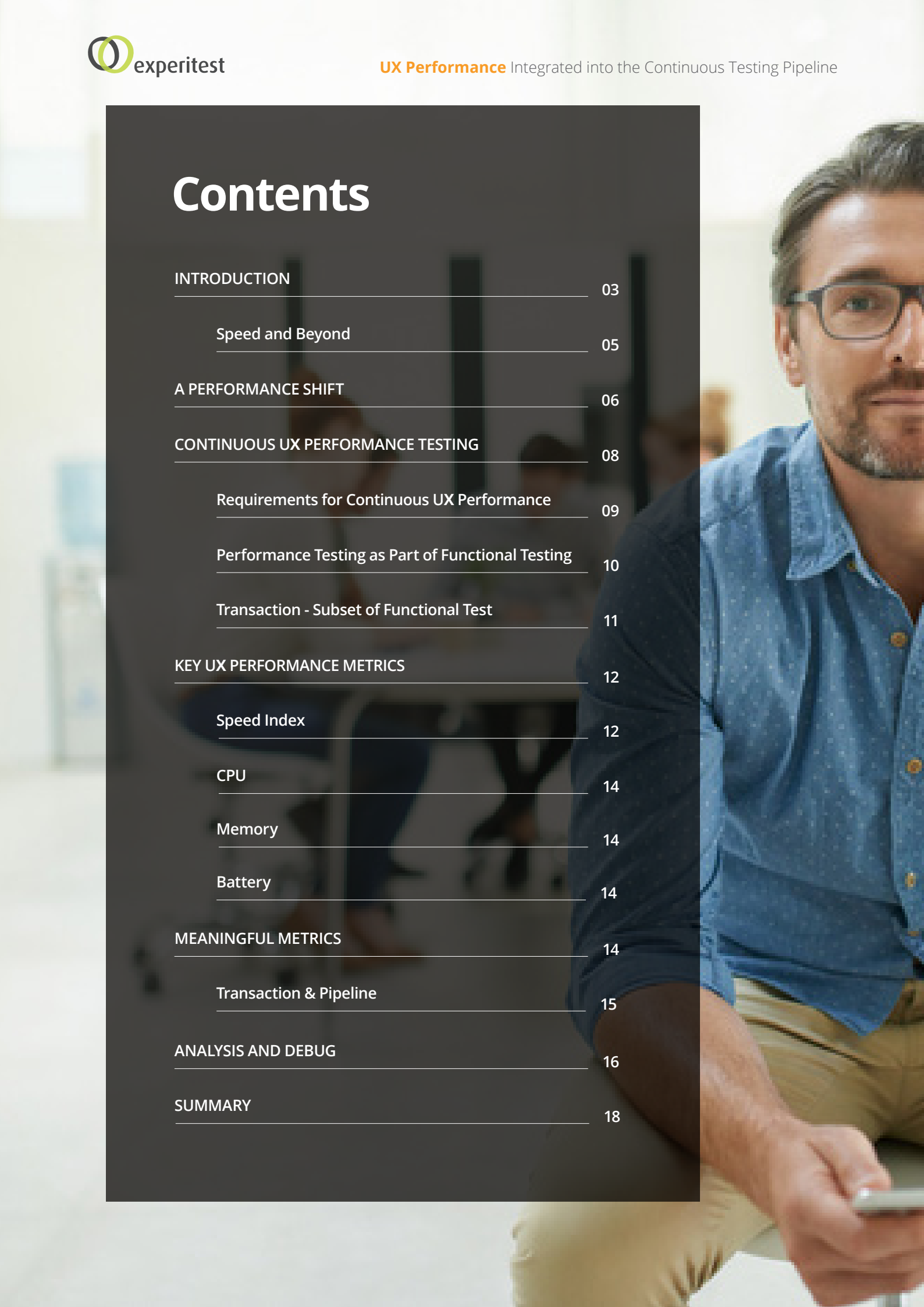
experitest

UX PERFORMANCE

UX Performance Integrated into the Continuous Testing Pipeline

Contents



Introduction

Digital experiences are front and center in a modern organization's business strategy. Application and website performance has a huge impact on digital customer experience, which directly affects business results. Poor user experience performance drives customers away and negatively affects the bottom line, while good performance can help attract and retain customers. And the difference between the two can make a significant impact on a company's bottom line.

Following are some examples ›

20%

A one-second delay in mobile load times can impact conversion rates by up to 20% (Feb, 2019)

5 STARS

App Store: While functional testing can earn 3 stars, UX performance testing is required to get to those 5 stars

31%

Telefónica

Telefónica improved load times for its mobile site by 70% — from 6 seconds to only 2 seconds on 3G connections. These improvements helped the company increase the click-through rate by 31%

15%

Pinterest increased search engine traffic and sign-ups by 15% when they reduced perceived wait times by 40%.

10%

BBC

The BBC found they lost an additional 10% of users for every additional second their site took to load

SITE SPEED

In 2018, Google implemented **site speed** as a ranking signal in its mobile search algorithm.

