General Coding Skills Evaluation Framework

Albert Sahakyan and Tigran Sloyan

CodeSignal
Skills Evaluation Lab

Published July 2019

Abstract - In 2019, US-based universities will issue more than 50,000 computer science degrees, and only a small fraction of those degrees are issued from "top tier" CS programs. However, given the lack of standardized technical assessments and certifications, many companies hiring for technical talent still use pedigree as the main proxy for skill. This means that most companies are fighting over the same small set of candidates from top schools and most qualified developers struggle to get a single interview at any of these companies. In this paper, we introduce a framework for creating a General Coding Assessment (GCA) and a scoring system (Coding Score) that maps programming skills to a score from 300 to 850, combining information about problem solving and code-writing skills as well as speed and code quality. GCA aims to be a foundational assessment for all developers; in future papers, we will introduce more specialized assessments that could be used for more targeted certifications, such as a JavaScript Specialized Coding Assessment (SCA). We provide data-driven results on the performance of GCA certifications thus far, and we dive into the logic behind the choice of the topics and questions.

Introduction

The status quo in today's technical recruiting process is to use resume as a proxy for skill, which leads to biased and inefficient recruiting and evaluation practices. Some forward-looking companies are using automated assessment tools to create their own tests and to send those to candidates as the first step of the interview process. However, there are two major problems with that approach. Firstly, the internal teams creating these assessments are not experts in test design, which may lead to creation of tests that are not EEOC-compliant, or focus too much on some skills that might not be as important for the role while neglecting others that could be critical. Secondly, after internal teams spend a substantial amount of time creating these assessments, many of these assessments are eventually posted online on sites like Glassdoor and Stack Overflow, making the validity of the test results questionable.

The General Coding Assessment (GCA) framework described in this paper can be used to create standardized tests to measure core programming and computer science knowledge that is shared between most software engineers and is taught in almost all CS programs throughout the US. The research that has gone into it makes sure that the tests themselves are highly consistent and EEOC-compliant. In addition, basing the test on a framework allows the GCA to be scaled with a large pool of questions that adhere to the same guidelines. On each test administration, questions are randomly selected from the pool, reducing the risk that a test taker

gains an unfair advantage by memorizing the questions in advance.

The Coding Score obtained from the test is a singular measure meant to give recruiters, hiring managers and educators – as well as the test-takers themselves – a quick view of the test-taker's skills. It measures the test taker's code-writing and problem-solving skills as well as the ability to produce clean code at a reason able speed. GCA and the Coding Score it produces are not language specific, and test-takers can choose any programming language to complete the tasks. The test should be administered in a proctored environment, and the solutions should be automatically checked for plagiarism.

In the next two sections we will discuss how the score maps to different skills and knowledge, and we will draw guidelines for creating tasks based on the frame work. Afterwards, we will illustrate how the score is calculated and discuss our empirical findings from us ing the GCA in practice. At the end there is a detailed description of the ideal test administration process as well as an FAQ.

The Framework

The maximum allowed completion time for the test is **70 minutes** and each test has **4 tasks** in it. Even though a longer test (2-3 hours long) could produce a more accurate measurement, test-takers are significantly less willing to complete the assessment when they have to allocate more than an hour. Given the empirical results (see below) observed while administering GCA thus far, 70 minutes is enough to produce a reliable signal while keeping the friction low.

Each test contains a very basic task, a simple data manipulation task, an implementation task (requires writing at least 25-40 lines of code but the description clearly explains what needs to be done) and a problem solving task (where you need to come up with the approach yourself). Using this framework, tasks can be easily generated while keeping the test results consistent.

Creating tasks for the GCA

The 1st Task

Test-takers who solve only the 1st task will have a Coding Score from **650-674**. The score may vary depending on the completion speed, code quality and the number of incorrect attempts.

The average time for solving this task should be **10 minutes**.

Requires writing 5-10 lines of code.

Expected Knowledge

- Working with numbers.
    - Basic operations with numbers.
- Basic string manipulation.
    - Splitting a string into sub-strings.
    - Modifying the elements of a string.
- Basic array manipulation.
    - Iterating over an array.

Can Include

- Tasks that require a combination of 2 to 3 basic concepts. For example:
    - Iterating over an array and taking into account some condition.

    - Splitting a string by some condition.
- Should usually be solvable using one loop.
- The task description should clearly state the implementation steps.

