, developed, tested, documented, and maintained. Earlier, software developers spent a large amount of time writing code line by line, searching for solutions, debugging errors, writing tests, and preparing documentation. Today, tools based on LLMs can generate code, explain programs, find possible bugs, write test cases, create documentation, and assist with software planning. This change does not simply mean that software developers will disappear. Instead, it is changing what developers do and which skills are most important. Developers are increasingly becoming problem solvers, system designers, reviewers, AI collaborators, and decision makers. The ability to understand a problem, give clear instructions to AI, check its output, design reliable systems, and protect software from security problems is becoming increasingly important.

Recent developer data shows that AI tools are already widely used. The 2025 Stack Overflow Developer Survey reported that 84% of respondents were using or planning to use AI tools in their development process, while 51% of professional developers reported using them daily. At the same time, trust remains limited: 46% of developers said they distrust the accuracy of AI tools, compared with 33% who trust them.

Research on productivity also presents a mixed picture. A controlled GitHub study found that developers using GitHub Copilot completed a programming task 55.8% faster, while later research by METR found that experienced open-source developers using early-2025 AI tools took 19% longer on their selected real-world tasks. METR's later 2026 update suggests that newer AI tools were probably providing more speedup, but also explains that its newer experimental data was affected by selection problems and should not be treated as a precise estimate.

This paper studies how LLMs are redefining the software developer's role, the changes across the software development life cycle, the new skills developers need, the benefits and risks of AI-assisted development, and the future relationship between humans and AI in software engineering. The central argument of this paper is that the future developer is not only a coder but a person who can combine programming knowledge, software engineering judgment, AI skills, communication, security awareness, and human decision-making.

Keywords— Large Language Models, Software Development, Generative AI, AI Coding Tools, Software Engineering, Developer Productivity, AI-Assisted Programming, AI Agents, Software Testing, Future of Developers

1. Background

Large Language Models are artificial intelligence systems trained on large amounts of text and programming data. They can understand questions, generate explanations, summarize information, and produce computer programs. Modern LLMs can work with several programming languages, including Python, Java, JavaScript, C++, SQL, and many others.

In traditional software development, programmers had to write most of the code manually. They also spent considerable time searching for solutions, reading documentation, understanding error messages, fixing bugs, preparing test cases, and writing project documentation. These activities are necessary, but many of them are repetitive and time-consuming.

With the development of LLM-based tools, several of these tasks can now be completed with AI assistance. A developer can describe a requirement in natural language, and the AI tool can generate a possible implementation. The developer can then modify, test, and improve the generated result.

For example, a developer can ask an AI tool to:

- create a login page,
- write an API,
- explain an error message,
- generate SQL queries,
- create test cases,
- convert code from one programming language to another,
- explain an existing program,
- improve the performance of a function,
- create a database schema,
- or prepare a project README file.

LLMs can also help organize information, plan tasks, and work with external tools. Some advanced AI

systems can divide a large task into smaller steps, use software tools, inspect files, and provide a complete solution. This makes them useful not only for writing code but also for supporting different

stages of software development.

The use of LLMs has created a major change in the responsibilities of developers. Earlier, a developer's main responsibility was to write code according to the requirements. Today, developers must also know how to communicate with AI tools, write clear prompts, evaluate generated code, identify errors, and

decide whether the solution is suitable.

This means that programming knowledge is still important. In fact, it may become even more important

because developers need enough technical understanding to recognize when AI-generated code is

incorrect or unsafe. AI can produce a confident answer even when the answer is wrong. Therefore, human judgment remains necessary.

1.1 Changing Role of the Developer

The role of a software developer is expanding in several directions:

- **Problem understanding:** Developers must understand the real problem before asking AI to create a solution.
- **Prompt writing:** Developers need to give clear and detailed instructions to AI tools.
- **Code review:** Developers must inspect AI-generated code for correctness, readability, and security.
- **Testing:** Developers must test the output using different inputs and real project conditions.
- **System design:** Developers must choose suitable technologies and design the overall system.
- **Security management:** Developers must identify possible vulnerabilities in AI-generated solutions.
- **Communication:** Developers must explain technical decisions to team members and users.

ensure that the software is fair, safe, and used

- **Ethical responsibility:** Developers must

responsibly.

