Industry Coding Skills Evaluation Framework

Dmitry Filippov, Vitaliy Plastinin, Sylvia Lin, Yi Zhuan Foong, and Frank Mu

CodeSignal
Skills Evaluation Lab

Published August 2021

Abstract – With the ever-increasing appetite for digital transformation across most companies, demand for software engineering talent, particularly at the mid-to-senior levels, continues to see exponential growth. Such widespread demand requires engineering, recruiting, and talent teams to adopt scalable hiring practices so they can efficiently assess the technical capabilities (i.e., coding skills) of job candidates. However, given the lack of standardized technical assessments and certifications, many companies rely on idiosyncratic and inefficient hiring practices, such as resume and phone screening, that are difficult to scale, costly to maintain, and provide poor signals about candidates. Such practices often slow down time to hire, result in a poor candidate experience, and run the risk of perpetuating systemic biases and inequality. This paper introduces the Industry Coding Assessment (ICA) framework, a standardized and industry-wide framework for assessing the technical skills of mid-to-senior software engineers at scale. The ICA simulates real-world coding projects, allowing candidates to objectively demonstrate their skills in implementing, designing, and maintaining codebases. The ICA framework is designed around the principles of scalability, validity, and fairness to provide high quality signals about candidates' coding skills. Thus, the ICA will enable modern hiring teams to efficiently identify top talent and dramatically scale hiring processes. Research on topics for the ICA framework and data from initial ICA tasks are outlined.

Introduction

Software engineering roles have exponentially increased over the last decade, and will continue to expand at a rapid pace as corporations invest heavily into building out digital infrastructure and scale their technical capabilities. While rapid digitalization has led the paradigm shift in terms of the way businesses operate, hiring processes remain relatively slow and ineffective [1].

This Industry Coding Assessment (ICA) paper describes a general framework for creating standardized assessments that can be used to measure general coding competencies for mid-to-senior software engineers, regardless of the existing variance in domain-specific expertise. Substantial research has been conducted to ensure that the assessment framework can produce signals about the coding competencies of senior software engineers in ways that are consistent, valid, and EEOC-complaint. Additionally, the modular nature of the framework allows for developing scalable assessments regardless of volume, while keeping all versions of the assessment to strict standards of quality and effectiveness.

Issues with Existing Hiring Practices

Presently, the industry-standard hiring practices for senior software engineers vary significantly across roles, companies, and industries. For example, when screening candidates on technical coding competencies, some employers might just review resumes,

some employers might use custom assessments built in-house, and some employers might use the traditional phone call. These differences are further exacerbated by the vastly different requirements expected of senior engineers, as well as the variance in the quality of their experience, working environment, and corporate culture. Although some tailoring may help to identify top talent for a specific role, such idiosyncratic hiring practices often fail both employers and candidates alike. On the one hand, employers may find it prohibitively costly to design and maintain effective methods of capturing signals about the coding competencies of candidates, leaving them with poor methods to make hiring decisions and unable to scale their hiring processes. On the other hand, candidates may face convoluted hiring practices that do not allow them to fairly showcase their talent, leaving them frustrated and with poor impressions of prospective employers.

Resumes are poor proxies for competency

When screening candidates on technical coding competencies, industry-wide hiring practices heavily leverage resumes as an indicator of competency which often introduces significant biases. With resume screens at the top of the funnel, self-reported experience, sometimes substantiated by pedigree, have been positioned as a proxy for a candidate's capacity to succeed in the role. Relying on such proxies can have significant flaws, as resume content are mainly designed for self-marketing and often not substantiated or objective. Likewise, pedigree bias can run the risk of perpetuating systemic discrimination and inequality by continuing to screen out candidates from underrepresented groups.

The self-authoring nature of resumes often calls into question the integrity of information reported on a resume. For example, some studies estimate that 36% of candidates lie on their resumes by misrepresenting their experience, skills and job duties [2]. With relatively little repercussion for fraud and prohibitive cost of due diligence, catching embellishments or outward lies on resumes is very challenging and often not worth the effort. Yet, employers face significant detriments to productivity when the candidate is hired based on lies and unable to perform their duties successfully.

Additionally, other research has consistently shown that resume bias continues to plague the hiring process in spite of its widespread and long standing adoption. For example, resumes containing racial minority cues tend to receive significantly lower rate of callbacks compared to resumes without such characteristics [3], and similar trends have been observed in age–related cues in resumes. Thus, resume screening can lead to additional unconscious bias against minority or older candidates despite those candidates having similar skill competencies as other candidates [4].