Speed and Beyond

While speed and load time is critical to user experience, battery life, data traffic, memory and CPU consumption are also important factors. Applications that are not optimized for consumption of these resources are often called "resources-hogging apps" and even " RAM, storage or battery vampires". An application that hogs resources is ten times more likely to be removed by customers.

Some examples:

- An application feature or process that monopolizes the CPU means that other processes are 'starving' for a chance to execute.
- Battery life is often considered the single most important aspect of the mobile user experience. A device without power offers no functionality at all. For this reason, it is critically important that apps be as respectful of battery life as possible.
- Apps that hog resources or have recently introduced (intentionally or not) a feature that consumes too much battery immediately risk being on "not the most" wanted lists such as "Top Battery Draining Apps to Avoid". Or being featured in an article such "Deleting This 1 App Can Literally Double Your Phone's Battery Life."

A Performance Shift

It is important to distinguish between Load and UX Performance. In the past, the main bottleneck to service performance was the backend systems, limited by physical or even virtual servers. Loading backend servers with tools such as JMeter or LoadRunner was the main focus of performance testing, highlighting issues related to load (scale), server CPU usage, and similar.

Recently the focus of performance testing has shifted from backend systems to the front-end, with the focal point being UX performance. There are two technological trends that are driving this shift:

The first is the **growing adoption of elastic container technology**, powered by the likes of Docker and Kubernetes, which enable backend systems to automatically scale to any required load. The applications using these solutions must be tested to ensure quality, but scaling issues are less of a bottleneck than they were in the past.

The second trend is the **growing complexity and size of digital applications** coupled with the growing variety in user conditions. Applications are heavier, richer and require more resources (network and processing power). At the same time, they are used in networks ranging from 2G to 5G, in varying degrees of coverage, and on devices differing in their capabilities and resources.

Defining UX Performance Testing

This shift in performance testing has resulted in a focus on UX performance testing, which measures the combined impact of the network, device, OS, and browser on the performance of an application.

UX performance testing is designed to evaluate how responsive and effective an application is to the end-user under various network conditions and on difference devices, OS and browsers.

Continuous UX Performance Testing

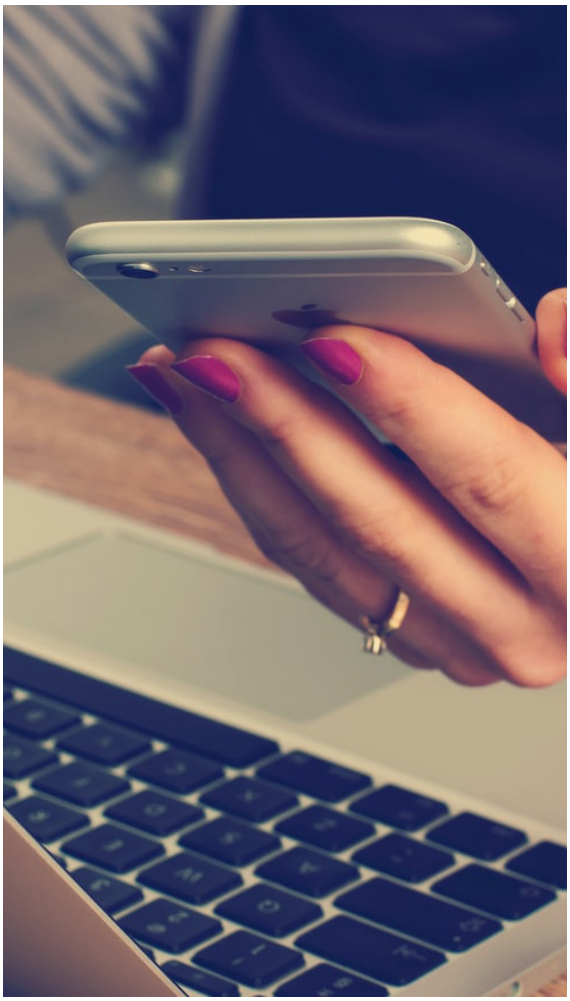
More organizations than ever are moving to a continuous testing model, where testing is performed early as part of the CI/CD pipeline, also called “shift-left performance testing”. In fact, there is a clear understanding that continuous delivery is impossible without continuous testing.

Yet performance testing is not yet integrated throughout the testing continuum and, in most cases, is still performed at a late stage, just before deploying to production. The result is that issues are identified very late in the Software Development Lifecycle (SDLC) when the cost and time of fixing them is much higher. Even worse, if organizations do not pay attention to UX performance testing, issues may remain undetected, until they are detected by users.

In addition, when carrying out functional testing, it's clear when a single functional test passes or fails. However, with performance testing it's much more important to detect small deviations or anomalies from the baseline. Building this baseline and then identifying even slight deviations can only be undertaken when it's done continuously and as part of the CI/CD pipeline. Test performance data should be stored together with other versions and test data in an analytics database.This enables comparison of the performance of the app on a specific build to the baseline created from many builds, and then analysis of the results.
To ensure a consistent user experience, organizations need to make UX performance testing part of their CI/CD pipeline and part of their continuous testing practice.