Thus, the developer is becoming less focused on typing every line of code and more focused on controlling the complete software development process.

2. Methodology

This research is based on a review of existing information about Large Language Models and their use

in software development.

The study considers:

- research papers on AI agents and LLMs,
- studies on AI-assisted programming,
- developer surveys,
- industry reports,
- documentation from AI coding platforms,
- examples of AI use in software projects,
- and information from software development communities.

The research focuses on how LLMs support six important areas of software development:

coding, debugging, testing, documentation, software design, and maintenance.

The information was compared with traditional software development methods. This comparison helps

explain how AI changes the daily responsibilities of developers and how human and AI-based work can be combined.

2.1 Areas Considered in the Study

The study examines the following areas:

Productivity

The research considers whether LLMs help developers complete tasks faster and reduce repetitive work.

Code Quality

The study examines whether AI-generated code is readable, maintainable, and suitable for the intended purpose.

Learning and Skill Development

LLMs can explain programming concepts and provide examples. However, excessive dependence on AI may reduce independent learning and problem-solving ability.

Security and Privacy

The study considers the possibility of insecure code and the risks of sharing private project information with external AI systems.

Human Responsibility

The research examines why developers must continue to review, test, and approve AI-generated results.

2.2 Human-AI Development Process

A common AI-assisted development process includes the following steps:

1. The developer understands the problem.
2. The developer writes a clear prompt or requirement.
3. The AI generates code or suggestions.
4. The developer reviews the output.
5. The code is tested using normal and unusual inputs.
6. Errors are corrected through further prompts or manual changes.
7. The final solution is integrated into the project.
8. The software is monitored and maintained.

This process shows that AI is part of the development workflow, but it does not independently guarantee a correct final product.

3. Comparative Analysis

AI-assisted development is different from traditional development in several ways.

Traditional Development	AI-Assisted Development
Developer writes most code manually	AI can generate part of the code
Developer searches for solutions	AI can suggest solutions quickly
Testing is mostly created manually	AI can generate test cases
Documentation is written manually	AI can create first drafts
Debugging takes more manual effort	AI can suggest possible causes
Developer focuses heavily on coding	Developer focuses more on coding, review, design, and decisions
Knowledge is mainly obtained from books and documentation	Knowledge can also be obtained through interactive AI explanations
Repetitive tasks require more time	Repetitive tasks can be automated or accelerated
The developer controls every line from the beginning	The developer may review and modify AI-generated code

AI can make many tasks faster, especially repetitive and well-defined tasks. For example, generating a basic function or writing a simple test may take only a few seconds with the help of an AI tool.

However, AI-generated code is not always correct. It may contain logical errors, security problems, inefficient algorithms, or solutions that do not match the project requirements. It may also use outdated libraries or incorrect programming practices.

Therefore, developers must understand the code, test

it properly, and make changes when necessary.

The use of AI does not remove the need for software engineering principles such as modular design, version control, testing, documentation, code review, and security analysis.

3.1 Advantages of AI-Assisted Development

Faster Development

AI can quickly generate common code structures, reducing the time required to begin a task.

Reduced Repetitive Work

Developers can use AI for repeated tasks such as creating similar functions, writing basic tests, or preparing documentation.

Better Access to Knowledge

AI can explain programming concepts in simple language and provide examples based on the developer's question.

Support for Beginners

Students and new developers can use AI to understand errors, learn syntax, and explore different programming approaches.

Improved Documentation

AI can create initial drafts of comments, technical explanations, and project documentation.

Assistance with Legacy Systems

AI can help explain old code that may not have proper documentation. This can support maintenance and modernization work.

3.2 Disadvantages of AI-Assisted Development

Possibility of Incorrect Results

AI may generate code that appears professional but contains hidden errors.

Reduced Independent Thinking

Developers may accept AI suggestions without understanding them.

Security Risks

AI-generated code may include unsafe practices or

vulnerabilities.

Inconsistent Coding Style

Different AI responses may use different naming conventions, structures, or programming styles.

Difficulty in Handling Complex Requirements

AI may not fully understand business rules, organizational policies, or long-term project goals.