Moreover, hiring on pedigree discriminates against candidates with non-traditional backgrounds. With the average student loan debt bordering on $30,000, many qualified candidates may possess sufficient coding skills but are being priced out by the exorbitant cost of traditional educational paths and

are thus carelessly screened out. With relatively lower barriers to entry, coding bootcamps and self-learning have proven to be effective ways for candidates to quickly learn advanced programming skills [5, 6]. In fact, recent surveys showed that 24% of software engineers were self-taught, and 10% graduated from bootcamps [7]. Hiring on pedigree excludes this significant group of candidates with non-traditional coding educations, and further perpetuates the inequality of opportunity for candidates who are unable to afford a hefty traditional college education in computer science.

Overall, despite widespread usage, resumes are poor proxies to assess the coding skill of software engineers due to lack of integrity and high potential for unfairness and adverse impact.

In-house assessments come with many hidden costs

Some forward-thinking companies have also pursued the route of building out their own internal assessment framework and content. In general, in-house assessments are designed by senior members of the hiring team and can range from cognitive and behavioral to skills-based assessments. However, since most in-house assessment designers are not explicitly trained in assessment design and validation, they are likely to lean on general heuristics and anecdotal experience to create content without referencing proper frameworks. When such assessments are put together, although they are well-intentioned, they often fail to consider objective guidelines for validation recommended by the US Equal Employment Opportunities Commission (EEOC), which would unwittingly lead to ineffective hiring decisions and adverse impact [8].

In addition, designing and testing assessments are highly costly and distract key technical personnel away from their main responsibilities. Even after going through the effort of developing assessments in-house, the content is expensive to maintain due to the high potential for leaks on programming-focused networks like StackOverflow, Glassdoor, and Blind. In particular, the high-stakes nature of such assessments would motivate many candidates to both seek and share content with other candidates, rendering the assessments effectively useless for differentiating candidates on their coding skills. With limited mechanisms to detect and flag plagiarism, lack of maintenance for in-house assessments further hamstrings hiring teams by providing false positive signals about the coding skill of candidates.

Overall, although in-house assessments are often well-intentioned, they are rarely designed and maintained effectively, as doing so can be prohibitively costly for many companies.

Technical phone screens are not a scalable answer

Due to inconsistencies with resumes and difficulty of building in-house assessments, many companies rely on technical phone screens to evaluate candidates on their coding skills. However, such practices are rarely scalable without incurring massive costs. Since key technical personnel (i.e., senior software engineers) need to spend considerable time conducting one-on-one inter-

views with candidates, asking them to conduct more interviews can only further distract them away from their main responsibilities.

Technical phone screens also inherit all of the hidden costs of building in-house assessments, as the content must be designed and maintained effectively to accurately capture signals about candidate coding skills. Even if the content were well-designed, live interviews are plagued with human-driven biases which hamper their ability to capture relevant signals about candidates [9], resulting in unfairness and adverse impact.

Many technical phone screening processes are unstructured and poorly designed, leaving candidates with the impression that the interviewers are just flaunting their technical knowledge by deliberately asking obscure questions [10]. When such poorly designed phone screens involve coding, many candidates report that the experience of having their code graded and scrutinized while coding creates considerable anxiety, leaving them with the impression that such interviews are stress tests rather than coding skill assessments [10]. Moreover, without a well-designed structure, the candidate experience is largely dependent on the connection that candidates can form with their interviewers, which can lead to biased and inconsistent experiences with potential to alienate senior engineering candidates.

Overall, technical phone screens are not a scalable answer to screening candidates on their coding skills, as phone screens inherit the same hidden costs as in-house assessments while running the additional risks of interview biases and alienating candidates.

Need for Senior Engineering Assessment Framework

What makes a senior engineer?

While the specific definition can vary across geographies, industries, and corporate structures, senior software engineers encompass a core set of skills that can range from making informed decisions, to leading projects independently, to providing mentorship to junior engineers within their team [11]. However, the most essential skill for senior software engineers is writing well-designed and efficient code. The realm of code writing encompasses other attributes that make a well rounded software engineer. This includes, but is not limited to, maintaining extensive engineering knowledge, testing and debugging code, staying mindful of memory consumption and performance, and preserving consistency in code design and style.

