Kobiton

A Guide to Performance Testing in Mobile

Performance Testing in Mobile

Introduction: Making a case for Mobile Performance Testing is paramount

The days of "Mobile-first" are upon us. When looking at industry research, half of all web traffic comes from a mobile device such as a smartphone or tablet. The revenue implications of this are massive. Taking e-commerce as an example, some industry research projects that, by the end of 2021, 73% of e-commerce transactions will take place on a mobile device (Loesche). This statistic lines up with what we have seen in 2020 with certain online holiday shopping and revenue split trends. Even with some predicting that mobile shopping would decrease as a result of people staying and working from home (now close to their personal desktop computers and not reliant on a smartphone to connect), Adobe found that, between November 1st and December 31st, almost 4% of all e-commerce revenue was driven by mobile, and over half of this spend came from online shoppers on Christmas day using a smartphone ("2020 Holiday Shopping Trends") (Steiner).

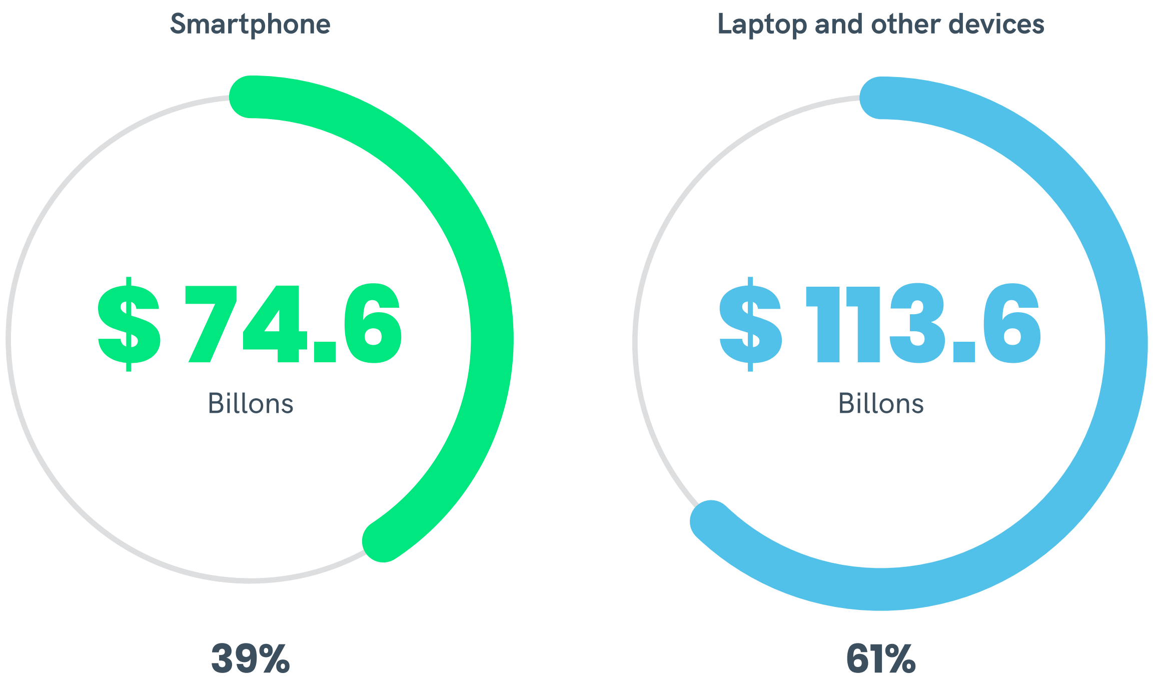
HOW MUCH CONSUMERS ARE SPENDING

Revenue dollars: 2019 vs. 2020
November 1 -December 31, 2020

2019 Historic	2020 Actual	2021 Predicted
$ 142.4	**$ 188.2**	**$ 184**
Billons	Billons	Billons

SHARE OF REVENUE BY DEVICE

November 1 -December 31, 2020



Where Performance becomes important

So, we know that mobile is becoming a major channel and touchpoint within the customer journey/ experience (as evidenced by our latest holiday shopping trends), but does Performance of your Mobile channel matter?

