Perfecto
by Perforce

How to Create a Mobile App Testing Strategy With Virtual & Real Devices

Introduction

Testing mobile apps — whether they're native, hybrid, or web — requires a solid strategy. The mobile landscape continuously evolves. It is complicated and inconsistent in its behavior, especially when tested together with other activities on the device. While the mobile ecosystem consists of two major platforms — iOS and Android — each platform is quite different, fragmented, and they each require a different testing process.

Developers and testers have to choose between two target platforms to test against upon each phase of their development life cycle: virtual platforms and real platforms. Compiling all testing types, including unit testing, integration testing, functional and non functional testing, production monitoring testing, and API testing into a single strategy based on proper considerations is a great challenge.

In this eBook, we will outline the key differences between the two types of platforms and provide a recommended practice around when and how to use each of them for a maximum test coverage objective that eliminates risks for escaped defects.

A virtual platform for the Android OS is called an emulator, while for iOS it's called a simulator. An Android emulator, as well as an iOS simulator, is a software tool that can emulate or run its respective operating system on your desktop or laptop machine to enable development and testing activities. Most of these virtual platforms are embedded within the core IDEs (Independent Development Environment), like Google's Android Studio and Apple XCode.

Developers can use these platforms to deploy their apps in early development stages, and even advanced stages, to understand how they behave and what they look like. They can also be used to perform extensive debugging from local machines. While this is a powerful, free, and easy solution to get started with, it is not a complete offering for the entire development lifecycle. In the next section, we will highlight some of the material differences between virtual platforms and real devices.
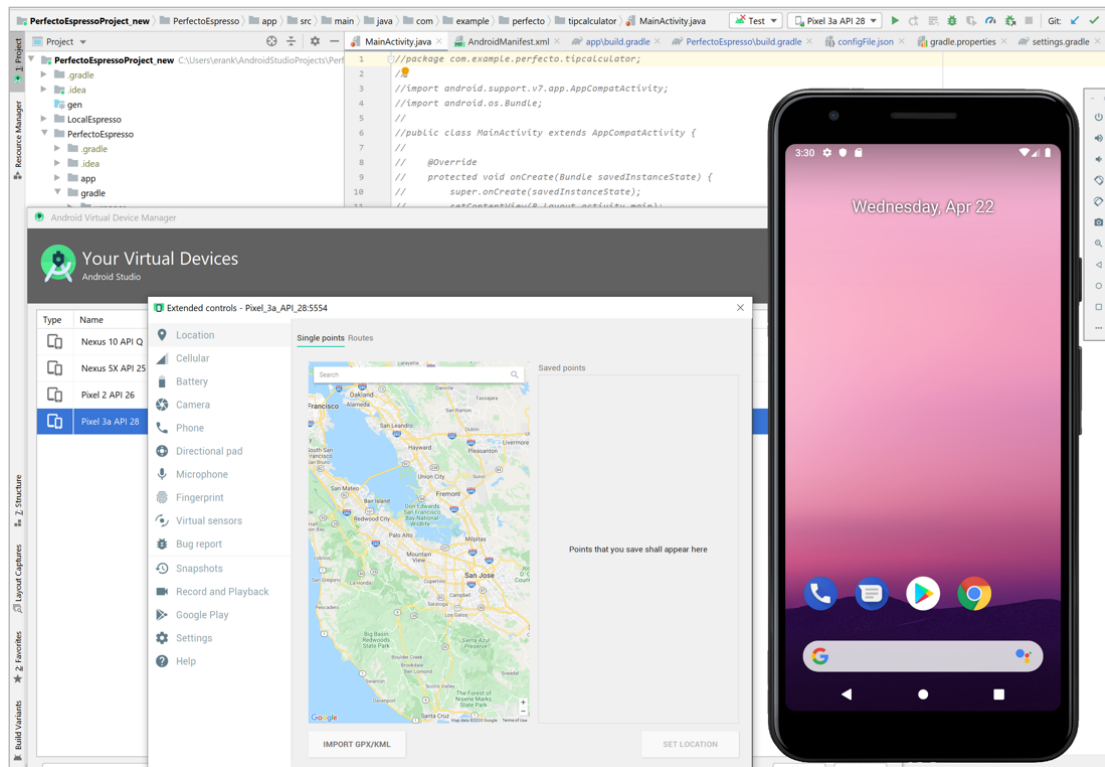
Android Virtual Device (AVD) Executed Within the Android Studio IDE
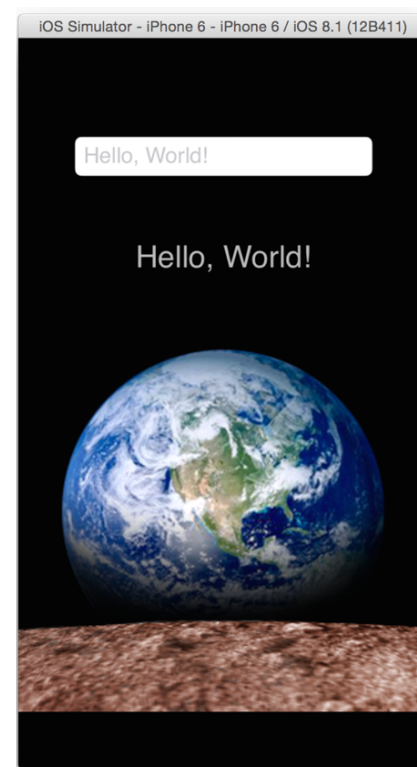
Apple's iOS Simulator

