Quality Assurance Engineering Skills Evaluation Framework

CodeSignal
Skills Evaluation Lab

Published October 2022

Introduction

In software development, Quality Assurance (QA) engineers are responsible for implementing test plans, creating automated tests, documenting identified defects in new software for application improvement, and test maintenance. The demand for this role is projected to grow 22 percent from 2020 to 2030, faster than the average for all occupations[1]. The growth in demand for QA talent requires organizations to implement innovative ways to accurately assess such talent at scale.

With QA engineers playing a key role in the entirety of the software development lifecycle, there has been a shift in industry required skills to include programming competencies in parallel with closer collaboration with software developers. This framework paper will outline core QA engineering skills, including understanding user stories, manual testing, creating automated tests, and test maintenance.

This paper will outline the core components of the QA Engineering Skills Evaluation Framework based on industry research and consultation with subject matter experts. It will also illustrate the scoring distribution across modules and how the given score will map to the core skills of a QA engineer.

The Framework

The framework is designed to model closely to what the QA engineer would be expected to perform on the job. It can be utilized across different methods of delivery, assessment or interview, while preserving its objectivity by automatically calculating the final score.

The maximum allowed completion time for the framework is **90** minutes and consists of **4** levels that mimic a real world scenario. These levels are sequential in nature where each subsequent level adds on additional complexity for a candidate to solve, with each level equally weighted at **250** points for a total score of **1000**. The scenarios presented have also been designed to introduce minimal industry-specific context so as to create an agnostic evaluation and reduce any bias within the candidate population.

Level 1 – Manual Quality Assurance

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

[1]Bureau of Labor Statistics, U.S. Department of Labor, Occupational Outlook Handbook, Web Developers and Digital Designers, at https://www.bls.gov/ooh/computer-and-information-technology/software-developers.htm (visited June 10, 2022)

Expected Knowledge

- Understanding a testing plan
- Understanding user stories and being able to replicate the user flow

Can Include

- Manual testing of a web application
- Manual Graphical User Interface testing
- Manual End to End testing

Should Exclude

- Scripting
- Unit tests
- Automation testing

Level 2 – Component Testing

The average time for solving this level should be **20** minutes. Typically, candidates should expect to write **10-15** lines of code for this level.

Expected Knowledge

- Basic programming knowledge and understanding of application code
- Understanding existing tests for sample application
- Implementing corrections for existing component tests

Can Include

- Understanding of functions, methods, and existing component tests for sample application
- Understanding of input and expected output
- Identifying corner cases in the application logic

Should Exclude

- Writing novel unit tests
- Writing production code
- Writing and testing new automated tests (integration, end-to-end, etc.)

Level 3 – API Testing

The average time for solving this level should be **30** minutes. Typically, candidates should expect to write **40-60** lines of code for this level

Expected Knowledge

- Ability to write tests that cover all user interactions across user stories
- Ability to read/understand a test plan
- Knowledge of how to asynchronously call remote APIs from the browser and handling errors and authentication

Can Include

- Everything from the prior level
- Implementing planned test cases from scratch
- Writing tests to cover all API end-points within a defined REST API based on API documentation and user stories

Should Exclude

- Writing production code
- Unit testing

Level 4 – Test Evolution

The average time for solving this level should be **30** minutes. Typically, candidates should expect to write **70-90** lines of code for this level.

Expected Knowledge

- Understanding new user stories and feature specifications

- Refactor and modify existing tests to satisfy new user stories and feature specifications
- Evaluating test errors to identify indications of false positive results
- Knowledge of advanced API concepts like pagination and rate limiting

Can Include

- Everything from the prior level
- Implementing novel test cases from scratch
- Divergent thinking to understand test cases to prioritize

Should Exclude

- Writing production code
- Test case planning and design
- Ability to prioritize test cases based on user requirements and code coverage

Scoring Methodology

Evaluating a candidate's test quality presents a novel challenge to general skills evaluation techniques, which is generally confined to logical or semantic comparisons between a functional output to a predefined output. Additionally, traditional quantitative testing metrics like code coverage and efficiency do not provide a strong signal on the objective quality of tests written by a candidate.

In order to circumvent these issues, the framework leverages mutation testing techniques to measure the quality of a candidate's tests. Mutation testing is a well-researched practice of creating artificial changes within a piece of software in order to evaluate if an existing test suite is able to detect the intentionally created changes.

Based on the user stories defined at each level, the framework thus introduces a series of intentional mutations to the application being tested by the candidate, which thus allows for an evaluation of the candidate's tests by quantifying the number of mutants correctly identified. These are then weighted appropriately for the level and attributed to the overall score for the candidate.

