O'REILLY®

Full Stack Testing

A Practical Guide for Delivering High Quality Software

Free Chapter

Gayathri Mohan

Foreword by Dr. Rebecca Parsons

Full Stack Testing

A Practical Guide for Delivering High Quality Software

This excerpt contains Chapter 1. The complete book is available on the O'Reilly Online Learning Platform and through other retailers.

Gayathri Mohan

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

Full Stack Testing
by Gayathri Mohan

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Acquisitions Editor: Melissa Duffield
Development Editor: Jill Leonard
Production Editor: Jonathon Owen
Copyeditor: Rachel Head
Proofreader: Liz Wheeler

Indexer: nSight, Inc.
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Kate Dullea

June 2022: First Edition

Revision History for the First Edition
2022-06-03: First Release

See *http://oreilly.com/catalog/errata.csp?isbn=9781098108137* for release details.

978-1-09810-813-7

[LSI]

Table of Contents

CHAPTER 1
Introduction to Full Stack Testing

In today's world, digitalization is required to sustain and grow any business. Many businesses are leading the way in this aspect, while some are in the early phases of modernizing their existing digital platforms.

Digitalization is key to broadening a business's reach from a local community to a global scale, translating to more adoption and more revenue. Almost all small and large-scale enterprises in various sectors, like health care, retail, travel, academics, social media, banking, and entertainment, devise plans to advance their digital strategies as a critical measure to reach new customer segments and yield higher profits.

In this journey toward digitalization and modernization, innovation becomes a crucial driver. The businesses that innovate constantly continue to stay relevant and thrive over many decades. Netflix is a classic example: it started as an online DVD rental portal in the 1990s, then ventured into online streaming in 2007, cannibalizing its own DVD rental business. It later started producing original content, called *Netflix Originals*. As of the end of 2021, Netflix was the largest online streaming service with well over 200 million global subscribers.

The technology space has been evolving in parallel with these innovative businesses to accommodate their increasingly advanced needs. Gone are the days where people were willing to wait in line to buy movie tickets, drive to a store in a remote location to buy a specialty product, or carry around a handwritten shopping list searching for specifics. Technology eases such everyday tasks. We can sit at home and stream our favorite entertainment programs at the touch of a button, try on a new dress virtually, schedule regular delivery of the items on our grocery lists, brew a coffee with voice command, and more.

With the rapid pace of evolution in technology, product strategies must be versatile, catering to different customer needs to fend off competition in the sector at hand. It's

no longer enough to just build a website; horizons must be broadened. Consider ride-hailing companies like Uber and Lyft, which provide varying ways of accessing their services: the web, Android and iOS mobile platforms, even a WhatsApp chatbot. This kind of versatile product strategy has helped these companies expand across the globe and outgrow their competitors.

Innovation and versatility help businesses be successful in acquiring a critical mass of customers. But the challenge then is to thrive further, earning more revenue and gaining more customers. We've seen how industry giants like Amazon leverage their existing customer base to cross-sell services and products as a strategy for expansion. Amazon, which started as an online bookstore, now cross-sells products ranging from fresh pantry items to electronics to apparel, jewelry, and more, meeting customer demand for consumer goods in nearly every market segment.

Why are we discussing these details in a book on software testing? Because today's software industry caters to all these business needs, providing the tools to innovate new product ideas, bring them to life, and scale them to reach new customer segments across the globe. Unquestionably, software development teams are standing on the edge, especially when the need of the hour is to deliver *with high quality*! Indeed, software quality has become a nonnegotiable criterion in today's competitive market. Compromising on software quality is equivalent to running a race knowing that you will lose it. This has been reinforced by many real-world examples. For instance, in October 2014 Indian ecommerce giants Snapdeal and Flipkart went head to head on their seasonal sale after months of marketing. Unfortunately, Flipkart's website crashed multiple times during the "Big Billion Day" sale due to overwhelming demand, causing it to lose many customers and a great deal of revenue to Snapdeal. Similarly, Yahoo! failed to keep pace with its competitors (in spite of being one of the first of its kind in the market), in part because it failed to pay attention to the quality of its search product and in part because of damage to the brand caused by poor security measures, which led to the biggest data breach in history, exposing 3 billion user accounts in 2013. Such is the impact of software quality today!