DIFFERENCES BETWEEN IOS AND ANDROID VIRTUAL PLATFORMS

| Capability	Android Emulator	iOS Simulator	Comments
App Store Testing	Supporting non NDK apps	Unsupported	
Testing Target (AUT)	.APK file	.APP file	
Sensors Support (GPS, mic, etc)	Camera requires instrumentation	No face recognition support	
Mimic Real Device HW	Partially	Partially	
How to Access It	Basic emulators are part of Android SDK/Android Studio. Perfecto supports Android emulators.	Apple XCode environment supports iOS simulators. Perfecto supports iOS simulators.	
Platform Testing Stability	PC/Server maintenance	PC/Server maintenance	As opposed to mobile device updates
App Store Installations	Supported (only non NDK apps)	Unsupported	
Push/System Notifications	Supported	Unsupported	
Performance Metrics	Unsupported	Unsupported	Virtual platforms don't reflect real device UX timers
MDM Support	Unsupported	Unsupported	Can't install MDM on virtual platforms
Calls/SMS/2FA	Very limited (via screen mock)	Unsupported	
Security	NA	APP file – used on any simulators	
Non-Native Browser Support	Unsupported	Unsupported	
Background Apps Support	Doesn't reflect same processes due to different architecture.	Doesn't reflect same processes due to different architecture.	
Device Settings	Supported	Unsupported	
End User OS Version Testing	Basic image, no device OEM OS	Supported	

Contents

Part 1: Virtual Platforms vs. Real Devices — A Comparison

Using a virtual platform for Android or iOS development and testing from the local IDE is a powerful solution. However, it cannot fulfil all of the mobile app testing and development requirements.

If we examine the two mobile OS platforms through their fragmentation, this alone shows a large scale of possible permutations with over 6 Android OS families and 3-4 iOS families that need to be considered for testing.

The two mobile OS platforms are distributed differently across different geographies. This means that in some countries, users leverage more and different Android smartphones than iOS devices, and vice versa — a fact that needs to be considered in the test coverage strategy.

To understand more on the core differences of the platforms, here is a high-level distinction between them.

ANDROID OS MARKET SHARE
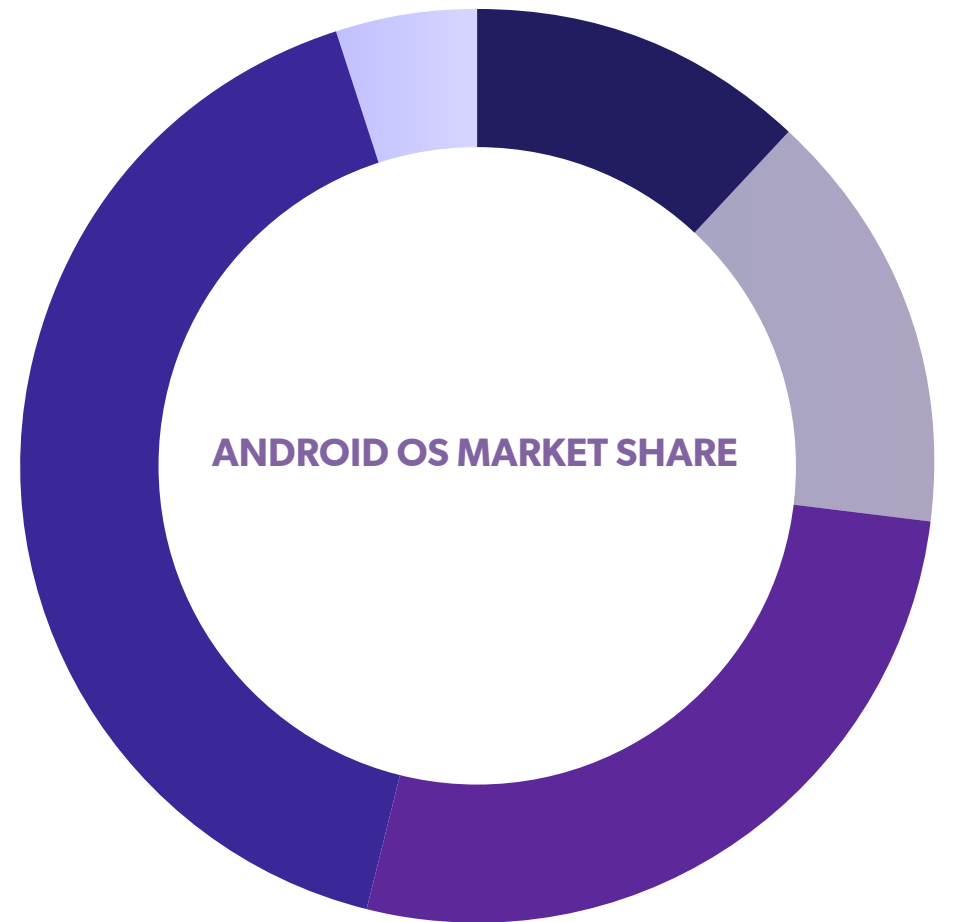
● Android 6.x (Marshmallow) ● Android 7.x (Nougat) ● Android 8.x (Oreo)

