The Complete Guide to End-to-End Testing

SMARTBEAR

Content

What is End-to-End Testing?

End-to-End testing is a methodology used in the software development lifecycle (SDLC) to test the functionality and performance of an application under test. It is a process implemented by quality assurance (QA) teams today to not only test the product that's being developed or the pieces that it's artificially been bro- ken down into, but to gauge how the product works from the ***end user perspective*** by asking the question, "How are the consumers interacting with the product and is it a seamless experience?"

Software today is complex, and applications are rarely standalone products. They are often built on a network of integrated sub- systems including databases, interfaces, and other applications.

When one fails, so does the overarching product. As a result, you need to not only test your software - how it looks and how it behaves - but also how those sub components behave.

End-to-end testing is designed to provide full test coverage for each of these pieces, for both the API and UI layers, and each of the connected networks and third-party apps. Executing end-to- end tests will not only optimize your testing cycles but will ensure you're confident in your next release. While similar to integration or system testing, the ultimate goal of running end-to-end tests has a different focus – user experience.

The Goal of End-to-End Testing

The end game is all about the user. As a tester, it can be hard to remember that there is another human being on the other end of the product. Taking a step back from looking at only the performance and functionality of various test criteria, and moving away from the metrics-only lens is hard, but a vital step to ensuring success.

Remember the user's journey and gauge the success of your product workflow from a different angle. You'll need to walk through the steps your potential customers might take to better understand how they will interact with the software.

Having this mindset is critical for building a successful end- to-end testing strategy and will ensure that you validate your system from start to finish.

Benefits of End-to-End Testing

1.

Confirmation of Application Health

You need to test the features of your application to validate their behavior. While we just stated that end-to-end testing focuses on the user experience, you still need to confirm that everything works, not just together, but also separately.

Some teams may refer to this as systems or integration testing. One pitfall teams tend to fall into is that they will execute these tests for only the back-end or the front-end. It's essential that you avoid this and test all the systems - from the back-end to the front-end.

2.

Expanded Test Coverage

Another benefit of end-to-end testing is expanded test coverage. When you're testing every single unit, function, or feature of your application, you may not always be considering how the user

is interacting with your product. You will need to ask yourself, "What are all of the different environments your end user may use to interact with your application?"

As software scales globally and as the number of available devices continues to rise, it can be hard to predict what browser or operating system your users will be using to access your product. While challenging to foresee, if you can conduct end -to-end testing and plan your test strategy with the consumer in mind, you'll be able to expand your test coverage by including environments previously not considered.

3.

Discovered New Bugs

If you're not testing the same way your users are experiencing your product, you won't detect every bug. However, they might. Despite your best efforts, consumers always seem to create new and never-before-heard-of ways to interact with software. Bugs in these scenarios can be extremely challenging to find as they can slip by your functional and performance tests undetected.

It's essential to your success and the happiness of your end-users that you detect bugs before they reach production. The most effective way of doing this, is through end-to-end testing. The process will ensure you test your application, end-to-end, the way it would be used in the real world.

4.

Reduced Testing Resources

A key question that always pops up is 'if you're adding end-to-end tests to your already existing strategy, wouldn't you be adding more resources?' The answer is no. While it may seem counterintuitive, you have to remember that resources aren't limited to just people and budget. Time is a valuable resource, and one of the few you will never be able to get back. By implementing a robust end-to-end strategy, you'll actually cut down on key time-consuming tasks.

Maintenance

By catching bugs before your end users do, you'll cut down on the amount of time spent fixing those bugs. It's much easier to fix issues before they're released.