There are many similar examples across the globe that reinforce the observation that businesses, despite whatever novel product ideas they might have, face a steep and slippery slope when quality is compromised, as customers quickly move on to more reliable competitors. At times, businesses may be forced to choose time-to-market over software quality, but they should be aware that they have only created a debt for themselves that needs to be resolved before their competitors use it to their advantage. Thus, we can firmly say that quality is quintessential for sustaining a business in the long term—and high quality can only be achieved through a combination of skillful development and meticulous testing, paying attention to every aspect of the application throughout its stack. To get you started on this path, this chapter will introduce what's involved in full stack testing for a typical web or mobile application.

Full Stack Testing for High Quality

To begin with, let us come together on a common understanding of *software quality*. Software quality was once equated to a bug-free application—but anyone in the software world today will agree that it's not just that anymore. If you ask end users to define quality, you will hear them speak of ease of use, look and feel, data privacy, swiftness in rendering information, and 24/7 availability of services. If you ask businesses to define quality, you will hear about return on investment, real-time analytics, zero downtime, no vendor lock-in, scalable infrastructure, data security, legal compliance, and more. All of these are aspects of what makes an application high-quality software today. Failings in any of these areas will affect the quality in some way or another, which is why we need to test for them meticulously!

Though the list of quality requirements looks tall, we have tools and methodologies to cater to most of these needs. So, the bridge to high quality is made up of knowledge of those tools and, more importantly, the skill to apply them in a given context, both from a development and a testing perspective. This book aims to help you build that bridge, teaching you the testing skills you need to deliver high-quality web and mobile applications.

Testing, in a nutshell, is a practice to validate that the behavior of the application is as expected throughout. For testing to be successful, it needs to be practiced at the micro and macro levels. It has to be entwined with the granular aspects of the application, such as testing every method in a class, every input data value, log message, error code, and so on. Similarly, it has to focus on macro aspects such as testing features, integrations between features, and end-to-end workflows. But we cannot stop testing there! We need to further test the holistic quality aspects of the application—security, performance, accessibility, usability, and more—to achieve the end goal of delivering high-quality software. We can encapsulate all of that by saying we need to do *full stack testing*! As represented in Figure 1-1, full stack testing involves testing different aspects of the application's quality in each layer (database, services, and UI), and the application as a whole.

