Appsurify

Test Selection with Appsurify's TestBrain

By James Farrier, Founder and CTO

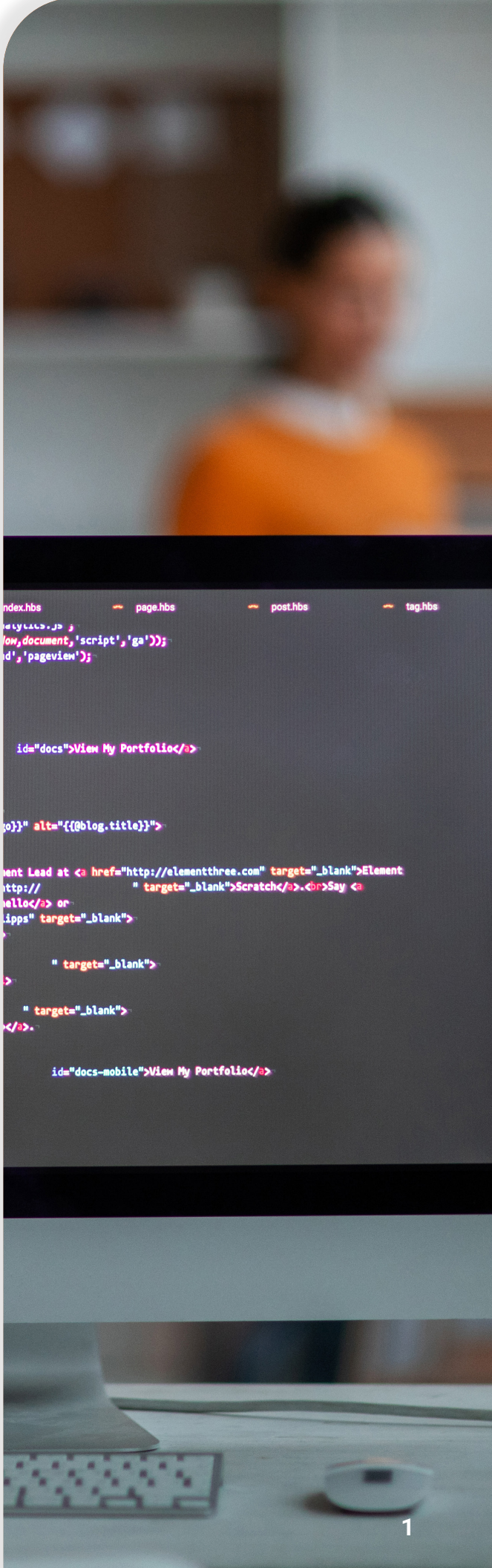
Introduction

Many organizations have a large number of automated tests. Automated tests are built to reduce the burden of manual testing, enabling companies to quickly regression-test their application, leaving manual testers to perform exploratory testing and more complex tasks that may be difficult to automate.

Current trends in software development involve companies shifting their testing left, getting results to developers faster to move towards a continuous delivery/deployment model. This involves having the automated tests run as part of the continuous integration pipeline. However, for the automated tests to be included in the pipeline, the automated tests need to execute in a short amount of time and the results need to be accurate.

When tests take longer than three minutes, studies have shown that developers become less efficient, either taking more breaks or moving on to other tasks as they wait. This leads to context switching, where productivity is reduced and the number of defects created is increased. Additionally, long test execution time means developers receive feedback on their commit later than anticipated. The later a defect is found, the longer it takes to fix.

Similarly, if the results of the automated tests are not accurate, teams may either release potentially buggy software or spend time debugging, ultimately decreasing productivity and increasing frustration.



To counteract long-running test suites, companies have the following options:

Parallelize the test execution. This, however, is limited by the number of processors available to execute the tests and increases the cost of running the test automation.

Reduce test suite size. This may either involve splitting the test suites into multiple, smaller test suites of different priorities, or removing tests completely. Regardless, this option may cause defects to be missed and can slow feedback to the developer.

Use test coverage to reduce the number of tests executed. Unfortunately, this option is often difficult to deploy, requiring the build and tests to be instrumented. Additionally, it will run any test, which covers the change regardless of whether the test actually tests the change. Essentially, this means that more tests are being run than necessary.

Continue with long-running test suites.

Appsurify's TestBrain offers an alternative solution that uses machine learning and the test execution history to reduce the number of tests that need to be executed for each change. This method also involves the automation of the defect management cycle and flaky test failure detection.

