End-to-end testing: Useful practice or frustrating time sink?

Lee Badal (badal@kth.se)
Daniel Grünler (grunler@kth.se)

April 2021

1 Introduction

There are few things as painful in the development cycle as pushing a broken build to production. This simple fact makes test automation a key aspect of the DevOps approach. Most developers will be familiar with unit tests; small tests that quickly check the validity of some isolated component of the software. End-to-end (E2E) tests can be considered the antithesis of unit tests; they test the application on a system level by testing the effects of user actions. E2E tests can be either manual or automated, but in this essay we will be considering automated E2E tests unless explicitly stated otherwise. [1]

We first encountered E2E tests while working on an open source project. It was quickly apparent to us that the approach was promising and we were excited by its prospects, but we soon encountered substantial setbacks. The tests were unstable and time consuming and we were left questioning whether this was a result of our tool and implementation or if it was intrinsic to E2E testing. In this essay we will discuss E2E testing; how it works, what tools exist, what the advantages and disadvantages to the method are and when you should use it with the goal of indicating if automated E2E testing is a useful method or a waste of time.

2 E2E Testing

E2E testing is essentially testing software from start to finish, including integrations, dependencies, availability, and everything else. We usually speak of E2E testing in web applications although it is possible to implement for mobile applications or desktop applications. In E2E testing often a test requirement can directly translate to a use case, in example being a customer buying a product from an e-commerce site. [2]

The purpose of an E2E-test case could be to verify that all steps the customer takes are functioning as intended, that could mean that when a customer clicks on “buy” a product is placed in the basket, the correct product is placed in the

basket, the basket image is updated and that when the customer is ready to place an order the correct information is parsed to the system handling orders. The number of details in the test case can vary, but usually you want to ensure all the steps work as intended for a use case. E2E testing is the practice of attempting to verify exactly this. The first thought that might come to mind is to simply have individuals execute the test cases and report the results, this is known as manual testing and has been common practice since the beginning of software testing and still is today. When we speak of E2E testing we often refer to an automation of testing the whole system. Many E2E frameworks function by programmatically visiting a web application through a browser and imitating clicks and or inputs to the website. Since E2E testing is a method of testing we also expect an output to analyze, this could be a dialog box opening, a search result or any other predefined result. E2E testing frameworks usually allow developers to not only verify if objects are present but if objects are missing or faulty.

The tools often enable verification of everything a user could theoretically verify by using a browser and some common utilities within a E2E framework include [3],[4],[5]:

- Accessing a browser
- Finding elements on a web page
- Keyboard & pointer actions
- Clocks for timing
- Storing element information
- Crawling
- Snapshot testing

2.1 Snapshot Testing

The most interesting and recent innovation of these functions is probably snapshot testing, it attempts to remove a lot of the manual coding in test cases. To explain snapshot testing we will use an example of how it works: Consider the e-commerce example that we previously had, usually a website contains tons of components. We probably know what it should look like visually but writing the code for it can be a tedious and time consuming task. Snapshot testing involves taking a snapshot of the structure during a specific time when we know that the components are displayed as intended, so that whenever we run the tests we have a structure snapshot we can refer to. Whenever we make changes in the code that influence the saved snapshot the test suite will fail, we can then view what has been changed and evaluate if the changes made are intended. If the changes are intended, we can simply update our snapshot and the test suite will pass, otherwise we can debug why these changes occurred. [6]

2.2 E2E Testing Approaches

The classical approach to E2E testing uses the Selenium WebDriver, an API that controls the browser and supports tests in multiple programming languages.[4] Another approach is to execute the tests directly inside the browser using JavaScript. One such framework is Cypress. [7] The most recent development is an increase in AI-based tools to automate E2E testing. [8]

3 Advantages And Disadvantages Of E2E Testing

Before deciding whether or not E2E testing is worth it, let's have a look at a few of the claimed benefits and drawbacks associated with the practice.