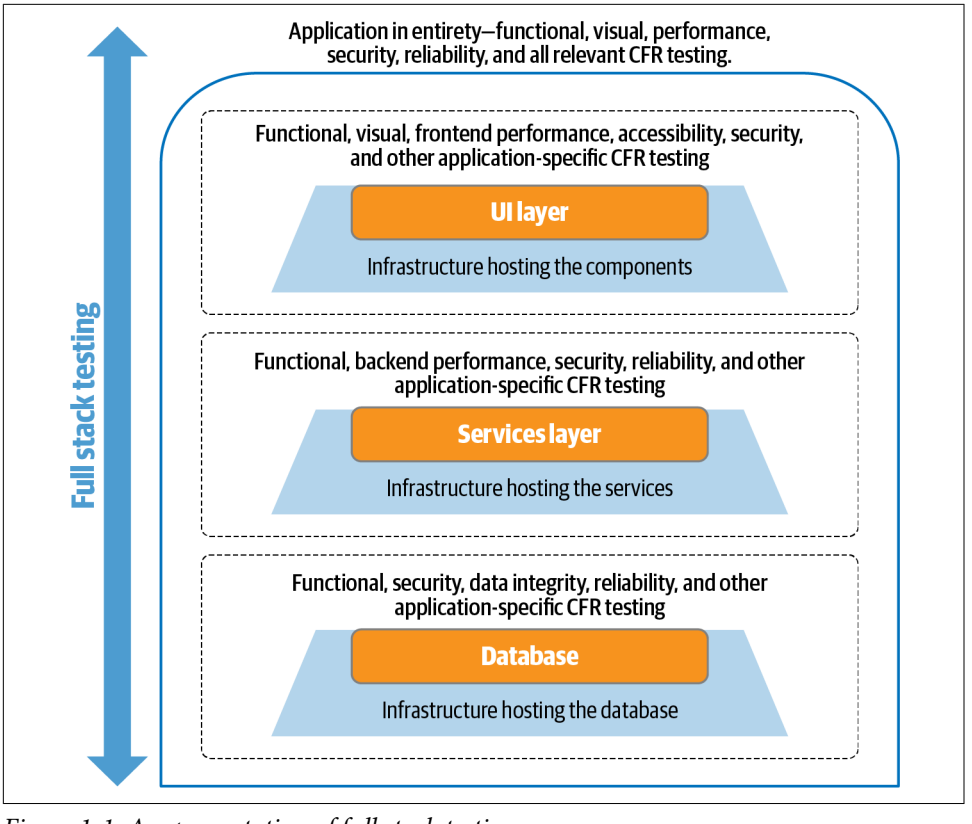

Figure 1-1. A representation of full stack testing

Indeed, full stack testing and development should be inseparable, like the two rails of a railway track. We must advance along both rails simultaneously to build quality into the product; otherwise, we are guaranteed to derail. For example, suppose we are writing a small block of code to calculate the total order amount for an ecommerce application. We need to test whether the code is computing the right amount and whether it is secure in parallel. If we don't do this, we might wind up with gaps in the railway line, and if we continue to develop on top of this fractured line we will have poor integration and suboptimal functionality. To ingrain testing at such an elementary level, teams need to stop thinking of it as a siloed post-development activity, as it was done traditionally. Full stack testing needs to begin in parallel with development and be practiced throughout the delivery cycle, giving faster feedback. The practice of starting testing early in the delivery cycle is referred to as *shift-left testing*, and it's a key principle to follow for full stack testing to yield the right results.

Shift-Left Testing

If we were to write down the sequence of activities in a traditional software development lifecycle, it would read *requirements analysis*, *design*, *development*, and *testing*, with testing coming at the end. As seen in Figure 1-2, shift-left testing suggests shifting the testing activities to the beginning of the cycle instead to produce high-quality results.

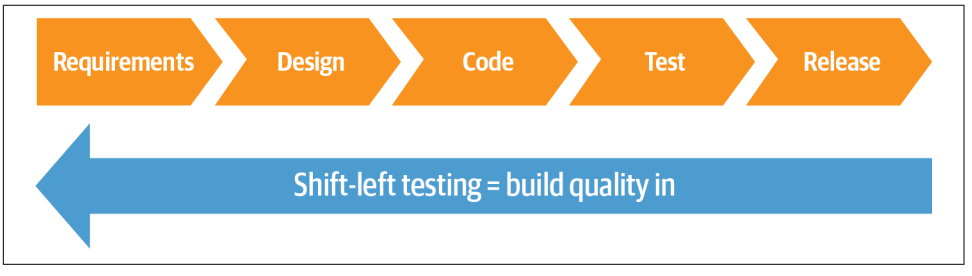

Figure 1-2. Shift-left testing

Let's consider an analogy to better illustrate this concept. Imagine your team is building a house. Does it seem sensible to complete the construction fully and only then check for quality? What if you find out the rooms are not of the correct sizes, or the interior walls are not strong enough to bear the load? Those are the kinds of issues shift-left testing tries to avoid, by implementing quality checks right from the planning stage and continuing them throughout the development phase. This allows for the end product to be of the highest possible quality.

Continuing quality checks throughout the development phase means repeating them iteratively for every small chunk of work, so that the needed changes can be incorporated smoothly. In the house construction analogy, it means performing these checks as each wall is built so that any issues are corrected immediately. To perform such extensive tests, shift-left testing relies heavily on automated testing and CI/CD practices, where the quality checks are automated at the micro and macro levels and continuously run against every small chunk of work in the CI server. This ensures the application is continuously tested, with minimal cost and effort compared to manually testing every small chunk of work for multiple quality aspects.

To see what this means in a software context, let's break down shift-left testing into day-to-day activities. Consider a software team that follows an iterative development cycle, such as in Agile development. Some of the quality checks they may do in different phases of delivery to shift testing to the left are captured in Figure 1-3.