● Android 9 (Pie) ● Android 10 (Q)

OPERATING SYSTEMS

It is a fact that the OS versions that are installed on an Android AVD and on real Android devices are different. Teams should realize the differences between the Android **stock OS** versions that run on AVDs and real Google Pixel devices versus the OS versions that are deployed on other real devices. Each device vendor (OEM) modifies the stock OS version and builds a unique flavor of it to run on its devices.

The table below shows a simple comparison between the **custom OS** versions that run on various Samsung, Huawei, LG, and Oppo devices and the one that runs on the Google Pixel device.

In addition to the OS version, developers and testers ought to know that the hardware configuration that is used on virtual devices is, in most cases, irrelevant and unrepresentative of what is running on Android or iOS devices. An Android generated AVD uses the **PC/ laptop resources** with a non-realistic simulation of memory and battery compared to a **Samsung S20 Snapdragon 865** chipset or an **iOS A13 bionic** chipset.

When you create an app, it is perfectly fine to use a few virtual device platforms for designing it and performing basic testing. Beyond this, however, it needs to be clear that the outcomes of testing on virtual vs. real devices are different by definition.

Device Model	Custom OS Implementation Versions	Base Stock OS
Samsung Galaxy S20 Ultra	One UI 2	Android 10
Samsung Galaxy S7	TouchWiz UI	Android 8.0
Samsung S8	One UI	Android 9.0
Samsung Note 9	One UI 2	Android 10.0
Google Pixel 4 XL	STOCK	Android 10
Huawei Mate 20 Pro	EMUI 9.1	Android 9.0
LG G8 Thin Q	LG UX 9.0	Android 10.0
Oppo Reno 5G	Color OS 6	Android 10
Huawei Y9 Prime	Magic UI 2.1	Android 10

FUNCTIONALITY SUPPORT

In addition to OS differences, there are functionalities that cannot be tested on virtual devices, only on real devices. Among these are, for example, unique popup notifications, MDM (Mobile Device Management) system integration, security-related features, such as **2FA** (two-factor authentication), real cellular network behavior, and more.

Because each app is designed to work on real devices, where a lot of real-life events take place, it is critical to validate the app under conditions that are as close to the production environment as possible. Features such as brightness and background color of the device display or GPU abilities are unique to real devices.

Virtual platforms support **biometrics** simulation (fingerprint and face ID). However, they cannot cover all the different types of real user options. Sending a pass/fail command to the virtual platform is one thing, but ensuring that two different human faces can log in to an app is something else. Performance of an app measured on a real device is far more accurate than what can be logged on an AVD/simulator.

For example, on iOS simulators, users cannot change device settings and see the impact on the app under test. While Android AVDs allow the configuration of some settings, on iOS, this is a real-device-only capability. Also, testing your app alongside production app store applications is quite limited across Android and iOS. For iOS, it is not supported at all on virtual platforms. For Android, it can be done through a Google Play extension for non-NDK apps.

Another important functional aspect is the application under test. Specifically for iOS, the app that is being tested on the simulator (iPhone/iPad) is of a different type (**.app**) compared to a real device app under test (**.ipa**). This means that the package that is being deployed and tested across the two types of platforms is 100% different.

As mentioned above, integrating virtual devices with an enterprise **MDM** system is unsupported. Therefore, if the testing of the app goes through the organizational MDM system, this will be a problem.

At the screen and UI layer, having the ability to test on real devices across multiple **screen sizes, resolutions, and "skins"** is by far more flexible and advanced than what Android and iOS virtual platforms can offer.

Lastly, testing apps on virtual devices vs. real devices can pose security threats. An iOS .app file that is targeted to simulators can be installed outside of the organization without being signed, while an .ipa app requires a signature prior to being installed on specific devices (pre-production).

There are also **browser**-related differences, for example, regarding **localization** and **interruptions** between native OS devices and virtual platforms. However, the above examples provide a great representation of some functional differences between the two platforms.

ENVIRONMENT CONDITIONS

Mobile apps are developed to be consumed by real end users that carry real devices, not virtual, and as such, apps are required to be fully tested against real end-user conditions. Such environment conditions consist of real cellular network connectivity, background production apps running on the device, and interference with app functionality, as well as real system notifications like push, security, and others.