Defect fix time

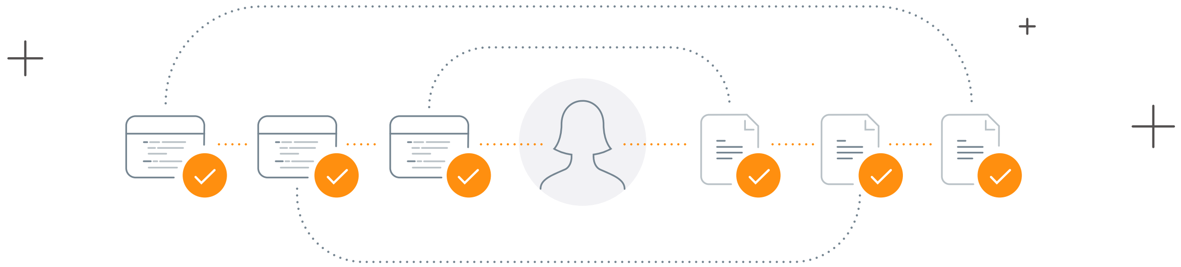
There is a huge benefit to finding bugs prior to production aside from avoiding maintenance hassles and unhappy customers. End-to-end testing provides detailed information on how individual pieces of an application work together in a specific context. When these tests fail, the bugs can be easier to find, making it much easier for developers to fix them. While this is not particularly a testing resource, having a development team that can fix bugs quickly makes it less likely that one bug could effect another. There are many moving pieces and the faster you fix one bug, the less likely it will be to cause further defects down the road.

Types of End-to-End Testing

There are two types of end-to-end testing – vertical and horizontal. While horizontal end-to-end testing is most commonly associated with this testing method, it's crucial to realize that both play a part in building a successful end-to-end strategy.

The process of horizontal end-to-end testing is used to verify that every workflow or transaction in an application occurs correctly. Historically, this may have occurred in a single application or interface, but software systems today interact with a myriad of external interfaces, both those developed internally and those owned by third parties. With horizontal end-to-end testing, it's essential your test environments are set up in advance to ensure a smooth testing process.

Vertical end-to-end testing entails thoroughly ***testing each sub system***, independent from one another, starting the bottom layer and working your way up through each series of connected pieces – from back to front-end. Let's walk through a real-world example to put the two into perspective.



How to Perform End-to-End Testing

We'll use a familiar example, ecommerce. As the end-user, the process of making an online purchase may look like the following:

1. Go to website
2. Navigate through the website until you find the pricing page
3. Add desired item(s) to your cart
4. Click "check-out"
5. Proceed to payment page
6. Insert user credentials, payment information, and address for shipping and billing
7. Click "submit"
8. Finalize purchase

Depending on the product you're purchasing, the vendor may need to send essential follow up information. These steps could look like the following:

1. User receives thank you email from vendor
2. User receives follow up email with the product to download
3. User receives a "Getting Started Guide"

The end-to-end tests that would be run in this example, starting with the website loading to ensuring the PDF of the guide downloads, define the process of horizontal end-to-end testing. The tests follow the consumer through their journey to make sure each step works flawlessly. You may not realize it initially, but there are a handful of vertical processes within each of these steps – the back-end processes needed to ensure various pieces and user actions such as the payment pages, inserting user credentials, and sending follow up emails all activate when prompted. Let's dig a little deeper.

Take a look at the **"Vertical vs. Horizontal"** diagram below. For each point in the journey along the horizontal axis, 'search & find,' 'order,' and 'purchase,' there is a duplicate vertical stack of systems above and below. For example, the 'search & find' step has database components connected to it. When you search for products on Amazon, the back-end systems trigger all sorts of APIs that will scour the site to pull previous behavioral data in order to provide suggestions of other products for you to buy. As you move along the horizontal line, the vertical stack moves with you and the back-end systems continue to fire to track your actions. This is how the site builds a profile on your purchasing habits.