Should Exclude

- Anything that requires noticing or proving a certain pattern.
- Anything that requires optimizing an algorithm.
- Anything that requires a knowledge of classic algorithms.
- Anything that requires good implementation skills.

Example

Given an array of integers a. The task is to return another array b where b[i] = a[i − 1] + a[i] + a[i + 1], (if an element does not exist it should be 0).

The 2nd Task

Test-takers who solve only the 2nd task will have a Coding Score from **688-712**. The score may vary depending on the completion speed, code quality and the number of incorrect attempts.

The average time for solving this task should be **15 minutes**.

Requires writing 10-20 lines of code.

Expected Knowledge

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

Can Include

- Tasks that require a combination of 3 to 5 basic concepts. For example:
    - Splitting a string into substrings, modifying each substring and comparing with other strings.
    - Iterating over an array to produce two new arrays given some conditions, modifying the second array and appending it to the beginning of the first array.
- Should usually be solvable using one to two nested loops.
- The task description should clearly state the implementation steps.

Should Exclude

- Anything that requires noticing or proving a certain pattern.
- Anything that requires optimizing an algorithm.
- Anything that requires a knowledge of classic algorithms.

Example

Given a list of words (consisting of lowercase English letters) and a complex word written in camelCase (consisting of English letters), check if the complex word consists of words from the given list.

The 3rd Task

Test-takers who solve only the 3rd task have a coding score from **719-743**. The score may vary depending on the completion speed, code quality and the number of incorrect attempts.

The average time for solving this task should be **20 minutes**.

Requires writing 25-40 lines of code.

Expected Knowledge

- Includes everything from the previous task.
- Splitting a task into smaller subtasks/ functions.
- Manipulating two-dimensional arrays.
    - Iterating over the elements in a particular order.
    - Modifying values.
    - Swapping rows/columns.
- Using hashmaps.
    - 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 that clearly explain what needs to be done and require translating the instructions into code.

Should Exclude

- Anything that requires noticing or proving a certain pattern.
- Anything that requires optimizing an algorithm with advanced data structures like binary indexed trees, etc.
- Anything from special topics like graphs, number theory, or dynamic programming.

Example

Given two strings, merge them with the following merge function: instead of comparing the characters in the usual lexicographical order, compare them based on how many times they occur in their respective strings.

Fewer occurrences mean the character is considered smaller; in case of equality, compare them lexicographically; in case of equality, take the character from the first string.

The 4th Task

Test-takers who solve only the 4th task have a coding score from **768-792**. The score may vary depending on the completion speed, code quality and the number of incorrect attempts.

The average time for solving this task should be **30 minutes**.

Requires writing 20-35 lines of code.

Expected Knowledge

- Includes everything from previous tasks.
- Working with trees.
    - Storing and traversing trees.

    - Transforming trees.
- Understanding hashmaps.
    - Solving tasks that require understanding the implementation of hashmaps.
- Fundamentals of discrete mathematics.
- Brute-force search.
    - Checking all possible solutions to find the optimal solution.

Can Include

- Tasks that require noticing an application of a certain algorithm, data-structure or technique.
- Optimizing some queries with the help of data structures like hashmaps or sets.
- Algorithms on trees like finding the longest path of a tree.

Should Exclude

- Brain teasers or tasks that require specialized knowledge of certain libraries or advanced algorithms like Dijkstra, Kruskal, FFT, etc.
- Tasks that require very long implementation. Unlike the second task, implementation is not key here; problem-solving is key, thus this task should ideally avoid taking up lots of implementation time.

Example

Create a data structure that allows the following operations.

insert x y - insert an object with key x and value y.

get x - return the value of an object with key x.

addToKey x - add x to all keys in map.

addToValue y - add y to all values in map.

Time complexity of each operation should be $O(\log(N))$

Results

The result of the GCA is a Coding Score based on the test-taker's solved tasks, speed, coding style and failed attempts. Coding Score is a number from **300** to **850**, and there are 7 score levels, as shown in Table 1.

Test-takers who solve all four tasks will usually get a coding score from **825-850**. Test-takers who solve the 1st, 2nd and 4th tasks will get a score from **800-824**. Test-takers who solve all the implementation tasks (the 1st, 2nd, 3rd) will have a score above **750**. Test-takers who solve the 1st and the 2nd task will be above the **700**. Test-takers who solve at least one task will be above **650**.