Requirements for Continuous UX Performance Testing

In continuous testing environments, tests are performed automatically, and as the name implies, continuously. The result is a large amount of data that requires analysis. Applications and tools that are capable of analyzing this data include open source tools such as Elasticsearch, Logstash, Kibana and Splunk.

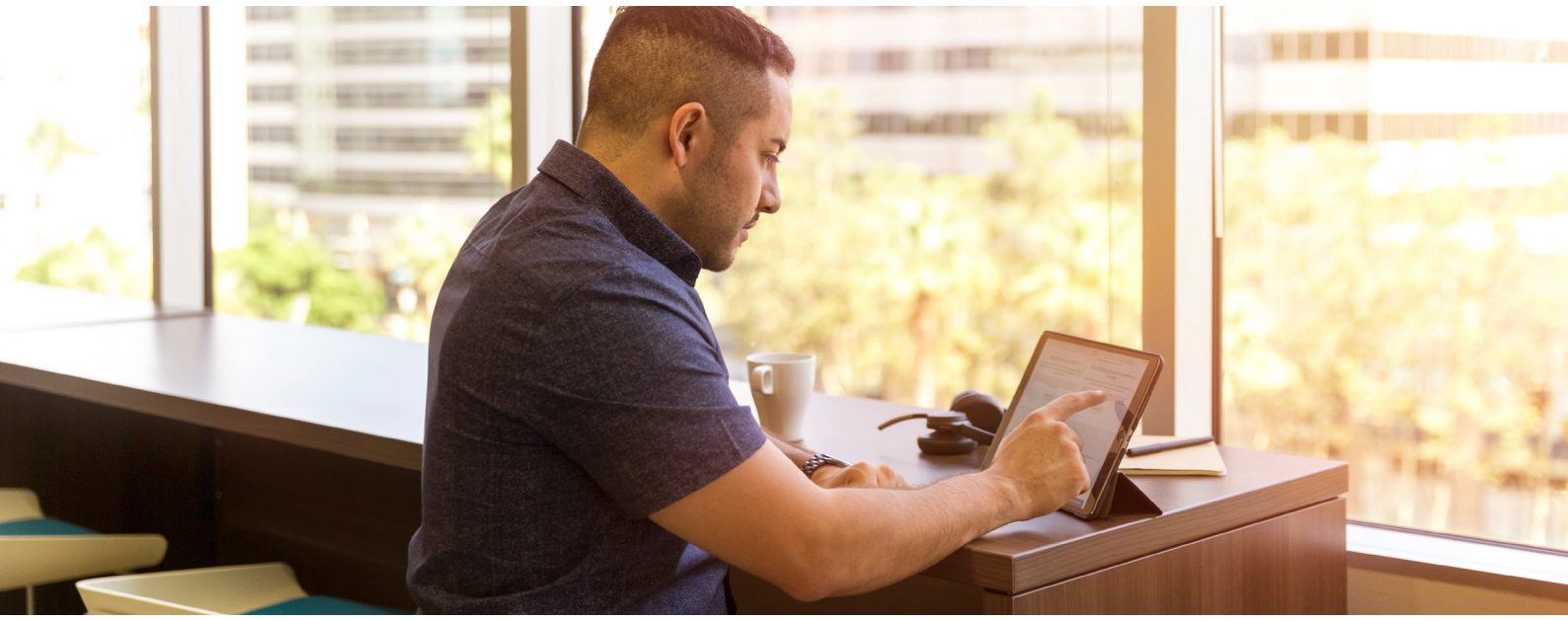


Effective UX Performance tests need to be consistent in what they measure, and the results need to be correlated with other test data in order to enable meaningful analysis. This methodology allows comparison of different performance indicators as they change across versions, builds, platforms or networks.

In the following sections, we cover some of the key factors that enable effective Continuous UX Performance testing.