Example

Below is an example of a question that is established based on the structure of the framework. Similar questions are also created and monitored on an ongoing basis to ensure overall consistency as well as preventing widespread cheating and plagiarisms.

Scenario: A candidate is asked to test a message posting application, similar to Twitter.

Level 1 – Manual Quality Assurance

You are currently working with an engineering team who is working to develop a new message posting application. The engineering team just delivered new features for the application with the following user stories.

Ask: Manually test the application and return up to two user stories that are faulty in the new application

User Stories
As a User, I want to be able to create new posts, so that I can share a message with other users.
As a User, I want to be able to delete existing posts, so that I can remove a message I no longer

want to share with other users.

As a User, I want to be able to edit existing posts, so that I can make updates to the message I am sharing with other users.

As a User, I want to be able to like an existing post shared by other users, so that I can show my support for their message.

Context: The candidate will be presented with an actual web application to test manually, and will input the user story id into the input fields provided.

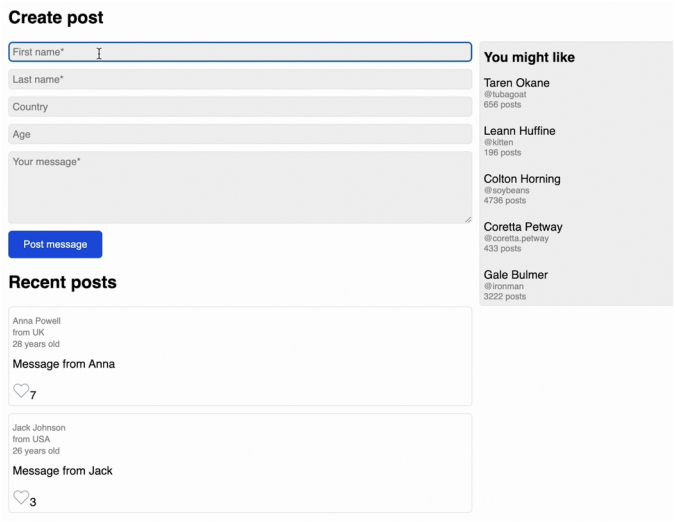

Figure 1: Sample Manual Testing Application

Level 2 – Component Testing

Engineering just finished the development and testing of the liking functionality of a message posting application, and you are now tasked with testing this functionality.

Ask: The test cases provided in the test file have been written against the table provided. The test file has listed one incomplete test case per user operation from the table.

Complete the automated test cases in the test file to ensure the tests are testing the new user logic.

The user story and the class are defined as follows:

User Stories

As a User, I want to be able to like an existing post shared by other users, so that I can show my support for their message.

As a User, I want to be able to unlike an existing post shared by other users that I have liked, so that I can withdraw my support for their message.

postingApplication.py

```
class Post:
    def __init__(self, postId, posterId, message,
likedUserIds):
        self.postId = postId
        self.posterId = posterId
        self.message = message
        self.likedUserIds = {}


class User:
    def __init__(self, userId, userName):
        self.userId = userId
        self.userName = userName


class postingApplication:
    def __init__(self):
        self.posts = {}
        self.users = {}

   def likePost(self, userId, postId):
        self.posts.get(postId).likedUserIds
            .add(userId)

   def unlikePost(self, userId, postId):
        self.posts.get(postId).likedUserIds
            .remove(userId, None)

   def getLikes(self, postId):
        self.posts.get(postId).len(likedUserIds)
```

As shown in the following table, we have collected actual user data from users of the `postingApplication` class to serve as the

basis of the component tests.

User Operation	Result
Like a message that has not been liked by the user	Post is liked by the user
Like a message that has already been liked by the user	No change in post like status
Unlike a message that has already been liked by the user	Post is no longer `liked` by the user
Unlike a message that has been not liked by the user	No change in post like status

Table 1: Test case user data for component tests of the postingApplication class

Context: The candidate will be presented a testing file with existing test cases to provide structure to the candidate. The candidate will have to fill in the code to make the tests run as defined in the user data table, which mimics a simplified testing plan.

Level 3 – API Testing

When a message is posted, this should propagate across a posting application. This feature is built around an API microservice that captures, retrieves and publishes the messages.

Ask: Write the automated test cases in the test file to ensure that the new API works as per the requirements described.

User Stories

As a User, I want to be able to view messages posted by all users, so that I can get updated on the overall activity of users on the application

As a User, I want to be able to view messages posted by a specific user, so that I can get updated on their activity only

As a User, I want to be able to create new posts, so that I can share a message with other users

The service operates at the URI: https://api.-codesignalcontent.com/postingApplication/ posts with the endpoints defined as follows:

Endpoint	Method	Description
posts/	GET	Returns a list of the latest messages posted by all users
posts/{userId}	GET	Returns the latest (up to 5) messages posted by the `userId`
posts/{postId}	POST	Publishes a `Post` object, assigned to the `userId` that calls the method. This should propagate across the application and be visible to other users.

