General Coding Skills Evaluation Framework

Technical Research Paper

Albert Sahakyan and Tigran Sloyan

CodeSignal

Skills Evaluation Lab

Published July 2019; Updated April 2023

Abstract – As the focus on digital transformation and software development grows in modern organizations, there is an ever-increasing demand for skilled software developers and engineers. This demand for talent has led to a surge in demand for scalable hiring solutions that can efficiently assess the technical capabilities of job candidates, especially their coding skills. However, due to the lack of standardized assessments, many organizations rely on idiosyncratic and inefficient hiring practices that are difficult to scale and provide poor signals about candidates. To address this, we introduce the General Coding Skills Evaluation Framework (aka General Coding Framework), a blueprint for creating Certified Evaluations that provide reliable, high-quality, fair, and objective signals about the core coding skills required for software engineering jobs across industries. Such evaluations produce Coding Scores, which are easily understood metrics that quantify candidates' core coding skills. We discuss research on core coding skills captured by the framework, and present population-level data from General Coding Framework Certified Evaluations. Adopting evaluations created from the General Coding Framework can revolutionize hiring processes, enabling organizations to identify top technical talent better and faster than the competition.

Introduction

In today's business landscape, digital transformation, especially in the context of software development, has become a primary source of competitive advantage for organizations. This is reflected in the widely-held belief that "every company is now a software company" [1]. To thrive in this environment, organizations must find ways to identify top technical talent better and faster than the competition. Accordingly, demand for technical talent, particularly software developers or engineers, has grown rapidly in recent years. In fact, the demand for software engineer jobs is projected to grow at an accelerated rate of more than 25% between 2021 to 2031 [2].

Unfortunately, the status quo in today's technical recruiting and hiring processes is to use resumes as a proxy for skill, which leads to biased and inefficient recruiting and evaluation practices. Even among organizations that don't rely on resumes, current hiring processes tend to be inefficient and ineffective, either by requiring too much time for senior engineers to manually vet candidates through time-intensive interviews or through the use of inefficient evaluations that do not accurately or consistently capture candidates' skills. As a result, organizations are becoming increasingly concerned with the structure, consistency, and scalability of their hiring processes for software engineers. Such concerns call for a standardized approach to creating automated evaluations, which will allow organizations to evaluate the skills of software engineering candidates with a high degree of accuracy, consistency, and fairness while enabling them to scale to meet the growing demand for technical talent.

This paper describes a framework for developing simulation-based evaluations that accurately capture high-quality signals of the

technical skills held by candidates applying to software engineering jobs at scale. Framework-based evaluations are expertly designed and highly structured, allowing engineering and talent teams to efficiently scale their hiring process and make effective hiring decisions while providing a fair and engaging experience for candidates.

Generally, the most essential task for software engineers is creating software by writing well-designed and efficient code that solves problems. To succeed, software engineers, particularly those who are in the initial stages of their careers, must possess core coding skills, including 1) basic coding concepts, 2) data structures and manipulation, 3) implementation efficiency and constraints, and 4) solving problems through algorithms.

Although some forward-thinking organizations have started to create automated assessment tools internally to evaluate these skills, there are two major problems with this approach. Firstly, the internal teams rarely include experts in measurement or test design, which may lead to the creation of assessments that are not job-relevant or legally defensible - i.e., may not be compliant with assessment standards prescribed by the Equal Employment Opportunities Commission (EEOC) in the US [3]. Secondly, after internal teams spend substantial time creating these assessments, candidates may easily leak the content online by posting the questions on sites like Glassdoor and Stack Overflow. Once leaked, candidates may easily get the answers beforehand, so the content may no longer be valid in evaluating technical skills.

The General Coding Skills Evaluation Framework described in this paper can be used to create Certified Evaluations to measure core coding skills. This Framework was developed based on researching software engineering jobs and consultation with subject matter experts. Certified Evaluations powered by this framework are designed to assess the key knowledge and skills that are: 1) generally taught in computer science programs (including coding boot camps) and 2) commonly required for software engineering roles across a wide variety of organizations and industries.

The research behind this Framework helps to ensure that the content is job-relevant and legally defensible or compliant with EEOC prescriptions. Moreover, using a Framework-backed approach allows the content to be scaled, with a large pool of question variations that adhere to the same specifications. With this approach, candidates are presented with different but highly consistent questions across Certified Evaluations. This scalability reduces the ever-growing risk of candidates gaining an unfair advantage by accessing leaked questions in advance.

All Certified Evaluations powered by the General Coding Framework provide hiring organizations with a Coding Score, a holistic and straightforward metric quantifying the candidates' core coding skills. Coding Scores can be used by recruiters and hiring managers to evaluate candidates at scale.

Framework Specifications

The General Coding Skills Evaluation Framework is designed to simulate the typical coding workflow so candidates can display their core coding-related knowledge and skills required for software engineering roles. The purpose of this framework is to provide a blueprint for developing valid and reliable evaluations of candidates' role-relevant skills for software engineering and related roles at scale. The framework can be utilized to create evaluations that span across

different delivery methods, such as pre-screen assessments or technical interviews, while providing objective signals by automatically generating scores to quantify candidates' skills.