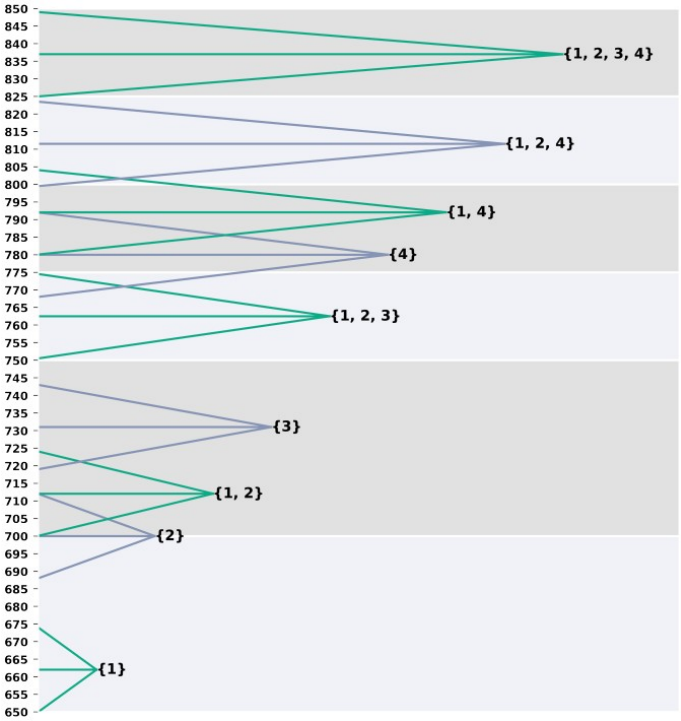

Figure 1: The score ranges based on which tasks a test-taker has solved.

Fig. 1 summarizes the scores based on which tasks a test-taker has solved. Only the most important outcomes are shown.

Fig. 2 and Fig. 3 are the distributions of scores on a sample of 1000 randomly selected Computer Science seniors currently studying at US-based universities. Fig. 4 is the scatter plot of the test-takers' scores (where at least one task was solved) in sorted order. Table 2 is a guide that describes some possible test results.

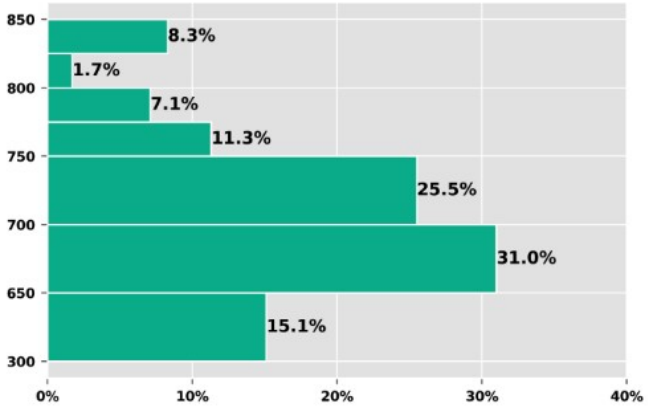

Figure 2: The percentage distribution of scores for a sample of 1000 students.

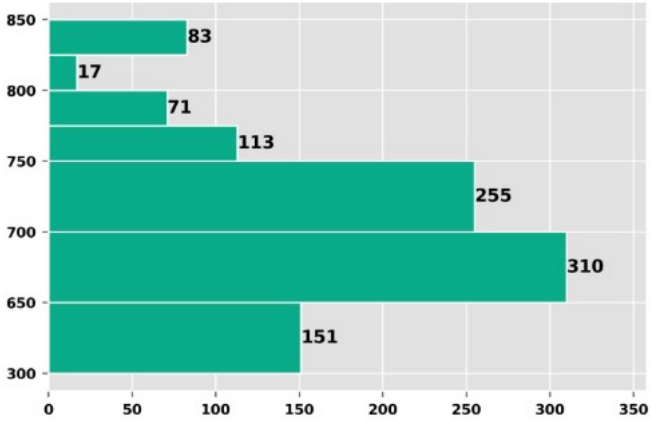

Figure 3: The histogram of scores for a sample of 1000 students.