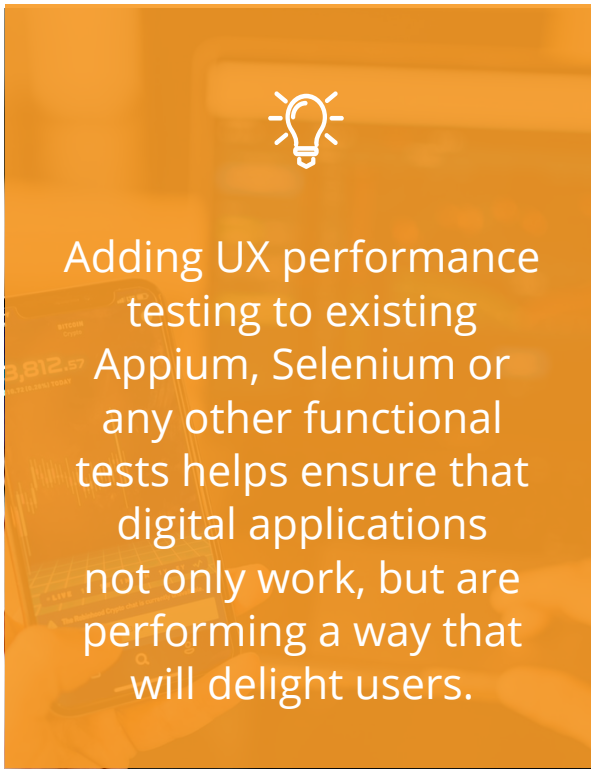
Functional Testing

Today, functional testing is the number one priority on QA teams' agenda for automation. Since it has the highest impact on application quality and user experience, significant effort and resources are going into incorporating functional testing into the CI/CD pipeline. Organizations are developing and maintaining test suites, and investing in the test labs that are required for high scale parallel execution and for device and browser coverage.



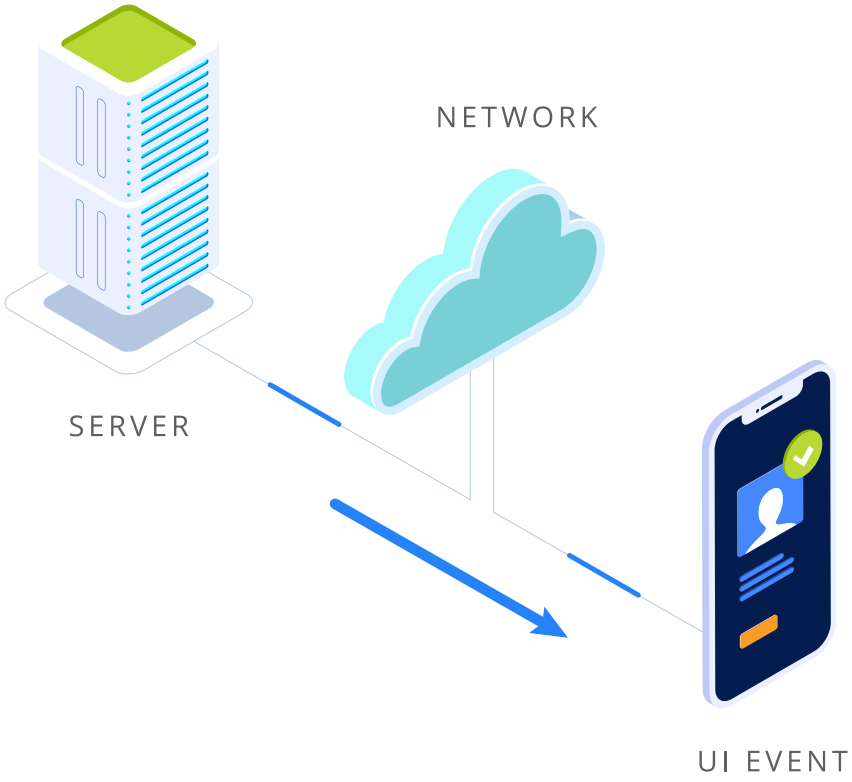
Performance Testing as Part of Functional Testing

The most effective and efficient way to implement UX performance testing is to leverage the efforts and investments in functional testing. Adding performance tests to an existing test suite saves test development and maintenance, and ensures the wide coverage required.
In this manner, users who create and run 'regular' functional tests can create and run UX performance tests on a continuous basis, without requiring the skills of a performance engineer. Combining functional tests and UX Performance tests also helps encourage team collaboration and helps ensure that performance testing is incorporated early in the SDLC.



Transaction - Subset of Functional Test

Starting to measure performance metrics requires breaking down users' interactions within an application to the level of each transaction. A transaction is a specific operation performed at the UI level, which leads to communication with the server and back. For example, clicking the Login button and waiting for the next screen to load can be considered as the 'Login' transaction. Other examples of transactions are actions like Search, generating a report, deleting an element and so on. Each of these types of transactions typically involves an interaction with the database.



Usually, in functional testing, tests contain also actions that are not part of a transaction. For example, if we have forms we need to fill in the application, filling a form is not part of the transaction; and the speed in which a user fills the form is mainly up to their typing speed and level of distraction. Many tests also load the application, or navigate to a specific area in a page.

When analyzing application performance, especially as part of Continuous Testing pipeline, we need to isolate transactions from other actions as well as from each other, in order to pinpoint specific issues and correct them.

Key UX Performance Metrics

UX performance can impact the user experience in different ways. How long a page takes to load in full, how long before the user starts engaging with a page, whether an app slows down, guzzles down battery, hogs CPU or device memory and more. An issue with one metric doesn't necessarily affects another metric. In addition, if the login transaction performance is flawless without any deterioration, it doesn't mean that the search transaction doesn't have a bug that led to a performance issue.

