Quality Assurance Engineer

RoadMap

“Let Consistency be your Superpower”

[June 27th, 2024]

Important Notice

This document has been meticulously crafted to accommodate individuals from diverse backgrounds and fields. It is designed with utmost flexibility, recognizing that the learning journey is boundless. The purpose of this document is to provide a foundational introduction to Software Quality Assurance (SQA) and guide you in starting from scratch. I encourage you to read through this document and delve deeper into related topics that extend beyond its scope, following the QA roadmap independently.

Owner's Signature

Name: Muhammad Hammad Rashid
Title: Software Quality Assurance Engineer
Date: June 27, 2024

Signature: Hammad Rashid

Introduction

Overview

Becoming a Quality Assurance (QA) Engineer is a rewarding career choice for those passionate about ensuring the quality and functionality of software products. QA Engineers play a pivotal role in the software development lifecycle by identifying and addressing bugs, ensuring that software meets user expectations and industry standards. Their work helps prevent costly errors and enhances the overall user experience.

This roadmap aims to provide a clear and structured path for aspiring QA Engineers, covering the foundational knowledge, practical steps to gain experience, and advanced techniques to master. Whether you are a recent graduate, a professional looking to switch careers, or someone in the tech industry aiming to specialize in QA, this guide will help you navigate the essential steps to become a successful QA Engineer.

Purpose

The purpose of this roadmap document is to provide a clear and structured guide for individuals aspiring to become Quality Assurance (QA) Engineers. It outlines the essential skills, knowledge, and steps required to enter and excel in the field. By following this roadmap, aspiring QA Engineers can systematically develop their expertise, gain practical experience, and navigate their career progression effectively.

Audience

This roadmap document is designed for recent graduates, professionals seeking a career change, and current tech industry employees who aim to specialize in Quality Assurance (QA). It will benefit anyone looking to build a successful career in QA Engineering by providing them with a structured path to develop the necessary skills, gain practical experience, and achieve their professional goals.

Overview of QA Engineering

A QA Engineer plays a vital role in the software development process by identifying bugs, ensuring the software meets requirements, and maintaining high standards of quality. Their responsibilities include designing test plans, executing tests, and collaborating with developers to resolve issues.

Skills and Knowledge

Technical Skills

Proficiency in programming languages such as Java, Python, or C#. Familiarity with testing tools like Selenium, JIRA, and TestRail is a plus.

Soft Skills

Strong analytical skills, keen attention to detail, and effective communication abilities.

Educational Background

A degree in Computer Science, Information Technology, or a related field is beneficial.

RoadMap

The following roadmap is a pathway for the people who wish to enter this field and want to explore more. They can always explore this and can improve their skill set accordingly. For ease of use, this roadmap is divided into multiple modules. The following roadmap is a weekly schedule for the beginners to follow. By applying this approach, you will be able to develop a sound understanding of what you yearn to learn and will make you an expert in this field.
For weekly division of the RoadMap, refer to table 1 in Appendix A.

Software Testing Basics

To begin with your QA journey, it is important to start with the basics of Software Testing and fundamental questions related to Testing. Here are a few examples to help you get started.

1. What is Software Testing?
2. What is SDLC?
3. What is STLC?
4. What is the difference between a bug, error, defect and failure?
5. What is a root cause and root cause analysis?
6. What are the 7 principles of Software Testing?
7. What is the difference between QA and QC?
8. What is verification and validation?

And many more. You can refer to the ChatGPT or https://guru99.com/software-testing.html for further questions.

Testing Approaches

A testing approach is considered the steps or the approach we tend to use when designing test cases. This includes the several techniques which we use when designing the test data and input data for testing a software application.

BlackBox Testing Technique

Black box testing is a method where testers assess the functionality of an application without any knowledge of the underlying code or logic. This approach involves creating test cases based solely on the requirements and specifications of the software. Testers focus on input and output, ensuring that the software behaves as expected without delving into the internal workings. Here are the 4 most important BlackBox testing techniques which every QA should know. I have explained these 4 types with examples in my medium articles. You can check them out by clicking the respective technique.

