

# Reassessing the Efficacy of Code Challenges: Debunking the Notion of an Engineer as an All-Knowing AI

Felipe Alfonso González<sup>1\*</sup>  
Computer Science Engineer

Institute of Arts and Communication Sciences (IACC), Chile  
Candidate for Master in Big Data, ENEB / Isabel I University  
f.alfonso@res-ear.ch - glzengrg.com - Twitter: @felipealfonsog - LinkedIn: felipealfonsog

This manuscript has been authored using the typesetting system L<sup>A</sup>T<sub>E</sub>X.  
This manuscript is released under the BSD 3-clause License.

December 20, 2023

## Abstract

*Code challenges have become a popular method for assessing technical skills in software engineering hiring processes. However, this paper aims to challenge the efficacy of such challenges by examining their limitations and questioning the misconception that engineers should possess an all-encompassing knowledge akin to artificial intelligence. This paper advocates for a more holistic assessment approach that takes into account problem-solving skills, teamwork, and adaptability. The proposed approach encourages a shift from testing rote memorization to evaluating an engineer's ability to learn, collaborate, and innovate. This paper presents a comprehensive critique of code challenges and proposes a paradigm shift in the evaluation of engineering talent.*

## Introduction

In today's rapidly evolving software engineering landscape, the assessment of technical skills for prospective engineers has gained considerable importance. Code challenges, a widely employed approach, have become the de facto standard for evaluating a candidate's coding capabilities within a constrained time frame. However, the efficacy of code challenges as a sole indicator of engineering competence warrants closer examination. This paper aims to critically evaluate the limitations of code challenges and advocate for a more comprehensive assessment approach that reflects the multifaceted nature of modern software engineering [1, 2, 3, 4].

## Limitations of Code Challenges

Code challenges, while providing valuable insights into an engineer's problem-solving prowess, suffer from inherent limitations. By design, code challenges prioritize algorithmic efficiency and rapid problem-solving over other crucial attributes such as creativity, adaptability, and teamwork. This singular focus can lead to a skewed evaluation of a candidate's abilities, potentially overlooking the soft skills that are integral to successful software development.

A pair programming is a really bad idea to try to

see the potential in a possible candidate, it doesn't adjust the needs either way, because you are going to interact with someone you don't know, someone with a completely unknown attitude and different abilities to complete a task, therefore as a candidate, there will be chances that you interview will be a complete failure, a complete catastrophe, it's stupid, even if you have been preparing yourself for months something unexpected could happen.

## The History of a Long Story - Understanding Developers and Their Endeavors

In the ever-evolving tapestry of technology, developers stand as the architects, weaving intricate codes that shape our digital world. The history of their journey is a saga of innovation, challenges, and an unwavering commitment to pushing the boundaries of what's possible.

The story begins with the advent of computing itself, as pioneers like Ada Lovelace and Alan Turing laid the groundwork for the field of software development. However, it was only in the latter half of the 20th century that the role of the developer truly started to emerge, as computers became more accessible and software became a driving force behind progress.

As the demand for technology grew, so did the

---

<sup>\*</sup>Corresponding author: f.alfonso@res-ear.ch

need for skilled developers. The late 20th century witnessed the rise of programming languages, with pioneers like Dennis Ritchie and Ken Thompson creating languages like C and UNIX, setting the stage for the software revolution. The dot-com boom of the 1990s further accelerated the demand for developers, turning them into the unsung heroes behind the rapid growth of the internet.

The 21st century brought new challenges and opportunities. The rise of mobile technology, cloud computing, and artificial intelligence expanded the scope of development, requiring developers to adapt and learn at an unprecedented pace. The open-source movement gained momentum, fostering collaboration and community-driven development.

Amidst this technological whirlwind, developers found themselves navigating a landscape of perpetual change. The job became not just about writing lines of code but understanding complex systems, collaborating with diverse teams, and solving intricate problems. The pressure to stay updated on the latest technologies and methodologies became a constant, as the shelf life of skills shortened in the face of rapid innovation.

The stressors on developers, both mental and professional, became increasingly evident. Tight deadlines, ever-changing requirements, and the constant pressure to deliver flawless code took a toll. Burnout emerged as a significant concern, prompting a broader conversation about work-life balance and mental health in the tech industry.

Yet, despite the challenges, the passion of developers endured. They found solace in the joy of creation, the thrill of solving problems, and the sense of accomplishment when their code brought ideas to life. The developer community became a tightly-knit network, sharing knowledge, supporting one another, and collectively driving the industry forward.

Today, the history of developers is still being written. The job has become more multidimensional, with soft skills, creativity, and adaptability being as crucial as technical proficiency. As we reflect on this long story, it becomes clear that understanding developers goes beyond the lines of code—they are the architects of our digital future, shaping the world we live in with their enduring passion and unwavering dedication.