Hardware constraints — such as battery, memory, sensor connectivity (Bluetooth, GPS, etc.) — and **sensors** — such as accelerometers, gyroscopes, or proximity sensors — are additional aspects of a real device environment setup.

In the below image, a real device running a location-based application (Waze) is shown with a full list of activities happening in the background — this image by itself can be considered a representation of a real-device testing environment:

- Apps running in the background.

- Apps pushing notifications to the users.

- Real cellular network conditions.

- Location services on.

- Battery not fully charged.

- Competition on resources from various apps.
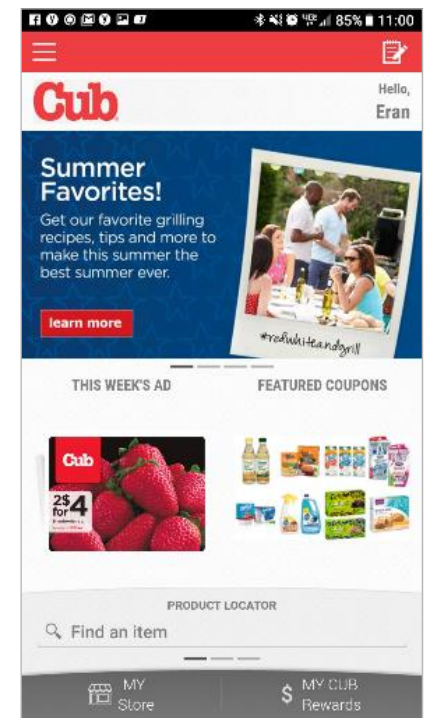
TESTING OBJECTIVES

It is essential to understand the roles of each platform type within the entire DevOps lifecycle. Early in the development cycle, a virtual platform is perfect and provides solid logging and debugging capabilities to the developer. But as the development progresses, the testing scope and requirements from both developers and testers grow and require more advanced abilities that exist only on real devices.

In addition, a test engineer is typically tasked with more rigorous and advanced testing types and activities than a developer. Unit and API testing can be done within or outside CI (continuous integration), while functional and non functional E2E testing is usually under the responsibility of the testing team.
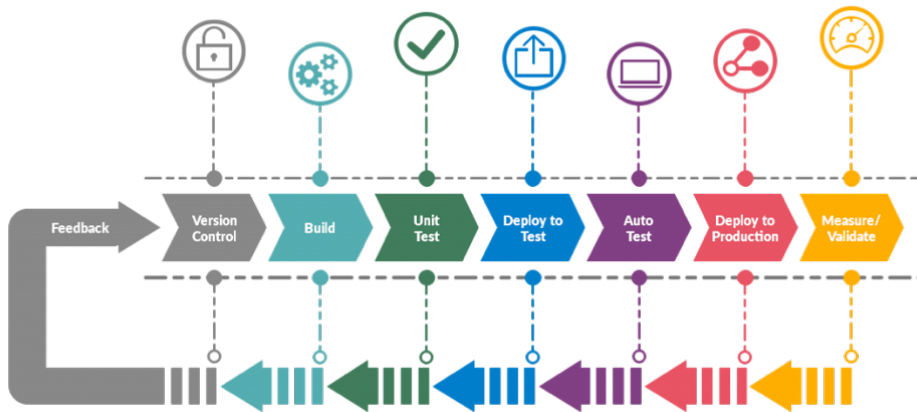
Finally, a Go/No-Go release decision can only happen after a solid testing cycle was performed on real devices and configured against real environment conditions.

COST

Any IT team looks at the overall costs of development and testing activities continuously. Android AVD emulators and iOS simulators are completely free and embedded within the IDEs — that's a given. On the other hand, real devices come with a cost to purchase, maintain, and manage.

The key to a successful and cost-effective strategy that includes both platforms relies on proper balance and sizing throughout all types of dev and test activities. As mentioned above, developers have unique and different use cases as well as objectives compared to test engineers — this needs to be considered and reflected in the scoping of the two types of platforms.

Later in this eBook, we will provide a recommended practice that looks at the test pyramid and ongoing activities and provides a prescriptive approach for when and how to use both platforms.

## SUMMING UP DIFFERENCES BETWEEN REAL & VIRTUAL DEVICES

In summarizing the differences between real devices and virtual platforms, it becomes clear that by nature, both serve different development and testing needs and are required in complementary phases of the development cycle. Real users report real defects on real devices, not on virtual ones; hence, it is essential that organizations properly balance the usage of virtual and real devices.

Many apps, throughout the App Store and across market segments (media, travel, financial, eCommerce, etc.), constantly reveal defects, bug fixes, and performance improvements that could have been eliminated in the first place through proper testing against the right mix of platforms.

Next, we'll walk through a mobile app testing strategy that combines both real and virtual devices.

The images to the right serve as real life example and a representation of defects that were reported on real devices both from a visual perspective as well as device-specific functionality.