We pulled research from multiple sources to find this out, and the results are astonishing. To begin with Adobe's research, we found that a lot of potential revenue was left on the table when it comes to mobile, as conversion rates were more than twice as low for Mobile as they were for Desktop web for all major tracked US shopping holidays.

The next step was to dive into the reasons why conversion rates are so much lower than web. When we dove into this research, we found the following:

85%	70%	40%
The mobile shopping cart abandonment rate is an **astonishing 85%.**	70% of smartphone app or site abandonment is due to pages **taking too much time to load.**	40% of mobile shoppers will wait **no more than 3 seconds** before abandoning a retail or travel site.

So, to wrap up this section, the industry research shows that:

- Mobile is a major point of contact with customers and should be prioritized as a revenue-driving channel
- There is still a lot of revenue left on the table when looking at discrepancies between conversion rates on Mobile and Desktop web
- Performance of your mobile product is a huge influencer on conversion rates, especially within the realm of eCommerce

What is Performance Testing?

The sum and its parts

Performance Testing is often one of the most misunderstood kinds of Testing, especially when it comes to Performance Testing in a mobile context. When we speak with members of the market and ask "What is Performance Testing to you?" we often hear the following responses:

- "Load Testing"
- "Stress Testing"
- "Measuring the speed of my application or website"

The truth of the matter is that none of the above answers are wrong. However, neither of them, individually, can encapsulate "Mobile Performance Testing," as Performance Testing truly requires a large combination of many kinds of testing and measurement tactics. At Kobiton, we define Mobile Performance Testing as the following:

"Mobile Performance Testing is a multi-faceted, and incredibly crucial, component of mobile testing. When testing performance, testers, developers, and performance engineers need to make sure that the device performs well under real-world conditions, interacts with the network seamlessly, and can handle load, stress, and volume at the Server/API level. Only then is mobile performance truly tested."

So, what are the different kinds of testing required for this? To put the answer in industry terms, here is a list of the different types of Performance Testing that are crucial within any organization:

- **Load Testing**
- **Stress Testing**
- **Endurance Testing**
- **Volume Testing**
- **Spike Testing**
- **Chaos Testing**
- **Front end UI/UX testing**

Let's dive a bit deeper into each of these kinds of Performance Testing, looking at what they are and why they are important.

Load Testing

What is it?

The most simultaneous with the term "Performance Testing," Load Testing refers to evaluating your app under test (AUT) given a certain amount of load that is virtualized at the Server/ API level. So, you essentially subject the server and AUT to differing levels of load on the API Server in order to reproduce real-world scenarios associated with user traffic. Your goal here is to measure and analyze how your application responds to varying levels of load.

Why this matters

Load Testing is incredibly important and a key component of your Performance Testing strategy. Teams that skip this crucial step may feel confident releasing their application to customers after that application passes Functional Tests, but then find that, as their app gains more and more popularity (and therefore more and more load at the Server/API level), those actions that once passed Functional tests either no longer work or work so slowly that users abandon that application and/or leave horrible reviews on the app store around performance and functionality.

Stress Testing

What is it?

Like Load Testing, Stress Testing also involves virtualizing load at the Server/API level and examining the function of your application under this load. However, the big difference here is the amount of load that your subject your AUT to. When Load Testing, you typically are subjecting your application to “normal” or “expected” amounts of load. When Stress Testing, you are essentially testing the absolute limits of your product by subjecting it to unrealistic or unexpected amounts of load. The goal here is to break the system and understand how the system behaves when it breaks.