In this paper, we will describe:

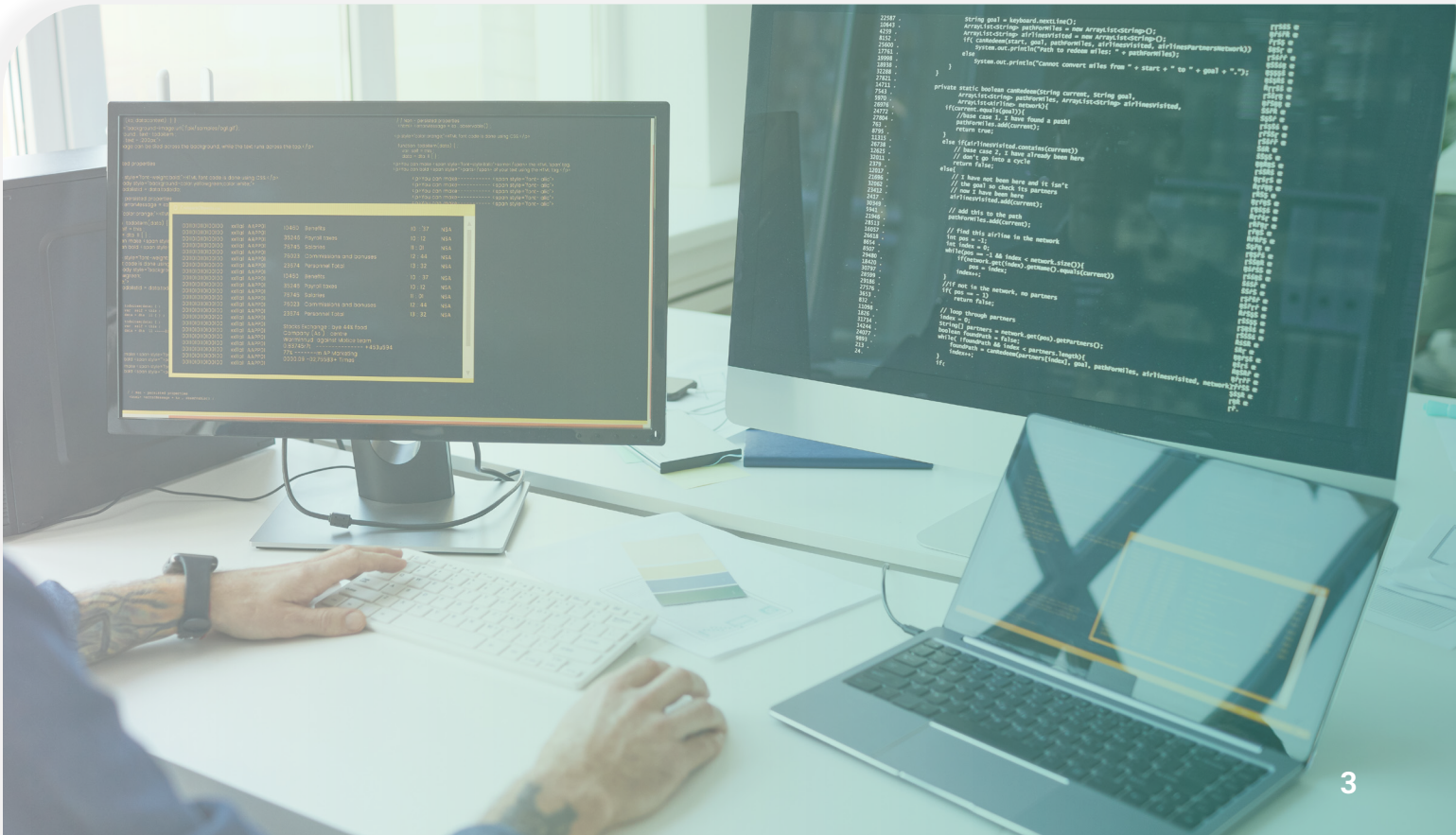
- The current developer workflow and how automated test selection can fit into this workflow.
- How tests are selected, including initial data setup to speed up the machine learning.
- How flaky tests and other defects are automatically created.
- Integration of Appsurify's TestBrain within a company's CI toolset.
- Expected results from Appsurify's TestBrain test selection.

Development Workflow

Almost all companies now use a continuous integration tool as part of their development process. When a developer commits his code the continuous integration (CI) system picks up the code change, builds the code, and runs the appropriate set of tests.

Each of these steps can be “gated,” meaning that they need to pass in order for the next step in the pipeline to be executed. When tests are a gated step in the CI pipeline, developers and testers need to wait for the tests to be executed before the build can be released.

Incorrect test results or long-running test executions tests are frequently removed from the CI pipeline. Unfortunately, this often leads to tests not being run, reducing the value of the test automation and potentially meaning defects are missed.



Selecting Tests

Appsurify's TestBrain uses a machine learning model to determine the likelihood of a test failing based on a particular change. As a result, TestBrain is able to produce a list of tests that are most likely to fail for each specific change.

