Financial Services Client

Portfolio valuations and risk web service

Problem

The customer had a SPA website to display a customer's live positions and valuations as well as those from the previous business day. All of the data was sourced from a webservice which exposed the data generated by an application stack responsible for generating the data.

As the main customer requirement was to show live, "ticking" risk, each instance of the service would be running on its own continuous loop of generating valuations and then repeating as soon as possible with any updated market and trade data. Therefore it was impossible to guarantee that any 2 instances would provide exactly the same live numbers.

Because customers were relying on this data to be able to make trading decisions, it was vital that any changes to their numbers were well understood (frequently there were intentional changes to reflect better modelling choices etc.) as any unexplained differences would cause consternation with customers, especially when they discovered the differences rather than being forewarned.

Because of all of these factors, every software release was time consuming and cumbersome which was slowing down the development team. There was a desire to be able to release more quickly whilst retaining the quality standards that the consumers had come to expect.

This pre-release testing was performed manually, with a human tester checking each of the portfolios for any impact by having 2 instances of the website, production and the candidate, on screen and comparing the numbers. This was time consuming, prone to operational risk and wasn't scalable.

In order to complete the process within the available time, not every portfolio could be checked and therefore the developers would warn the testers where they expected there to be impacts and this is where the testing was targeted. This meant that where changes occurred unexpectedly, they were frequently overlooked until discovered by the end user, triggering a rollback of the release until the investigations could be completed. This didn't mean that the change was wrong and that the updated value was wrong, merely that there wasn't enough time to confirm that to the end client prior to this rollback decision being taken.

Solution

The desires here were:

- Increase the number of portfolios which could be tested
- Increase the range of functionality which could be covered
- Use the manual testing resources more effectively to test

As it was very easy to spin up new instances of the main stack / web service, we created a new tool which could do the following:

1. Call the webservice and ask for the complete list of portfolios[1]
2. Iterate over the list of portfolios and, per portfolio:
    a. Call the live production instance for the EOD valuations for the previous business day
    b. Call the candidate instance for the same criteria
    c. Process the returned results[2] to generate flattened representations of the per-trade results using the BorsukSoftware.ObjectFlattener framework.
    d. Compare the per trade results using the BorsukSoftware.ObjectComparison framework.
    e. Repeat the previous 2 steps for the summary values returned
    f. Output a payload file containing both a summary of the differences found[3] and the drill down payload[4].
3. Iterate through all of the payload files to generate a single differences file for all portfolios to help with any required analysis[5].

The CI process was then updated so that the new tool was called[6] on the newly spun up candidate instance and the test results were uploaded to Conical. Where no changes were detected, the pipeline stayed green. Where changes were detected, the pipeline[7] went red which flagged up the need for the developers / testers to investigate.

This additional pipeline could be run both on demand during the day for developers to double check the impact of any changes which had occurred as well as overnight so that the testers could have the results available to them in the morning.

---

[1] This was security controlled etc. Done by the appropriately permissioned user on their web service.
[2] The customer's result structure was based on 'per trade' results as well as group level summaries.
[3] E.g. For the trade level figures, 546 matched, 3 differences, 2 additional, 1 missing etc. A similar summary could be provided for summaries.
[4] So for each trade with a difference, the trade identification information was provided alongside the values which were different etc.
[5] Typically, in the case where a change was made and it was expected that it would have an impact on every portfolio, e.g. a new value was added or one was removed, then every portfolio will go red. In order to avoid the case where this hid other, unexpected changes, the testers could look at this macroscopic file and see which keys were different and then drill in on any which were unexpected.
[6] The tool could also be run on local instances of the stack so that developers could run the same analysis work (whilst going for a coffee etc.) to be certain of the impact that their change would have on the actual customers' portfolios.
[7] This was an additional, separate to the code compilation pipeline so that all of the artefacts etc. weren't lost

The net effects of these changes were:

Improved Developer confidence

Because developers could see the impact, either by running locally prior to check-in or on the CI server, that their changes would have on the end customers' numbers, they were able to reduce the length of the development-validation cycle to a matter of minutes compared to days beforehand.

Better use of manual testing resources

Because the testing process was no longer as manual as possible, and the testers then subsequently knew where they should be investigating and where they didn't need to investigate, they had more time to use their analysis skills to investigate why there were differences rather than simply identifying that there were differences.

They were then able to get in touch with any impacted customers ahead of time to work with them / inform them etc. so that there were fewer surprises.

Fewer rollbacks

Because the developers / testers were then able to see the impact ahead of time, developers and testers knew what the impact was going to be and therefore any decision about whether to not release a particular change could be made as part of the normal release decision rather than there needing to be a "panic induced" rollback after release prior to any investigation having been completed.

This meant that the success rate of releases[8] went up to something approaching 100%.

Faster release cycles

Because of all of the above, the length of time it took to be able to perform a release went down from being measured in days, to being measured in a low number of hours[9]. This allowed the release process to be moved from weekly with a high chance of rollback to daily with a virtually guaranteed chance of success.

This allowed the customer to be much more responsive to their customers' needs.

[8] The number of releases which didn't need to be rollbacked
[9] There was still some manual testing required of other components etc. so it wasn't 100% automated

Future steps

The testing could have been taken further by looking to cover more of the comparison of the "live" numbers. The problem here is that it would have potentially thrown up a large volume of false negatives due to differences[10] in trade populations[11] / market data[12], these would not have been insurmountable but the primary value had already been achieved.

Summary

By reducing the amount of time taken to manually verify that numbers hadn't changed, the client was able to improve the quality of testing performed in a dramatically reduced time period allowing them to move from an effective release rate of fortnightly[13] to daily releases with very infrequent rollbacks.

All of this was done without needing to make any changes to the web service being tested.

[10] The application being tested operated on a continuous loop so there was no guarantee that the timestamp for the inputs to any 2 instances would have been the same.
[11] This wouldn't have actually been a major problem but would have potentially added to noise unless these differences could be masked, but that would have been at the cost of not testing that functionality through the integration tests. The trade functionality itself was relatively easy to cover with unit tests at the service level so it might not have been a major issue, but it would have required careful consideration
[12] These could have been handled by adding per-value tolerances etc.
[13] Assuming half of weekly releases needed to be rolled back for whatever reason, then this leads to a developer being able to realistically expect their software to be available to clients after a fortnight.