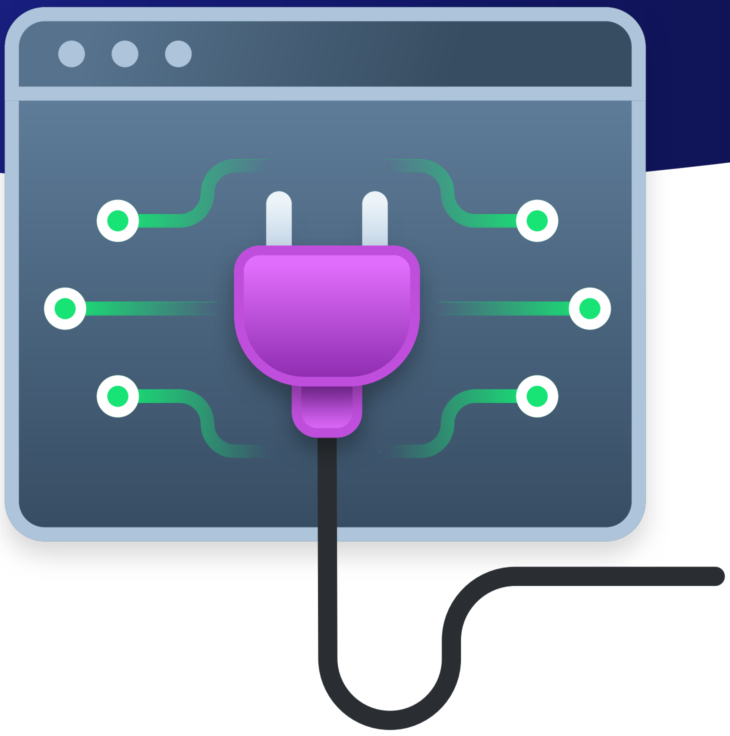
When you subject your application to these levels of stress, you might:

- Examine the order of behaviors and actions (i.e. does the call to log-in completes before the log-in screen actually renders when load is unexpectedly high)
- Ensure that the server is still reachable
- Check the app server logs for things such as exceptions that the user cannot see

Why this matters

While Load Testing helps you confirm that your application can and will function under normal amounts of load, Stress Testing is going to help you best understand the absolute limits of your application, the risks associated with these limits, how your application “breaks” under too much load, and, if you have the right tooling available, how to best make your application less susceptible to load and/or break in a way that isn’t completely disruptive to the end-user. This is crucial, as most developers, when testing their applications, are testing the “happy path” in their normal workflows, and this “happy path” does not cover certain real-world scenarios where load might be unexpectedly high.

Endurance Testing



What is it?

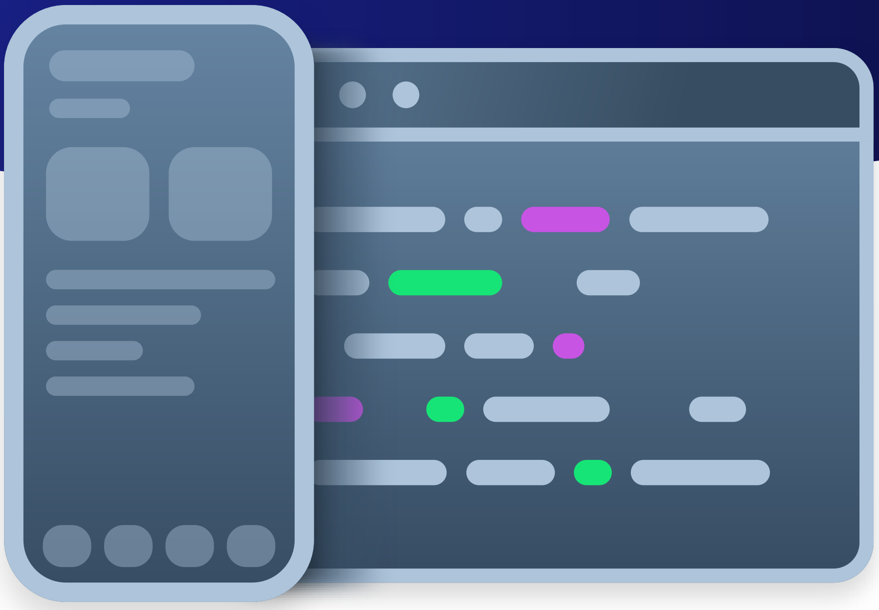
When you subject your AUT to Endurance Testing, you are essentially trying to see how the system responds to a certain amount of load for a given long amount of time. You are essentially allowing the system to "soak" in a given amount of lead for a long time, hence why some teams refer to Endurance Testing as "Soak Testing."