Table 2: Posting Application API Endpoints

Examples:

GET post/

200 Response

```
{
  "posts":
    {
      "postId": 5,
      "posterId": 5,
      "message": "Stop spamming!!!",
      "likedUserIds": {2}
    },
    {
      "postId": 4,
      "posterId": 1,
      "message": "Testing 123",
```

```
      "likedUserIds": {1}
    },
    {
      "postId": 3,
      "posterId": 2,
      "message": "Hello World",
      "likedUserIds": {2, 3}
    },
    {
      "postId": 2,
      "posterId": 2,
      "message": "This is so cool!",
      "likedUserIds": {}
    },
    {
      "postId": 1,
      "posterId": 1,
      "message": "Just setting up my account",
      "likedUserIds": {1, 2}
    }
}
```

GET post/user_id=1

200 Response

```
{
  "posts":
    {
      "postId": 4,
      "posterId": 1,
      "message": "Testing 123",
      "likedUserIds": {1}
    },
    {
      "postId": 1,
      "posterId": 1,
      "message": "Just setting up my account",
      "likedUserIds": {1, 2}
    }
}
```

POST post/1

Request

```
{
   "status": "success"
}
```

200 Response

```
{
  "post":
    {
      "postId": 1,
      "posterId": 1,
      "message": "Just setting up my account",
      "likedUserIds": {}
    }
}
```

Context: This level is similar to level 2 but introduces API testing. The candidate will be presented a testing file with a single test case defined to provide structure to the candidate. Following which the candidate will have to create additional tests, based on the user stories and API end-points provided.

Level 4 – Test Evolution

The engineering team has added on a newly implemented feature, that unfortunately impacts the testing suite that you have developed earlier.

Ask: You are to review the existing tests from the previous level, and implement end-to-end tests for additional newly implemented features, using the following data.

Assume that the postingApplication class has been both unit and component tested, and no bugs have been detected.

User Stories

As a User, I want to be able to share another user's post, so that I can share a message with other users.

As a User, I want to be able to navigate through messages posted by a specific user, so that I can get updated on their historical activity.

A rate limiting feature has also been implemented to protect the service against Denial of Service (DoS) attacks. Hence, users calling the API from the same IP address will only be able to do so up to a maximum of 50 instances within a 24 hour period.

In addition, please assume that the testing plan is not comprehensive, and implement additional tests that would be relevant.

Endpoint	Method	Description
posts/	GET	Returns a list of the latest messages posted by all users
posts/{userId}	GET	Returns the latest (up to 5) messages posted by the `userId`
posts/{userId}?page={pageNum}	GET	**If the `userId` has more than 5 posts:** Returns the messages posted by the `userId` corresponding to the page defined in the GET request **If the `userId` has less than 5 posts:** Returns the latest (up to 5) messages posted by the `userId`
posts/{postId}	POST	**If `postId` doesn't exist:** Publishes a `post` object from the user_id that calls the method. **If `postId` exists:** Reshares the `post` object and pushes it to the top of the message list. In both cases, this should propagate across the posting application

Table 3: Posting Application Updated API Endpoints

EndPoint	Request/Response Data
GET posts/1	Response Data User exists `{ "post": { "postId": "1", "posterId": 1, "message": "Just setting up my account", "likedUserIds": {1, 2} "reposted": {1, 2} } }`
GET posts/3	Response Data User does not exist `{ "status": "user not found" }`
GET posts/2?page=2	Response Data User has more than 5 posts `{ "post": { "postId": "8", "posterId": 2, "message": "Amazing tacos", "likedUserIds": {4, 5} "reposted": {} }, { "postId": "17", "posterId": 2, "message": "Going to the beach", "likedUserIds": {5} "reposted": {} }, { "postId": "6", "posterId": 2, "message": "Love this concert", "likedUserIds": {2, 3, 4} "reposted": {} } }`
POST posts/1	Request Data `{ "status": "success" }`

Response Data

Post exists
{
"post":
{
"postId": "1",
"posterId": 1,
"message": "Just setting up my account",
"likedUserIds": {1, 2}
"reposted": {1, 2}
}
}

Table 4: Test Data

Context: This level introduces a new feature that causes potential breaking changes to the candidate's tests written in level 3. This means that candidates will need to both refactor existing test cases and implement new test cases in this level. Additionally, advanced API concepts like the rate limiting feature in this example, is included to encourage candidates to design and implement additional test cases that might not have been explicitly defined in the testing plan as part of the evaluation.