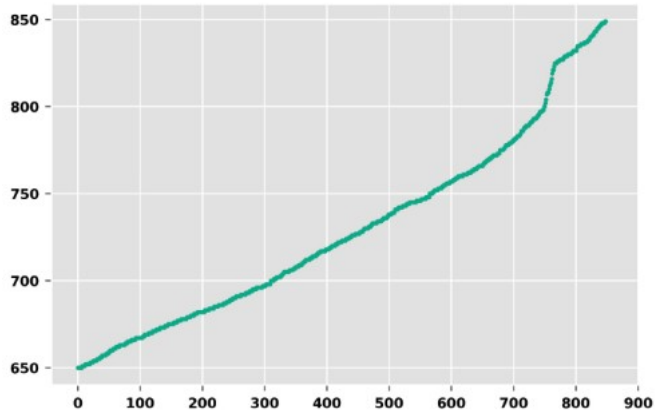

Figure 4: Student Coding Scores (where at least one task was solved) in sorted order.

Deep Dive

In the following sections we will describe the reasoning behind some of the topic choices for GCA and draw parallels between other similar standardized assessments.

Since GCA is designed to measure skills that are important for almost all software developers, we have aimed to find the common denominator between three different sources of data:

1) What are the most common topics

taught in different CS programs at 4 years Universities in the US?

2) What are the most common topics covered during technical interviews at successful US-based companies?

3) What are the most common questions asked on Stack Over ow that are about general programming and not specialized domain knowledge?

We've used MIT OCW [1], EdX [2], Coursera [3] and Udacity [4] course catalogs as a source for #1. We've used the book "Cracking the Coding Interview" [5], CodeSignal Interview Practice Mode [6], Leetcode [7], CareerCup [8] and Glassdoor [9] Interview Questions sections to identify #2. And Stackoverflow public API [10] for #3.

Based on the research, the common denominator expectation for all professional software developers (including all new grads) is the following:

1) **Basic Coding**

Ability to write a basic 5-10 lines of code that does one operation like transforming an array in some simple way. This way we can distinguish all the programmers from the rest.

2) **Data Manipulation**

Ability to work with standard data structures (like arrays and strings), which is covered by question 2 of the GCA test. Based on interview question analysis it seems that being highly comfortable with this skill seems to be enough to be successfully hired at most non-tech companies seeking engineers.

3) **Ease of Implementation**

Ability to translate specifications into clean code at a reasonable speed. This is usually required at non tech companies where tech plays a core role or tech companies where the product or the role doesn't have high technical complexity.

4) **Problem Solving**

Ability to apply more advanced algorithmic techniques to find an optimal solution to the given problem. This skill is required at most prestigious tech companies like Google and Facebook, independent of the technical role you are being hired for.

Level 1	Level 2	Level 3	Level 4	Level 5	Level 6	Level 7	Level 8	Level 9	Level 10
300-399	400-499	500-599	600-649	650-699	700-749	750-774	775-799	800-824	825-850

Table 1: Score Levels

Score	Solved Tasks	Description
660	1st	The test-taker is familiar with programming and can write simple code to do some operations.
710	1st, 2nd	The test-taker has good implementation skills, can work with built-in data types and implement the desired solution. Most tech companies require only these skills for the job.
760	1st, 2nd, 3rd	The test-taker has great implementation skills and can develop large applications. The candidate may not have much experience on solving algorithmic tasks.
810	1st, 2nd, 4th	The test-taker has great problem solving skills and can implement ideas that don't require too much coding.
840	1st, 2nd, 3rd, 4th	The test-taker has excellent algorithmic, problem-solving as well as implementation skills.

Table 2: Score Guide for the General Coding Framework

The second column is the set of solved tasks. The first column is the expected score. The actual score may vary depending on the performance.

Coding Score Calculation

In this section we describe the general approach used to calculate the Coding Score based on the test-taker's solutions of GCA tasks or other similar tests.

For each test we need to define the following metrics:

- Maximum average score (*ms*) which is the max score one will get by solving all tasks in average time.
- Maximum quiz score (*mq*) which is the maximum score one will get by solving only the quiz tasks (if any).
- Max variation (*mv*) which is how much the score may vary (from the score that one would get when solving that set of tasks in average time, without incorrect attempts) based on speed, code quality and incorrect attempts.