Why this matters

While Load and Stress Testing are able to show you how your application might function (or not function in the case of a crash) given certain amounts of load, Endurance testing gives you the ability to understand and optimize your app to deal with certain amounts of load over long periods of time. For example, the product you built might function as planned for an hour, but once a consistent amount of users have been using it for 4, 6, or 12 hours, your application might behave improperly, causing it to either be unusable or provide a poor mobile experience. At Kobiton, we have a customer that runs one of their automated tests over a period of twelve hours, takes a 15-minute break, and then runs that test for another 12 hours while subjecting their application to a given amount of load, as their users are doctors and nurses in hospitals who need to keep this app open constantly so as to garner and track life-saving applications.

This kind of testing is crucial for many kinds of applications or websites (think video streaming, crucial back-end business systems that must keep running, etc.), and will only become more important for mobile applications as more and more shopping, media consumption, work (think about the hospital example above), etc. is done via smartphones, tablets, and other mobile devices.

Volume Testing



What is it?

Unlike Load Testing and Stress Testing, Volume Testing has nothing to do with load at the Server/API level. Instead, Volume Testing has you increase the amount and size of data at the database level. So, you increase the amount of data, and you check to see whether this impacts the expected function and performance of your AUT.

Why this matters

When analyzing response times, functional, display, etc., it is important to stress more than just the API/ Server with certain amounts of load. While load can impact performance in meaningful ways, the amount of data at the database-level is also a major culprit in many application issues. For example, for each test step, as you go from a gig in a table to a 100 gig, you might find issues such as missing indexes, data being stored incorrectly in the db, etc. This also helps inform the rest of your organization around how to best maintain and staff datacenter/ database resources so as to avoid costly emergency fixes associated with fixing database issues after your solution is already in production.

Spike Testing

What is it?

The goal of spike testing is to measure how your App or system under test responds to major increases and decreases in load in a short amount of time. Performance Engineers and/or Performance Testers will virtualize load at a normal level, and then rapidly increase that amount, and then drastically reduce load. Most of the time, Performance Engineers and/or Performance Testers are looking to see how an app recovers as a result of the rapid fluctuation in load.

Why this matters

Spike Testing matters because it replicates yet another very realistic real-world scenario. Take, for example, an eCommerce application or website (let's say the Amazon app) that might be launching a once-a-year special promotion or discount (let's say Amazon Prime Day). Amazon Performance Testers and Engineers will absolutely need to know how their application functions and performs under what will be very sporadic but significant, increases and decreases on load as new Prime Day deals hit every few hours.

Chaos Testing

What is it?

When Chaos Testing, Performance Testers and/or Engineers will artificially apply abnormal environmental conditions to the device and/or AUT so as to see how the device and/or application responds. The important factor here is that these “abnormal environmental conditions” refer to much more than just load on the server or amount of data at the database-level. When it comes to Mobile, these environmental conditions might refer to reproducing extremely poor or chaotic network settings, network switching, draining the battery down, performing certain user actions in a strange order, etc. Usually, this will be done until the application “breaks.” This is all done to test not just the function of your AUT, but also the resiliency of your application.

Why this matters

Chaos Testing is crucial in the world of mobile applications due to the very nature of mobile applications being mobile. What I mean by this is that your application is going to go wherever your phone goes, will be away from power and reliable internet connection for as long as you are, etc. With these variables in place, the AUT, while unknown to the user, has to constantly adapt to varying conditions. As conditions change rapidly, will your application be fault tolerant? If not, then your application stands a good chance of being abandoned early. Chaos Testing allows you to not only prevent this from happening, but also understand the variables that cause issues in function so as to optimize your mobile application going forward.

Front-end and UI/UX Testing

Wait, what? How is UI Testing relevant here?

There aren't many sources that include UI/UX Testing as a part of what they are calling "Performance Testing." However, we believe that this is a mistake. Teams can simulate load, stress, crazy conditions, etc., and test whether or not the application fails, crashes, etc. However, even if your application can run in these conditions, teams must also know how the UI and UX are impacted. This can only be done by testing your UI on real devices whilst reproducing conditions like heavy load, stress, or data injection.