Evaluations based on this framework consist of four modules, with one question each, that require candidates to write code based on specified requirements. Each module has a slightly different focus, but all modules are designed to capture one or more of these core coding skills:

1. Basic Coding
2. Data Manipulation
3. Implementation Efficiency
4. Problem Solving

Candidates are given an opportunity to demonstrate their skills by effectively solving questions within the modules. For example, solving Basic Coding questions demonstrates skill in writing basic code to conduct basic operations such as working with numbers, strings, and arrays. Similarly, solving Problem Solving questions demonstrates understanding of challenging computing problems and knowledge of algorithms that can efficiently solve those problems (e.g., greedy, two pointers, etc.).

To balance the breadth and depth of the evaluation content with the goals of fostering a positive candidate experience, **the maximum allowed time for this framework is 70 minutes (for 4 code writing questions)**. Longer evaluations allow for increased measurement precision and improve the quality of signal–however, the more time-intensive evaluations become, the more reluctant candidates are to complete them. Moreover, solving the questions in the given timeframe is an important indicator of skill and a key factor in differentiating between candidate skill levels. This time-constrained process simulates on-the-job demands, as software engineers often balance multiple tasks simultaneously. Additionally, offering a limited, 70-minute timeframe helps prevent candidates from engaging in behaviors such as spending time searching for answers online, further promoting the validity of evaluations powered by the framework.

The following sections outline specifications for each module within the General Coding Framework at a high level. These specifications can be used to create variations of questions while ensuring evaluation results are comparable across candidates and attempts.

Module 1 – Basic Coding

This module contains **one coding question** focusing on basic coding concepts and operations. On average, candidates are expected to write **5-10 lines of code** and solve this within **10 minutes**.

Expected Knowledge

- Basic operations with numbers
- Basic string manipulation, such as splitting a string into substrings or modifying the elements of a string
- Basic array manipulation, such as iterating over an array

Can Include

- Questions that require a combination of 2 to 3 basic concepts, such as conditionally iterating over an array, or conditionally splitting a string
- Questions that should generally be solvable using a single loop
- Clear descriptions of implementation with step-by-step instructions

Should Exclude

- Questions that require noticing or proving patterns
- Questions that require knowledge of basic algorithms or optimization
- Questions that require designing or figuring out implementation details

Module 2 – Data Manipulation

This module contains **one coding question** focusing on manipulating data structures. On average, candidates are expected to write **10-20 lines of code** and solve this within **15 minutes**.

Expected Knowledge

- Working with numbers, including
  - Basic operations with numbers
  - Splitting numbers into digits
- Basic string manipulation
  - Splitting a string into substrings
  - Comparing strings
- Modifying elements of a string
  - Concatenating strings
  - Reversing strings
- Basic array manipulation
  - Iterating over an array
  - Modifying the elements of an array
  - Reversing an array
  - Concatenating two arrays

Can Include

- Questions that require a combination of 3 to 5 basic concepts, for example:
  - Splitting a string into substrings, then modifying each substring and comparing each with other substrings
  - Iterating over an array to split into two arrays, then modifying the second array and appending it to the first array
- Questions that should generally be solvable using 1 to 2 nested loops
- Clear descriptions of implementation with step-by-step instructions

Should Exclude

- Questions that require noticing or proving patterns
- Questions that require knowledge of basic algorithms or optimization

Module 3 – Implementation Efficiency

This module contains **one coding question** focusing on implementing solutions that can run efficiently and adheres to execution time limits. On average, candidates are expected to write **25-40 lines of code** and solve this within **20 minutes**.

Expected Knowledge

- Includes everything from module 1 and module 2
- Splitting overall requirements into subtasks or functions
- Manipulating multidimensional arrays, for example:
  - Iterating over elements within nested arrays in a given order
  - Transposing or pivoting the rows and columns values in a 2D array
- Using built in hashmaps to store strings or integers as keys

Can Include

- Implementing a specific comparator for strings
- Implementing a specific merge function for arrays
- Other implementation challenges which require translating step-by-step instructions into code

Should Exclude

- Questions that require noticing or proving patterns
- Questions that require algorithms with advanced data structures, such as binary indexed trees
- Questions that require complex topics, such as graphs, number theory, or dynamic programming

Module 4 – Problem Solving

This module contains **one coding question** focusing on applying algorithmic techniques to implement optimal solutions. On average, candidates are expected to write **25-35 lines of code** and solve this within **30 minutes**.