Then we have the following metrics for each task: coding difficulty cd_k, problem-solving difficulty ps_k and task difficulty $d_{k=}\ \frac{psk+cdk}{2}$ which is the score one would get when solving only that task (we assign these scores by looking at the skills that are needed to solve that task and the score range one should get when solving that task).

For GCA we have $ms = 837$, $mq = 0$, $mv = 12$ and task difficulties $d_1 = 662$, $d_2 = 700$, $d_3 = 731$, $d_4 = 780$. To make the test results consistent we also have a predefined recommended time for each task t_k that shows the expected time of solving the task. Solving it faster grants additional points. Quiz score and code-writing score are independent, so the scores are summed.

Consider the case when the test contains only code writing tasks (which is the case for GCA). Let's sort the tasks in increasing order of the difficulties $d_1 \le d_2 \le \cdots \le d_n$. Then, if the test-taker solved some set of tasks $\{i_1, \ldots, i_k\}$, $i_1 < \cdots < i_k$ then the general score (without taking into account speed and code quality) will be calculated calculated by $gs = d_{ik} + p_{i1} + \cdots + p_{ik-1}$ where $p_1, \cdots, p_{n-1}$ are the additional scores calculated by assuming that $d_n + p_1 + \cdots + p_{n-1} = ms$ and that $p_1, \ldots, p_{n-1}$ should have the same difficulty ratios ($\frac{p_i}{p_j} = \frac{di-600}{dj-600}$)

so if $r = \frac{ms - d_n}{\sum_{i=1}^{n-1}(d_i - 600)}$ then $p_i = r \cdot (d_i - 600)$

After we have *gs* we should calculate how speed and code quality increase or decrease this score. So we have

$$speedScore = \frac{\sum_{i \text{ is solved}} s_i \cdot cd_i}{\sum_{i=1}^{n} cd_i}$$

where s_i is a number from -1 to 1 depending on how fast the test-taker solved the task (-1 means two times slower and 1 means two times faster than the average time). We have *attemptScore* which depends on the number of incorrect attempts and is multiplied by some coefficient. And we have

$$codeQualityScore = = \frac{\sum_{i \text{ is solved}} cq_i \cdot cd_i}{\sum_{i=1}^{n} cd_i}$$

where c_{qi} is a number from -1 to 1 and shows how good the code quality is (-1 means bad code quality and 1 means good code quality) for each task. We calculate a variation score:

$vs = codeQualityCoef \cdot codeQualityScore + speedCoef \cdot speedScore + attemptCoef \cdot attemptScore$

Where: $codeQualityCoef + speedCoef + attemptCoef = 1$ and then $vs \cdot mv$ is added to the general score $codingScore = gs + vs \cdot mv$. Note that $-1 \le vs \le 1$ so *codingScore* can vary from *gs* by at most *mv*.

Scoring Quiz Tasks

When we have quiz tasks (usually multiple choice questions with 4 options) in the test, we group them by topic and score each group as a unit. Scoring quiz questions by group prevents score in inflation due to guessing.

Let's suppose we have a group of *n* quiz tasks and one would get *p* points if they answer all of them correctly (if we have *Q* quiz tasks in total then $p = (\frac{n}{Q} \cdot mq)$. Then test-takers should get

$$f(k) = \frac{p}{c^{n-k}} \cdot \frac{k}{n}$$

points if they answer k questions correctly ($f(0) = 0$, $f(n) = p$) and $c = \frac{3}{2}$ (the reasoning behind the choice of this constant will be explained shortly). When using the standard approach of granting fixed points for solving a single quiz task, the score function would be

$$g(k) = p \cdot \frac{k}{n}$$

Here is a proof that *f* is a better scoring function than *g*: let $q = \frac{p}{n}$ and *X* be a random variable which is the number of correctly answered questions for the group. For each task the probability that one will answer it correctly is $\frac{1}{4}$ and $E(X) = \frac{n}{4}$.

$$E(g(X)) = q \cdot E(X) = \frac{p}{4}$$

Let's calculate $E(f(X))$

Claim.

$$S(a, n) = \sum_{k=1}^{n} k \cdot a^{(n-k)} \cdot \left(\frac{n}{k}\right) = n \cdot (1 + a)^{(n-1)}$$

and when $a = 2$ *it is equal to* $n \cdot 3^{n-1}$.