Why this matters

Without the ability to actually run UI Tests during the virtualization of load & stress or the reproduction of real-world environmental conditions, your teams are never going to understand how these different factors influence the overall UI and UX of your application that your end-user actually experiences.

When is Performance Testing helpful?

There really isn’t a perfect answer here, as Performance Testing is helpful almost all of the time. While much of Performance Testing is thought of as being done “in production” (think Shift Right), there are also plenty of reasons to Shift your Performance Testing Left into Dev and Test (most of what we have been talking about in this guide refers to testing and optimizing performance at these sections of the SDLC):

- Fixing any Performance issues before releasing your product to your customers minimizes the risk of poor user experiences and app abandonment (remember: one out of four mobile applications are abandoned after their first use)
- Defects caught in Production cost much more (some estimates state 10x more) than issues found during test/development
- With the right tooling and processes in place, Shifting your Performance Testing Left is easily done

Manual vs Automated Performance Testing

What is Manual Performance Testing?

Manual Performance Testing is self-explanatory. It's doing what you'd try and accomplish with an automated test, but it's done manually. Typically, this would involve a Performance Tester and/or Performance engineer manually reproducing real-world scenarios (virtualizing load, stress, data, etc.) and then manually looking at how this might affect the function, behavior, response time, etc. of the front-end UI/UX as well as seeing how the backend might be affected.

Let's take a look at the pros and cons of this approach

Pros/Cons of Manual Performance Testing

PROS	CONS
• Exploratory testing is often the best way to gain insight into how performance issues might impact the e2e user experience	• It is unfeasible (costly, time-consuming, etc.) to check for performance impacts across the hundreds or thousands of device permutations • The human eye cannot detect smaller visual discrepancies in the UI associated with performance issues on the backend, and this can lead to visual issues (that might even render functionally) leaking into production • No way to make this kind of testing continuous alongside Continuous Integration and Continuous Deployment/ Delivery initiatives

There really aren't very many "Pros" of this approach other than the fact that Manual Testing does often open up the door for Exploratory Testing. Because there are so many different device types and permutations, manual Performance Testing has a nearly impossible task of capturing performance impacts across the vast amount of device permutations, either leading to slow and costly release cycles and/or poor overall coverage. Both of these lead to poor user experiences and/or loss of market share to faster-moving competitors.

What is Automated Performance Testing?

Automated Performance Testing typically consists of doing the same sort of virtualization of load and stress, as well as synthesis of data, but combining that with an automated test execution/run in order to see where issues might arise in the UI/UX as well as where backend issues might occur.

Let's take a look at the pros and cons of this approach.

Pros/Cons of Automated Performance Testing

PROS	CONS
• With the right tooling, you can still combine the depth of Exploratory testing with the speed and efficiency of automated tests • Same valuable time and effort as opposed to manual testing • Cover for more device types, OS's, etc. • Find more performance regressions before products are pushed to production	• Automation can be incredibly difficult to implement • Automation can be incredibly difficult to maintain • Most performance testing tooling still lacks the ability to deliver reliable results when measuring UI/UX impacts during an automated test run.

The pros of automating your Performance Testing are quite clear, with the only downside being the difficulties associated with automation. However, these difficulties are totally dependent on the tooling and process implemented. With the right solutions in place, these difficulties can be mitigated and/or eliminated entirely. If using the right solutions, the automated Performance journey is clearly the winner overdoing the majority of your testing manually.

Performance Testing Tools

When considering Performance Testing tools, you have to keep in mind the comprehensive nature of your testing. For Performance Testing, you will need to consider tooling that both allows for you to generate load and stress and a solution that allows you to test Function, behavior, and response time on the front-end UI/UX of the application. Here are a few solutions that are worth your consideration:

Load and Stress Generation

NeoLoad by Neotys

One of the most popular load and stress testing tools, NeoLoad allows you to:

- Virtualize load and stress at the server/API level
- Create codeless Performance Tests
- Convert a Functional Test into a Performance Test
- Analyze performance of your application with in-depth reports and analytics

Perhaps the most exciting aspect of NeoLoad is it's ability to seamlessly integrate and communicate with other applications, such as the Kobiton integration mentioned above, integrations with monitoring solutions like Dynatrace and New Relic, and its ability to repurpose functional tests from solutions like Kobiton, Tricentis Tosca, Worksoft, Appium, etc.