1. Equivalence Class Partitioning
2. Boundary Value Analysis
3. Decision Table Testing
4. State Transition Testing

WhiteBox Testing Technique

White box testing, also known as clear or glass box testing, is a software testing method where the tester has full knowledge of the internal workings, code structure, and implementation details of the application. This approach allows testers to design test cases that specifically target the internal logic, pathways, and individual components of the code. By thoroughly examining the code, white box testing helps in identifying vulnerabilities, improving code quality, and ensuring that all possible paths and conditions are tested. This method is particularly useful for optimizing performance and verifying the correctness of algorithms and data structures.
The two commonly used techniques are:

1. Statement Testing
2. Branch Testing

You can explore more about them by clicking on the respective techniques.

Once you have done these techniques, you can refer to sample questions for practice purposes in order to enhance your skill and make your concepts clearer with respect to the above techniques.

Cyclomatic Complexity and Control Flow Graphs

Control Flow Graphs

A control flow graph (CFG) is a graphical representation of all paths that might be traversed through a program during its execution. Each node in the graph represents a basic block, a

straight-line sequence of instructions with no branches, while the edges represent the control flow paths between these blocks. CFGs are crucial for various aspects of software development and analysis, including optimization, testing, and debugging. They help in identifying unreachable code, potential infinite loops, and the overall complexity of the code. By visualizing the control flow, developers and testers can better understand the structure and behavior of the program, making it easier to identify areas that require thorough testing and optimization.

Cyclomatic Complexity

Cyclomatic complexity is a quantitative measure of the complexity of a program's control flow, used in software engineering to indicate the number of linearly independent paths through a program's source code. It is a key metric in assessing the maintainability and testability of software. Cyclomatic complexity is calculated using the formula:

$$V(G) = E - N + 2$$

where V(G) is the cyclomatic complexity, E represents the number of edges in the control flow graph, N is the number of nodes, and P stands for the number of connected components or exit points in the graph. Another common formula is:

$$V(G) = P + 1$$

where P is the number of predicate nodes, or decision points, such as if statements or loops, in the program.

Higher cyclomatic complexity indicates a higher potential for defects and a greater difficulty in understanding and testing the code, making it a crucial factor in software quality assurance.

Explore more about cyclomatic complexity and control flow graphs in my article here.

Exploratory Testing

Exploratory testing is an unscripted, hands-on approach to software testing where testers actively explore the application to identify defects and issues without predefined test cases. This technique relies on the tester's creativity, experience, and intuition to uncover hidden problems. There are several types of exploratory testing, including session-based, free style or ad-hoc and scenario-based. You can read more about exploratory testing and its sub types here.

Testing Techniques

There are two types of testing techniques. They are static testing techniques and dynamic testing techniques. Let's dive into them one by one and explore them.

Static Testing

Static testing is the type of testing done without executing the code. For further details refer to the article here.
Static testing covers three important genres. The rest of them somehow come under the three main categories listed below.

Documentation

Reviewing documentation in static testing is a critical process that involves meticulously examining project documents to identify errors, inconsistencies, and areas for improvement without executing the code. This phase typically includes evaluating requirements specifications, design documents, test plans, and user manuals. The primary goal is to catch defects early in the development cycle, thereby reducing the cost and effort associated with fixing issues later on. By conducting thorough reviews, teams can ensure that the project's foundational documents align with user needs and technical constraints. This proactive approach not only enhances the quality and reliability of the final product but also fosters better communication and understanding among team members.

Design

Reviewing design in static testing involves a detailed examination of design documents, architecture diagrams, and interface specifications to identify potential issues before implementation begins. This step is crucial for ensuring that the design aligns with the requirements and is feasible, scalable, and maintainable. During this process, reviewers check for adherence to design standards, completeness, correctness, and consistency. They also assess whether the design can meet performance, security, and usability goals. By identifying and addressing design flaws early, teams can avoid costly changes during later stages of development, streamline the development process, and improve the overall quality of the software. This proactive review not only enhances the robustness of the design but also facilitates better planning and resource allocation for the subsequent phases of the project.