Hiring senior engineers

Macro-trends show a strong correlated increase in the demand and salary of software engineers, which has remained resilient inspite of other global events, such as the general volatility stemming from the global COVID-19 pandemic that started in 2020 [12]. In fact, pandemic-driven tailwinds have accelerated the digitalization appetite across most corporations, massively increasing the general demand for engineering talent. At the same time, engineering and recruiting teams are met with challenges to quickly adapt to the normalization of remote working and virtual hiring of diverse candidate back-

grounds [13].

Senior software engineering roles take the number two spot in the most in demand tech listings on popular employment networks like Indeed and Linkedin, logically second only to software engineering roles. This is exemplified by how mid-level software engineers (i.e., with 4 to 10 years of experience) receive an average of 35% more job interviews than other roles with similar experience across most industries [14]. In the face of the burgeoning demand and stiff competition for senior engineers, companies are starting to focus on candidate experience throughout the hiring process as well as time to hire as key competitive advantages to keep candidates engaged and eventually convert to full-time hires.

Hence, hiring organizations are often faced with a false dichotomy of choice—whether to reduce the time to hire at the expense of a more comprehensive interview process or to reduce the number of reviewers and fill vacancies quickly. As leaders in the organization, the impact senior engineers will likely have in terms of decision-making and hiring at the company could result in significant tangible costs in the event of a poor hire, further compounding the indecisiveness in the hiring process [15, 16].

An industry-wide standardized skills assessment framework would be the key to alleviating the issues discussed above. Assessments developed from this framework can easily be scaled without requiring human interviewers, reducing time to hire while reliably capturing relevant signals about the coding skills of senior software engineering candidates. This will ensure a strong correlation between performance on the assessment and performance on-the job without negatively impacting candidate experience.

The Industry Coding Framework

The Industry Coding Assessment (ICA) is built to assess senior engineers around the key principles of validity, scalability, and fairness. Each ICA contains 1 domain-agnostic, project-based task with 4 progressive levels. The maximum allowed completion time for the assessment is 90 minutes; however, candidates are not necessarily expected to complete all tasks within this time limit. While longer assessments allow more accurate measurement of candidate skills, the willingness to complete assessments decreases dramatically for tests longer than 2 hours. Moreover, a major factor in assessing candidates' skill levels is to see how far they can progress within the given time frame.

The requirements for each level build on the requirements from previous levels, so the overall task progressively increases in complexity. The first level requires basic implementation operations while accounting for corner cases. The second level introduces data processing functions, such as calculations or exports. The third level extends the previous functionality to support new advanced features. And finally, the fourth level culminates the project with further extension of functionality. Given the progressive nature of the requirements, candidates must reuse, encapsulate, and refactor earlier code to maintain backward compatibility.

The next sections will delve deeper into

the specifics of the Industry Coding framework, starting with the design of each level and details about how the core competencies map against their requirements. Then, the methodology for determining ICA design is discussed. Finally, data from pilot research, which was conducted to test and iterate on the Industry Coding Assessment from real world usage is presented to provide an overview of score norms.

Building ICA Tasks

Each ICA task is designed around a set of four progressive levels that increase in complexity. At each level, new methods and entities are introduced while retaining the integrity of previously implemented method contracts. This ensures that candidates will not have to completely change their existing implementations, but they will be required to refactor the code to replicate a real-world working scenario and iterative software development methodologies.

While each level might extend the functional requirements of existing methods, these are designed to be logical continuations of existing methods and are always aimed to be backward compatible. Hence, candidates with more thoughtful implementation design would also benefit in later levels, similar to how code styling and organization in a legacy codebase significantly impacts the complexity of dependencies for future features.

Overall Design

All ICA tasks will require candidates to write code to implement required functionality described in the task. The overall goal for candidates is to write code efficiently while accounting for increasingly complex requirements. As such, all tasks explicitly describe the project-based and progressive nature of the levels, and encourage candidates to complete as many requirements as possible within the time limit. This simulates the evolving and deadline-driven nature of real-world coding projects.

A sample ICA task is described in the Appendix.

Level 1

The first level assesses general programming abilities and the use of basic data structures. At this level, candidates are expected to implement 3-4 simple methods. Typically, candidates should expect to spend **10-15 minutes** on this level while writing 15-20 lines of code.

Can Include

- Basic implementation (conditions, loops, type conversions, strings, etc.)
- Basic data structures (1-2D arrays, lists, hash tables)
- Covering corner cases
- Error handling

Should Exclude

- Advanced data structures
- Advanced implementation
- Any complex algorithms, problem solving, optimizations

Level 2

The second level introduces the implementation of data processing functions, such as calculations, and aggregations, or exporting, while also assessing the ability to reuse code

from Level 1. Specifically, this level is focused on implementation skills, and does not require advanced algorithms, problem solving, or optimizations. At this level, candidates are expected to implement 1-2 additional methods of medium difficulty. Typically, candidates should expect to spend **20-30 minutes** on this level, and write 30-45 lines of code for both Level 1 and Level 2.