E2E testing comes with a number of drawbacks. Chief among these is that E2E tests are often fragile. E2E tests interact with the graphical user interface (GUI) of the web application. As such small changes to the page layout such as adding choices to a drop-down list could end up breaking tests. This will result in a much higher need for maintenance than other approaches such as unit tests. [9] The fragility issues do not end there. As E2E tests incorporate the entire system they often suffer from so-called "flakiness" where tests pass or fail nondeterministically. This might force the developer to re-run the test multiple times or spend time investigating the failure. [10]

Additionally E2E tests are slow. Running an E2E test requires running the entire application. Even discounting the time it takes to start the application E2E tests cover substantially more code than unit or integration tests and therefore take much longer to run. Adding these factors together an E2E test is often several orders of magnitude slower than other tests.

The above factors contribute to one of the major disadvantages of using E2E tests: cost. E2E tests are expensive. The large and flaky tests gobble up resources, force developers to investigate the cause of failures and there can be substantial costs associated with continuously maintaining and rewriting the tests.

E2E tests are also a poor choice for identifying the root causes of bugs. As black box tests at a high level of abstraction they say very little about what caused the test to fail. An E2E test might cover thousands of lines of code which makes the developers' task of isolating and fixing the bug incredibly hard. [10]

So why then would anyone use E2E tests?

The most obvious advantage of E2E tests is that they directly test use cases using automation. E2E testing is the optimal way to test that your application actually works as intended as it simulates the end users' interaction with the entire system. This is not feasible using lower level testing approaches as they focus on isolated parts of the system. The use case approach makes E2E tests more likely to spot complex bugs and bugs that the end user would experience when interacting with the application as it was intended to be used.

E2E testing also offers multiple advantages over manual testing, the main alternative to finding high level bugs. E2E tests are much faster than manual tests and they can be run at any time whereas a manual tester might not be able to work during the night. They can log test data automatically and they minimize human error. But the main advantage of E2E testing over manual testing is cost of scaling. Paying a tester to manually click through the same user scenarios thousands or millions of times would be prohibitively expensive and getting a developer to do it would not only be a massive waste of resources but also most likely force them away from your company. As such it is much better to leave the bulk of the use case scenarios to automated E2E testing.

4 When To Use E2E Testing

So when should we use E2E testing? As we've seen from the previous section E2E testing offers some benefits over other testing approaches but suffers from some severe drawbacks. High costs and well-known fragility issues make E2E tests a daunting option but is it all worth it if we can more closely test the application from a user perspective?

The industry opinion seems to be yes - but with a caveat. E2E tests can not replace lower level tests as they serve different purposes. E2E tests should instead be considered a complement to other testing approaches. A common guideline for determining the proportion of different tests in a project is the so-called Testing Pyramid (figure 1). It comes in various flavors but commonly includes three layers from bottom to top: unit tests, integration tests and E2E tests. The higher you go into the pyramid the larger the tests get and the fewer tests you should have in your project. [11]