Requirement Analysis

Reviewing requirement analysis in static testing is a fundamental activity that focuses on evaluating the requirement documents to ensure they are clear, complete, consistent, and testable. This process involves scrutinizing functional and non-functional requirements, use cases, user stories, and any associated diagrams or models. The objective is to identify ambiguities, contradictions, missing requirements, and feasibility issues before any design or development work begins. By conducting thorough reviews, stakeholders can confirm that the requirements accurately reflect the needs and expectations of the end users and align with

business objectives. This early validation helps prevent costly changes and rework in later stages, enhances communication among project team members, and lays a solid foundation for a successful project. Effective requirement analysis reviews ultimately contribute to the creation of a more reliable, high-quality product that meets stakeholder expectations.
Refer to the RoadMap image for further details.

Dynamic Testing

Dynamic testing is done by executing the code snippet. Refer to my article here for further details.

Functional Testing

Functional testing refers to testing the components for their functionality. In simpler words, we can say that we tend to the component and ensure that it acts according to the requirements provided by the business stakeholders. Functional testing has several subtypes some of which are as follows. You can explore other types from the roadmap provided in the image.

1. Exploratory Testing
2. Unit Testing
3. Integration Testing
4. Smoke Testing
5. Sanity Testing
6. Regression Testing
7. Acceptance Testing

It is necessary to go through these types in detail so as to perform the testing process thoroughly and seamlessly.

Non Functional Testing

Non functional testing tests the non functional aspects of the code. These aspects are not concerned with the functionality the code snippet provides but it deals with how the code snippet fulfills the given functional requirement. Non functional testing includes the following subtypes.

1. Performance Testing
    a. Load Testing
    b. Stress Testing
    c. Endurance Testing
    d. Spike Testing
    e. Volume Testing
2. Compatibility Testing
    a. Device
    b. Browser
3. Maintainability Testing
4. Security Testing
5. Responsive Testing
    a. Tablet
    b. Mobile
    c. Desktop

Explore more about them and delve deeper into the details!

Understanding SDLC Models

SDLC stands for Software Development Lifecycle. This phenomenon tends to help the developers, testers and the business owners to develop the projects and seamlessly launch them in the market for public use. The most commonly used SDLC models are as follows:

Waterfall Model

Waterfall model is like a waterfall flowing from top to bottom. This means that once you reach the bottom, you cannot go back to the top. In this model, the requirements are locked and then the design phase is started. Similarly, once the design is done, the development part is started and testing starts once the product is fully developed according to the requirements. This model is also known as sequential model, in which every step occurs sequentially.

V-Model

Also known as the Verification Validation model. The steps take place in a V shape. In other words, there is a testing phase against each dev phase.

Agile

The iterative process of developing software products is referred to as the agile paradigm. This means that a team breaks down a huge project into smaller sections and delivers these completed smaller parts in regular cycles rather than delivering a large project only when all components are complete.

You are free to explore more about the SDLC models and go through any other model other than those mentioned above.

Manual Testing

Manual testing involves manually testing the system and checking it against the user provided requirements in order to confirm the functionality of the system and see if it can be launched into the product. This testing technique involves manually looking at the system, testing it out for bugs and issues with the naked eye. We can manually test the functional requirements of a system but unfortunately, we do not have any way to test out the non functional requirements of a system manually, except usability. Usability testing involves checking the usability of any product, software, application, system with real time users and observing them how they use it. Commonly we use functional testing for the following testing types.

UI/UX Testing

UI/UX testing involves testing and checking the UI requirements of a software such as color palettes, font consistency, text correctness, styling and layout of the software and many more. It mostly deals with the design and layout structure of the application and is more concerned about the look and feel of the final product.

Compatibility Testing

Compatibility testing involves testing the product across multiple devices such as mobile, laptop, desktop and tablets. This ensures that the system is responsive across all device types and performs well across every screen resolution.