Can Include

- Intermediate implementation (data processing, statistical functions, etc.)
- Processing large streams of data (projection, filtering, aggregation, etc.)
- Reusing and building on existing code
- Advanced built-in data structures (counters, linked lists, sorted sets, etc.)
- Manipulate data representations based on commonly used formats (JSON, CSV, etc.)

Should Exclude

- Complex or niche algorithms (binary search, two pointers, dynamic programming, etc.)
- Data optimizations
- Use of third-party libraries
- Use of non-built-in advanced data structures
- Parsing data files (JSON, CSV, etc.)

Level 3

The third level requires candidates to extend and maintain their existing codebase from Level 1 and Level 2. This level will assess the ability to refactor or encapsulate functionality from the previous levels to support new features. The difficulty of this level depends on the quality of the previously implemented methods – the more reusable the previous methods are, the easier it will be to implement new functionality. At this point in the project, code design will have an intermediate impact on performance, as inefficient designs will be costly to refactor. At this level, candidates are expected to implement 3-5 additional methods of medium to advanced difficulty, some of which are encapsulations of previous functionality with additional logic. Candidates may also be expected to add some helper methods and classes upon implementation. Typically, candidates should expect to spend **30-60 minutes** on this level, and write 90-130 lines of code for Level 1, Level 2, and Level 3.

Can Include

- Refactoring or encapsulation techniques to incorporate additional functionalities while maintaining backward compatibility for existing code
- Advanced implementation and problem solving without complex algorithms
- Basic software design patterns and principles
- Advanced built-in data structures (sorted maps, linked lists/queues, stacks, etc.)

Should Exclude

- Advanced techniques (concurrency, parallelism, distributed computing, etc.)

- Complex or niche algorithms (binary search, two pointers, dynamic programming, etc.)
- Use of third-party libraries
- Use of non-built-in advanced data structures

Level 4

The fourth level is the final level, which finalizes the project by implementing 1-2 additional methods that enhance functionality and are backward compatible with the existing architecture designed in previous levels. Similar to Level 3 but to an even greater extent, the difficulty of this level will be dependent on the scalability and reusability of the previous levels—the more reusable the previous methods are, the easier it will be to refactor and implement additional functionality. At this point in the project, efficient code design will have a significant impact on performance, as an inefficient codebase will be almost impossible to refactor and encapsulate within the time limit. At this level, candidates are expected to implement 1-2 additional methods of medium to advanced difficulty, depending on their previous implementation. Typically, candidates should expect to spend **30-60 minutes** on this level, and write 110-160 lines of code to complete the entire assessment.

Can Include

- Adjusting previous functionalities without regressions
- Advanced implementation and problem solving without complex algorithms
- Refactoring or encapsulation techniques to incorporate additional functionalities while maintaining backward compatibility for existing code
- Optimal software design patterns and principles
- Advanced built-in data structures (sorted maps, linked lists/queues, stacks, etc.)

Should Exclude

- Advanced techniques (concurrency, parallelism, distributed computing, etc.)
- Complex or niche algorithms (binary search, two pointers, dynamic programming, etc.)
- Use of third-party libraries
- Use of non-built-in advanced data structures

Scoring ICA Tasks

Code submissions are automatically scored by a series of test cases, similar to how unit tests are designed to gauge code functionality and catch for edge case problems. To ensure consistency, each level within an ICA task will have 10 test cases, summing to a total of 40 test cases per task. All ICA tasks will produce an overall coding score based on the number of levels completed and number of unit tests passed within each level.

ICA tasks are progressive by design, such that higher-level requirements depend on effective implementation of lower-level requirements. However, to reflect requirements in real-world coding projects, dependencies will be less likely for edge or corner cases, such that some submissions could fail

lower-level edge cases while passing higher-level core test cases. As candidates are challenged to complete as many requirements as possible, all 40 test cases are weighted equally to allow candidates flexibility in terms of what they can focus on. Specifically, each test case will be worth 25 points, and the overall coding score will be a number ranging from 0-1000. Moreover, the final score will be the highest score achieved throughout the assessment to avoid excessive penalties if candidates run into errors while refactoring their code. Table 1 illustrates some common and meaningful outcomes arranged by ability levels (expected scores).