Expected Knowledge

- Includes everything from module 1, module 2, and module 3
- Implementing common algorithms to optimize solutions, such as greedy, divide and conquer, and two pointers
- Implementing abstract data types such as hashmaps within solutions
- Discrete mathematics fundamentals

Can Include

- Questions that require implementing an appropriate algorithm, data structure, or technique
- Questions that require optimizing queries using data structures like hashmaps or sets

Should Exclude

- Questions designed like brain teasers
- Questions that require knowledge of specialized or advanced algorithms, such as Dijkstra, Kruskal, or Fast Fourier transform (FFT)
- Questions with complicated or time-consuming implementation steps that would be difficult to optimize

Framework Example Content

Below are example questions for each module within the framework. Similar questions are developed in accordance with framework specifications on an ongoing basis to minimize the impact of leaks that could result in cheating or plagiarism, as well as provide relevant and fair candidate experiences through changing industry standards.

Module 1 – Basic Coding

Given an array a, your task is to output an array b of the same length by applying the following transformation:

- For each `i` from 0 to `a.length - 1 inclusive`, `b[i] = a[i - 1] + a[i] + a[i + 1]`
- If an element in the `sum a[i - 1] + a[i] + a[i + 1]` does not exist, use `0` in its place
- For instance, `b[0] = 0 + a[0] + a[1]`

Example

For `a = [4, 0, 1, -2, 3]`:

- `b[0] = 0 + a[0] + a[1] = 0 + 4 + 0 = 4`
- `b[1] = a[0] + a[1] + a[2] = 4 + 0 + 1 = 5`
- `b[2] = a[1] + a[2] + a[3] = 0 + 1 + (-2) = -1`
- `b[3] = a[2] + a[3] + a[4] = 1 + (-2) + 3 = 2`
- `b[4] = a[3] + a[4] + 0 = (-2) + 3 + 0 = 1`

So, the output should be `solution(a) = [4, 5, -1, 2, 1]`.

Sample Solution (python)

```
1   def solution(a):
```

```
    n = len(a)
    b = [0 for _ in range(n)]
    for i in range(n):
        b[i] = a[i]
        if i > 0:
            b[i] += a[i - 1]
        if i < n - 1:
            b[i] += a[i + 1]
    return b
```

Module 2 – Data Manipulation

You are given two strings: `pattern` and `source`. The first string pattern contains only the symbols `0` and `1`, and the second string source contains only lowercase English letters.

Your task is to calculate the number of substrings of source that match pattern.

We'll say that a substring `source[l..r]` matches pattern if the following three conditions are met:

- The pattern and substring are equal in length.
- Where there is a `0` in the pattern, there is a vowel in the substring.
- Where there is a `1` in the pattern, there is a consonant in the substring.

Vowels are `["a", "e", "i", "o", "u", "y"]`. All other letters are consonants.

Example

For `pattern = "010"` and `source = "amazing"`, the output should be `solution(pattern, source) = 2`.

- `"010"` matches `source[0..2] = "ama"`. The pattern specifies "vowel, consonant, vowel". `"ama"` matches this pattern: `0` matches `a`, `1` matches `m`, and `0` matches `a`.
- `"010"` doesn't match `source[1..3] = "maz"`
- `"010"` matches `source[2..4] = "azi"`
- `"010"` doesn't match `source[3..5] = "zin"`
- `"010"` doesn't match `source[4..6] = "ing"`

So, there are `2` matches.

For `pattern = "100"` and `source = "codesignal"`, the output should be `solution(pattern, source) = 0`.

- There are no double vowels in the string `"codesignal"`, so it's not possible for any of its substrings to match this pattern.

Guaranteed constraints:

- `1 ≤ source.length ≤ 103`
- `1 ≤ pattern.length ≤ 103`

Example Solution (python)