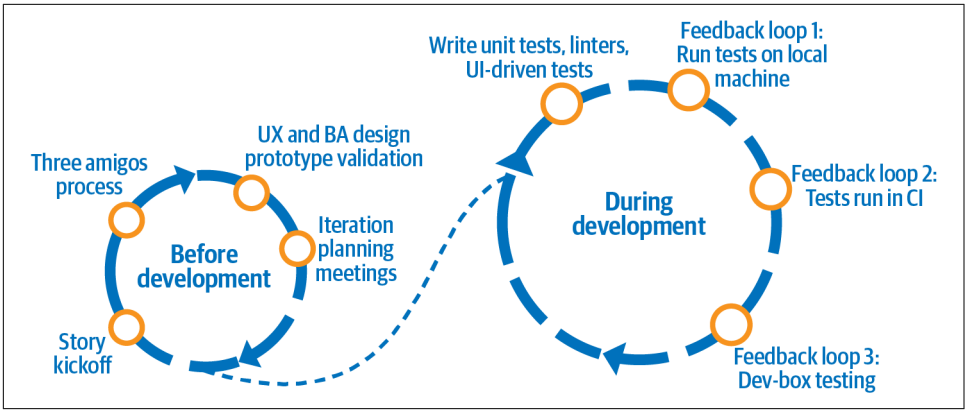

Figure 1-3. A set of quality checks shifted left

Reading Figure 1-3 from the left, it begins with highlighting a set of quality checks that are carried out by the team before a user story is considered ready for development:

- A ceremony called the *three amigos process* is conducted in the analysis phase. Here the business representatives, developers, and testers gather briefly to mull over the feature thoroughly. The process aims to collect all three roles' perspectives so that integrations, edge cases, and other business requirements don't get overlooked. This is the first step in shifting left, where the requirements of a feature are validated to begin with.
- In parallel, the business representative on the team works with the user experience (UX) designer to validate and improve the application design.
- Once these two steps are completed, an *iteration planning meeting* (IPM) is conducted at the beginning of the iteration/sprint to discuss the user stories of that iteration in detail. This provides an open space for the team to collectively validate the requirements once again.
- During the iteration, just before a user story is picked up for development, a *story kickoff* happens. The story kickoff is a minified version of the three amigos process where the focus of discussion gets deeper into that particular user story's requirements and edge cases. By this stage, we can fairly say that the team has tested/validated the requirements diligently.

Similarly, while developing a user story, the following quality checks are implanted and utilized to get fast feedback:

- Developers write unit tests as part of each story and integrate them with CI. They also add linting tools and plug-ins for static code analysis and integrate them with CI to get continuous feedback.

- In some teams, developers also write the UI-driven functional tests as part of user story development and integrate those with CI. In other teams, testers write them post-development; both are common practices.
- Before committing the latest changes, developers run a set of automated tests on their local machines to get the first level of feedback.
- The second level of feedback is obtained from the suite of automated tests (unit, service, UI, etc.) that are run during CI for every commit.
- The third level of feedback is received from a process called *dev-box testing*, where the testers and the business representatives do a quick round of manual exploratory testing on a developer's machine to quickly verify the newly developed functionality.

With such rigorous focus on providing faster feedback, the team will get almost half of the feedback that would have otherwise been gained through manual testing post-development before the user story even gets to the testing phase itself. In other words, the team just shifted testing to the left, in the process giving the testers on the team the liberty to fully explore the user story for various quality aspects rather than just verifying the expected functional behaviors.

Thus, shift-left testing both enables defect prevention (by having multiple rounds of validation on the requirements) and assists in catching any defects that do creep in early, either on a local developer's machine or in CI. In addition, it ensures that high-quality software is delivered by giving testers the space to explore various quality aspects in depth.

Extreme Programming (XP) is a flavor of Agile software development framework that incorporates shift-left testing. If you'd like to delve further into XP methodologies and practices, Kent Beck's *Extreme Programming Explained* (Addison-Wesley Professional) is a recommended read.

This concept of incorporating testing earlier in the delivery cycle is not restricted to functional application testing. It can be applied to testing in general, including security testing, performance testing, and more. For example, one of the many ways to shift security testing to the left is to use a pre-commit scanning tool like Talisman, which scans the commit for secrets and alerts even before checking in the code. In each of the upcoming chapters, you will see practical approaches to shift-left testing.