Initial ICA tasks have been reviewed and piloted within the hiring process for various senior software engineering positions across different companies and industries. Figure 1 presents the score distribution for updated ICA tasks from a sample of 5,164 candidates.

Expected Score	Levels	Description
125	1	The candidate can implement basic operations and work with simple data structures, but is unable to write efficient code that accounts for corner cases.
250	1	The candidate displays novice implementation skills and can implement basic operations to efficiently use basic data structures while accounting for corner cases.
375	1+2	The candidate displays novice skills in implementation and maintaining legacy code and can build on top of some existing code to support some intermediate operations with advanced built-in data structures.
500	1+2	The candidate displays intermediate skills in implementation and novice skills in maintaining legacy code and can effectively build additional functionality on top of existing code to fully support all intermediate operations with advanced built-in data structures.
625	1+2+3	The candidate displays strong skills in implementation, intermediate skills in maintaining legacy code, and basic skills in code design, being able to refactor some existing code from basic and intermediate operations to support some advanced operations.
750	1+2+3	The candidate displays strong skills in implementation and maintaining legacy code and intermediate skills in code design, being able to effectively refactor and encapsulate all existing code to fully support basic, intermediate, and advanced operations simultaneously.
1000	1+2+3+4	The candidate displays exceptional skills in implementation, maintaining legacy code, and code design, being able to flexibly adapt to changing requirements by effectively refactoring code without regressions to fully support all required operations simultaneously.

Table 1. Descriptions of Common ICA Score Outcomes

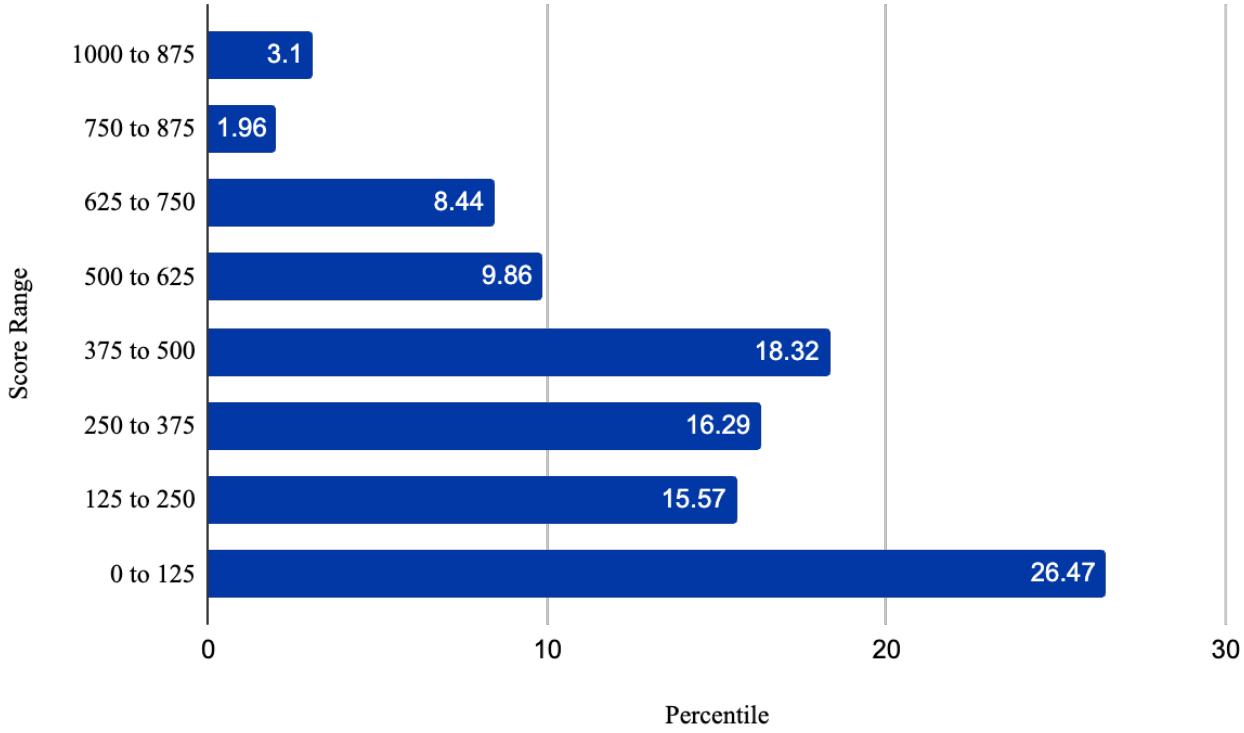

Figure 1. Distribution of ICA scores from a sample of 5,164 candidates.