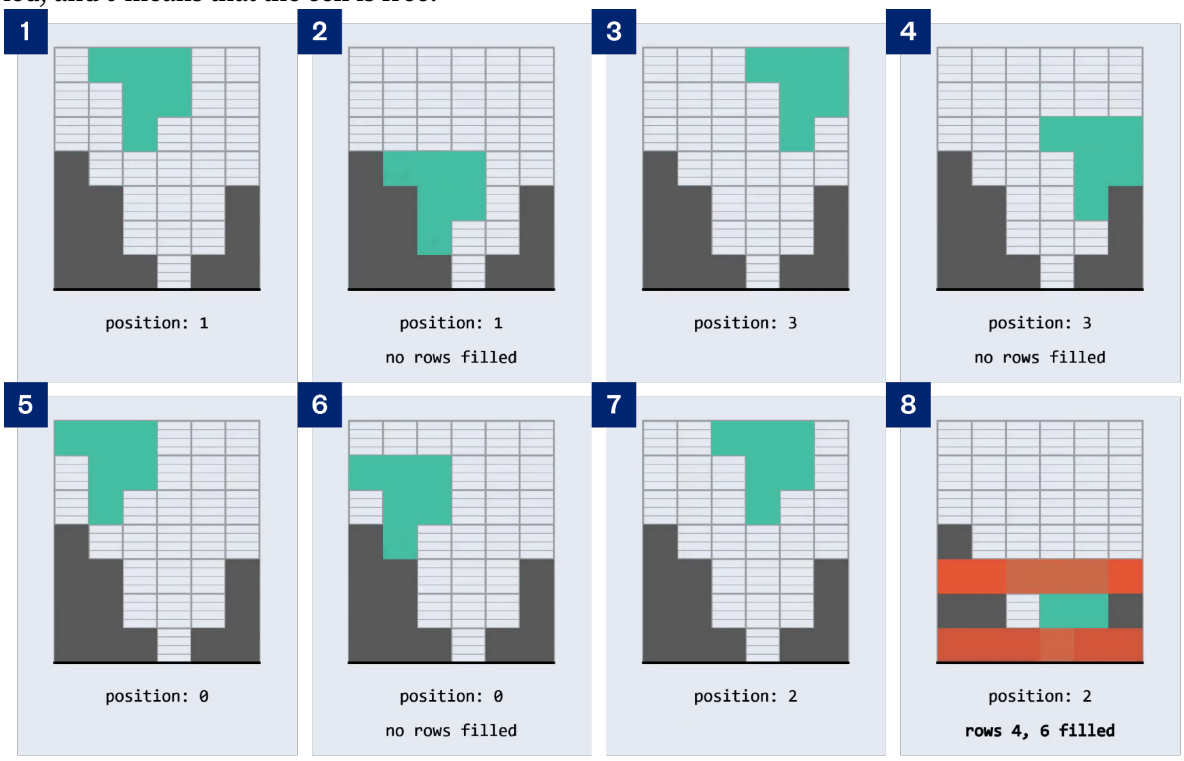
```
vowels = ['a', 'e', 'i', 'o', 'u', 'y']

def check_for_pattern(pattern, source, start_index):
   for offset in range(len(pattern)):
       if pattern[offset] == '0':
           if source[start_index + offset] not in vowels:
               return 0
       else:
           if source[start_index + offset] in vowels:
```

```
                return 0
        return 1
    def solution(pattern, source):
        answer = 0
        for start_index in range(len(source) - len(pattern) + 1):
            answer += check_for_pattern(pattern, source, start_index)
        return answer
```

Module 3 – Implementation Efficiency

You are given a matrix of integers field of size height × width representing a game field, and also a matrix of integers figure of size 3 × 3 representing a figure. Both matrices contain only 0s and 1s, where 1 means that the cell is occupied, and 0 means that the cell is free.



[1]

You choose a position at the top of the game field where you put the figure and then drop it down. The figure falls down until it either reaches the ground (bottom of the field) or lands on an occupied cell, which blocks it from falling further. After the figure has stopped falling, some of the rows in the field may become fully occupied.

Your task is to find the dropping position such that at least one full row is formed. As a dropping position, you should return the column index of the cell in the game field which matches the top left corner of the figure's 3 × 3 matrix. If there are multiple dropping positions satisfying the condition, feel free to return any of them. If there are no such dropping positions, return -1.

Note: The figure must be dropped so that its entire 3 × 3 matrix fits inside the field, even if part of the matrix is empty.

[1] The actual image is presented to candidates in an animated gif format, which can be viewed here.

Examples

For

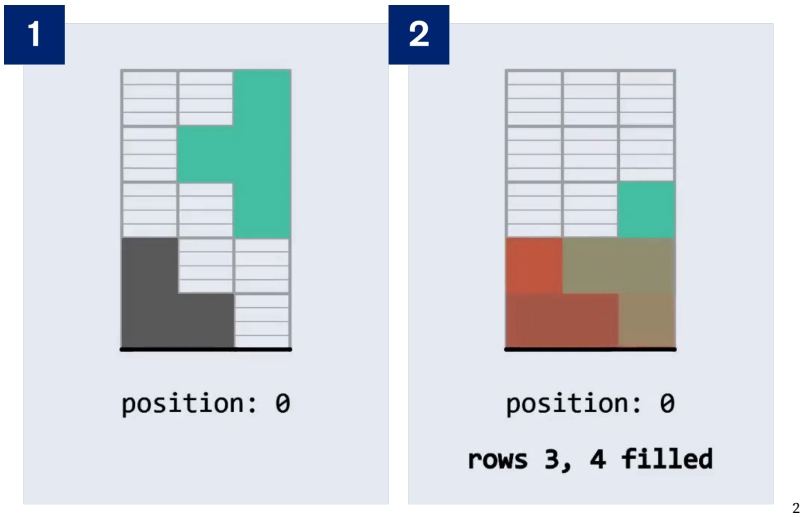
```
field = [[0, 0, 0],
         [0, 0, 0],
         [0, 0, 0],
         [1, 0, 0],
         [1, 1, 0]]
```

and

```
figure = [[0, 0, 1],
          [0, 1, 1],
          [0, 0, 1]]
```

The output should be `solution(field, figure) = 0`.

Because the field is a `3 x 3` matrix, the figure can be dropped only from position `0`. When the figure stops falling, two fully occupied rows are formed, so dropping position 0 satisfies the condition.