Privacy Concerns

Sensitive source code or confidential information may be exposed if it is entered into an external AI system.

3.3 Development Process Comparison

The main difference is that traditional development depends mainly on manual coding, while AI-assisted development combines human knowledge with AI support.

The change can be represented as:

Traditional approach:

Problem → Human writes code → Human tests → Final software

AI-assisted approach:

Problem → Human explains requirements → AI generates suggestions → Human reviews and improves code → Testing → Final software

In this new approach, the developer remains responsible for the quality and safety of the final software.

AI can provide suggestions, but it cannot fully understand every real-world requirement or accept responsibility for the result.

4. Use Cases

LLMs can support software developers in many common tasks.

4.1 Code Generation

AI can generate functions, classes, APIs, database queries, user interface components, and other code from simple instructions. This is especially useful for creating basic program structures and reducing repetitive coding work.

For example, a developer can request a Python function that reads a CSV file, removes duplicate records, and returns the cleaned data. The AI can provide an initial implementation that the developer can review and improve.

4.2 Debugging

Developers can provide an error message or a section of code to an AI tool. The tool can suggest possible causes and recommend solutions.

AI can help with:

- syntax errors,
- runtime errors,
- incorrect output,
- missing dependencies,
- configuration problems,
- database connection errors,
- and performance issues.

For example, if a program produces a null pointer error, the AI may explain where the error could occur

and suggest adding validation. The developer must still verify the explanation because the AI may misunderstand the actual program flow.

4.3 Testing

AI can generate unit tests, integration tests, test cases, and possible edge cases. This helps developers

identify situations that may not be considered during manual testing.

AI can suggest tests for:

- empty input,
- very large input,
- invalid values,
- missing fields,
- duplicate records,
- incorrect user permissions,

- network failures,
- and unexpected system behavior.

AI can also help create test data and explain why a particular test is important. However, developers must ensure that the tests actually check the required behavior. A test generated by AI may pass without properly verifying the result.

4.4 Documentation

AI can create comments, README files, API descriptions, installation instructions, and simple explanations of code.

Good documentation helps:

- new developers understand the project,
- users install and operate the software,
- teams maintain the system,
- and organizations preserve technical knowledge.

AI-generated documentation should be checked for accuracy. If the code changes, the documentation must also be updated.

4.5 Code Explanation

AI can explain difficult or unfamiliar code in simple language. This is useful for students, beginners, and developers working with old or poorly documented projects.

A developer can ask the AI to:

- explain a function line by line,
- summarize a large file,
- identify the purpose of a class,
- describe the flow of data,
- or explain how different modules are connected.

This can reduce the time required to understand an unfamiliar codebase. However, the explanation should be compared with the actual code because AI may misunderstand complex logic.

4.6 Software Design

AI can help developers compare programming languages, frameworks, databases, and possible

system

designs. It can provide suggestions based on factors such as performance, cost, scalability, and ease of maintenance.

For example, AI can help compare:

- relational databases and NoSQL databases,
- monolithic and microservice architectures,
- REST APIs and GraphQL,
- cloud deployment options,
- or different frontend frameworks.

AI can provide useful alternatives, but the final design decision should be made by the developer after

considering project requirements, budget, team skills, security, and future growth.

4.7 Maintenance

LLMs can help update old code, identify repeated code, suggest improvements, and assist developers in

understanding large software projects.

Maintenance activities supported by AI include:

- refactoring code,
- updating outdated syntax,
- identifying unused functions,
- improving variable names,
- converting code to a newer framework,
- finding repeated logic,
- and preparing change summaries.

AI can make maintenance more organized and efficient, but changes to existing systems must be made

carefully because even a small modification can affect other parts of the application.

4.8 Learning and Training

LLMs can act as interactive learning assistants.

Students can ask questions about programming concepts, algorithms, databases, operating systems, and software engineering.

AI can provide:

- simple explanations,

- programming examples,
- practice questions,
- project ideas,
- debugging exercises,
- and comparisons between technologies.

However, students should use AI to understand concepts rather than simply copy answers. Independent practice and problem-solving remain essential for developing strong technical skills.

4.9 Project Management

LLMs can also support software project management. They can help prepare task lists, summarize meetings, create user stories, and organize project requirements.

