workable | hackerearth

AI-dentifying Your Top Tech Talent:

Evaluating coders and developers in the age of AI

Introduction

In the rapidly evolving landscape of technology, the impact of artificial intelligence (AI) on developer workflows has been nothing short of transformative.

From automating tedious tasks (like writing documentation and test cases) to optimizing code with a click of a button, AI has revolutionized how developers work, enhancing productivity and opening new possibilities.

Recent data and statistics attest to AI's significant influence on the tech industry, and it continues to reshape various aspects of the software development lifecycle.

Generative AI can potentially accelerate the software development life cycle, unlocking greater productivity. Workers spend about 40% of their time engaged in manual and repetitive tasks, equating to nearly two full work days each week.

And current estimates say that 50% of manual software development tasks can be automated, doubling productivity.

The integration of AI-powered solutions can optimize manual code development processes, freeing up developer bandwidth to focus more on high-level tasks, problem-solving, and innovation.

How do you evaluate candidates now?

The impact of AI is particularly pronounced, and also impacts how tech candidates are evaluated. Traditional methods of assessing technical skills during hiring often involve time-consuming and subjective evaluations.

However, with the advent of generative AI, the potential for transforming tech skill evaluation has grown exponentially.

HackerEarth CTO Vishwastam Shukla says:

“Generative AI has split the tech candidate evaluation right down the middle. While objective, standardized and unbiased assessments were the hallmark of assessment products, the mode of delivering this value will differ. “Assessments conducted with AI-assisted environments vs. those conducted without AI will convey different skill signals about the candidates. Discerning employers will choose both forms of evaluation to hire the best candidates."

And Panos Korros, the SVP of Engineering at Workable, highlights the importance of hiring tech candidates who can walk the line between AI and human-contributed work:

Panos Korros
SVP Engineering - Workable

“We want good engineers and people that can do the job. Knowing how to take advantage of AI to increase their personal productivity is a positive thing for us and something that we want our engineers to do. There is a lot of mundane work that AI can take care of, and the engineer can focus on what is important.”

The use of generative AI in tech skill evaluation benefits organizations and provides candidates with a fairer and more transparent assessment process. It allows candidates to showcase their skills and potential without the limitations of traditional evaluation methods, leading to a more inclusive hiring environment.

Let's explore the various applications of generative AI in tech hiring, interviews, and overall tech management in this new time.

The role of generative AI in coding

With their ability to analyze patterns, learn from vast amounts of data, and generate human-like responses, generative AI tools can play a big role in how code is generated, debugged, and optimized.

Let's look at how AI can factor into the work itself:

1. AI in code generation

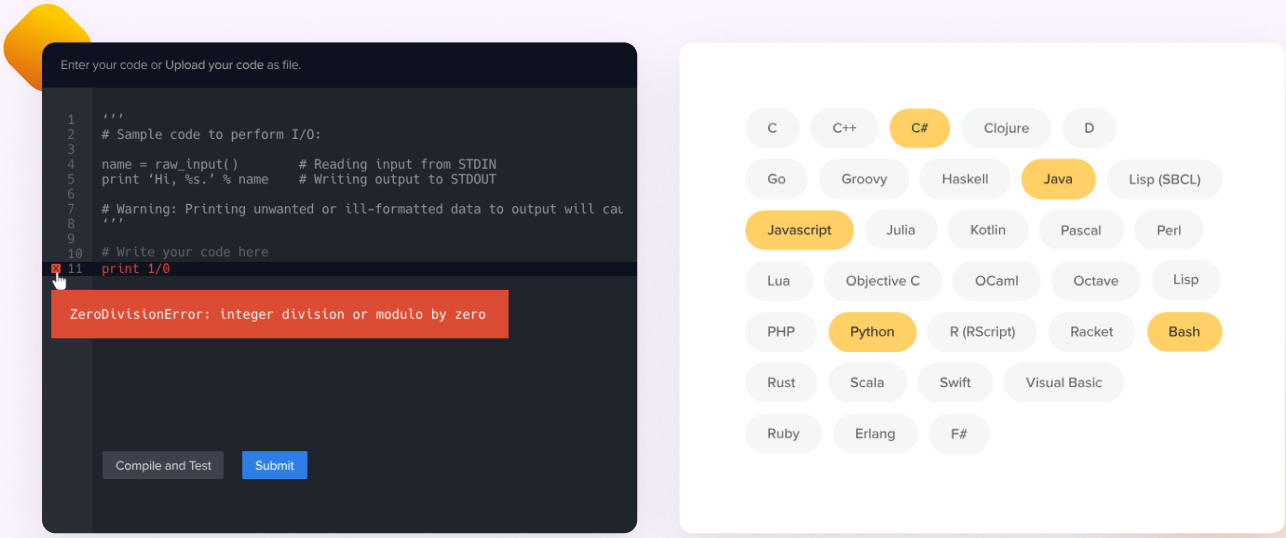
Traditionally, developers have had to manually write code from scratch, a time-consuming and error-prone process. However, with AI-powered code generation, developers can leverage vast existing code patterns to automate this task.