[2]

For

```
field =  [[0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0],
          [1, 1, 0, 1, 0],
          [1, 0, 1, 0, 1]]
```

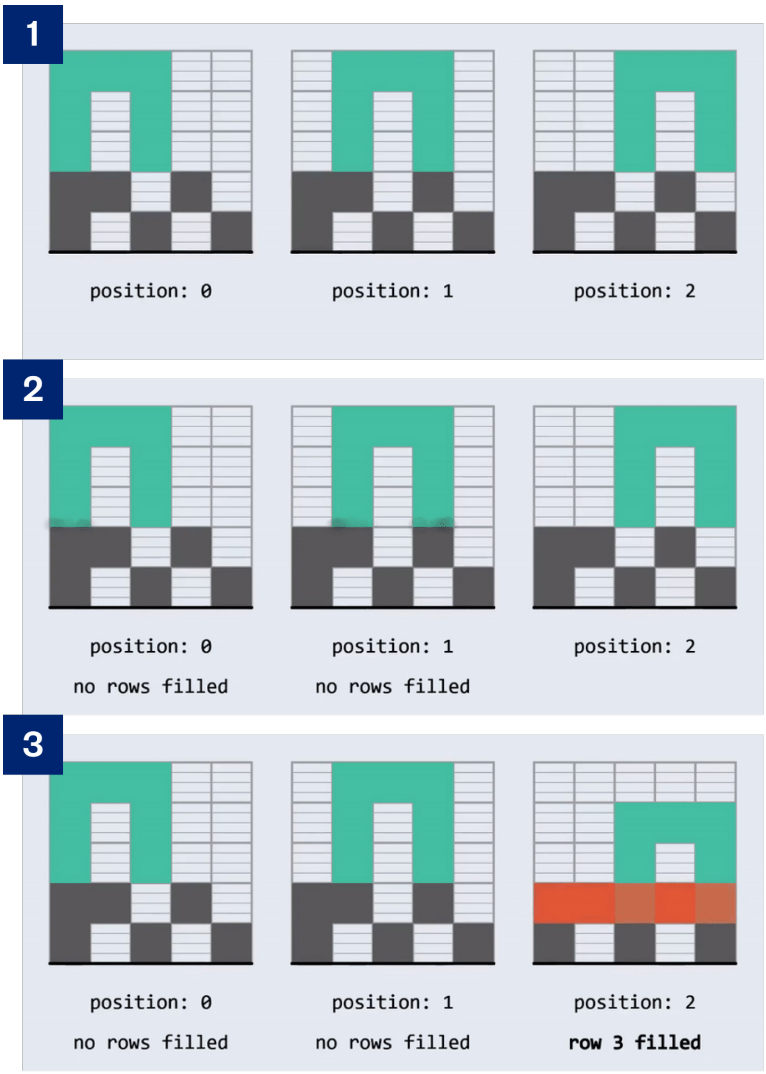
and

```
figure = [[1, 1, 1],
          [1, 0, 1],
          [1, 0, 1]]
```

the output should be `solution(field, figure) = 2`.

The figure can be dropped from three positions: `0`, `1`, and `2`. As you can see below, a fully occupied row will be formed only when dropping from position `2`:

[2] The actual image is presented to candidates in an animated gif format, which can be viewed here.



[3]

Sample Solution (python)

```python
def solution(field, figure):
    height = len(field)
    width = len(field[0])
    figure_size = len(figure)

    for column in range(width - figure_size + 1):
        row = 1
        while row < height - figure_size + 1:
            can_fit = True
            for dx in range(figure_size):
                for dy in range(figure_size):
                    if field[row + dx][column + dy] == 1 and figure[dx][dy] == 1:
                        can_fit = False
            if not can_fit:
                break
            row += 1
        row -= 1

```

[3] The actual image is presented to candidates in an animated gif format, which can be viewed here.

```
        for dx in range(figure_size):
            row_filled = True
            for column_index in range(width):
             if not (field[row + dx][column_index] == 1 or
                     (column <= column_index < column + figure_size and\
                   figure[dx][column_index - column] == 1)):
                 row_filled = False
            if row_filled:
                return column
    return -1
```

Module 4 – Problem Solving

Given an array of unique integers numbers, your task is to find the number of pairs of `indices (i, j)` such that `i ≤ j` and the sum `numbers[i] + numbers[j]` is equal to some power of `2`. Note: The numbers 20 = 1, 21 = 2, 22 = 4, 23 = 8, etc. are considered to be powers of 2.

Examples

For `numbers = [1, -1, 2, 3]`, the output should be `solution(numbers) = 5`.

- There is one pair of indices where the sum of the elements is 20 = 1: (1, 2): `numbers[1] + numbers[2] = -1 + 2 = 1`
- There are two pairs of indices where the sum of the elements is 21 = 2: `(0, 0)` and `(1, 3)`
- There are two pairs of indices where the sum of the elements is 22 = 4: `(0, 3)` and `(2, 2)`
- In total, there are `1 + 2 + 2 = 5` pairs summing to powers of `2`.