To build the machine learning model, data is extracted from the change, including the files changed and the author of the change. Then, riskiness of the change is calculated by TestBrain's software. Additionally, the test result data is added to the model, mapping which changes in the past have caused tests to fail.

To increase the accuracy of the test selection, Appsurify's TestBrain automates the creation of defects, including determining the type of defect. This data may also be overwritten by the test team. Using this data, we are able to determine the real reason the tests failed when compared to those which have failed due to flakiness.

Once the tests have been selected, they are further prioritized based on the likelihood of finding a defect per second (i.e., a test which is five percent likely to find a defect and takes one minute to execute will be prioritized higher than a test that is six percent likely to find a defect and takes two minutes to complete). By prioritizing the tests in this way, we are able to further reduce the time taken to find a failure and thus reduce the time taken to provide feedback to the developers.

Data Setup

To reduce the time required to produce accurate results, Appsurify's TestBrain allows users to provide the following information to development teams to improve the results of its test prediction:

Test set to functional area mapping
Mapping which sets of tests map to which sets of files within the application.

Functional area dependencies
Mapping sets of files that then depend on other sets of files.

Initial Heuristic

Initially, before the machine learning model is capable of returning accurate results, Appsurify's TestBrain can use the data input above to determine which set of tests to run. In this case, TestBrain looks at which set of files have changed in the commit being analyzed and then compares this to the data created above, determining which tests to run based on the dependencies and the test set mapping. While in Heuristic Mode, TestBrain will also use defects to determine other potential tests to run that may not be included in the mapping.

Machine Learning

TestBrain initially uses a heuristic to determine which tests to execute. However, after each test execution, TestBrain compares the results to those from the machine learning model. When the model produces results that are more accurate than the heuristic, TestBrain then switches to start using the machine learning model.

TestBrain uses a classification model that uses the following features (and others not listed) to determine which tests to execute

Test History

This includes details about the defect created from the test result. The defects created can be various types that are detailed in the following section. As a result, TestBrain is able to prioritize tests that are more likely to find “real” defects as opposed to flaky failures.

Files Changed

Which files have been changed by the commit.

Data Setup

The data that can be added in the previous section.

Change Risk

Details of the risk of the change can be found in TestBrain’s previous white paper.

Defect Density

Which areas of the code base are most likely to contain defects.

Flaky Tests and Other Defects

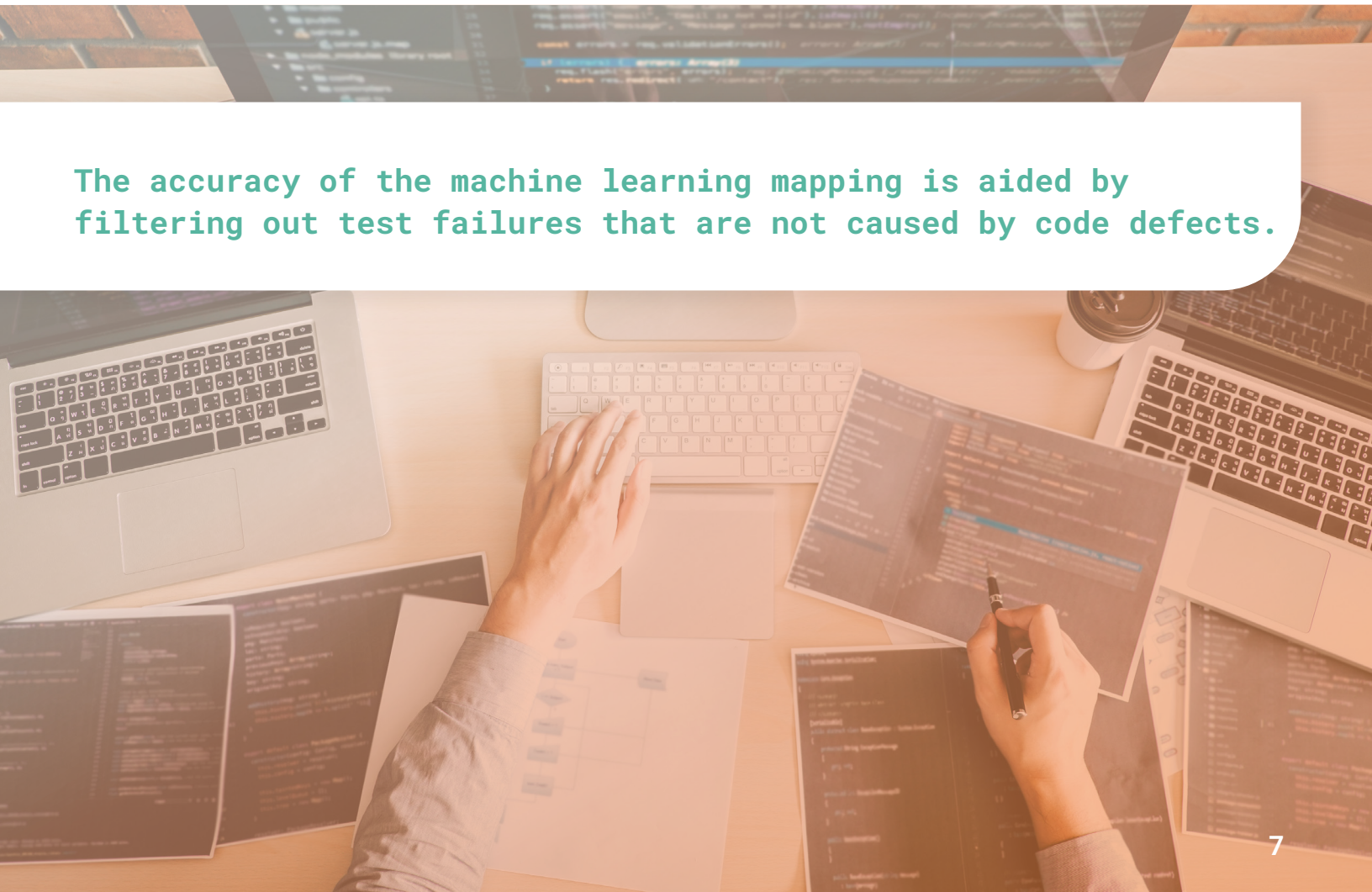
When a test fails, TestBrain raises an appropriate defect based on the type of test failure. The defect types that can be raised include:

Code defect:
Real defects in code.

Flaky failure:
The failure was non-deterministic, it may be that the test itself does not create a stable result or the application may also cause unstable results. This is frequently caused by race conditions, but a large number of other factors can cause flaky failures.

Invalid test:
The test is no longer valid due to changes to the code. The change made has caused the test to be invalid (i.e., it is out of date as this failure was expected).

Outside scope:
The test failure was outside the scope of what the test was attempting to validate.



The accuracy of the machine learning mapping is aided by filtering out test failures that are not caused by code defects.

The classification of a defect is determined via either manual classification, a heuristic, or machine learning model.

Heuristic: TestBrain can be configured to automatically rerun any failures for a specified number of times. If it passes once on retry, the failure is determined to be flaky.

TestBrain analyzes the test results and compares it to previous results to determine the type of defect. The model compares new failures to the previous failures based on the following factors (and others not listed):

Average run time of the test when passing vs. run time for this failure (i.e., if a test typically takes one minute to run and this time fails after five seconds, it is highly likely that something went wrong setting up the test itself, indicating a flaky failure).

Test name

Defect failure message

Logs

Area of the code changed

Recent history of the test shows many different results

Defect results can be manually updated by the user to the correct defect type and the machine learning model is adjusted.

Appsurify's TestBrain Integration

Appsurify's TestBrain is simple to integrate into the developer workflow.

There are two points of integration:

TestBrain needs to see data from the repository, which can be connected either by:

- A script that pushes metadata to TestBrain which can be run either on a VM or as part of the CI.
- A direct connection to the repository via multiple methods. For more information, see the TestBrain knowledge base.

Integration with the method by which tests are executed, often the CI system.

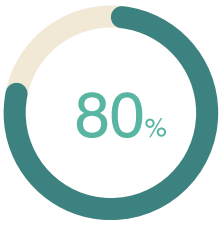
Appsurify's TestBrain has an API that returns the set of tests which should be run for the specified change. The format of the returned tests can be changed based on the parameters supplied in the API request.

For easy integration, TestBrain supplies a script which, based on parameters used, will automatically call out to TestBrain's APIs to determine which tests to run, run those specific tests, and verify the results.

Conclusion

Appsurify's TestBrain has managed to achieve excellent results on both open and closed source projects. On the jsoup open source project, TestBrain was able to reduce the number of tests executed per test run by **98 percent** while catching all defects. Initial customers have also had impressive reductions in their testing, with companies being able to reduce their testing by **80 percent** from the first test run.

As Appsurify's TestBrain consumes more data the accuracy of its results increases. Using the data setup detailed above,



TestBrain is quickly able to reduce testing by upwards of 80 percent, moving test execution time from 24 hours to one hour in a single instance.

After roughly 500 test runs, the accuracy allows testing to be reduced by upwards of 95 percent.

By reducing the amount of testing required by such substantial amounts, Appsurify's TestBrain not only is able to speed up getting results to developers, but also increases productivity and reduces the damage caused by defects as well as the costs of testing.

To learn how Appsurify's TestBrain can meet your specific software testing needs, visit our website or give us a call.

appsurify.com | 1.650.402.1400