Generative AI models can analyze code repositories, identify common patterns, and generate code snippets or complete functions based on the provided context, speeding up development, reducing human errors, and improving code quality and efficiency

2. AI for code debugging

Identifying and fixing bugs can be a challenging and time-consuming process. Generative AI models can assist developers by analyzing code for potential issues and suggesting changes.

Machine learning and pattern recognition can significantly speed up debugging, enabling developers to locate and rectify errors more efficiently.



3. AI for code optimization

Generative AI tools can analyze code structures, identify areas that can be optimized for improved performance or reduced resource consumption, and provide recommendations accordingly.

Using generative AI in such a manner helps developers enhance the efficiency of their code without requiring extensive manual analysis and trial-and-error.

4. From manual coding to efficient use of AI tools

As AI tools like ChatGPT and Claude become more sophisticated and accessible, there is a shift in focus from manual coding to efficient use of these tools to speed up coding work.

It's important to note that AI tools shouldn't be seen as an outright replacement – but rather, as complementary to coding work. While AI can assist with code generation, debugging, and optimization, human expertise and oversight is still needed.

Developers must exercise caution and critical judgment when utilizing AI-generated code and recommendations, ensuring that they align with the specific requirements and standards of the project.

The ground rules for generative AI in coding

For efficient use of AI for coding, developers – and managers of developer teams – need to adapt the holistic overview before getting into the nitty-gritty of hiring and evaluating for tech talent in this new era of AI-supported coding.

The large-scale changes can be daunting and intimidating. But that doesn't mean you and your team can't keep up. You absolutely can, with the following ground rules in mind:

1. Understand the strengths and limitations of AI tools

You should clearly understand what AI tools can and cannot do. While AI can automate certain tasks and provide suggestions, it is important to recognize that it is not a substitute for human expertise.

You should leverage AI tools as assistants rather than relying solely on their output.

2. Consider the context and domain-specific knowledge

AI models operate based on the data they are trained on. You should ensure sufficient context and domain-specific knowledge of AI tools to generate accurate and relevant code.

This includes providing clear instructions and specifying desired outcomes – in other words, prompt engineering best practices – and aligning the AI's output with the project requirements.

3. Ensure oversight, validation, and review

It is crucial to validate and review the output generated by AI tools. Your team should carefully review the code snippets or suggestions provided by the AI and assess their suitability for the specific use case.

This includes checking for functional correctness, efficiency, adherence to coding standards, and compatibility with the existing codebase.

4. Balance automation and human expertise

You should balance leveraging AI for automation and exercising their own coding skills and judgment.

While AI tools can assist in generating code, your team should actively contribute their expertise, problem-solving abilities, and critical thinking to ensure the code's quality, scalability, and maintainability.

5. Remember ethical considerations

AI tools should be used responsibly and ethically. Your team needs to be mindful of potential biases in training data and outputs and ensure compliance with privacy and security regulations.

You should know the ethical implications of relying solely on AI-generated code and consider the potential impact on code quality, reliability, and user safety.