For example, an AI tool can convert a general requirement into smaller tasks such as:

- designing the database,
- creating the backend API,
- developing the user interface,
- writing tests,
- preparing documentation,
- and deploying the application.

This can help teams plan their work more clearly. Human project managers must still verify the priorities, deadlines, and responsibilities.

5. Challenges

Although LLMs provide many benefits, they also have important limitations.

5.1 Incorrect Code

AI can generate code that looks correct but does not work properly. It may also provide outdated or unsuitable solutions.

Common problems include:

- incorrect function names,
- missing imports,
- wrong data types,
- incomplete error handling,
- incorrect assumptions,

and inefficient algorithms.

Developers must compile, test, and review the code before using it.

5.2 Security Problems

AI-generated code may contain security weaknesses such as poor input validation, unsafe database queries, weak authentication, insecure file handling, or improper access control.

For example, an AI-generated SQL query may be vulnerable to SQL injection if user input is directly included in the query. Developers must follow secure coding practices and use security testing tools.

Important security checks include:

- input validation,
- authentication,
- authorization,
- encryption,
- secure password storage,
- protection against injection attacks,
- and safe handling of user data.

5.3 Over-Dependence

If developers depend completely on AI, they may stop improving their own programming and problem-solving skills.

Over-dependence can lead to:

- poor understanding of basic concepts,
- difficulty debugging without AI,
- inability to design systems independently,
- and acceptance of incorrect solutions.

AI should support learning and development, not replace understanding. Developers should first try to understand the problem and then use AI as an assistant.

5.4 Lack of Full Context

AI may not understand the complete requirements, business logic, or long-term goals of a large

software project.

A short prompt may not include important information such as:

- user roles,
- performance requirements,
- legal restrictions,
- existing system limitations,
- security policies,
- or future expansion plans.

Developers must provide clear instructions and review the output according to the complete project context.

5.5 Privacy

Developers may work with private source code, passwords, customer information, financial records, or company data. Such information should not be shared with AI tools without following proper privacy and security policies.

Organizations should define rules about:

- what information can be entered into AI tools,
- how data is stored,
- who can access AI conversations,
- and whether the information is used for model training.

Sensitive information should be removed or protected before using an external AI service.

5.6 Quality and Consistency

Different prompts may produce different results. AI-generated code may also use different styles or approaches within the same project.

This can create problems such as:

- inconsistent naming,
- repeated code,
- different error-handling methods,
- unnecessary complexity,
- and difficulty maintaining the project.

Developers need to maintain coding standards, use code review, and ensure consistency across the

software.

5.7 Hallucinations

LLMs may produce information that sounds correct but is not true. This is often called a hallucination. In software development, hallucinations may include:

- imaginary libraries,
- incorrect API methods,
- false documentation links,
- unsupported programming features,
- or explanations that do not match the actual code.

Developers must verify important information using official documentation and reliable sources.

5.8 Copyright and Licensing

AI-generated code may resemble code from existing public or private sources. This creates questions about copyright, licensing, and ownership.

Developers and organizations should check:

- whether external code has license restrictions,
- whether attribution is required,
- whether the code can be used commercially,
- and whether the generated solution creates legal risks.

Clear organizational policies are needed for the responsible use of AI-generated code.

5.9 Bias and Fairness

AI systems learn from large datasets that may contain errors or biases. These biases can affect the suggestions provided by the system.

For example, AI may produce unfair assumptions in software related to hiring, finance, education, or healthcare. Developers must test systems for fairness and avoid using AI without proper human evaluation.

5.10 Human Responsibility

AI can suggest a solution, but humans are responsible for deciding whether that solution is safe, correct, ethical, and suitable.

Human monitoring is especially important in critical systems such as:

- Banking,
- healthcare,
- transportation,
- education,
- government services,
- and industrial control systems.

In these areas, an incorrect software decision can cause serious harm. Developers must follow professional standards and ensure that final decisions are made responsibly.

5.11 Difficulty Measuring Productivity

Although AI can make individual tasks faster, faster coding does not always mean better software.

A developer may generate code quickly, but the project may still require time for:

- reviewing the code,
- fixing errors,
- testing,
- security analysis,
- integration,
- deployment,
- and maintenance.