Overall, this approach embodies the aphorism "Quality is the team's responsibility," as performing quality checks at every phase of the software development life cycle—validating application design prototypes, requirements, and so on, as discussed earlier—has to be owned by different team members. So, we can say that building the relevant

testing skills to perform various quality checks is crucial for all the roles in a team to deliver high-quality software successfully!

Ten Full Stack Testing Skills

When we think of testing skills, we tend to consolidate them into two broad skills—manual and automated testing. But technology has evolved over the course of the last several decades, and these broad terms mask the essential new skills that one has to learn to perform various quality checks and deliver high-quality web and mobile applications. Figure 1-4 shows the 10 full stack testing skills that will enable us to perform full stack testing efficiently.

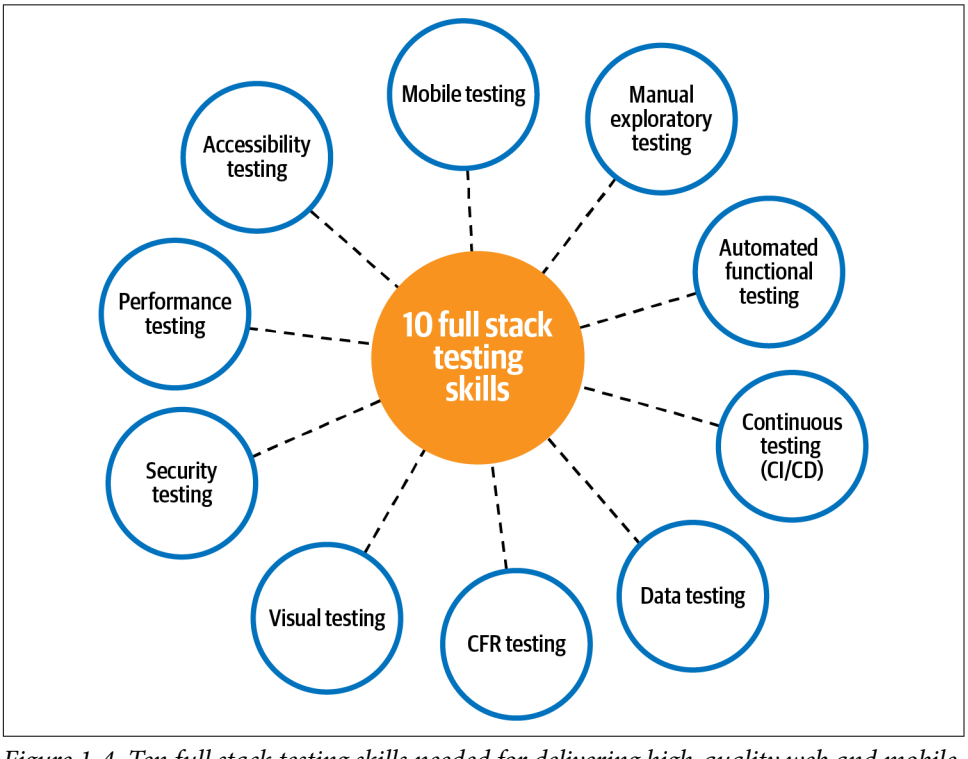

Figure 1-4. Ten full stack testing skills needed for delivering high-quality web and mobile applications

Let's explore these 10 skills, and why you should care about learning them:

Manual exploratory testing

Firstly, to clarify, manual exploratory testing is different from manual testing. The latter refers to verifying a given list of requirements and doesn't necessarily demand an analytical mindset. In contrast, manual exploratory testing is the skill

of delving into the application details, coming up with different real-life scenarios apart from what is documented in user stories, simulating them in a test environment, and observing the application's behavior. It demands a logical and analytical mindset and is the first and foremost skill required to create a bug-free application. There are various methodologies and approaches that can be learned to structure these exploration sessions, which we shall discuss in Chapter 2.