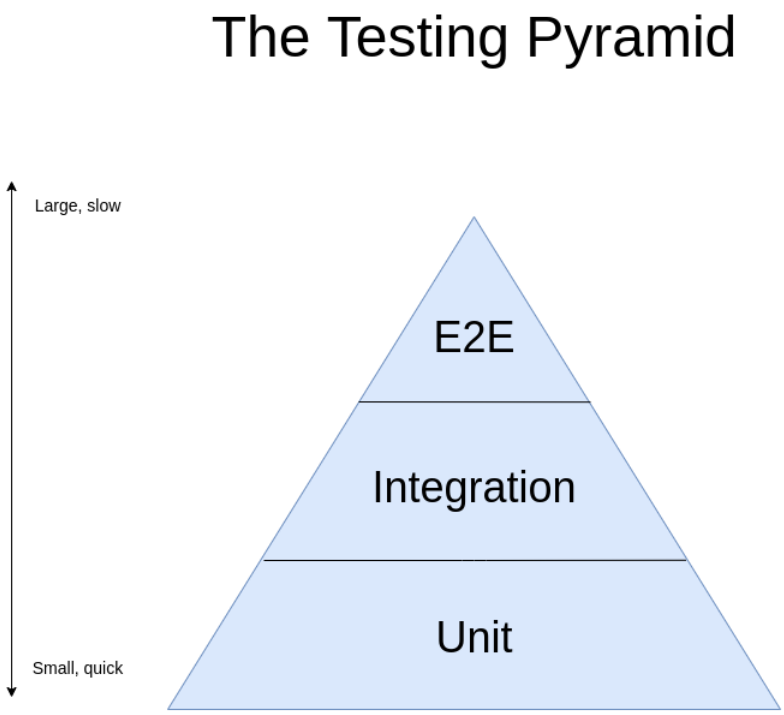

Figure 1: The Testing Pyramid.

By following the Testing Pyramid you ensure that E2E tests does not consume too much of your testing resources. This will lower your costs and reduce the time it takes your tests to run while still ensuring some automated system-level tests.

Finally any project should consider their scale before applying E2E tests. For smaller projects the cost of creating and maintaining the tests might not be worth it, and they might be better served by manually testing the application.

5 Conclusion

In theory end-to-end (E2E) testing offers a fantastic proposition. Being able to automatically test your application on a system level according to use case

scenarios is a very exciting proposal for any web application project. In practice this proposition starts to break down. E2E tests are slow, expensive and unreliable compared to other automated tests and when they fail they do a poor job of identifying what caused the issue. These issues have limited the role of E2E tests in application testing.

Should you then use E2E tests for your project? The answer is not that simple; we believe there is a time and place for E2E testing and it can indeed find crucial errors that are undetectable using unit tests or integration tests. However, in its current state E2E testing is a complex and costly affair which can lead individuals to waste valuable time and money.

We believe that your first focus should be on building your application and test it using other methods such as unit, integration and manual testing. If the manual testing becomes a chore, your team is large or you're planning to scale up your application then you should consider implementing E2E tests. Just make sure not to overdo it or the maintenance costs and stability issues might make you regret your decision. If you're a small team working on a hobby project it might not be worth it in the first place and you should consider other options.

References

[1] Sten Pittet. *The different types of software testing.* URL: `https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing`. (accessed: 29.04.2021).

[2] Dan Widing. *What is End-to-End Testing?* URL: `https://prodperfect.com/blog/end-to-end-testing/what-is-end-to-end-testing/`. (accessed: 28.04.2021).

[3] *Why Cypress?* URL: `https://docs.cypress.io/guides/overview/why-cypress#Features`. (accessed: 30.04.2021).

[4] *Webdriver.* URL: `https://www.selenium.dev/documentation/en/webdriver/`. (accessed: 28.04.2021).

[5] *Puppeteer Documentation.* URL: `https://devdocs.io/puppeteer/`. (accessed: 30.04.2021).

[6] Gleb Bahmutov. *End-to-End Snapshot Testing.* URL: `https://www.cypress.io/blog/2018/01/16/end-to-end-snapshot-testing/`. (accessed: 29.04.2021).

[7] *Testing has been broken for too long.* URL: `https://www.cypress.io/how-it-works`. (accessed: 28.04.2021).

[8] *5 Popular AI-powered tools for test automation.* URL: `https://www.nextgenerationautomation.com/post/ai-powered-tools`. (accessed: 29.04.2021).

[9] Filippo Ricca, Maurizio Leotta, and Andrea Stocco. "Three open problems in the context of E2E web testing and a vision: NEONATE". In: *Advances in Computers*. Vol. 113. Elsevier, 2019, pp. 89–133.

[10] Mike Wacker. *Just Say No To More End-to-End Tests*. URL: `https://testing.googleblog.com/2015/04/just-say-no-to-more-end-to-end-tests.html`. (accessed: 27.04.2021).

[11] Shreya Bose. *Testing Pyramid: How to jumpstart Test Automation*. URL: `https://www.browserstack.com/guide/testing-pyramid-for-test-automation`. (accessed: 28.04.2021).