Both issues could have been observed prior to the application release into the App Store. And they could have easily been avoided.
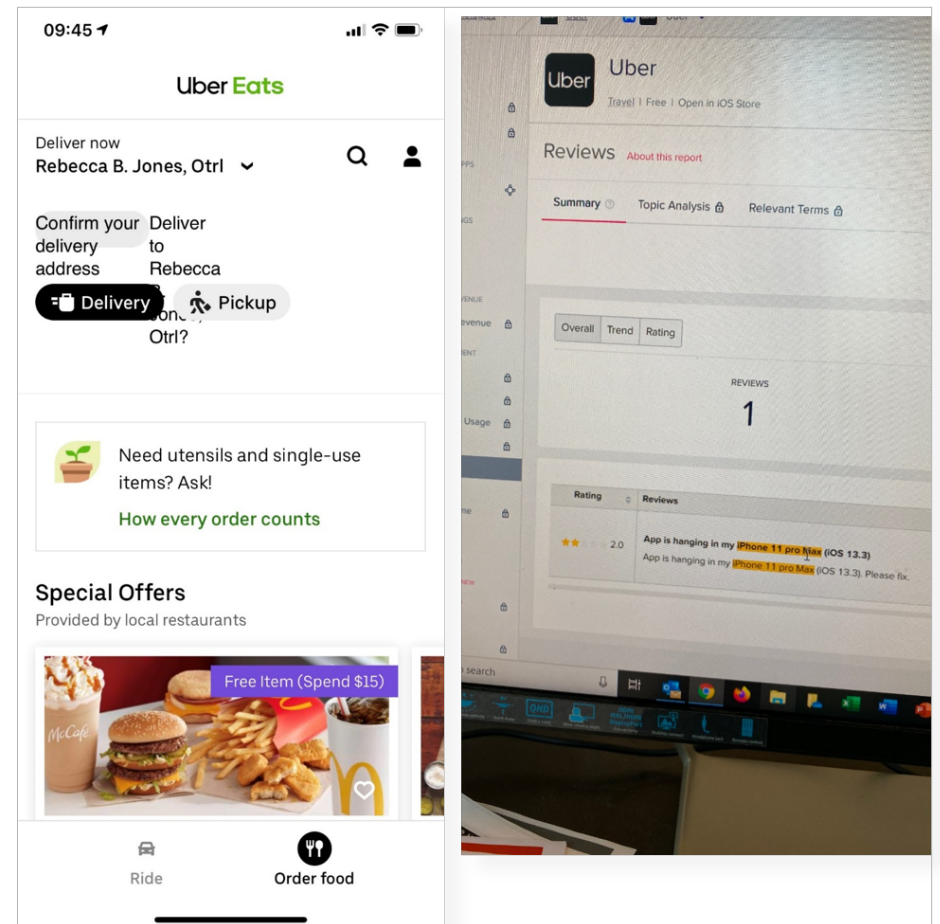
Uber Eats and Uber Ride Share Application Issues on Real Devices

Part 2: Testing Strategy Recommendations for Virtual & Real Devices

Regardless of the testing methodology that your organization follows, whether it is Agile, BDD, TDD, or something else, the testing coverage needs to be solid and adhere to product requirements, end-user conditions, and target geographies. In addition, the coverage must meet the different phases of the development lifecycle.

In the below table, we have gathered as an example the different platform considerations per major DevOps pipeline stages. **Early in the feature development cycle**, when development happens on a

local machine, using a small subset of platforms that are **virtual** can be sufficient and provide the necessary feedback and debugging/log information.

Android developers can build Android .apk files and execute their **Espresso** or **Appium** tests on Perfecto's **emulators** and other Android **AVDs**. Similarly, iOS developers build their target .app files and run them against simulators in the Perfecto cloud or local XCode simulators to obtain fast feedback.

PLATFORM COVERAGE		Unit Testing P1		Build Acceptance Test P1 + P2	Acceptance Test Regression & Non-Functional	Production P3
	Test Platform	1 iOS (device/simulator) 2 Android (device/emulator) 2 Desktop Browsers		Essential (Top 10) Mobile & Web Platforms	Enhanced/Extended Coverage (Top 25-32 Platforms)	2 iOS 2 Android 2-4 Browsers
	Trigger	Per-commit	Post-commit	Scheduled Daily	Scheduled Nightly	Scheduled Hourly
	Environment	Dev Workstation		Continuous Integration Server		Production

As the app matures and requires more **integration, functional, and non functional testing** as part of the CI phase, use **real devices**. In this phase, configuring CI or other data providers (TestNG, etc.) against the Perfecto cloud will allow you to scale up testing and obtain feedback on the actual .ipa/.apk builds.

Close to the **release phase**, and as part of the acceptance testing, adding additional **real device/OS configurations** that match the client usage analytics and market trends is advised.