6. Maintain continuous learning and adaptation

AI technologies are evolving at a brisk pace, and your team should stay updated with the latest advancements in the field. Your team should invest time learning and understanding AI tools and keeping track of new releases, updates, and best practices.

Regularly experimenting, adapting, and refining the utilization of AI tools make for more efficient and effective coding practices.

7. Collaborate and communicate

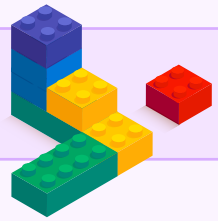
Utilizing AI tools for coding should not hinder collaboration among team members. Your team members should ensure open communication channels, sharing insights, discussing AI-generated code, and seeking colleague feedback.

Collaboration helps validate AI outputs, leverage collective expertise, and maintain code coherence within the team.

Essential new skills for developers in the age of AI

Now that you have established new ground rules and best practices for AI in coding, it's time to think about what the skill set now looks like for developers and how that's changed from before.

Hard skills



Let's start with looking at the hard skills:

1. Data science and machine learning

Developers need to have a strong understanding of data science and machine learning to build and deploy generative AI applications.

This includes understanding how to collect, clean, and prepare data, as well as how to build and train machine learning models.

2. Natural language processing

Generative AI applications often rely on natural language processing (NLP) techniques to understand and generate text.

Developers need to understand NLP well to build effective generative AI applications.

3. Computer vision

Generative AI applications can also create images, videos, and other visual content.

Developers need to understand computer vision well to build effective generative AI applications that can create high-quality visual content.

4. Cloud computing

Generative AI applications can be computationally expensive to train and deploy. Developers need to be familiar with cloud computing platforms to build and deploy generative AI applications that can scale to meet the demands of users.

5. Security

Generative AI applications can be used to create synthetic data that can be used to train machine learning models. This raises security concerns, as synthetic data can create fake news, spam, and other forms of malicious content.

Developers must be aware of the security risks associated with generative AI and take steps to mitigate these risks.

Soft skills

Hard skills are not the only skills that change – soft skills are also affected by the emergence of AI tools in the tech world.

While innate knowledge of technologies and coding languages are absolutely core to success and are themselves updating, soft skills are growing significantly in importance.

They include:

1. Critical problem solving and system design

These are not new skills per se, but rather a change in skill expectations. Previously, software developers could focus primarily on lower-level coding tasks, leaving the architectural design to team leads or senior developers.

Now that AI technology can automate certain coding tasks, developers need a deeper understanding of the broader software development process – including a holistic approach to system design.

2. Technical aptitude for AI

Developers should possess a strong technical aptitude for integrating and leveraging various generative AI platforms. This includes understanding these platforms' functionalities, APIs, and programming interfaces.

They must be proficient in working with the specific tools and frameworks associated with generative AI, enabling them to effectively incorporate these technologies into their workflows.

3. Troubleshooting skills

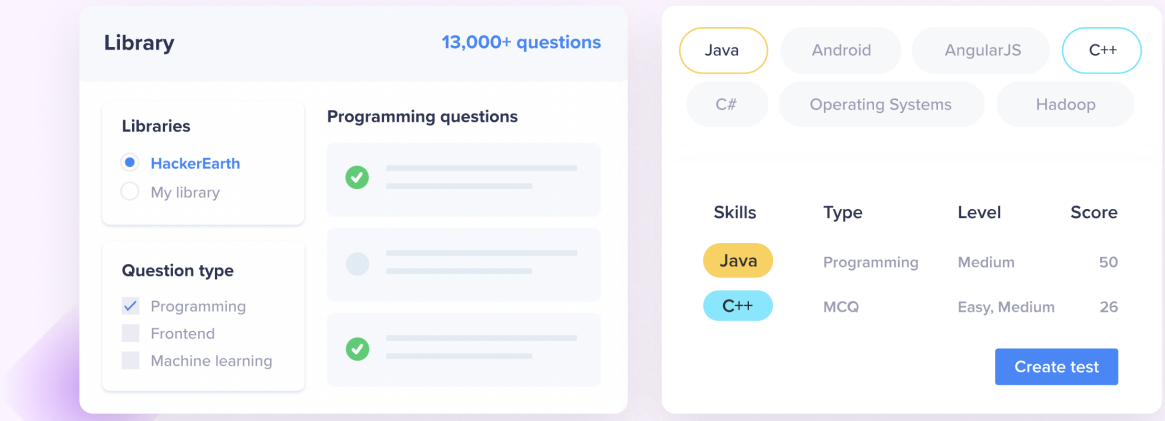
Troubleshooting skills are now a must for handling human-generated and machine-learned programming errors. Regardless of whether the code is generated by AI or written by another human, developers should focus on further developing their code debugging and analysis skills.