JMeter

JMeter was first developed by Stefano Mazzocchi of the Apache Software Foundation. He sat down and made it primarily to test the performance of Apache JServ, which the Apache Tomcat Project later replaced. JMeter is the most popular performance testing tool provided by Apache Software Foundation.

There a few reasons why teams choose JMeter as their load and stress testing tool of choice:

1. Apache JMeter is an open-source load and performance tool. It's a pure Java application designed to load functional test behavior and measure performance. It is platform-independent, and to make this tool stronger Apache JMeter development community works to enhance the JMeter features.
2. Zero acquisition cost: Simply download the binaries from their website.
3. Support for various Protocols: JMeter is not a browser- it works at the protocol level, supports multiple protocols such as HTTP, HTTPS, SOAP / XML-RPC, POP3, IMAP, SMTP, JMS, FTP.
4. Ease of Learning: Anyone with software testing experience or knowledge at any level will find JMeter easy to learn and use.

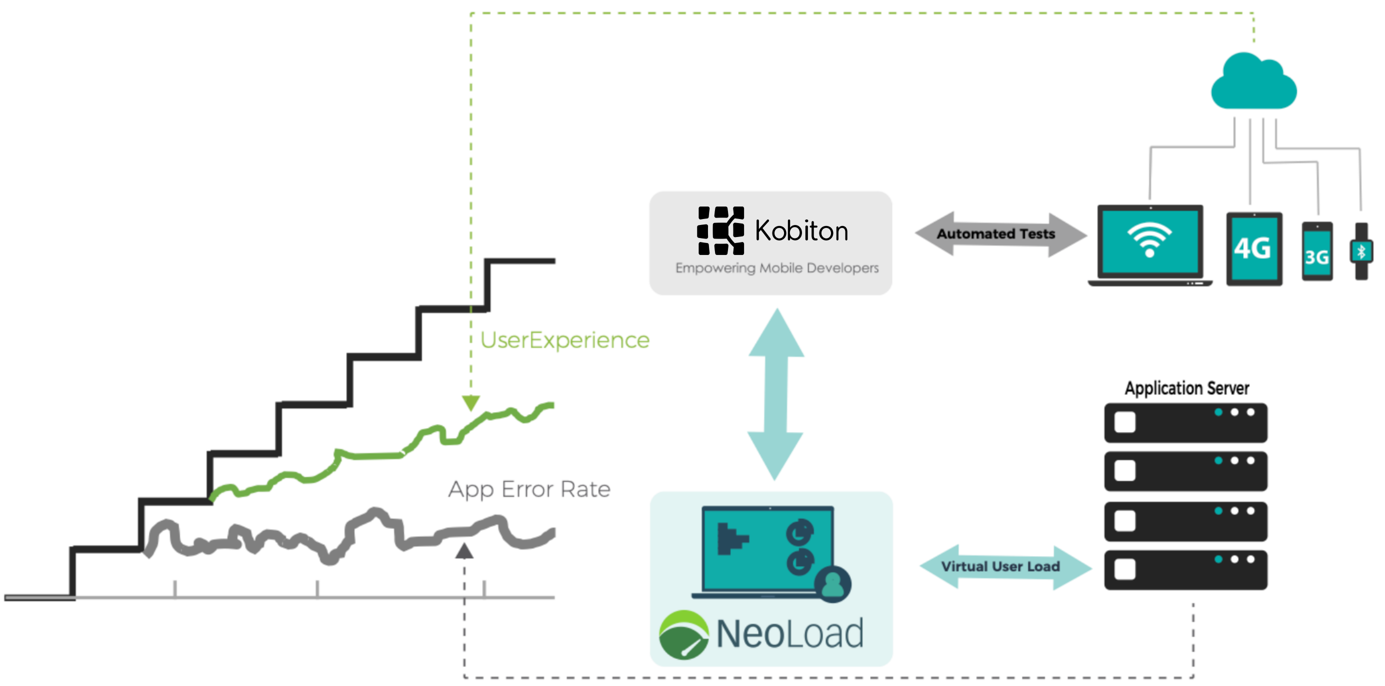
Front end UI/UX

Kobiton Mobile Experience Platform

Kobiton offers uniquely flexible real device access options that include a public cloud, private cloud, and the ability to "cloud-ify" real devices. Kobiton also offers a strict on-premises deployment option as well. Kobiton even offers the ability to test against real devices and have video streamed back to the tester at 30FPS for in-depth analysis and information around the performance and function of dynamic content and videos.