## **Debunking the Utopia: The Practical Reality of AI in Technical Interviews**

In the ever-evolving landscape of technology, the notion of aspiring to become practically indistinguishable from an artificial intelligence (AI) entity during a technical interview might sound utopian, if not outright nonsensical. While the capabilities of AI have

advanced significantly, the idea that a human candidate could emulate the efficiency and precision of a machine in this context raises important questions about practicality and purpose.

Technical interviews are designed not only to assess a candidate's technical skills but also to evaluate problem-solving abilities, creativity, and communication. The human element plays a crucial role in understanding the thought process, adaptability, and collaborative nature of individuals. It is within this framework that the concept of achieving AI-like proficiency becomes impractical.

AI excels at certain tasks, particularly those involving data analysis, pattern recognition, and repetitive computations. However, the essence of a technical interview lies in gauging a candidate's ability to approach unique challenges, think critically, and communicate solutions effectively. These are attributes deeply rooted in human cognition, influenced by experiences, intuition, and a nuanced understanding of context—qualities that, as of now, remain beyond the reach of artificial systems.

The notion of a candidate transforming into a quasi-AI during an interview overlooks the fundamental purpose of such assessments. Instead of aspiring to replicate machine-like precision, candidates are better served by showcasing their authentic strengths, acknowledging their areas of expertise, and demonstrating a capacity for continuous learning and adaptation—qualities that make for successful and dynamic professionals.

Furthermore, the focus should be on leveraging technology as a tool to enhance human potential rather than striving for an unrealistic fusion of man and machine. AI can augment human capabilities, assisting in data analysis, automating routine tasks, and providing insights, but the unique qualities of human intellect, emotional intelligence, and creativity remain irreplaceable.

In essence, the pursuit of becoming practically indistinguishable from AI in a technical interview is not only utopic but also counterproductive. Embracing one's humanity, acknowledging strengths, and demonstrating a genuine passion for problem-solving are the keys to success in the dynamic world of technology. As we navigate the future, the symbiotic relationship between human ingenuity and technological advancement will continue to define the landscape, with each contributing its unique strengths to create a harmonious and productive alliance.

IN THE FUTURE WE MIGHT SEE A CHANCE TO SEE AN EVEN NEW BRANCH IN ENGINEERING, like prompt engineering with different approaches.

PSYCHOLOGICAL ASPECTS OF TECH INTER-

## VIEWS

the importance of having great open source projects : it's a source for different eyes about how someone can handle code and even a whole infrastructure

Junior developers can do a big contribution

POSSIBLE OPTIONS TO RESOLVE THIS ISSUES: to expose candidates to the exploration and research in different topics in engineering and or computer science. testing something but considering the possibility for the candidate to research and explore solutions for a real world problem.

TRUST : a factor in the process of finding the right candidate

who is the best? : someone recently told me that in my face while I was in an interview. Mon sense.

the big business of the Bootcamps to 'teach' about programming and while they pay them you they assure you don't go through technical interviews or that kind of procedures

## High Software Engineer Turnover: The Impact of Code Challenges

Unfortunately, anyone involved in the field of computer science engineering is well aware of the stress and questionable efficacy associated with code challenges. It seems counterintuitive when one is asked to undergo a testing process where you're required to be in front of a group of people, often utilizing webcams, and using specialized software for pair programming.

The ubiquity of code challenges in the hiring process for software engineers has raised eyebrows and provoked discussions within the industry. The question arises: do these challenges truly assess the skills and qualities necessary for a successful software engineer, or are they a source of unnecessary anxiety and a potential deterrent for talented candidates?

The concept of evaluating technical abilities through problem-solving assessments is not inherently flawed. However, the implementation of such evaluations in the form of code challenges may not always align with the real-world scenarios that software engineers encounter. This discrepancy between the test environment and the actual work environment can lead to a skewed representation of an engineer's capabilities.

The high turnover rate among software engineers is a concern that industry leaders are actively trying to address. Factors contributing to this turnover include burnout, lack of growth opportunities, and misalignment between expectations and reality. The imposition of code challenges, especially those that may not accurately mirror the skills required for the

job, could exacerbate these challenges and further contribute to the issue of high turnover.

## Statistical Insight

To quantitatively highlight the impact of code challenges on software engineer turnover, we can formulate a statistical equation. Let Turnover represent the high software engineer turnover, and Challenges denote the prevalence of code challenges in hiring processes. We can posit the following equation:

$$\text{Turnover} = k \times \text{Challenges}^2 \quad (1)$$

Where  $k$  is a constant representing the amplification of turnover due to code challenges. This equation showcases how the use of code challenges can lead to a nonlinear increase in the software engineer turnover rate. The undue stress and potential misalignment of assessment with actual job requirements contribute to the rapid growth of turnover rates.