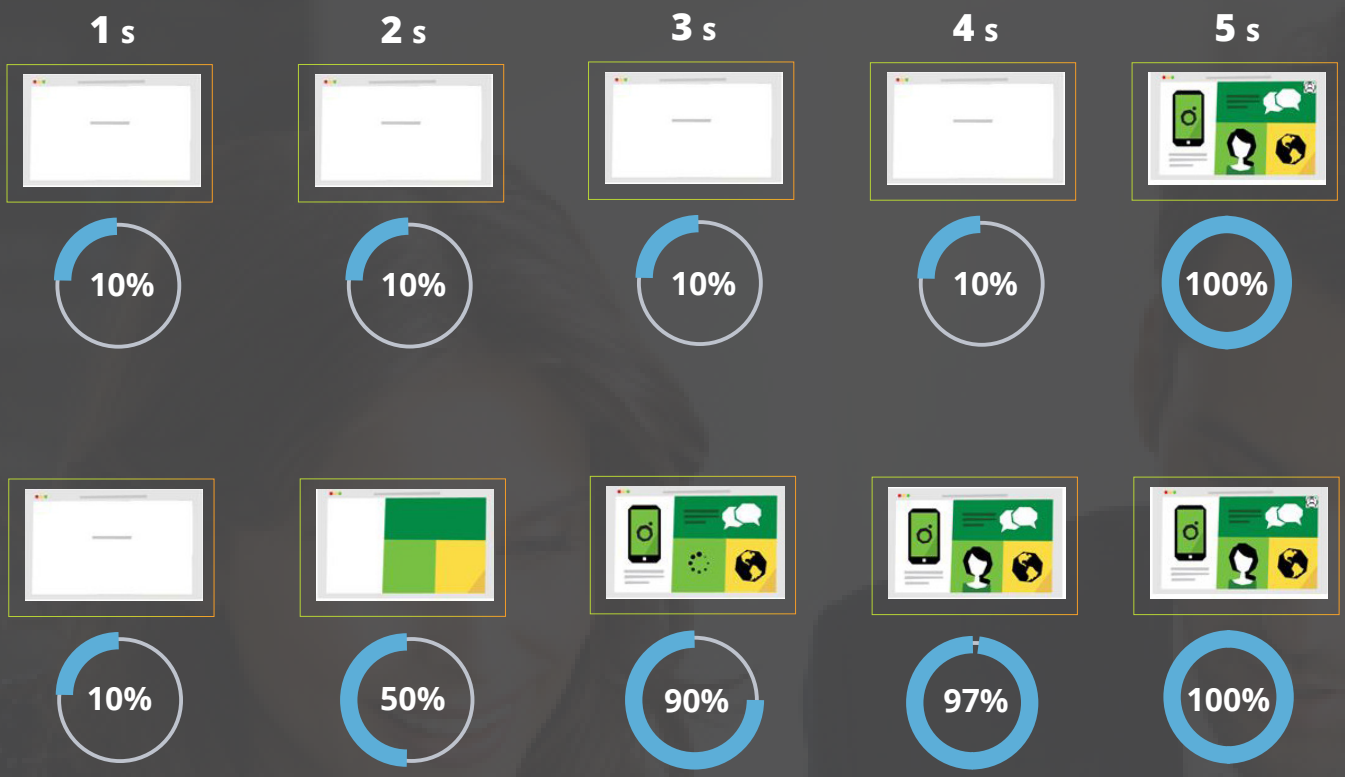
This is why it's important to **continuously** monitor all transactions and all key performance metrics and compare them to the established baseline.
Following are the key performance metrics:

Transaction Time

Transaction time is the full duration of the performed operation, starting with the click of the 'Submit' button until all the information was rendered back to the user.

Speed Index

The Speed Index is the average time at which visible parts of the page are displayed. It is expressed in milliseconds and dependent on the size of the viewport.
The following example shows two different web pages that load second by second, taking five seconds for the page to load in full:



In the first example the user sees nothing during the load time, and only in the 5th second does the material appear as the entire page is being loaded all at once.

In the second example, after 2 seconds, the user is able to see the full frames, by the 3rd second most of the content is available and the user can start to analyze the page and find the main content on the page.

The user experience in these two cases is very different and it emphasis why a meaningful alternative for 'page load time' is needed.

CPU

The CPU (Average, Maximum) of the device/application during the transaction time.

Memory

The Memory (Average, Maximum) of the device application during the transaction time.

Battery

The Battery (Average, Maximum) of the device/application during the transaction time.

Network

The uploaded and downloaded data during a transaction.

Meaningful Metrics

On their own, performance metrics are meaningless. Is a transaction duration of 5 ms good or bad? To generate value from performance metrics, one approach is to require meeting the target values for these metrics, or “thresholds”. If the target duration of the Login action defined by the business owners is 10 ms, 5 ms is great. If the defined duration is 4ms, this does not meet the mark.

A better approach would be to achieve these “target values” or “thresholds” is by building a baseline from many builds for that same transaction, with different conditions. Then the system can automatically identify anomalies and deviations from the baseline.

Transaction data needs to be stored in a central repository, so it can be analyzed to identify trends and issues, which brings understanding of the root causes of the issues. This way we can also understand whether a deviation happened only for a specific device or network conditions, in a specific build, following a certain update, and so on.

