Industry Coding Skills Evaluation Framework

Technical Brief

CodeSignal

Skills Evaluation Lab

Published August 2021

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.

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.

The Industry 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 commonly required for experienced software engineering roles across a wide variety of organizations and industries, including:

- Software design patterns
- Code implementation
- Problem solving
- Data structures and data processing
- Maintaining codebases via refactoring and encapsulation

Framework Specifications

The Industry Coding Skills Evaluation Framework is designed to assess the coding skills of senior software developers or engineers around the key principles of validity, scalability, and fairness. Each evaluation contains **1 language-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 project-based 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.

Solving Industry Coding Framework 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.

Level 1 – Initial Design & Basic Functions

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 – Data Structures & Data Processing

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 – Refactoring & Encapsulation

The third level requires candidates to ex-

tend 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 – Extending Design & Functionality

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 project.

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 back-

ward 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

Framework Example Content

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:

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

Level 1 – Initial Design & Basic Functions

- `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 – Data Structures & Data Processing

- `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 – Refactoring & Encapsulation

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 lifetime 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