As the use of AI-generated code becomes more prevalent, it is important for developers to strengthen their ability to identify and rectify issues in both types of code. This enables them to identify and debug issues that may arise during the development process, distinguishing between errors caused by human code and those stemming from AI-generated components.

4. Understanding data requirements

Developers must also grasp the data requirements for training generative AI models specific to their software projects. They need to identify and curate datasets representative of the problem domain, ensuring that the models are trained on relevant and high-quality data.

They should also be knowledgeable about data preprocessing techniques and the best practices for training AI models to achieve optimal performance and accuracy.



Assessing for these new skills

The debate revolves around whether to embrace the future where generative AI is an integral part of coding or to prioritize demonstrating fundamental skills beyond AI-generated code.

As the landscape evolves, it is crucial to consider the implications and strike a balance between utilizing generative AI for coding efficiency and ensuring that fundamental skills are not compromised.

So, designing assessments that allow for the use of AI tech tools by developers while effectively evaluating their fundamental skill level requires a thoughtful approach.

Here are some considerations for tech companies to follow:

1. Clearly define the assessment goals

Determine the specific skills and competencies you want to evaluate in candidates.

This could include problem-solving abilities, critical thinking, algorithmic understanding, and coding proficiency.

2. Incorporate a multi-stage assessment process

Divide the assessment into multiple stages to evaluate different aspects of a candidate's skills.

For example, you can have an initial stage that focuses on fundamental coding skills without the use of ChatGPT or another AI tool.

This stage can assess a candidate's ability to write clean, efficient code and solve algorithmic problems on their own.

3. Include a stage for problem-solving with an AI tool

Introduce a separate stage that allows candidates to leverage generative AI tools.

This stage can involve more complex and open-ended problems where the emphasis is on utilizing the AI tool effectively to demonstrate problem-solving capabilities.

The focus here is not solely on the code generated but on the candidate's ability to approach and solve real-world problems using AI as a resource.

4. Provide clear guidelines and instructions

Clearly communicate the rules and guidelines for using AI tools during the assessment stage and how it would factor into the final evaluation.

5. Assess problem-solving strategies and explanations

Request candidates to explain their problem-solving approach, rationale, and reasoning during the assessment stages.

This allows evaluators to gain insights into a candidate's thought process, decision-making abilities, and understanding of the problem domain.

This also helps distinguish between candidates who merely rely on generative AI for code generation and those who can effectively apply AI as a tool while showcasing their fundamental skills.

6. Consider a technical interview or code review

Supplement the assessment process with a technical interview or code review.

This enables evaluators to have direct interactions with candidates and evaluate their understanding of concepts, their ability to optimize and debug code, and their response to feedback and questioning.

Criteria for evaluating assessment outcomes

Systemizing your approach also includes fair and consistent analysis of how the candidate performs in assessments.

When a candidate completes an assessment, ask the following questions in order to best identify the strengths and weaknesses of that candidate:

1. Problem-solving approach

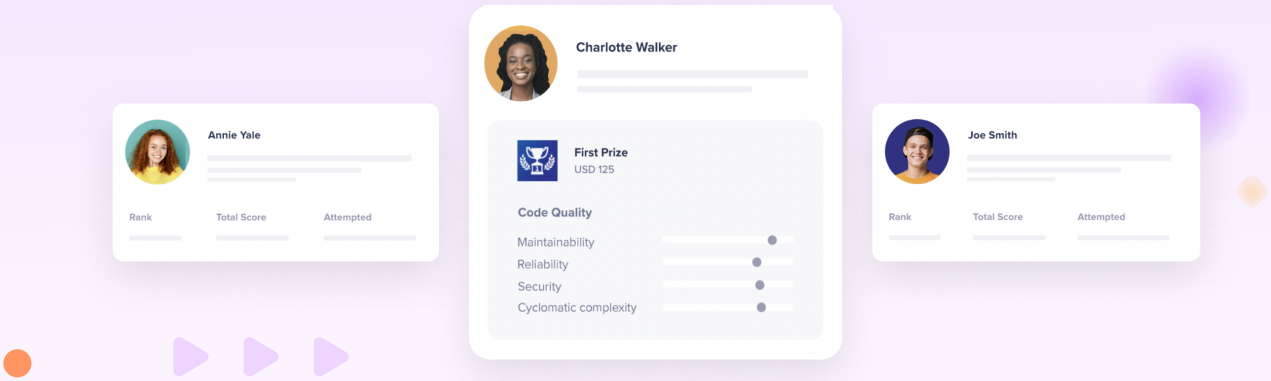
- Did the candidate demonstrate a systematic and logical approach to solving the problem?
- Did they analyze the problem requirements and constraints effectively?
- Did they consider alternative solutions or approaches before using generative AI tools?
- Did they explain their problem-solving strategy clearly and concisely?

2. Utilization of AI tools

- Did the candidate effectively leverage generative AI tools to generate code or solutions?
- Did they appropriately integrate the output of these tools into their solution?
- Did they demonstrate an understanding of when and how to use these tools in the problem-solving process?
- Did they utilize AI tools to enhance their solution rather than relying solely on it?

3. Code quality and readability

- Did the candidate produce clean, well-structured, and readable code?
- Did they follow coding best practices and industry standards?
- Did they appropriately comment and document their code to enhance readability?
- Did they demonstrate an understanding of code optimization and efficiency?



4. Fundamental understanding

- Did the candidate showcase a strong foundation in fundamental programming concepts?
- Did they exhibit an understanding of algorithms, data structures, and programming paradigms?
- Did they demonstrate knowledge of relevant programming languages, libraries, and frameworks beyond generative AI tools?

5. Adaptability and creativity

- Did the candidate showcase adaptability in using generative AI tools to solve different problem types?
- Did they demonstrate creativity in their problem-solving approach and solution design?
- Did they go beyond the basic requirements and provide additional enhancements or improvements to the solution?

6. Communication and explanation

- Did the candidate effectively communicate their problem-solving approach and rationale?
- Did they provide clear explanations of the code or solutions generated with AI?
- Did they articulate their understanding of the problem domain and their reasoning behind the choices made?

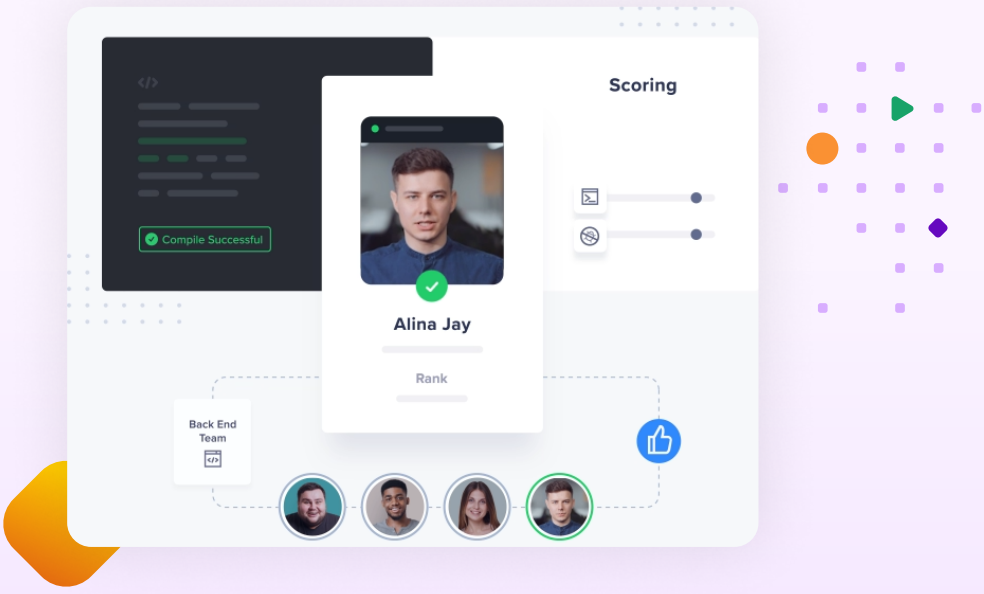
Fair and applicable testing is still essential