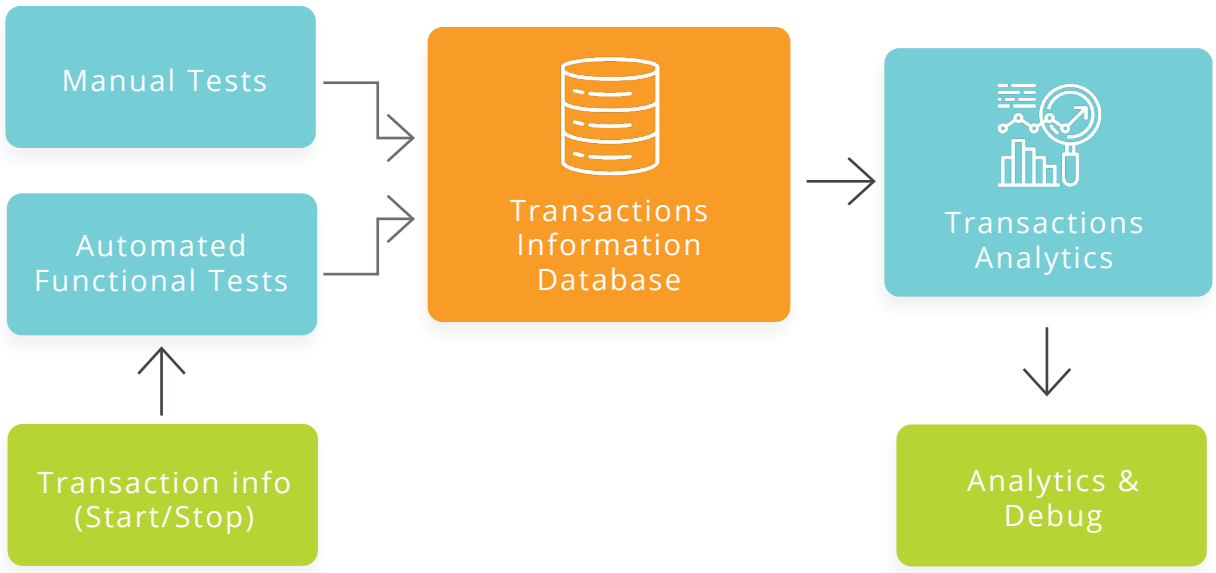
Transaction and Pipeline

To add application performance to the CI/CD pipeline:

1. **Start collecting transaction information.** The information can be collected both from manual flows as well as automation (functional) flow. In order to add it to automation flows, map the transactions into the relevant functional tests by adding 'start transaction' and 'end transaction' commands to these functional tests. Make sure to call same transaction by the same name, for example if the transaction is user login, call it "user login" (and not once "user login" and once "login").

2. Store the information in a centralized repository specific to each application. The stored information should include device information, application information and obviously the performance measurements like Transaction Time, Network Download and Upload, Speed Index, CPU, Memory and Battery usage.

3. Perform analysis that will enable determination of the baseline, and thresholds. This analysis enables views of trends and provides a view of transactions over builds and versions. For example, analysis could show that for a specific build, iOS 13 login duration time was consistently longer by 5% whereas other transactions and/or this login transaction for other versions of the operating systems were not affected in this way. In this case the build can be defined as failed under these circumstances.