For `numbers = [2]`, the output should be `solution(numbers) = 1`.

- The only pair of indices is `(0, 0)` and the sum is equal to 22 = 4. So, the answer is `1`.

For `numbers = [-2, -1, 0, 1, 2]`, the output should be `solution(numbers) = 5`.

- There are two pairs of indices where the sum of the elements is 20 = 1: `(2, 3)` and `(1, 4)`
- There are two pairs of indices where the sum of the elements is 21 = 2: `(2, 4)` and `(3, 3)`
- There is one pair of indices where the sum of the elements is 22 = 4: `(4, 4)`
- In total, there are 2 + 2 + 1 = 5 pairs summing to powers of `2`.

Guaranteed constraints:

- `1 ≤ numbers.length ≤ 105`
- `-106 ≤ numbers[i] ≤ 106`

Sample Solution (python)

```
from collections import defaultdict

 def solution(numbers):
   counts = defaultdict(int)
   answer = 0
   for element in numbers:
       counts[element] += 1
       for two_power in range(21):
           second_element = (1 << two_power) - element
           answer += counts[second_element]
   return answer
```

Evaluation Scoring

To effectively quantify core coding skills required for software engineering roles, questions within the General Coding Framework prompt candidates to design and create software by understanding user requirements and writing code to create functional applications. Consistent with this design, General Coding Framework Certified Evaluations are scored via unit testing. Given user requirements on expected functions or behaviors of software, a set of test cases verify the functionality of different components, or units, within the software by providing sample input (i.e., data, context, conditions) and checking whether the software produces expected outcomes consistent with user requirements. Specifically, test cases to verify whether the code submitted by candidates as solutions can meet the requirements of that specific question by producing expected outputs. Moreover, since each test case generally has a binary correct vs. incorrect outcome, all questions are designed with a set of test cases to cover complex and multifaceted requirements within questions.

For Certified Evaluations created from the General Coding Framework, the solution to each question is scored by calculating the proportion of test cases passed/solved over the entire set of test cases for that question. All test cases are equally weighted within and across questions. Since scores for individual questions are aggregated into a Coding Score, overall candidate performance is ultimately based on the total number of test cases passed by their solutions across all questions within the Framework.

All evaluations created from the General Coding Framework will produce a Coding Score[4], which is designed to be a valid, reliable, and straightforward representation of a candidate's core coding skills. **Coding Scores range between 200 (lowest) to 600 (highest)**, increasing as candidates partially or fully solve each of the questions within the Framework. Figure 1 presents the distribution of Coding Scores among a large sample of candidates who have attempted a General Coding Framework Certified Evaluation, and Table 1 describes skill profiles of candidates based on which modules they fully solved in their evaluation.

Although the Coding Score does not explicitly incorporate speed of execution or efficiency in its calculation, efficiency and speed are implicitly captured due to the time limit. Candidates who are able to work efficiently will be able to complete more requirements and solve more questions in the allotted time frame, resulting in higher scores than candidates who are not able to work efficiently.

Research Identifying Core Coding Skills

Our primary aim in developing the General Coding Skills Evaluation Framework is to facilitate the creation of standardized, job-relevant, and simulation-based measures that can effectively evaluate the core technical skills required for software engineering roles across industries and at scale. Since the primary activity shared across most software engineer roles is writing code to create software, we started by identifying the most common topics relevant for coding skills via the following research questions:

[4] The latest version of CodeSignal's Coding Score system is called Coding Score 2023, which differs from previous versions of Coding Score. Please see this support article for more details.