Proof.

$$S(a, n) =$$

$$n \cdot \sum_{k=1}^{n} 1^{(k-1)} \cdot a^{(n-1-(k-1))} \cdot \left(\frac{(n-1)!}{(k-1)! \cdot (n-1-(k-1))!}\right)$$

$$S(a, n) = \quad n \cdot (1+a)^{(n-1)}$$

Note that $P(X == k) = \quad \left(\frac{n}{k}\right) \cdot \left(\frac{3^{(n-k)}}{4^n}\right)$ (the probability that a test-taker would answer exactly k questions correct when selecting randomly), so using the claim we will get

$$E(f(X)) = \quad \frac{p}{4} \quad \cdot \quad \left(\frac{3}{4}\right)^{(n-1)} \quad = E(g(X)) \cdot \quad \left(\frac{3}{4}\right)^{(n-1)}$$

Let's compare f and g for the simple case of $n = 4$, $Q = 16$, and $mq = 100$ (which is mostly true for specialized assessments) hence $p = 25$.

For g, $E(X) = 1$, $E(g(X)) = \quad \frac{p}{4}$ and the expected score for the whole test is p.

$$g(1) = 0.25 \cdot p = 6.25$$

$$g(2) = 0.5 \cdot p = 12.5$$

$$g(3) = 0.75 \cdot p = 18.75$$

$$g(4) = p = 25$$

The expected additional score for the whole test is 25 from 100 when selecting randomly.

For f, $E(X) = 1$, $E(f(X)) = \quad \frac{p}{4} \quad \cdot \quad \left(\frac{3}{4}\right)^3 \quad \approx \quad \frac{p}{9.5}$ and the expected score for the whole test is approximately $\frac{p}{2.3}$.

$$f(1) \approx 0.07 \cdot p \approx 1.85$$

$$f(2) \approx 0.22 \cdot p \approx 5.6$$

$$f(3) = 0.5 \cdot p = 12.5$$

$$f(4) = p = 25$$

The expected additional score for the whole test is approximately 10.5 from 100 when selecting randomly.

Test Validity

This section includes instructions on how GCA should be administered to achieve maximum validity and reliability.

Test Security and Integrity

GCA should be administered in a proctored environment and the following actions should be banned during the test:

1. Switching to other browser tabs.
2. Disconnecting from the internet.
3. Using an external IDE.
4. Looking away from the screen for prolonged periods of time.
5. Moving out of the frame of the camera.
6. Accessing any outside materials or devices.
7. Interacting with anyone.
8. Copying or rewriting others' solutions.

Plagiarism Check

Even with many different question sets and continuous copyright-based leak monitoring, there will still be some test-takers that try to use someone else's solution as their own when completing the GCA. Hence, it is necessary to have an automated plagiarism check system that can do the following:

- Understand the syntax of programming languages.
- Compare solutions by removing comments and extra spaces while taking into account token types.
- Find similar solutions from a large database of solutions and flag the most similar solution if they pass a certain

similarity threshold while giving a visual diff-based comparison between the test-taker's solution and the similar solution.

Code Quality

The system used to administer GCA needs to support automated code quality computation since quality rating is one of the required attributes of the assessment. The assessment of code quality cannot, however, be based on biased or subjective criteria. Hence, only basic criteria for code quality that are universally accepted should be applied in the automated code quality computation (such as having consistent indentation and braces, descriptive variable names etc.). Each styling rule (some of them are from Google Style Guides [11]) is given a weight depending on how important the rule is in that language.

FAQ

Q:1 Why aren't Dynamic Programming (DP) or other more advanced algorithmic techniques covered in GCA?

A: Even though DP is taught as part of the standard CS program at some universities, the number of companies that expect this knowledge as part of the inter view process has decreased over the years. In addition, since GCA is meant for software engineers from all backgrounds, and advanced techniques like DP are not taught at many CS programs or non-traditional educational institutions like coding bootcamps, including those topics in GCA would create an unfair disadvantage for a large population of software engineers.

Q:2 How do you prevent GCA tests from being leaked on the web?

A: The advantage of having a framework-based assessment is that you can make endless copies of the same test containing different questions that measure the same skills yet are different enough to prevent copying solutions. In addition to having many different question sets, we also submit all GCA questions to the United States Copyright Office [12] and continuously monitor sources like GitHub, Stack Over ow and Glassdoor. If and when a GCA question appears online, we request an official takedown.