Below is a visualization of the famous testing pyramid that captures most types of testing based on the phases in the lifecycle, with a recommended type of platform for each.
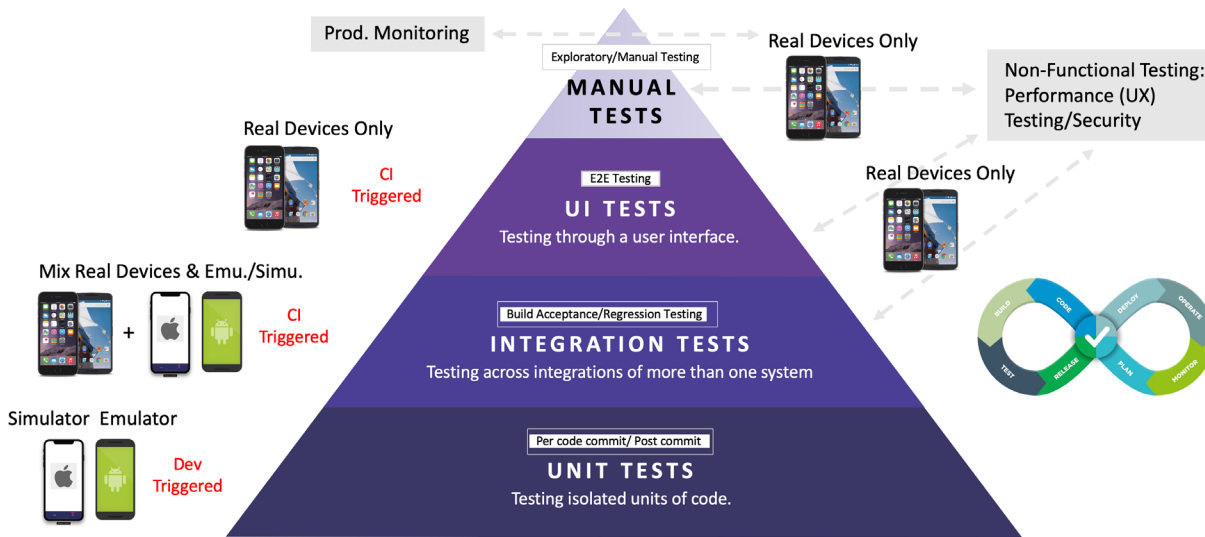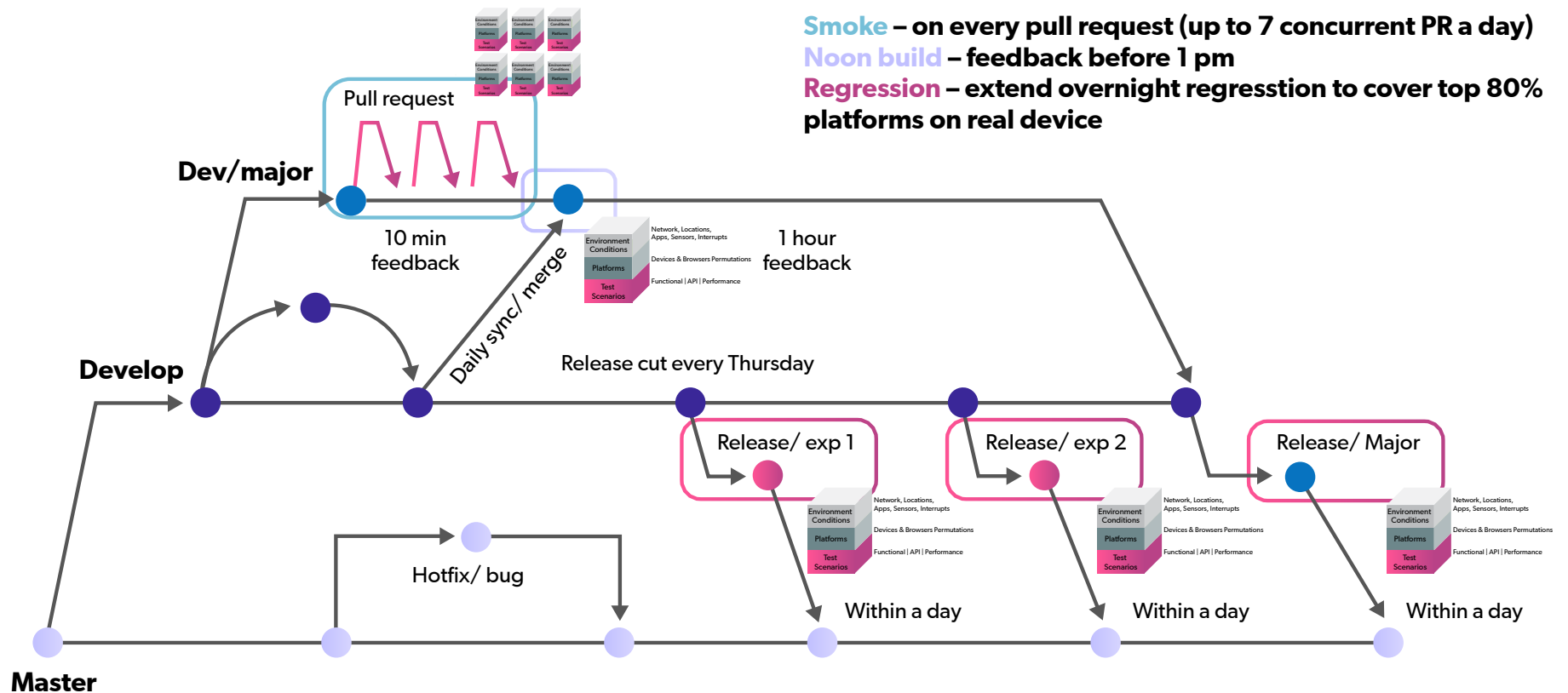
In many organizations, there are exceptions to the testing models. When time is short to deliver feedback on a large amount of pull requests (PRs) and code changes, it forces QA engineers and developers to scale, mix, and size their labs differently to prevent escaped defects to production and cope with the amount of product changes that are happening.