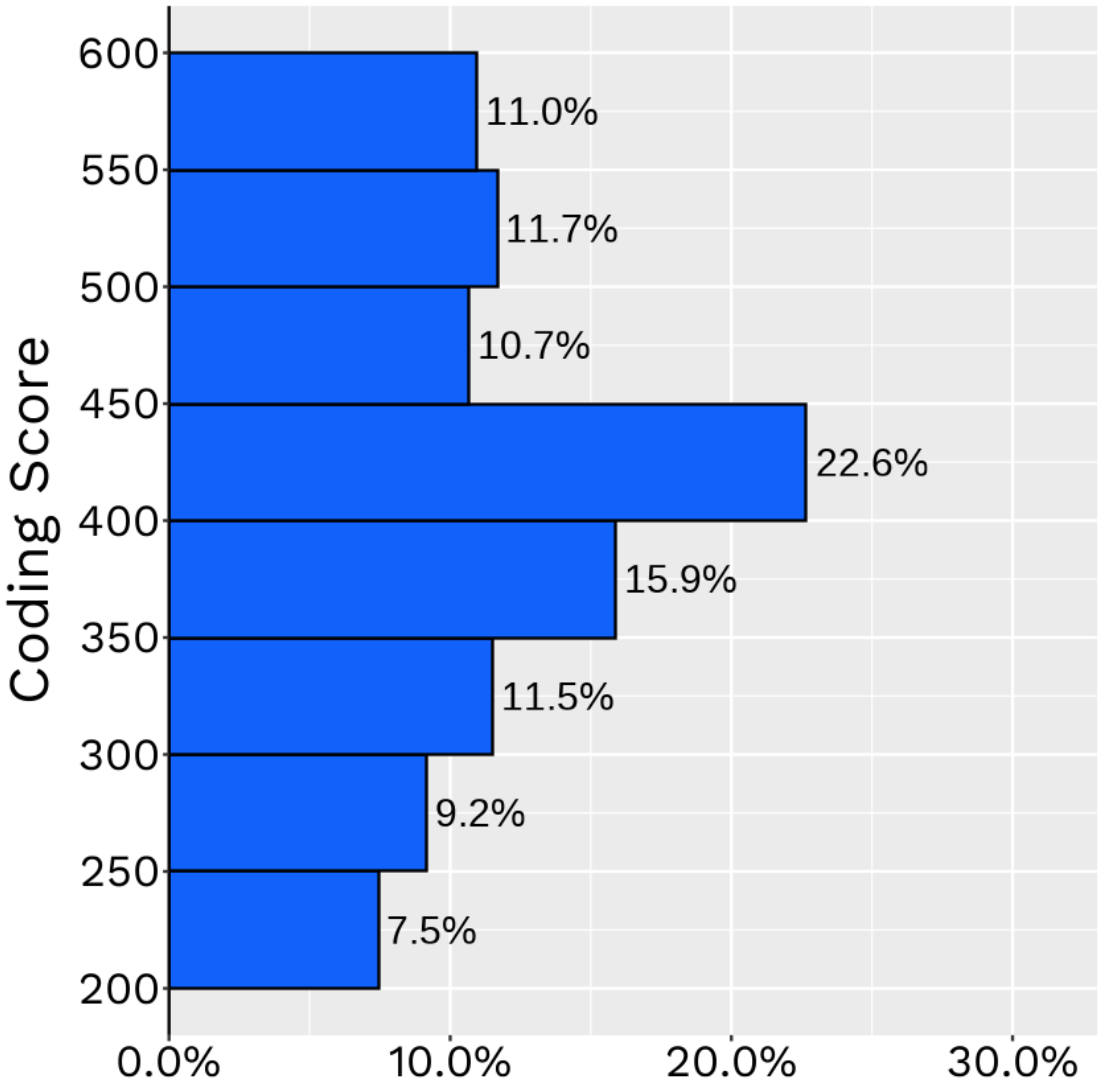


Figure 1. Coding Score distribution for a sample of N > 200,000 candidates who have attempted a General Coding Framework Certified Evaluation.

1. What are the most common topics taught in reputable computer science programs at post-secondary educational institutions in the US?
2. What are the most common topics covered during technical interviews at innovative and accomplished organizations in the US?
3. What are the most frequently asked questions on online communities for developers, such as Stack Overflow, on general programming concepts and not specialized domain knowledge?

To address question 1, we reviewed syllabi from courses on MIT OpenCourseWare (OCW) [4], EdX [5], Coursera [6], and Udacity [7]. To address question 2, we reviewed questions from technical interview preparation resources [8], CodeSignal Interview Practice Mode [9], Leetcode [10], CareerCup [11], and

Glassdoor [12]. To address question 3, we scraped data from Stack Overflow's public API [13]. After gathering a wide range of topics, we conducted thematic analysis [14] to identify higher-order themes through an iterative process. Specifically, we identified themes by:

1. Thoroughly reviewing all of the documented topics.
2. Analyzing similarities and trends among the topics.
3. Grouping topics together based on semantic similarities and relationships with other topics
4. Iteratively repeating the above steps until coherent themes were formed
5. Naming and defining themes based on underlying topics

Score	Solved Questions	Candidate Skill Profile*
200 – 279	None	The candidate may be able to write simple code to perform some operations.
296	1	The candidate is familiar with programming and can write simple code to perform some operations.
396	1 & 2	The candidate has solid implementation skills, can solve some algorithmic tasks, and can work with built-in data types and implement the desired solution. Most tech companies require only these skills for the job.
496	1, 2, & 3	The candidate has good problem-solving skills, is familiar with algorithms, and can implement ideas that don't require a high degree of innovative thinking.
500	1, 2, & 4	The candidate has great algorithmic, problem-solving, and implementation skills and can develop complex applications.
600	1, 2, 3, & 4	The candidate has excellent algorithmic, problem-solving, and implementation skills and can develop large, complex applications efficiently.

Note: Actual candidate skill profiles will vary depending on individual performance factors.

Table 1. Score Guidelines for General Coding Framework Certified Evaluations

Results of the thematic analysis highlighted **4 major themes** that represent core aspects of coding skills required by all professional software engineers jobs, especially for early talent or entry-level jobs. The first major theme is **Basic Coding.** This was derived from foundational topics and concepts that allow candidates to write 5-10 lines of

basic code which executes a simple operation, such as knowing the syntax for using built-in functions to transform an array in a simple way. Because such topics set the foundation for all code writing activities, this is an important skill for all software engineering jobs.