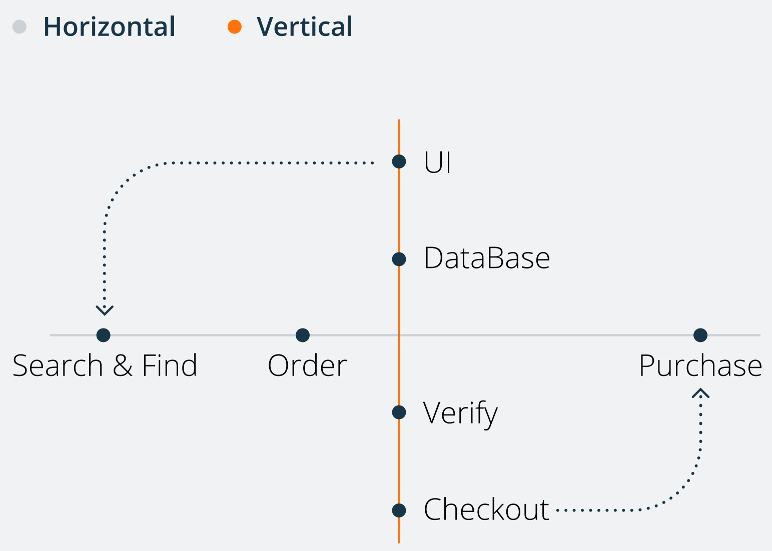
There are at least three sub systems that are triggered by one purchase – product delivery, marketing, and operations. To provide a seamless user experience, it's vital that this entire chain of processes is tested. That is how you will be successful with end to end testing.

However, when building your end-to-end strategy, you may ask yourself, "do I really need to test every single layer for every step?" The answer is no. Certain sub systems may be less important to different user actions. However, your end-to-end strategy is highly dependent on ***your application and your user journey.***

In our ecommerce example, we may not be as concerned about testing the database systems post-purchase. Ensuring the operations and marketing systems that are designed to deliver the product and vital onboarding information work properly may be more critical.

So which is better – horizontal or vertical end-to-end testing? To build a robust and successful strategy you need both. The end goal is to provide the user with a bug-free experience and in this case, it all starts with ensuring your webpage doesn't 404.

Horizontal & Vertical End-to-End Testing Example



How To Be Successful: Your Top-Down Strategy

Now that we've had the chance to walk through why end-to-end testing is important, the various types needed to build a robust strategy, and how to conduct the tests, let's take it a step further. What is the approach you should take to implement end-to-end testing?

Before diving into the 'top-down' approach, let's quickly review what the 'bottoms-up' approach entails and why we recommend a 'top-down' approach.

The 'bottoms-up' approach to end-to-end testing typically involves teams stitching together already-existing tests: API tests, UI tests, databases, and unit tests from developers. While the approach works and can be assembled quickly, you lose focus on the end user experience. When piecing together tests like a puzzle in this manner, you're essentially looking at the success of your end-to-end strategy as the sum of your functional tests.

The most successful end-to-end tests are built by starting with defining the experience you want your end users to have and what the workflows will look like based off these experiences. You can then break the workflow down and determine from there, whether or not you have existing tests that address each stage of the user journey and if not, that's what you need to build next.

These are the three key steps in implementing a top-down approach:

1. Define workflows from the user perspective
2. Decompose the workflows
3. Decide what you can do

With the top down approach, there are two vital SLDC pieces you'll need: well defined requirements and a solid framework.

Well Defined Requirements:

Well-defined requirements are rare. While good user stories can drive these, most teams aren't that lucky. In today's fast- paced environment, it can be a toss-up between spending time flushing out requirements and getting started. In many cases, user stories and requirements tend to focus on the functionality of pieces along the journey, and not define the workflow from start to finish. Having the user experience in mind when defining the requirements forces you to approach the problem.

Testing Frameworks:

Testing Frameworks are essential to the success of your end-to-end strategy. They can reduce maintenance costs and improve team efficiency by facilitating the reusability of code, standardizing processes, and maximizing test coverage. Building one starts with your user stories.

Once you've defined your requirements, you can decompose your workflows and start making key decisions, such as what tests should be automated, which tests should remain manual, and what environments your tests need to be run in. Having reusable code or test components will enable you to build quickly. You will otherwise constantly be rebuilding and redefining. You also need to be able to build the workflows in the same environments you have your tests set up in.