In light of this, teams may find themselves creatively resizing their lab with more or less real/virtual devices in various stages and at peak times within the development life cycle.

Bringing the above-mentioned strategy into reality, below is a real-life product release train flow that iterates software testing a few times a day, triggering validations upon code changes, during lunch breaks, and on a daily basis.

Such a workflow balances the use of both real and virtual platforms and provides appropriate fast feedback upon each code change several times a day. This is a highly-recommended model to follow and scale or size your testing environment and lab accordingly.

Smoke – on every pull request (up to 7 concurrent PR a day)
Noon build – feedback before 1 pm
Regression – extend overnight regresstion to cover top 80% platforms on real device

Scaling a lab and sizing it based on the above workflow depends on the availability of the following parameters:

1. Number of test scenarios per each type of test cycle.

2. Average test scenario duration.

3. Number of of target platforms required.

Based on these factors and the time windows that are given (e.g., 1 hour for noon build testing), parallel testing can be planned to fit this constraint.

Profile	Location	Smartphone	Tablet	Network Conditions	Orientation	Background Apps
Georgia	New York	iPhone 11 Pro Max	iPad Pro 11''	5G & Wi-Fi	Vertical + Horizontal	
Ross	Florida	Samsung Galaxy S7	X	Average 4G LTE	Vertical	
Peter	UK	Huawei P30 Pro	X	Poor 4G LTE	Vertical	
Sam	Chicago	Samsung Galaxy Note 10	Samsung Galaxy Tab S5E	Average 3G & Wi-Fi	Vertical + Horizontal	
Sara	California	Google Pixel 3A	X	Poor 5G & Wi-Fi	Vertical	

All of the above testing must also consider the real user conditions. Above is an illustration of various personas that can serve as testing environments for your mobile apps, mostly focusing on real device testing.
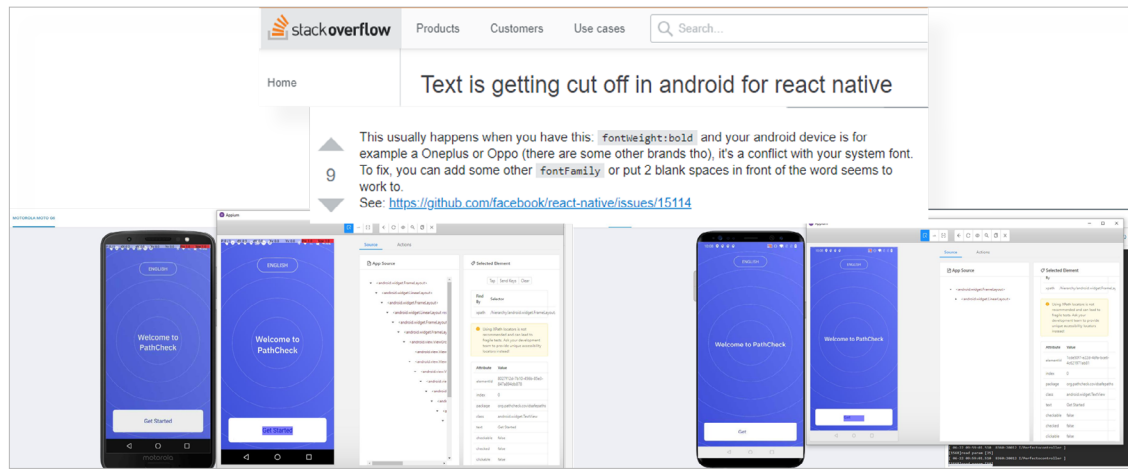
The defect above is an example taken from a react-native mobile application, that works fine on Android emulators, real Google Pixel and Samsung devices, however, when being installed and ran on OnePlus 7T or Oppo devices, text is getting cut from the app screens - This is a great example to why the mix of platforms is key for defects and quality risks mitigation.