Q:3 Who has verified the framework for EEOC [13] compliance?

A: We've partnered with Barkley Research Group [14] who specialize in assessment validity and compliance research to validate that the approach and the structure of GCA assessments are fully compliant with EEOC and other similar regulations.

Q:4 Are there other examples of commonly accepted tests that use a similar framework-based approach?

A: Every major standardized assessment is using a framework-based approach. Some examples include SAT (they call it Test Specifications [15]), TOEFL and other standardized language assessments use the Common European Framework for Language [16], GRE [17].

Q:5 Should GCA be used only with new grad and junior developers?

A: Not necessarily. It works very well with more junior developers since the knowledge measured in GCA covers almost everything expected from junior developers; but it can

also be used with more senior developers as an initial evaluation of core skills. However, for more senior or specialized developers (e.g. an iOS developer), GCA would need to be coupled with a more specialized interview or a Specialized Coding Assessment (e.g. iOS SCA) that uses a similar framework but focuses areas of measurements to the specific domain in question.

Acknowledgement

We are very thankful to the engineering advisory board of the #GoBeyondResumes [18] movement, including Anima Anandkumar - Director of ML Research @ NVIDIA and Bren - professor at Caltech, Chris Kanaan - SVP of Eng @ Ripple, Jessica McKellar - CTO @ Pilot, Kah Seng Tay - VP of Eng @ Drive.ai, Lei Yang - VP of Eng @ Quora, Nate Kupp - Director of Eng @ Thumbtack, Nimrod Hoo en - Director of Eng @ Facebook, Surabhi Gupta - Director of Eng @ Airbnb, Yoann Roman - Director of Eng @ Yelp for kindly reviewing and providing their comments and feedback on the GCA framework. Also Aram Shatakhtsyan - CTO @ CodeSignal, Eduard Piliposyan - Director of Assessments @ CodeSignal and Michael Newman - Director of Engineering @ CodeSignal for their detailed review and feedback on this paper.

References

[1] MIT OpenCourseWare. [Online]. Available: https:// ocw. mit.edu/index.htm

[2] edX. [Online]. Available: https://www.edx.org/ [3] Coursera. [Online]. Available: https://www.coursera.org/ [4] Udacity. [Online]. Available: https://www.udacity.com/ [5] G. L. McDowell, Cracking the Coding Interview: 189

Programming Questions and Solutions. CareerCup, 2015. [Online]. Available: https://www.amazon.com/Cracking-Coding-Interview-Programming-Questions/dp/0984782850?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0984782850

[6] CodeSignal. [Online]. Available: https://app.codesignal.c om/interview-practice

[7] LeetCode. [Online]. Available: https://leetcode.com/ [8] CareerCup. [Online]. Available: https://www.careercup. com/

[9] Glassdoor. [Online]. Available: https://www.glassdoor.co m/index.htm

[10] Stack Exchange API. [Online]. Available: https:// api.stac kexchange.com/

[11] Google Style Guides. [Online]. Available: https:// google .github.io/styleguide/

[12] U.S. Copyright O ce. [Online]. Available: https:// www. copyright.gov/

[13] Equal Employment Opportunity Commission. [Online]. Available: https://www.eeoc.gov/laws/regulations/index. cfm

[14] BRG Applied Technology | BRG Drive | Service Platform. [Online]. Available: https://www.thinkbrg.com/ [15] Test Speci cations for the Redesigned SAT. [Online]. Available: https://collegereadiness.collegeboard.org/pdf/ test-speci cations-redesigned-sat-1.pdf

[16] Common European Framework of Reference for Languages — Wikipedia, The Free Encyclopedia. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Common_European_Framework_of_Referenc

e_for_Languages&oldid=875947854

[17] GRE. [Online]. Available: https://www.ets.org/gre/revise d_general/about/content/

[18] Go Beyond Resumes. [Online]. Available: http:// gobeyo ndresumes.org/

Authors

Albert Sahakyan is a Data Science researcher at Yerevan State University and a Sr Machine Learning Engineer 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.

Tigran Sloyan is the CEO of CodeSignal and the founder of the #GoBeyondResumes movement. He has three degrees from MIT (in Computer Science, Mathematics and Economics) and has worked in Engineering and Product Management roles at companies like Google and Oracle.