Deep Dive into ICA Research

As discussed above, senior software engineers are expected to possess a wide range of skills, and will often play a key role in their engineering teams. As both key contributors as well as mentors for fellow team members, their coding skills will be a fundamental factor contributing to the development and success of the organization that they are a part of.

This section expands upon the core design principles of the ICA as well as how the core skills being measured by this assessment were researched and incorporated into the assessment design process. This research process involved gathering insights to answer the following questions:

1. What are the key skills required by a senior software engineer across industries?
2. What differentiates an entry-level engineer from a senior software engineer?
3. What responsibilities are senior software engineers expected to perform in their work?
4. What questions/tasks do large US-based tech companies use in technical interviews for hiring mid-senior level software engineers?

Key insights were compiled from a variety of sources to answer these questions, including:

- Definitive definitions of “computer programmers”/”software developers”

from the U.S. Bureau of Labor Statistics [17, 18] and U.S. Department of Labor O*NET Program [19]

- Industry trends from leading professional networks for software engineers like StackOverflow, Linkedin, Indeed, and Hired [20, 21]
- Industry trends from consulting firms like Bain & Company and McKinsey & Company [22, 23], and labor market forecasts from the World Economic Forum [24]
- Review of 19,107 distinct job descriptions for "senior software engineer" and "senior software developer" posted by US-based companies to job search platforms, including LinkedIn (28%), Indeed (32%), Hired (24%), Glassdoor (7%), ZipRecruiter (6%), and Monster (3%), between January 2020 and December 2021
- Interviews with 23 senior engineering hiring managers in various companies across industries (i.e., tech, FinTech, e-commerce, finance)

In answering the research questions, key skills required for senior software engineers were identified. Digging into the data further revealed fundamental and domain-agnostic coding skills shared by mid-to-senior software engineers across industries, including:

1. **Implementation**
   - 1.1. Ability to design, develop, test, deploy, maintain and improve software applications and/or functions across a wide variety of domains (e.g., cloud, network systems, integration, infrastructure)
   - 1.2. Converting general product vision, user stories, and business requirements into working applications that operate at production-level scale
2. **Programming**
   - 2.1. Ability to leverage built-in data structures (e.g., arrays, sets, hash tables, lists) and algorithms (e.g., search, sort and traversal) appropriately
   - 2.2. Understanding of methods to implement statistical calculations and data aggregations
   - 2.3. Manipulating and formatting data into commonly used standards (e.g., CSV, JSON, YAML)
3. **Code Literacy**
   - 3.1. Ability to read and understand code written by other developers
   - 3.2. Knowledge of code writing best practices (e.g., style guidelines, accuracy, testability and efficiency)
   - 3.3. Understanding of language-specific best practices, advantages, and limitations
4. **Software Design**
   - 4.1. Ability to identify optimal design patterns and principles to effectively implement high-quality code
   - 4.2. Understanding existing code dependencies across complex systems
   - 4.3. Ability to encapsulate and

refactor existing code for backward compatibility to ensure new features and functionality can be built on top of legacy codebases

4.4. Writing thorough and reusable code to design modularised software and decoupling application liabilities, making software easier to understand and maintain

5. Problem Solving

5.1. Understanding computational, storage, resource, architectural, and design considerations that impact software maintenance, scalability, and performance

5.2. Ability to identify optimal approaches to various technical problem sets, as well as considering various edge cases

The ICA framework was built based on these skills, ensuring that content designed under the framework will be relevant and valid for assessing the coding skill of senior software engineering candidates across a wide range of roles, companies, and industries.

Conclusion

CodeSignal's Industry Coding Assessment is the tool to help companies move fast in evaluating the core coding skills of mid-to-senior engineers in a more consistent, accurate, and fair manner than a traditional technical phone screens or other hiring practices. The ICA is a research-backed framework to build standardized assessments that can reliably and accurately assess the coding skills of mid-to-senior software engineering candidates at scale. The assessments from this framework mimics real-world scenarios with multilevel tasks that start with basic requirements and gradually increase in complexity, requiring candidates to write, test, and refactor code. The ability to automate and validate the assessment and its results allows engineering teams to gain a stronger understanding of candidates' coding skills while reducing biases in the hiring process. The ICA also reduces internal engineering time spent designing and maintaining questions, conducting technical phone screens, and scoring responses. For candidates, the experience is more seamless than ever before. They are able to work with realistic, project-based tasks that allow them to effectively demonstrate their coding skills while moving faster through hiring pipelines.