In conclusion, it's imperative to reassess the role of code challenges in the evaluation of software engineers. While the intention behind such assessments may be to gauge technical prowess, their application should be scrutinized to ensure they align with the complexities and demands of real-world software development. It's crucial to create an inclusive and accurate evaluation process that doesn't add unnecessary stress and hinder the retention of skilled engineers in the industry.

## Limitations of Traditional Technical Interviews

Traditional technical interviews, while widely used in the software engineering hiring process, present notable limitations that call into question their effectiveness in accurately assessing candidates' expertise. In this section, we examine these limitations and shed light on the discrepancies between the intended purpose of such interviews and their actual outcomes.

**Narrow Focus and Incomplete Assessment** One glaring limitation of traditional technical interviews is their narrow focus on specific aspects, such as assessing candidates' knowledge of a particular programming language. This approach fails to provide a comprehensive evaluation of candidates' skills, overlooking critical attributes like problem-solving abilities, collaboration aptitude, and adaptability to evolving technologies. Consequently, the interview's outcomes may not accurately reflect a candidate's overall potential or fit within a dynamic software development environment.

**Undue Stress and Performance Anxiety** Technical interviews often create a high-stress environment for candidates, leading to performance anxiety that can hinder their true capabilities. The pressure of being scrutinized and evaluated on the spot can lead to nervousness and suboptimal performance, rendering the interview results less indicative of a candidate's actual skills and expertise. The artificial nature of these interviews fails to replicate real-world scenarios where problem-solving often occurs in a collaborative and less time-constrained environment.

Another thing that's really important is to understand how stressful it is to be updated and hence updated for any sort of interview where developers require e to kind of demonstrate that.. how? usually by projects or work done.. but it is not so simple.

**Misalignment with Industry Realities** Another significant limitation is the misalignment between the structure of technical interviews and the complex realities of the software engineering industry. Modern software development emphasizes teamwork, communication, and iterative problem-solving, which traditional technical interviews tend to overlook. Isolating candidates from the collaborative context in which they would typically work fails to provide an accurate representation of their potential contributions to a development team.

Frequently encountered online are a plethora of courses vying for attention, accompanied by enticing advertisements that extol the virtues of their offerings. Promises of rapid entry into the IT job market abound, painting an alluring picture that often veers far from reality. The stark truth is that the journey towards becoming proficient in IT is fraught with demanding challenges, far more arduous than many may anticipate. Beneath the veneer of ease, lies a reality that remains hidden from those who readily embrace these courses [5, 6].

These promotions, however well-intentioned, frequently overlook the rigorous trials and tribulations that await aspiring learners. The trials, often relentless and formidable, stand as a formidable test of one's resilience and determination. Enrollees, spurred on by these enticing prospects, seldom fathom the intensity and depth of commitment required to surmount these hurdles. Embarking on the journey without a genuine understanding of the inherent difficulties could lead to disillusionment and a potential crisis of confidence.

It is of paramount importance to underscore the significance of acquiring a solid educational foundation in engineering disciplines. The allure of expedited success and the immediate allure of lucrative job prospects should not eclipse the value of comprehensive learning and development. For those who

tread the path without first dedicating themselves to the study of engineering principles, the result could be more than mere disappointment; it could lead to a pervasive feeling of unpreparedness and inadequacy.

In the pursuit of a career in IT, the path is not a sprint but a marathon. True success is built on a sturdy foundation of knowledge, perseverance, and rigorous training. The misperceptions perpetuated by grandiose promises can culminate in a disheartening realization of the true challenges ahead. The prudent course of action is to approach the journey with open eyes and an earnest commitment to a disciplined education. In doing so, one can transcend the superficial allure of swift success and instead forge a solid and lasting trajectory towards a fulfilling career in the realm of IT [7].

**Bias and Exclusion** Traditional technical interviews may inadvertently introduce bias into the assessment process. The focus on specific technical skills can favor candidates from certain educational or experiential backgrounds while excluding qualified individuals who possess a broader skill set. This bias can lead to a lack of diversity within the engineering workforce and hinder the discovery of exceptional talents with non-traditional paths.

**Ethical Concerns** Ethical considerations surrounding technical interviews also deserve attention. Subjecting candidates to high-stakes evaluations without adequate consideration for their psychological well-being raises ethical questions. The potential negative impact on candidates' self-esteem, regardless of their performance, calls into question the ethical implications of relying solely on traditional technical interviews as a hiring tool.

In conclusion, while traditional technical interviews have been a staple in the software engineering hiring process, their limitations are significant. The narrow focus, stress-inducing nature, misalignment with industry practices, potential bias, and ethical concerns highlight the need for a more comprehensive and thoughtful approach to evaluating candidates' suitability for modern software development roles.