Functional Testing

Functional testing is testing the application for the functional requirements ensuring that the functionality provided is being fulfilled or not. It includes everything such as forms, CTAs, page directions and many more.

Development Types

Normally there are three development types.

Test Driven Development

Creating an automated unit-level test case that fails, creating just enough code to pass the test, reworking both the test and production code, and then repeating the process with a new test case is known as test-driven development (TDD).

Behavior Driven Development

Behavior Driven Development, more commonly known as BDD, is an approach where we tend to write the functional requirements of the system using Natural Languages, such as English to improve the communication between the developers, testers and other stakeholders.
Tip: Explore Gherkin Language and you will understand how BDD works.

Acceptance Test Driven Development

Developers, testers, and business representatives collaborate during the acceptance test-driven development (ATDD) process to determine requirements, identify potential hazards, and lower the likelihood of errors before coding starts. The ATDD addresses the query, "Is the code doing what it's supposed to do?" from the user's point of view.

Testing Activities

The testing activities are a set of steps followed while executing STLC. These steps determine the life cycle of the testing phases and how the system is tested. There are 7 steps in this activity that are as follows.

1. Test Planning
2. Test Monitoring and Control
3. Test Analysis
4. Test Design
5. Test Implementation
6. Test Execution
7. Test Closure

Explore each of these steps and learn about them in detail. This is an important step in the pathway and carries several benefits as well.

Test Cases

Test case refers to the actions required to verify a specific feature or functionality in software testing. A sample test case can have a Test ID, Reviewer, Date, Priority, Preconditions, Post conditions, data, steps and status.

Automation Testing

Automation testing refers to the ability to test the software product by executing a script ensuring if the functional requirements of the software are being met or not.

Language Selection

The most fundamental thing when starting with automation testing is learning a new scripting language. If you know programming, well and good. But if you don't have a CS background, then you will have to learn a programming language to begin with. Here are a few options to help you get started.

1. Python
2. JavaScript
3. TypeScript
4. Java
5. C#
6. Ruby
7. Perl

Considering that you have a CS background, it won't be that hard for you to learn a programming language.

Integrated Development Environment

Once you have hands-on experience with a programming language, you need to have an IDE installed with you. IDE or Integrated Development Environment is a software application that helps programmers develop software code efficiently. There are multiple options but I will suggest you to go for Visual Studio Code as it has extensions available for each and every language.

Automation RoadMap

Frontend Automation

Frontend includes everything which can be executed on a browser or on any real time device. The users interact with the frontend in order to use the application and benefit from it. In order to execute frontend automation, we can use 2 options.

Browser add ons

Browser add ons include software frameworks or technologies that extend the functionality of a browser to perform a specific task. At frontend automation testing level, they include:

1. Selenium IDE
2. Ghost Inspector

Frameworks

Frameworks include tools and technologies which are installed on the local system in order to benefit from them. They include:

1. Cypress(my personal favorite)
2. Selenium WebDriver
3. TestCafe
4. Cucumber
5. NightWatch Js

There are other options as well for E2E testing and UI testing and you are free to explore them as well.

Backend Automation

Backend is everything which is hidden from the end users. It covers the logical functionality of the application, server side logics, backend programming, database handling, APIs(Application Programming Interfaces) and architecture of the system.

Application Programming Interface

Application Programming Interface, also known as API, is an interface for communication between two layers in an application. They make use of specific protocols and methods to call each other and communicate. There are multiple types of APIs but mostly we tend to use REST APIs in development.

REST APIs

An architectural style for application programming which makes use of HTTPS protocol. In order to learn about REST APIs, refer to the following topics.

1. Basics of REST APIs
2. How do REST APIs work?
3. Difference from other API types(such as SOAP, Open APIs)
4. Different methods and protocols used in REST APIs.

Frameworks

In order to test for APIs, we normally use Postman to test the APIs and their working. Here is a link for you to learn more about REST APIs and Postman.

For Database testing, you can explore more here.

