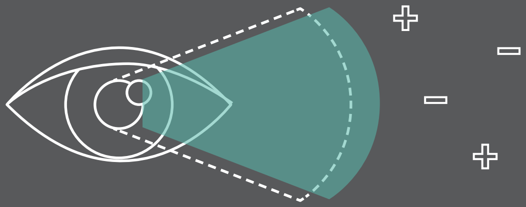
5 Visual Testing Best Practices

AUTHOR
Eric Goebel
Software Development
Engineer In Test (SDET) III

accusoft®



During my time in the domain of quality assistance, I've seen many teams struggle to get value from visual testing. Visual testing, at its core, is just making sure that the UI looks right to users. So with that in mind, I'll elaborate on a short list of best practices which I have found help teams avoid typical stumbling blocks.

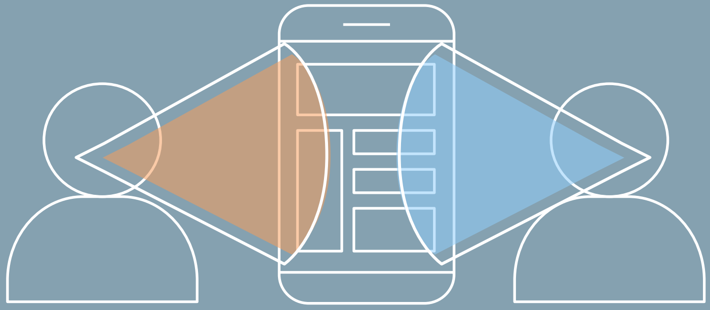
1

Don't Do It

This may seem to counter the "best practices" claim, but in reality it does not. Visual testing is a tool on your team's utility belt just like any other testing method. Knowing when and where to use the right tool is part science and part art; and when it comes to visual testing, the same rules apply. I've seen many teams eager to adopt visual testing early on without investing effort into tests at a level which precedes the system level. This phenomenon is understandable because visual testing is easy to conceptualize. So convincing teams to take it on is less difficult than some other endeavours.

Take, for example, a scenario in which we want to test the output of a process that takes multiple forms of input, processes it, then returns some output on a web page. The number of permutations for something like this can end up being huge. If the way this data is presented to the user on the system level does not differ much from case to case, then it may make more sense to offload the majority of the tests somewhere else. With a more comprehensive set of tests in place at a lower level, we could focus a smaller amount of testing at the system level where we can do a visual check that the returned data is visible. The key point here is that we want to visually check that the data is there but we don't validate the content of the returned data. We would be running through huge data tables of input at the contract test level or integration test level to validate the content of the returned data.

Given that even the best implemented visual tests can require more maintenance on average than other tests, efforts should be made to reduce the reliance on them whenever possible. Promoting the perception that visual testing techniques are "the big guns" or "the nuclear option" helps teams push the effort for testing closer to the bottom of the stack.



2 Be Consistent in the Language of Testing

It is important to decide early on about the format of testing steps and the terminology used. The saying "a picture is worth a thousand words" rings especially true when reviewing visual tests. If the objective of the test is even marginally unclear in either the description or the steps of execution, then it leaves the interpretation of what is being seen in a failure up to the reviewer who may be looking at a particular test failure for the first time. Just imagine that two people are individually given the task of describing a screenshot of the same test failure. How likely is it that the same exact description (word for word) will be given? Some clarity is forcibly introduced when automating visual tests due to the fact that good assertions are fairly concrete but precision in language early on saves time further down the road.

There are quite a few things to cover in terms of language consistency, but some key points that I have seen help teams overcome ambiguity is defining, either officially or unofficially, the schema for the construction of the test steps. If there is a standard model to follow in either the existing test suite or a documented pattern to follow from a wiki - and those tests are constructed in the same pattern in comparison to each other - then they generally read better.

Many behavior driven models will also express the importance of tense in the steps provided. Test writers should get used to using "present perfect" tense wherever possible when defining steps and specifically in steps which directly precede an assertion. One of the reasons this is important is because if you are looking at a screenshot depicting the state of the application under test, it is much more difficult to troubleshoot something that is potentially happening rather than something that has happened. So writing in this manner helps to enforce that assumption in both writing the automation around the steps and reviewing any potential failures.

3 Narrow Your Focus

When trying to continuously integrate and continuously deliver, every moment spent manually reviewing test failures is an expensive distraction. It may be appealing to test the full page/ full screen under the guise of testing multiple things at once but … it's a trap!

Okay, it's not always a trap, there are cases in which you want to do that at a system level but generally, not as part of a pipeline. To minimize the time an engineer is required to review a failure, we will always end up zeroing in our tests to make assertions about the application under test are which more and more specific. If your test is doing some type of image comparison to a baseline, ground truth image, the images which are compared should not contain a navigation sidebar if we don't expect the functionality being tested to alter the navigation sidebar in any way (you should have other tests that look at the navigation sidebar).

If, for some reason, the navigation sidebar looks slightly different because of some external factor like a browser update (also preventable) then the test fails and you have effectively triggered a manual investigation into a false positive. Not only is there a time cost in investigating but also in refactoring tests or re-baselining.

Image comparisons should also be avoided. The methods used to do comparison on images vary in effectiveness and complexity, so it makes sense in many cases to opt for asserting aspects of the layout instead of a direct comparison. If you do choose to do image comparison (like a pixel to pixel comparison between two images), consider what types of differences you would allow and know how difference thresholds will work for your situation.

For example, if I wanted to allow for up to a 3.5% difference between images strictly measuring by pixel, then having one row of pixels different in any direction can make a huge difference if the size of the captured area is different.

Example: A nearly indistinguishable difference is treated differently in automated pass/fail scenarios.

Device	Image Size	1 row of pixels changed	Difference %	
Laptop	500x500 (250,000 area)	500 total	4	Fails
Mobile device	300x300 (90,000 area)	300 total	3.3	Passes

In the table above, we could imagine that the same object is being rendered on two different devices that are using different screen resolutions, but our capture method has no way of acting on such granularity. In this reasonable scenario, a browser update causes both objects to have one row of pixels render differently but the failure is only reported on 1 platform. Out of context, a manual review might only trigger a re-baseline of the laptop scenario but leave the mobile as-is which may later incorrectly show that our expectation is out of sync.

4

011001
101000
001010

Logging

We can't always determine what went wrong when something fails visibly. Sure, we know when something doesn't look right, but how do you answer why? It is imperative that the reviewer is provided the means to understand what went wrong without having to replay the tests locally. There is no singular method that must be adhered to for good logging, only that it must be done in a way that provides value.

When visual testing is first adopted, there is sometimes an underlying assumption that since it is visual then the answer will be obvious. In many cases, that is true but for more complex applications, an image just does not answer enough questions about the actions that took place leading up to the failure. Timestamps help narrow the window of investigation but without some key identifier in the logs that can be tied to the visual error the time investment to fully realize the issue could be large.

5

Make the Secret Sauce

Basically, you should pursue novel and unique approaches to problems you face as a team when it comes to visual testing. There is plenty of room for innovation in the visual testing space if you choose to develop your test framework in-house. Once you get beyond the typical pitfalls, there are going to be challenges that are specific to your environment. At Accusoft, we've found that looking inward at our talent for answers, in many cases, worked better than looking outward at the industry space. For the time being, though, our secret sauce must remain a secret.

accusoft®

www.accusoft.com