References

[1] Occupational Outlook Handbook, Software Developers, Quality Assurance Analysts, and Testers. Bureau of Labor Statistics, U.S.: https://www.bls.gov/ooh/computer-and-information-technology/software-developers.htm#tab-6

[2] Lying on a resume (2020 study), ResumeLab: https://resumelab.com/resume/lying

[3] Whitened Résumés: Race and Self-Presentation in the Labor Market. Kang, S. K., DeCelles, K. A., Tilcsik, A., & Jun, S. (2016), Administrative Science Quarterly: https://doi.org/10.1177/0001839216639577

[4] Implicit Age Cues in Resumes: Subtle Effects on Hiring Discrimination. Derous, E. & Decoster, J. (2017), Frontiers in Psychology: https://doi.org/10.3389/fpsyg.2017.01321

[5] Coding Bootcamp Market size to grow by USD 772.04 million | Increase in Student Enrollments as a Key Driver | 17000+ Technavio Reports. Cision PR Newswire: https://www.prnewswire.com/news-releases/coding-bootcamp-market-size-to-grow-by-usd-772-04-million--increase-in-student-enrollments-as-a-key-driver--17000-technavio-reports-301425560.html

[6] See 10 Years of Average Total Student Loan Debt. Kerr, E., & Wood, S. (2021), U.S. News: https://www.usnews.com/education/best-colleges/paying-for-college/articles/see-how-student-loan-borrowing-has-risen-in-10-years

[7] 2021 Developer Survey. StackOverflow: https://insights.stackoverflow.com/survey/2021#overview

[8] Employment Tests and Selection Procedures. U.S. Equal Employment Opportunity Commission (EEOC): https://www.eeoc.gov/laws/guidance/employment-tests-and-selection-procedures

[9] Types of Interviewing Bias and How To Minimize It. Indeed (2021): https://www.indeed.com/career-advice/interviewing/interviewing-bias

[10] Tech Sector Job Interviews Assess Anxiety, Not Software Skills. North Carolina State University (2020), ScienceDaily: https://www.sciencedaily.com/releases/2020/07/200714101228.htm

[11] What Makes a Great Software Engineer. Li, P. L. (2016), University of Washington: http://hdl.handle.net/1773/37160

[12] The Future of Jobs Report 2020. World Economic Forum (October, 2020): https://www3.weforum.org/docs/WEF_Future_of_Jobs_2020.pdf

[13] 2021 Recruiting Trends Shaped by the Pandemic. Maurer, R. (2021), Society for Human Resource Management (SHRM): https://www.shrm.org/resourcesandtools/hr-topics/talent-acquisition/pages/2021-recruiting-trends-shaped-by-covid-19.aspx

[14] 2021 State of Software Engineers. Hired: https://hired.com/state-of-software-engineers

[15] How, When, and Why Bad Apples Spoil the Barrel: Negative Group Members and Dysfunctional Groups. Felps, W., Mitchell, T. R., & Byington, E. (2006), Research in Organizational Behavior: https://doi.org/10.1016/S0191-3085(06)27005-9

[16] How Long Does it Take to Hire? Interview Duration in 25 Countries. Chamberlain, A. (2017), Glassdoor: https://www.glassdoor.com/research/time-to-hire-in-25-countries/

[17] Occupational Outlook Handbook, Computer Programmers. Bureau of Labor Statistics, U.S.: https://www.bls.gov/ooh/computer-and-information-technology/computer-programmers.htm

[18] Occupational Outlook Handbook, Software Developers. Bureau of Labor Statistics, U.S.: https://www.bls.gov/ooh/computer-and-information-technology/software-developers.htm

[19] 15-1252.00 - Software Developers. O*NET OnLine, Department of Labor, U.S.: https://www.onetonline.org/link/summary/15-1252.00

[20] Top Interview Questions for Software Engineers. LinkedIn: https://business.linkedin.com/talent-solutions/resources/interviewing-talent/software-engineer?trk=lts-pros-TOFU-2020-tech&veh=lts-pros-TOFU-2020-tech#hardskills

[21] Your Guide to Hiring a Software Engineer. LinkedIn: https://business.linkedin.com/talent-solutions/resources/talent-acquisition/how-to-hire-guides/how-to-hire-a-software-engineer#skills