While having a robust testing framework and well-defined requirements are vital to ensuring your end-to-end tests are successful, there are eight other steps you should take to optimize your time.

Metrics for Success

As with any testing process, you'll want to know your time and efforts haven't been wasted on running end-to-end tests. The best way to do this is by tracking your progress with metrics. There is a vast array of numbers you can use to track progress and measure success, but there are four key metrics you should keep in mind to specifically gauge your end-to-end testing efforts.

8 Steps for Success

1. **Review the requirements you'll be using.** Any good testing process requires that teams have a basic understanding of the requirements.
2. **Set up test environments.** Having an idea, or making educated assumptions as to where your end users will be using your application, is the first step in ensuring complete test coverage.
3. **Define the processes** of your system and your sub systems.
4. **Describe the roles and responsibilities** for each system.
5. **Outline the testing methodologies** you plan on using. Ask yourself, what types of testing are you going to be doing - manual, automation, exploratory? What test cases should you be automating?
6. **Standardize testing processes.** You'll optimize your team's efficiency if everyone is aligned on what languages to script in, how to handle data, and what processes are in place.
7. **Create requirements for tracking.** We'll dig into this next, but it's essential that you have metrics tied to your requirements so you can gauge how successful your efforts are.
8. **List input and output data** for each system.

Test Case Preparation

This number, usually a percentage, is tracking the number of test cases that are being prepped against the number that you've planned.

Test Progress

This number measures how many tests you've completed against the total number planned, as well as whether or not they've passed or failed.

Defect Status

This number is often the percentage of defects that were opened and closed in a given time frame, as well as the severity and priority of each. Good end to end testing should enable you to fix defects faster, and you would ideally notice that here.

Test Environment Availability

This metric tracks the amount of time you think you'll need to run tests in a specific environment, how much time is actually needed for that environment, and how many total environments you need to run your tests in. You'll want to track this in measurements of hours or days.

Top 3 Most Common Mistakes in End-to-End Testing

After learning which steps you can take to be successful, that last piece of the puzzle is understanding what not to do. Avoiding these top 3 common mistakes in end-to-end testing will solidify your strategy and ensure you're getting the largest return on invest (ROI) for your efforts.

1 | Not including Environments

It can be easy to lose focus on the end user and not think of the context your application is being used in. This will result in poor test coverage and inevitably, bugs in production.

2 | Focusing only on functionality

A user experience is built on a combination of functionality, performance, and ease- of-use. While testing each feature to ensure the button is in the correct location on the UI, or that the API is calling the right information, ensuring that these fit together from the user perspective is vital. Good end-to-end testing should be focused on experience.

3 | Executing end-to-end tests prior to functional tests

Think of this as driving a car. You don't test the system while you're driving it down the road. You can't independently check the functionality of your app while it's in use, so it's important to do this before systems or integration testing. Feature bugs will be fixed faster when you're focused on finding them and when there are less variables at play.

Conclusion

Conducting end-to-end testing is critical to reducing business risks and essential to the success of your product. By taking a user-centric approach to testing software, you'll cover different scenarios and environments, effectively expanding test coverage. Finding bugs earlier in the process will also reduce testing cycles and minimize the costs associated with constant maintenance and upkeep.

End-to-end testing will validate your application is functional at every level, from the API to the UI layer, and ensure that the workflow between each piece works flawlessly. You'll feel confident your next release is bug- free and production-ready.

SMARTBEAR
TestComplete

Functional Test Automation
for Desktop, Mobile and Web

Get Started

SMARTBEAR
Zephyr

Native Jira & Real-time
Test Management

Get Started

SMARTBEAR
CrossBrowserTesting

Web-based, Mobile
and Desktop Browser Testing

Get Started

SMARTBEAR
SoapUI Pro

API Automated Funtional &
Security Testing

Get Started

SMARTBEAR