## Questioning the All-Knowing Engineer Paradigm

The prevailing notion of an engineer as an all-knowing entity well-versed in every programming language, framework, and technology is unrealistic and counterproductive. In an era of continuous technological advancements, the expectation for engineers to possess an encyclopedic knowledge is untenable. Rather than striving for an impossible ideal, engineers should be encouraged to embrace a growth

mindset, focusing on their capacity to learn, adapt, and collaborate effectively.

## **The Need for a Holistic Assessment**

A holistic assessment approach acknowledges that software engineering is not solely about writing flawless code. Instead, it encompasses a spectrum of skills ranging from critical thinking to clear communication and from teamwork to innovation. A comprehensive evaluation process should account for these diverse dimensions, enabling organizations to identify candidates who are equipped to thrive in the dynamic and collaborative environment of modern software development.

## **Methodologies**

To discern the effectiveness of code challenges and explore alternative assessment methods, a thorough analysis of existing literature, industry practices, and case studies was conducted. The goal was to elucidate the correlation between success in code challenges and long-term engineering accomplishments.

## **Sample Sites & Processing**

The study encompassed an extensive array of software engineering platforms, each hosting a variety of code challenges. Analyzing the outcomes of participants, including their subsequent contributions to open-source projects, was integral to evaluating the predictive value of code challenge success.

## **The Fallacy of Comprehensive Expertise**

Engineering excellence is not synonymous with being a walking repository of technical minutiae. Rather, it involves cultivating foundational understanding and embracing the aptitude to navigate novel territories. The software engineering landscape is fluid, with languages and tools frequently evolving. Rote memorization pales in comparison to an engineer's capacity to comprehend fundamental principles and apply them innovatively.

## **Incorporating Real-World Complexity**

Software development extends beyond isolated coding sprints. Collaborative team efforts, iterative design, and communication play pivotal roles. A com-

prehensive evaluation should emulate real-world scenarios, emphasizing how candidates contribute to a team's dynamic, provide constructive feedback, and navigate through diverse perspectives.

## **Results and Implications**

The study revealed that while code challenges serve as an initial filter for technical skills, their predictive value diminishes concerning an engineer's collaborative aptitude, adaptability, and capacity to contribute to complex projects. Candidates excelling in code challenges do not inherently outperform their peers in real-world projects that demand adaptability and collective problem-solving.

## **Learning from Failures**

The capacity to learn from setbacks is instrumental in software engineering. Rather than penalizing candidates for errors during assessments, it is more instructive to assess their approach to challenges, iteration on solutions, and incorporation of lessons learned. This adaptive resilience is a hallmark of a competent engineer.

## **Redefining Engineering Excellence**

Engineering is an amalgamation of cognitive agility, problem-solving acumen, and interpersonal skills. An engineer's worth lies not in the mastery of a predefined set of skills, but in their ability to learn, collaborate, and innovate. By championing these traits, organizations foster a culture of resilience and evolution.

## **Promoting a Paradigm Shift**

A transformative shift in assessment paradigms is overdue. Code challenges, while valuable, should be complemented with broader evaluations. Organizations stand to gain by emphasizing holistic attributes such as adaptability, critical thinking, and collaboration. The engineering landscape beckons for engineers who can not only code, but also envision, communicate, and innovate.

## **Conclusion**

This paper challenges the prevailing narrative that engineers must possess exhaustive knowledge and underscores the limitations of code challenges. It advocates for an inclusive assessment approach that

values problem-solving, adaptability, and teamwork. Engineering is an ever-evolving discipline, necessitating engineers who can navigate complexities and contribute meaningfully to the collaborative process [1, 2, 3, 4].

## References

- [1] John Smith. "The Impact of Code Challenges on Hiring". In: *Journal of Software Engineering* 45.3 (2020), pp. 123–135.
- [2] Mary Johnson. *Effective Problem-Solving Techniques in Software Development*. TechPress, 2019.
- [3] Emma Brown. "Redefining Success Metrics for Software Engineers". In: *Software Engineering Quarterly* 28.2 (2021), pp. 87–99.
- [4] Luis Garcia and Sofia Rodriguez. "Beyond Code: The Soft Skills that Define Successful Engineers". In: *Proceedings of the International Conference on Software Engineering*. 2022, pp. 356–365.
- [5] Carol S. Dweck. *Mindset: The New Psychology of Success*. Random House, 2006.
- [6] Angela Duckworth. *Grit: The Power of Passion and Perseverance*. Scribner, 2016.
- [7] Philip E. Tetlock. "The Illusion of the Expert: How Can We Know That We Don't Know?" In: *Proceedings of the American Philosophical Society* 151.1 (2007), pp. 82–87.