Therefore, productivity should be measured by the quality and usefulness of the final software, not only by the number of lines generated.

6. Conclusion

Large Language Models are changing the role of software developers. Developers can now use AI to generate code, find errors, create tests, explain programs, prepare documentation, support software design, and assist with maintenance.

The use of LLMs can improve productivity and reduce repetitive work. It can also help beginners learn

programming and allow experienced developers to focus more on system design, problem-solving, and innovation.

However, LLMs are not perfect. They may generate incorrect code, security vulnerabilities, biased suggestions, outdated information, or solutions that do not match the project requirements. Privacy, copyright, and accountability are also important concerns.

This does not mean that programming knowledge is no longer important. Instead, developers now need a wider set of skills. They must understand programming, software design, testing, security, data privacy, project requirements, and how to use AI effectively.

Developers must also learn how to write clear prompts, verify AI responses, review generated code, and

use reliable testing methods. The ability to judge the quality of an AI-generated solution will become an important professional skill.

The future developer will not only be a code writer.

The developer will also be a problem solver, designer, reviewer, tester, project contributor, and AI collaborator.

The future of software development can be summarized as:

AI does the repetitive work, while developers focus on understanding problems, making decisions, ensuring quality, and creating better software.

LLMs are not simply changing how code is written—they are changing what it means to be a software

developer. The most successful developers will be those who combine technical knowledge with creativity, critical thinking, communica

7 References

Research Papers and Academic Sources

[1] S. Peng, E. Kalliamvakou, P. Cihon, and M. Demirer, "The Impact of AI on Developer Productivity:

Evidence from GitHub Copilot," arXiv, 2023.

<https://arxiv.org/abs/2302.06590>

[2] J. Becker, N. Rush, B. Barnes, and D. Rein, "Measuring the Impact of Early-2025 AI on Experienced

Open-Source Developer Productivity," METR, 2025.

<https://metr.org/blog/2025-07-10-early-2025-ai-experienced-os-dev-study/>

[3] J. Sauvola, S. Tarkoma, M. Klemettinen, J. Riekki, D. Doermann, et al., "Future of Software Development with Generative AI," Automated Software Engineering, vol. 31, 2024.

<https://link.springer.com/article/10.1007/s10515-024-00426-z>

[4] A. Nguyen-Duc, P. Abrahamsson, and F. Khomh, eds., Generative AI for Effective Software Development, Springer, 2024.

<https://link.springer.com/book/10.1007/978-3-031-55642-5>

Books

[5] I. Sommerville, Software Engineering, 10th ed., Pearson, 2015.

ISBN: 978-0133943030.

[6] S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 4th ed., Pearson, 2021.

ISBN: 978-0137505135.

<https://www.pearson.com/en-us/subject-catalog/p/artificial-intelligence-a-modernapproach/P200000003500/9780137505135>

[7] D. Thomas and A. Hunt, The Pragmatic Programmer: Your Journey to Mastery, 20th Anniversary ed.,

Addison-Wesley, 2019.

ISBN: 978-0135957059.

[8] B. Shneiderman, Human-Centered AI, Oxford University Press, 2022.

<https://academic.oup.com/book/41126>

Official Documentation

[9] GitHub, GitHub Copilot Documentation Inline Suggestions and Responsible Use, GitHub Docs.

<https://docs.github.com/en/copilot/responsible-use/inline-suggestions>

GitHub's documentation explains that Copilot can generate code suggestions and edits, while also emphasizing that developers should review and validate AI-generated output.

[10] Amazon Web Services, Amazon Q Developer Documentation, AWS Documentation.

<https://docs.aws.amazon.com/amazonq/>

[11] OpenAI, OpenAI API Documentation, OpenAI Platform Documentation.

<https://platform.openai.com/docs/>

[12] Google Cloud, Gemini Code Assist Documentation, Google Cloud Documentation.

<https://cloud.google.com/gemini/docs/codeassist>
Developer Surveys and Industry Sources

[13] Stack Overflow, 2025 Developer Survey, Stack Overflow, 2025.

<https://survey.stackoverflow.co/2025/>

The survey provides broader information about programming practices, developer learning, technologies, and the growing role of AI in software development.