Mobile Automation

Mobile automation testing includes testing the mobile applications on both iOS and Android. It covers testing the complete application for functional, non functional, backend, frontend, UI/UX and API testing. There are multiple tools used for mobile automation testing globally but the most popular is Appium. It covers both iOS and Android Operating Systems and is free to use as well.

Performance Testing

Performance testing covers all the non-functional aspects of any application/ software such as load testing, stress testing, endurance testing, spike testing and volume testing. There are multiple tools used for performance testing but the most popular among all of them is JMeter. Here is a link to the guide for beginners on how to use JMeter. You can consult it for your own use as well. Besides, you should also be familiar with Google Lighthouse and how it works while checking the page speed, web analytics and the SEO score of a website.

Miscellaneous

Collaboration Tools

These tools are meant for effective communication between the team members. Mostly, just a single tool is used in an organization, but there might be a possibility that the client requires you to use a specific channel, so for that, you need to be fully prepared. Here are some commonly used tools:

1. Slack
2. Discord
3. Microsoft Teams
4. Bitrix
5. Email communication

Project Management Tools

These tools are used to manage, centralize and organize the projects. These tools are not limited to a specific role or department but can be used in any role, in any department, in any area across an organization. Some of the tools include:

1. Jira
2. ClickUp
3. Asana
4. Trello
5. Monday.com

Version Control Systems and Repository Hosting

These softwares tend to help the users manage the commits to the code and keep a track of how the code is being developed and pushed into the main branch. These tools include:

1. GitLab
2. GitHub
3. BitBucket
4. SVN
5. GNU Bazaar

The most common used are GitLab and GitHub and these two are mostly preferred in the industry.

Appendix

Appendix A: RoadMap

Table1: RoadMap for QA Engineers on a Weekly Basis

Week	Topics
Week 1	Software Testing Basics
Week 2	Testing Approaches(examples and sample practice questions)
Week 3	Testing Techniques(Functional Testing)(examples and sample questions)
Week 4	Testing Techniques(Non-Functional Testing)(examples and sample questions)
Week 5	Understanding SDLC models and explore other models
Week 6, 7, 8	Manual Testing and manual testing techniques(sample questions and practice)
Week 9	Development Types and Testing Activities
Week 10	Test Cases and write sample test cases as practice for a website.
Week 11*	Automation Testing(Programming Language and IDE setup), if you are new to programming and don't know any programming language, you should spend the next 4 weeks on learning a new programming language from scratch and practice it so that you become adept with it
Week 12, 13, 14	Frontend Automation(Basics on how to do it and learn a tool and practice)
Week 15, 16, 17	Backend Automation(Basics, REST APIs, Postman and Database Testing)
Week 18, 19, 20	Mobile Automation(Basics, learn Appium and practice)
Week 21, 22, 23	Performance Testing(Basics, learn and practice JMeter)
Week 24	Miscellaneous concepts(Collaboration tools, Project tools, VCS)
Week 25	Practical application of all the concepts studied

* If you belong to a non CS background, you can invest the next 4 weeks in learning a new language from scratch. I'll suggest start with Python as it is beginner friendly and easy to learn. After learning Python, you can opt for Selenium WebDriver as Python is supported in the Selenium framework. Thus, your roadmap will have an additional 4 weeks for learning a new language.

Appendix B: Articles

Here are the links to a few articles which you can refer to for your own learning.

1. https://medium.com/@hammad.rashid_73904/essential-tools-for-qa-engineers-ccb59e22c244
2. https://medium.com/@hammad.rashid_73904/mastering-quality-assurance-essential-principles-for-qa-beginners-01cc5c95543f
3. https://medium.com/@hammad.rashid_73904/7-principles-of-software-testing-0b7c1a1f409c
4. https://medium.com/@hammad.rashid_73904/visualizing-code-complexity-analyzing-cyclomatic-complexity-with-control-flow-graphs-4d839545a87e

Appendix C: Samples

Here is a link to access any images, roadmaps and sample reports. You can download them by clicking here.