[22] The Tech Talent War Is Global, Cross-Industry, and a Matter of Survival. Frick, J., George, K. C., & Coffeman, J., Bain & Company (2021): https://www.bain.com/insights/tech-talent-war-tech-report-2021

[23] Developer Velocity: How software excellence fuels business performance. Srivastava, S., Trehan, K., Wagle, D., & Wang, J., McKinsey & Company (2020): https://www.mckinsey.com/industries/technology-media-and-telecommunications/our-insights/developer-velocity-how-software-excellence-fuels-business-performance

[24] The Future of Jobs Report 2020. World Economic Forum (2020): https://www3.weforum.org/docs/WEF_Future_of_Jobs_2020.pdf

Author Bios

Dmitry Filippov is the Technical Assessment Lead at CodeSignal, where he manages content development for all of CodeSignal's technical assessments. He received his Master's degree in Computer Science from ITMO University, and is an active member of the International Collegiate Programming Contest community for over 10 years.

Vitaliy Plastinin is a Technical Assessment Developer at CodeSignal, where he creates and maintains the content used in a wide variety of technical assessments. He received a degree in Applied Math and Programming from St. Petersburg Polytechnic University. Vitaly has experience in several areas of programming, including competitive programming, industrial C++ development, and signal processing.

Sylvia Lin is a Senior Solutions Engineer at CodeSignal, where she advises fast-growing companies across a variety of industries on best practices for implementing scalable talent solutions. She received a Computer Science degree and a Business degree from Brandeis University. Prior to CodeSignal, Sylvia worked as a Sales Engineer at Pegasystems.

Yi Zhuan Foong is a Solutions Engineering Lead at CodeSignal, where he consults with leading companies across technology, finance, and education industries to solve the talent problem. He is also a published author, course instructor, and blockchain analytics advisor.

Frank Mu is a Senior Assessment Research Manager at CodeSignal, where he conducts research to ensure the scientific rigor behind CodeSignal assessments in terms of how they are developed, monitored, and used. He received his PhD in Industrial-Organizational Psychology from the University of Waterloo and is an active member in the Society for Industrial-Organizational Psychology (SIOP), serving as a volunteer on the Membership Analytics Subcommittee.

Appendix: Sample Task

Scenario

Your task is to implement a simplified version of a file hosting service.

All operations that should be supported are listed below. Partial credit will be granted for each test passed, so press "Submit" often to run tests and receive partial credits for passed tests. Please check tests for requirements and argument types.

Implementation Tips

Read the question all the way through before you start coding, but implement the operations and complete the levels one by one, not all together, keeping in mind that you will need to refactor to support additional functionality.

Please, do not change the existing method signatures.

Task

Example of fire structure with various files:

```
----- [server34] ----- 24000 Bytes Limit -------
---------------------- Size -------
+- file-1.zip 4321 Bytes
+- dir-a
| +- dir-c
| | +- file-2.txt 1100 Bytes
| | +- file-3.csv 2122 Bytes
+- dir-b
| +- file-4.mdx 3378 Bytes
```

Level 1

- `FILE_UPLOAD(file_name, size)`
  - Upload the file to the remote storage server
  - If a file with the same name already exists on the server, throws a runtime exception
- `FILE_GET(file_name)`
  - Returns size of the file, or nothing if the file doesn't exist
- `FILE_COPY(source, dest)`
  - Copy the source file to a new location
  - If the source file doesn't exist, throws a runtime exception
  - If the destination file already exists, overwrites the existing file

Level 2

- `FILE_SEARCH(prefix)`
    - find top 10 files starting with the provided prefix. Order results by their size in descending order, and in case of a tie by file name.

Level 3

Files now might have a specified time to live on the server. Implement extensions of existing methods which inherit all functionality but also with an additional parameter to include a timestamp for the operation, and new files might specify the time to live - no ttl means life-time being infinite.

- `FILE_UPLOAD_AT(timestamp, file_name, file_size)`
- `FILE_UPLOAD_AT(timestamp, file_name, file_size, ttl)`
    - The uploaded file is available for ttl seconds.
- `FILE_GET_AT(timestamp, file_name)`
- `FILE_COPY_AT(timestamp, file_from, file_to)`
- `FILE_SEARCH_AT(timestamp, prefix)`
    - Results should only include files that are still “alive”

Level 4 – Extending Design Functionality

- `ROLLBACK(timestamp)`
    - Rollback the state of the file storage to the state specified in the timestamp
    - All ttls should be recalculated accordingly