The second major theme is **Data Manipulation**. This was derived from topics related to working standard data structures in computing operations, such as strings and arrays. Prevalence of such topics in interview questions suggests that proficiency in manipulating standard data structures is emphasized in hiring processes for organizations that are not explicitly focused on technology. As such, this is an important skill for most software engineering jobs.

The third major theme is **Implementation Efficiency.** This was derived from topics related to translating user requirements into functional code that runs at a reasonable speed. Specifically, given that efficiency is a core tenant in software development, writing code that accounts for time complexity and can execute within a strict time limit is an important activity for software engineers. Thus, this is an important skill required by many software engineering jobs. The fourth major theme is **Problem Solving**. This was derived from topics related to applying algorithmic techniques to create optimal solutions for moderately complex computing problems. Given that algorithms provide a structured approach to finding solutions that are efficient, accurate, and consistent, they are a crucial aspect of software engineering and development. Thus, this is an important skill required by many software engineering jobs.

After uncovering the core themes discussed above, we verified the relevance of these skills for software engineering and development jobs with an advisory board of subject matter experts in hiring software engineers across a wide range of organizations and industries (see Acknowledgements). Based on expert consensus, we finalized the themes and topics that represent the core coding skills captured by the General Coding Skills Evaluation Framework.

References

[1] Gnanasambanda, C., Palaniappan, J., & Schneide, J. (2022). Every company is a software company: Six 'must dos' to succeed. *McKinsey Digital*: https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/every-company-is-a-software-company-six-must-dos-to-succeed

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

[3] U.S. Equal Employment Opportunities Commission. (2007). *Employment Tests and Selection Procedures*: https://www.eeoc.gov/laws/guidance/employment-tests-and-selection-procedures

[4] MIT OpenCourseWare: https://ocw.mit.edu/

[5] edX: https://www.edx.org/

[6] Coursera: https://www.coursera.org/

[7] Udacity: https://www.udacity.com/

[8] McDowell, G. L. (2015). *Cracking the Coding Interview: 189 Programming Questions and Solutions*: https://isbnsearch.org/isbn/0984782869

[9] CodeSignal: https://app.codesignal.com/interview-practice

[10] LeetCode: https://leetcode.com/

[11] CareerCup: https://www.careercup.com/

[12] Glassdoor: https://www.glassdoor.com/

[13] Stack Exchange API: https://api.stackexchange.com/

[14] Braun, V., & Clarke, V. (2012). Thematic analysis. In H. Cooper, P. M. Camic, D. L. Long, A.

T. Panter, D. Rindskopf, & K. J. Sher (Eds.), *APA handbook of research methods in psychology, Vol. 2. Research designs: Quantitative, qualitative, neuropsychological, and biological* (pp. 57–71). American Psychological Association. https://doi.org/10.1037/13620-004

Acknowledgments

We are extremely grateful to the engineering advisory board of the #GoBeyondResumes movement for kindly reviewing and providing feedback on the General Coding Framework, including: Anima Anandkumar, Director of Machine Learning Research @ NVIDIA and professor @ Caltech; Chris Kanaan, SVP of Engineering @ Ripple; Jessica McKellar, CTO @ Pilot; Kah Seng Tay , VP of Engineering @ Drive.ai; Lei Yang, VP of Engineering @ Quora; Nate Kupp, Director of Engineering @ Thumbtack; Nimrod Hooen, Director of Engineering @ Facebook; Surabhi Gupta, Director of Engineering @ Airbnb; Yoann Roman, Director of Engineering @ Yelp.

We'd also like to thank the following for their detailed review and feedback on this paper.: Aram Shatakhtsyan, CTO @ CodeSignal; Eduard Piliposyan, Director of Engineering @ CodeSignal; and Michael Newman, VP of Engineering @ CodeSignal.

Authors

Albert Sahakyan is a Data Science researcher at Yerevan State University and an Engineering Manager at CodeSignal. He is an internationally recognized Software Engineer, who won a Bronze medal at the International Olympiad in Informatics in 2013 and was one of the World Finalists in the International Collegiate Programming Contest that took place in Rapid City, South Dakota in 2017. Albert received his Ph.D. in Mathematics from Yerevan State University in 2022.

Tigran Sloyan is the Co-Founder and CEO of CodeSignal, a leading technical interview and assessment platform that helps companies go Beyond the Noise™ with early-stage assessments, technical screens, and live coding interviews. As an active member of the Forbes Technology Council, Tigran regularly contributes to Forbes, Morning Brew, Fast Company, Recruiting Daily, and other major publications. Tigran is a frequent industry keynote, a TED speaker, and a thought leader in the technical hiring industry, commenting on trends in talent acquisition, diversity, and innovation. Tigran received BS degrees in both Mathematics and Computer Science with a minor in Economics from the Massachusetts Institute of Technology (MIT).