Automated functional testing
: This is one of the core skills for shift-left testing, as discussed earlier. Doing automated testing also significantly reduces manual testing effort, especially when the application grows to include more features. In simple terms, the skill here is to write code to test feature requirements automatically, without human intervention. To do this we need tools, and therefore knowledge of different tools that can be used to write tests at different application layers is required to acquire this skill. It doesn't stop there, however; one also needs to know what antipatterns to look for in automated testing and how to stay clear of them. We'll discuss the different aspects of this skill in Chapter 3.

Continuous testing
: Continuous delivery is a practice where features are delivered incrementally to end users in short cycles, instead of through a single big-bang release. By continuously delivering, the business earns profits early and can assess and retune its product strategy quickly based on end user feedback. To power continuous delivery, we have to test the application continuously so that it is always in a ready-to-be-released state. As obvious as it may seem, the wise way to do this is to automate and integrate quality checks into your CI/CD pipelines and run them frequently to ease the testing process. The skill of continuous testing involves determining which types of automated tests should be run at each stage of the delivery cycle so that the team can get faster feedback and integrate them effectively into the CI/CD pipelines. These essentials are discussed in detail in Chapter 4.

Data testing
: You may have heard the sayings "Data is money" and "Data is the new oil." These ideas highlight how important testing for data integrity is today. When users' data is lost, or the application shows the wrong data to end users, they lose trust in the application itself. The skill of data testing requires knowledge about the different types of data storage and processing systems typically used in web and mobile applications (databases, caches, event streams, etc.) and the ability to derive appropriate test cases. Chapter 5 discusses these topics, and how data flow between the application components creates new test cases apart from the functional flows.

Visual testing
: The look and feel of the application is a major contributor to the business's brand value, and especially when it comes to big business-to-customer (B2C) products used by millions, low visual quality can impact brand value instantly. Therefore, it is essential to validate that end users have a harmonious and pleasant visual experience by conducting visual testing of the application. Visual testing requires an understanding of how the UI components interact with each other and with the browser, for web applications. Such checks can be automated too, using tools that are different from those used for automated functional testing. We will talk about this skill, and about the stark differences between these two types of automation, in Chapter 6.

Security testing
: Security breaches have become all too prevalent in today's world, and not even giants like Facebook and Twitter are excluded from such attacks. Security issues have a heavy cost for both the end users and the business in terms of loss or exposure of sensitive information, legal penalties, and brand reputation. So far, security testing has been viewed as a niche skill in the industry, with qualified penetration testers typically engaged only toward the end of the development cycle to look for security issues. But with the lack of available professional security testing talent and growing incidence of security breaches, software teams are well advised to incorporate basic security testing as part of their day-to-day work. We will discuss how to think like a hacker and seek out security issues in application functionality in Chapter 7, along with tools to automate security scans.

Performance testing
: Even a slight drop in application performance can lead to huge financial and reputational losses for a business—recall the Flipkart example discussed earlier. The skill of performance testing involves measuring a set of key performance indicators at different application layers. Performance tests can also be automated and integrated with CI pipelines to get continuous feedback. We will discuss a shift-left performance testing strategy along with relevant tools in Chapter 8.

Accessibility testing
: Web and mobile applications have become everyday commodities. Making them accessible to people with permanent or temporary disabilities is not only mandated by legal regulations in many countries, but also ethically the right thing to do. In order to acquire the accessibility testing skill, we must first understand the accessibility standards required by law. We can then use both manual and automated accessibility auditing tools to validate whether those standards are met. We will discuss this skill, and why incorporating accessibility features may even be a lucrative option for businesses, in Chapter 9.

Cross-functional requirements testing

We've seen that end users and businesses have a tall list of quality requirements, such as availability, scalability, maintainability, observability, and so on, apart from just needing bug-free functionality. These are called the *cross-functional requirements* (CFRs) of an application. Although functional requirements generally grab the most attention, it is the CFRs that imbue quality into the application, and failing to test these will lead to unsatisfied business or software teams, end users, or both. Therefore, CFR testing skill is a fundamental testing skill. We will discuss the testing methodologies and tools for validating different CFRs in Chapter 10.

CFRs are also referred to as *non-functional requirements* (NFRs) by many in the industry. We will discuss the subtle differences between these two terms in Chapter 10.