With Kobiton, you get the ability to integrate your real device testing with Load and Stress testing solutions, but run your actual tests on the Kobiton side in order to see the impacts on UI/UX when it comes to function (Kobiton Functional Testing), visual discrepancies (Kobiton Visual Testing), Network Payload Request and Response time, and you will have any Performance regressions that hit a certain threshold identified automatically and made available for remediation within the Kobiton UI. This can be done manually, with automation via script-based solutions like Appium, and using our AI-driven scriptless engine.

With Kobiton, you have a platform solution that allows for insight into app and device specific performance metrics that easily integrates with solutions like NeoLoad or JMeter. In fact, Neotys recently found that the integration with Kobiton and NeoLoad led to 10x faster Performance Test creation and maintenance (Neotys). See the diagram below for how this relationship works:



HeadSpin (including Nimbledroid)

An early innovator in the space, Headspin offers globally located real devices for remote cloud and on-premises access. When running tests on the Headspin tool, you are able to gather insight into an array of performance metrics and indicators associated with audio, video, response time, etc. Headspin's offerings are driven by AI and ML that is able to track and recommend strategies around optimization for digital experiences

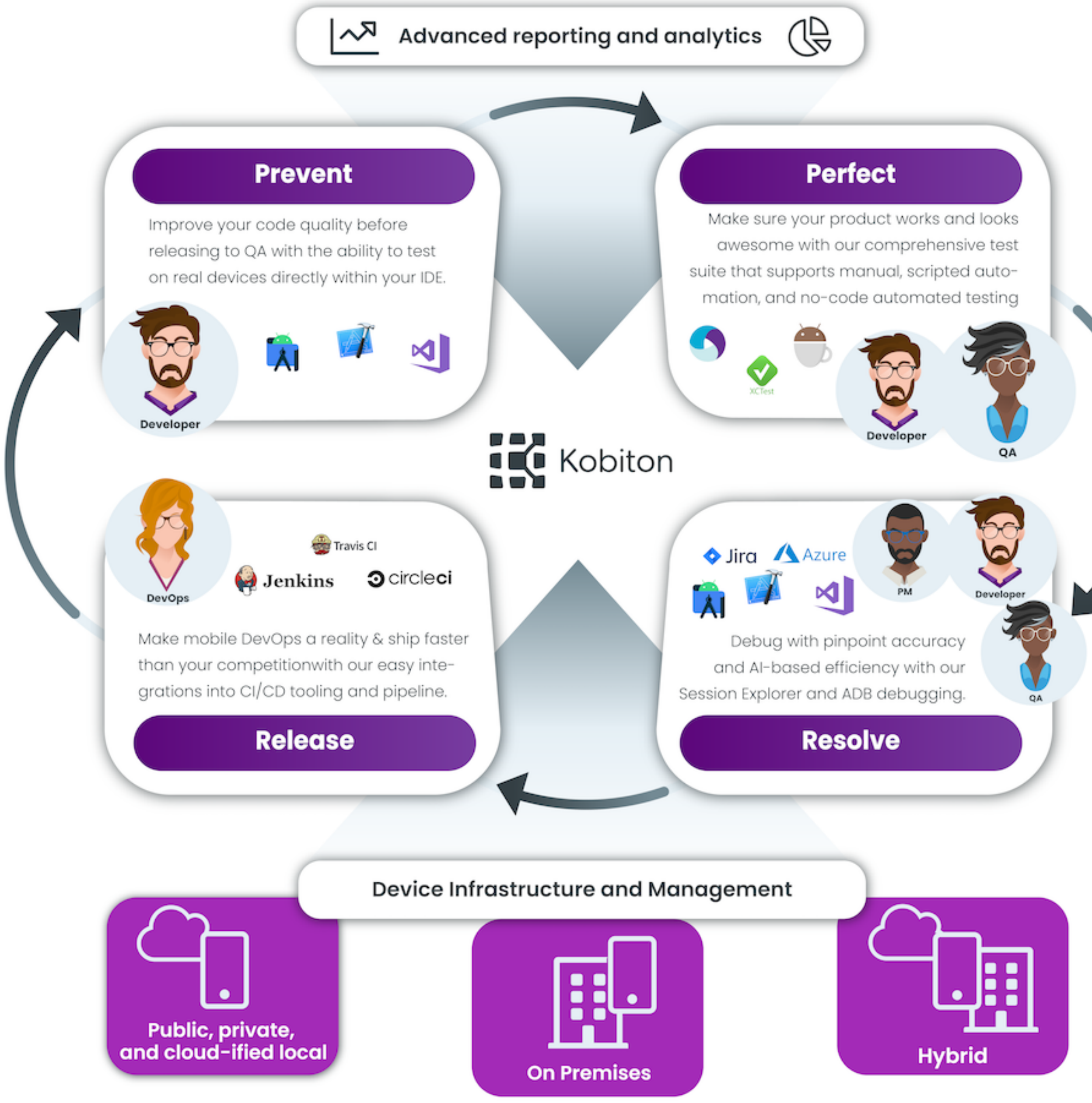
How do these vendors compare?