While generative AI skills are crucial for certain aspects of software development, it is important to design fair and relevant assessments for the specific skills being evaluated. Assessments should be carefully crafted to focus on the core competencies and knowledge required to work with generative AI.

For areas where generative AI isn't applicable or necessary, such as assessing foundational programming skills or algorithmic thinking, don’t include AI tools in the process. This ensures that developers are evaluated on their fundamental skills without the confounding influence of generative AI.

In addition to these specific approaches, it’s important to consider other relevant evaluation methods when assessing developers' skills in working with generative AI.

These may include code reviews, project evaluations, or interviews that explore developers' understanding of AI concepts, data requirements, and ethical considerations related to generative AI.



AI-dentifying your top talent:
Your new evaluation handbook

The advent of generative AI in software development marks the start of a transformative era, redefining skill sets and methodologies in the tech industry.

As AI becomes commonplace in routine coding tasks, developers must be proficient in AI-driven technologies on top of their traditional coding knowledge. This involves a great deal of adaptability and ability to learn more optimized ways of doing their work in coding.

Concurrently, the assessment methods for developer competencies must evolve if your company is to remain competitive. Companies must strategically integrate generative AI into their assessment processes, allowing candidates to demonstrate AI-augmented problem-solving while ensuring these tools don't overshadow essential programming expertise.

That’s a delicate balancing act. But the ultimate goal is clear – it’s to cultivate a new generation of developers who can harmoniously blend AI tools, existing coding best practices, and human ingenuity all in one powerful package. The best of the pack will lead the charge in an AI-augmented future without compromising the foundational pillars of software development – the bread and butter of SaaS companies worldwide.

From the Experts:
On licensing and IP infringement issues attached to AI-generated code

As AI-generated code becomes more prevalent, licensing issues and concerns around intellectual property (IP) infringement are areas developers need to be mindful of. The ownership of AI-generated code is a complex legal topic.

Generally, the creator or user of the AI model is considered the owner of the code it generates. However, some platforms may claim ownership or rights to the generated code, while others may grant usage rights but retain ownership. Understanding the licensing terms, agreements, and any associated restrictions is essential before using AI-generated code in commercial or proprietary projects.

AI models are trained on large datasets, including publicly available code repositories. Developers must respect the licenses and copyrights of the original code used to train the AI model. They should review and adhere to the licensing terms of any third-party code or libraries incorporated in the generated output when using AI-generated code.

Open-source licenses, such as the GNU General Public License (GPL) or the MIT License, may have specific attributions and sharing modification requirements. Compliance with these licenses is crucial to avoid legal consequences.

Software developers should also be cautious about the potential infringement of IP rights when using AI-generated code. AI models trained on proprietary code may inadvertently reproduce protected elements, leading to copyright or patent infringement.

Some helpful tips to avoid IP infringement include:

1. Carefully review and modify the AI-generated code to add original elements, customize the code for specific needs, and reduce the risk of unintentional infringement.

2. Consider licensing considerations for the models themselves, especially when creating and training your AI models. Open-source licenses, such as the Apache License or the Creative Commons License, can be used to specify the terms under which the AI model and its generated code are shared or used by others.

3. Conduct due diligence to ensure that AI models are not trained on proprietary or copyrighted code to mitigate the risk of IP infringement.

4. Consult legal experts familiar with intellectual property laws to ensure compliance and mitigate potential legal issues.