Mobile testing

The sheer number of apps available on the leading app stores (Google Play and the Apple App Store) in 2021 may come as a surprise—a total of 5.7 million. The explosion in the number of mobile apps stems mainly from our increased usage of mobile devices. Indeed, the web analytics company Global Stats announced in 2016 that their data showed mobile and tablet internet usage across the globe had surpassed desktop usage. So, the ability to test mobile applications and the compatibility of websites across mobile devices is a critical skill today.

Although all of the previously mentioned skills are required for testing mobile applications, it requires a change in mindset too. Additionally, a whole set of mobile-specific testing tools have to be learned in order to perform various quality checks on mobile applications. Therefore, mobile testing is carved out as a separate skill here. We will traverse the nuances of the mobile landscape in Chapter 11.

Together, these 10 full stack testing skills will enable you to test the full scope of holistic quality aspects of web and mobile applications. As mentioned earlier, it's important for every role in the team to acquire some degree of competency in each of these skills. The book will show you how, skill by skill, with practical examples.

Key Takeaways

Here are the key takeaways from this chapter:

- Software quality cannot be equated to just bug-free functionality anymore. An application can be deemed suboptimal in quality if its holistic quality dimensions (security, performance, visual quality, etc.) are not on par.
- Full stack testing refers to testing all the quality dimensions of an application holistically at every layer, thereby delivering high-quality software.
- For full stack testing to meet its goal of delivering high-quality software, teams should shift testing to the left, so that it begins in parallel with analysis and continues throughout the delivery cycle.
- Shift-left testing embodies the aphorism "Quality is the team's responsibility," as it demands that every role in the team take ownership of performing certain quality checks at different phases of delivery. This requires all team members to upskill themselves, acquiring relevant testing skills at varied competency levels.
- The two classic monolithic categories of testing skills, manual and automated, mask a vast set of new testing skills required to perform full stack testing efficiently. This chapter introduced 10 different testing skills that are essential for delivering high-quality web and mobile applications today, which we will explore over the course of the following chapters.

About the Author

Gayathri Mohan is a passionate technology leader with expertise across multiple software development roles and technical and industrial domains. Gayathri has proven her mettle by successfully managing large quality assurance (QA) teams for clients at Thoughtworks, where she is now principal consultant at Thoughtworks. While working as the company's global QA SME, she defined career pathways and the desired skill development structure for QAs at Thoughtworks. As office tech principal, Gayathri cultivated local tech communities, organized technical events and developed thought leadership across technical themes.

Gayathri is also coauthor of *Perspectives of Agile Software Testing*, released by Thoughtworks on Selenium's 10th anniversary.

Colophon

The animal on the cover of *Full Stack Testing* is a lowland streaked tenrec (*Hemicentetes semispinosus*). These small insectivorous mammals are one of many species of tenrecs found on the island of Madagascar. Lowland streaked tenrecs are typically found in scrubland, tropical lowland rainforests, agricultural land, and even some rural gardens on the eastern side of the island.

Lowland streaked tenrecs are easily identified by their long, pointed black snouts and small, tailless bodies striped with black and yellow quills. A crest of yellow spines covers the back of their necks. Their barbed quills are detachable and can be used as a defense mechanism; tenrecs also use the quills to communicate by rubbing them together, producing a high-pitched sound. Fully grown lowland streaked tenrecs are about five to seven inches long and weigh between four and ten ounces.

Lowland streaked tenrecs are social and gather in groups of up to 20. They dig connected burrows for nesting and forage for earthworms and insects individually or in small groups. In the winter, they go into torpor, a state of reduced body temperature and decreased metabolism. Females are only fertile for one year and are reproductively active at 25 days old, making them the only species of tenrec that can breed in the same season in which they were born. Lowland streaked tenrecs are classified as a species of least concern by the IUCN due to their widespread distribution, high abundance, and high tolerance to areas with large numbers of humans. Many of the animals on O'Reilly covers are endangered; all of them are important to the world.

The cover illustration is by Karen Montgomery, based on a black and white engraving from *English Cyclopedia*. The cover fonts are Gilroy Semibold and Guardian Sans. The text font is Adobe Minion Pro; the heading font is Adobe Myriad Condensed; and the code font is Dalton Maag's Ubuntu Mono.