	Kobiton	HeadSpin	NeoLoad	JMeter
Real device access	✓	✓	✗	✗
Functional Testing	✓	✓	✗	✗
Automated Visual Testing	✓	✗	✗	✗
Network Payload Capture "in the UI"	✓	✓	✗	✗
Load and Stress Virtualization	✗	✗	✓	✓
Scriptless test automation	✓	✗	✗	✗
Price	$$	$$$$	$$	Free (open-source)

About Kobiton

Kobiton is the mobile and IoT experience platform trusted by leading organizations globally. Our best-in-class software platform helps drive improved revenue on the mobile and IoT channels by lowering app abandonment, improving quality and reducing time-to-market.

Used by over 60,000 developers and testers worldwide, Kobiton is transforming the way companies deliver mobile apps and IoT devices through innovative applications of Artificial Intelligence, Real-Device Testing and the industry's first and only mobile scriptless automation solution.


Advanced reporting and analytics
Prevent
Improve your code quality before releasing to QA with the ability to test on real devices directly within your IDE.
Developer
Perfect
Make sure your product works and looks awesome with our comprehensive test suite that supports manual, scripted automation, and no-code automated testing
XCTest
Developer
QA
Kobiton
DevOps
Travis CI
Jenkins
circleci
Make mobile DevOps a reality & ship faster than your competitionwith our easy integrations into CI/CD tooling and pipeline.
Release
Jira
Azure
PM
Developer
QA
Debug with pinpoint accuracy and AI-based efficiency with our Session Explorer and ADB debugging.
Resolve
Device Infrastructure and Management
Public, private, and cloud-ified local
On Premises
Hybrid

Drive quality across the entire SDLC. Prevent bugs before pushing test to code with advanced ADB debugging and access to real devices within your IDE. Perfect your application site with the Kobiton Intelligent quality Suite, and build, execute, and report against scriptless and/or scripted automated Functional, Visual, and Performance Tests. When issues are found in test, you can easily resolve them with AI-assisted remediation and ADB debugging. And, all of this is seamlessly integrated with your CI/CD pipeline/tooling so that you can kick-off tests with confidence and ultimately release faster than your competition.

Run your tests on the industry’s most flexible and high-performance real device cloud or on premises solution that offers 30FPS video streaming, in-depth session exploration, and analytics solutions that offer visibility and traceability throughout your entire testing process.

With Kobiton, build, test, deploy, and release better mobile apps, websites, and IoT devices. There's a reason the world's mobile elite choose Kobiton to deliver perfect mobile and IoT experiences for their users.

Kobiton