Part 3: Test Automation on Virtual Platforms & Real Devices

In most cases, test automation does not differ much across platforms. Most platforms support common test automation frameworks. **Appium** can be used across iOS and Android. **Espresso** can be used on Android emulators and real devices. And **XCUITest** can be used on iOS simulators and real devices. The key differences between automation testing on real devices vs. virtual ones are the environment settings, capabilities, and extended test automation scenarios.

Below is a focused example of an Appium test automation framework that can be used to create and execute test automation artifacts agaist both virtual and real devices.

APPIUM

Appium is the de-facto cross platform test automation framework for testing native and hybrid iOS and Android apps. When testing on virtual platforms with Perfecto, clients should specify a unqiue command in the desired capabilities code block to pick a virtual platform, rather than a real device.

Below is the specific command for a virtual iOS iPhone 11 Pro simulator with the "useVirtualDevice" option set to true.

```
capabilities.setCapability("deviceName", "iPhone 11 Pro");
capabilities.setCapability("automationName", "Appium");
capabilities.setCapability("useVirtualDevice", true);
```

Without specifying this option when running the test on the Perfecto cloud through Appium, the default will be to run the test against a real iOS device.

As mentioned above, and specific to iOS, testing on a virtual platform requires an .app file, while running the test on a real device requires an .ipa file. All other configurations and the test code are the same.

To learn more on Perfecto's support for virtual devices, required configurations, and more, please refer to the Perfecto documentation portal.

Summary

The mix of virtual and real devices is not an option, but rather it is a reality and a recommended best practice. The balancing act between using the two types of platforms is what needs to be stratigically considered.

As highlighted in this eBook, each platform brings unique benefits and values, and each platform can be used at a varying scale per each code change, phase in the development lifecycle, and other product objectives.

At the end of each testing activity per software iteration, the goal is to continuously bring value to the end users without compromising quality. Mixing the two serves the goal of fast feedback, high quality, and cost-effective continuous delivery of mobile apps.

RELATED RESOURCES

- Mobile & Web Test Coverage Index
- The 2020 State of Test Automation
- The Buyer's Guide to Web & Mobile Test Automation Tools
- Testing Apps on a Simulator vs. Emulator
- Mobile Testing With Real Devices vs. Emulators vs. Simulators
- Emulation vs. Simulation

```java
public class IOS_simulator_app_demo {

    public static ReportiumClient reportiumClient;
    protected static IOSDriver<IOSElement> driver;

    protected static final String PERFECTO_APP_LOCATION = "PUBLIC:Simulators/InvoiceApp.app.zip";
    protected static final String BUNDLE_ID = "io.perfecto.expense.tracker";

    private static By userName = MobileBy.name("login_email");
    private static By password = MobileBy.name("login_password");
    private static By login = MobileBy.name("Login");
    private static By addExpenseButton = MobileBy.name("list_add_btn");
    private static By editHead = MobileBy.name("edit_head");
    private static By editHeadDropDown = MobileBy.className("XCUIElementTypePickerWheel");
    private static By amount = MobileBy.name("edit_amount");
    private static By editCurrency = MobileBy.name("edit_currency");
    private static By editCurrencyDropDown = MobileBy.className("XCUIElementTypePickerWheel");
    private static By editCategory = MobileBy.name("edit_category");
    private static By editCategoryDropDown = MobileBy.className("XCUIElementTypePickerWheel");
    private static By date = MobileBy.name("edit_date");
    private static By save = MobileBy.name("add_save_btn");
    private static By hamburgerMenu = MobileBy.name("list_left_menu_btn");
    private static By logout = MobileBy.name("list_logout_menu");


    @BeforeClass
    public static void beforeClass() throws IOException {

        String host = "demo.perfectomobile.com";
        String token = "";


        DesiredCapabilities capabilities = new DesiredCapabilities();

        capabilities.setCapability("securityToken", token);
        capabilities.setCapability("deviceName", "iPhone 11 Pro");
        capabilities.setCapability("automationName", "Appium");
        capabilities.setCapability("app", PERFECTO_APP_LOCATION);
        capabilities.setCapability("scriptName", "Perfecto Expense Tracker - Demo");

        capabilities.setCapability("useVirtualDevice", true);
//          capabilities.setCapability("appiumVersion","1.15.1");
```

About Perfecto

Perfecto by Perforce enables exceptional digital experiences and helps you strengthen every interaction with a quality-first approach for web and mobile apps through a cloud-based test platform. The cloud is comprised of real devices, emulators, and simulators, along with real end-user conditions, giving you the truest test environment available.

Our customers, including 50 percent of the Fortune 500 companies across banking, insurance, retail, telecommunications, and media rely on Perfecto to deliver optimal mobile app functionality and end-user experiences, ensuring their brand's reputation, establishing loyal customers, and continually attracting new users. For more information about Perfecto, visit **perfecto.io**.

TRY PERFECTO

Perfecto