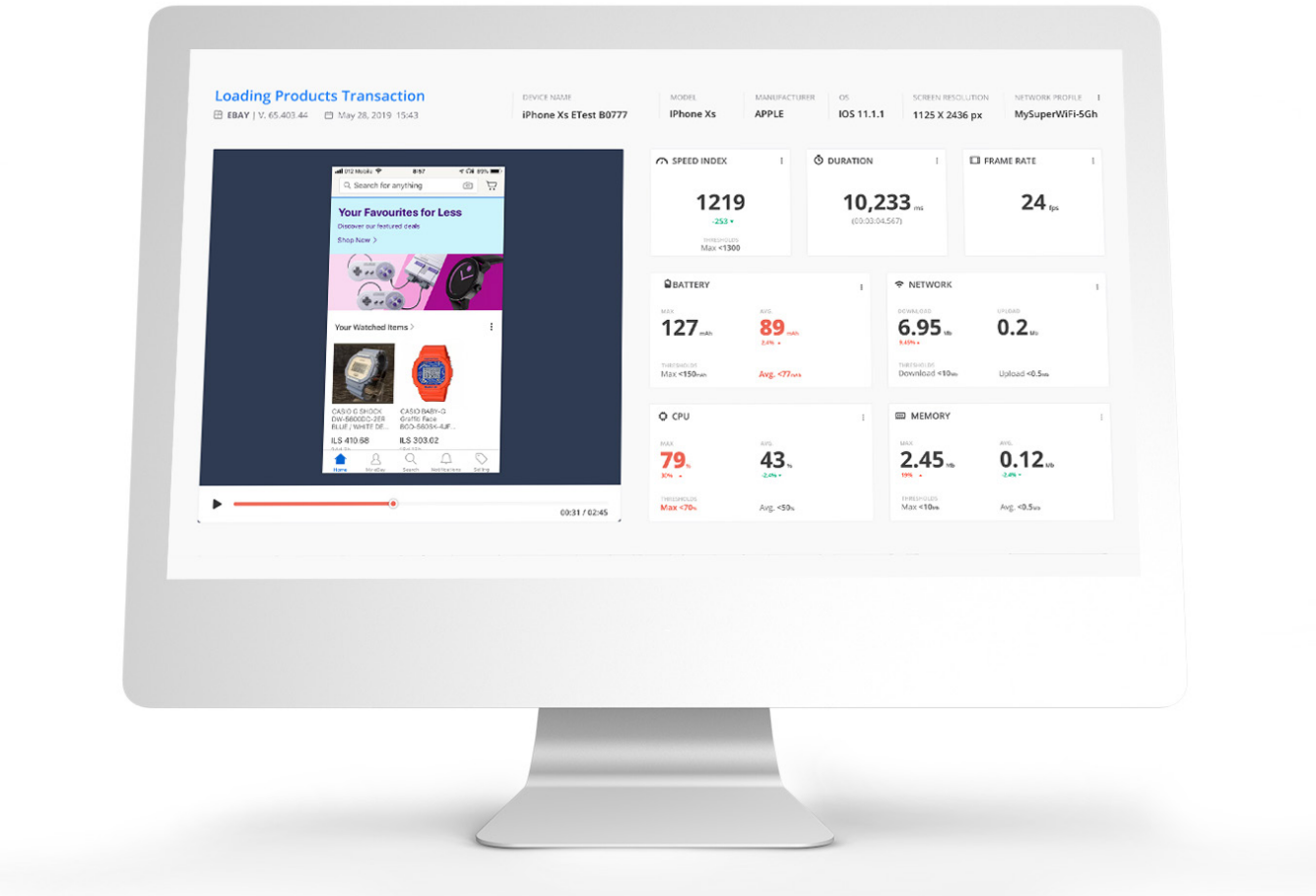


Analysis and Debug

Once the analytics of the transactions identify a trend and raise suspicion for regression, the next step is to analyze the change in the behavior and debug it. This involves trying to identify the reason for the performance issue, and to identify its root cause. There are many reasons why a build could have an issue with transaction performance, including:

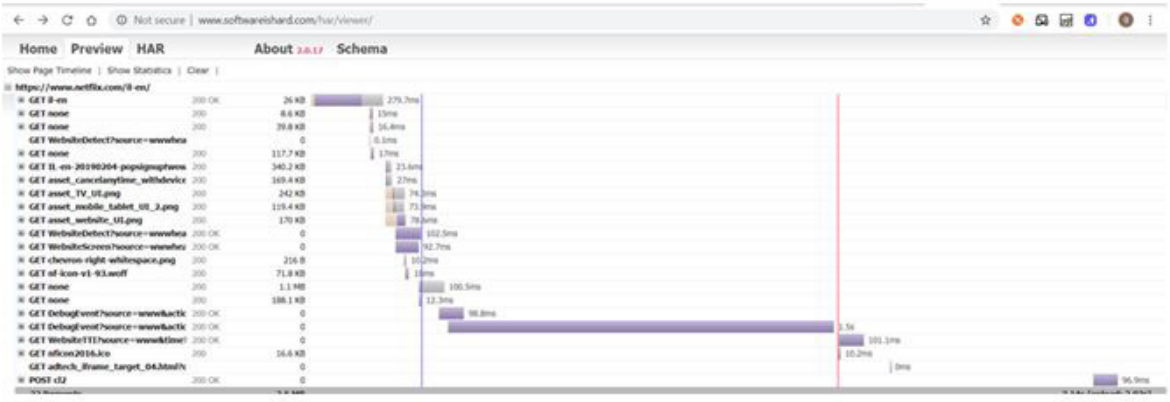
- A loop in the server that goes on and on, or a large image that takes too long to download
- A DNS issue
- An issue with an SSL handshake
- HTTP requests not being sent in parallel

Poor app performance can be detrimental for any company, especially when performance issues take too long to identify. To quickly identify the cause of a performance issue of a transaction, a new set of tools is required.



The transaction report, in conjunction with a video of the transaction, are critical in pin-pointing issues. The transaction report displays the range of parameters accumulated (Battery max, average, CPU, max, average, Speed Index, Network download, upload, Memory, etc.)

In addition, a key tool to be used is the waterfall view of a HAR file. HAR, short for HTTP Archive, is a format used for tracking information between a web browser and a website and can show all the network requests from the application to different services.



The ability to compare two views from two transactions can be very beneficial.

Also, automatic analysis of the network requests against common guidelines and best practices can be very helpful for new performance engineers.

Summary

To ensure excellent great digital customer experiences, organizations should embrace UX performance testing that incorporates not only speed and load time, but also considers battery life, network data traffic, memory, and CPU consumption.

To incorporate performance tests within test suites:

1. Identify the transactions to be monitored (these can be a dozen or even hundreds of such transactions).
2. Product owners then map these transactions.
3. Add the definition of the start and end of transactions to existing functional tests.
4. Integrate performance data into reports and analytics and take the required actions to correct the issues.

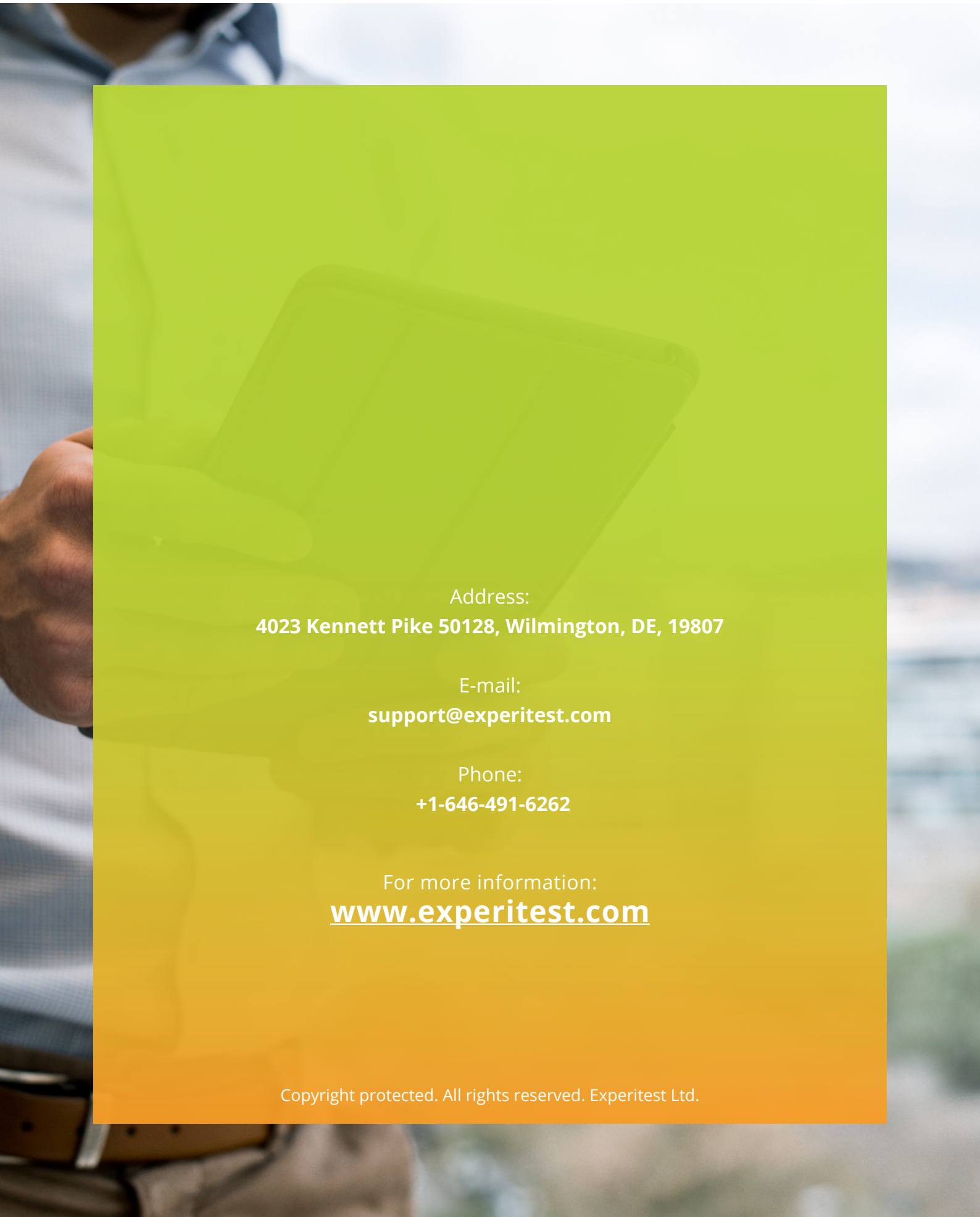
For UX performance testing to succeed it must be integrated into the CI/CD pipeline in the following manner:

Continuous
Make UX performance testing part of existing UI functional testing. Add UX performance tests to standard Appium and Selenium tests, triggered by the CI pipeline.

Consistent
Focus on transaction performance, not test performance. Add transaction definitions to the test code. Important transactions and their target performance should be defined by business owners and shared with DevOps teams.

Meaningful & Comparable
Test performance data should be stored together with other versions and test data in an analytics database, and compared to the established baseline.

Actionable
Leverage comprehensive reports and analytics for rapid root-cause analysis; leverage deep network and test data for in-depth investigation.



Address:
4023 Kennett Pike 50128, Wilmington, DE, 19807

E-mail:
support@experitest.com

Phone:
+1-646-491-6262

For more information:
www.experitest.com