Evolving Test Practices at Microsoft

11/13/2017 • 9 minutes to read

In this article

Testing at Microsoft circa 1990s

By Munil Shah

Testing at Microsoft circa 1990s

For a long period of time Microsoft shipped software with a basic setup for the engineering team. Every product team had three distinct disciplines. *PMs* owned understanding customer requirements and writing feature specs, *Devs* owned designing and coding the features. *Testers* owned testing the features. Teams had a ratio of 1 dev to 1 test to 0.5 PM, with some variance among teams.

The test discipline had two separate roles: Software Design Engineer Test (SDETs) who developed the automation, the test infrastructure etc., and then Software Test Engineer (STEs) who ran the automation or who ran manual tests. We hired SDETs from the same places we hired SDEs. They had very similar qualifications and skills when they got hired. They went to the same colleges. If we hired from industry, we'd pretty much hire developers and then convert them into SDETs.

How Did It Work in Practice?

It worked reasonably well back in the days. We achieved commercial success with big products like Windows and Office. The test discipline provided product signoff based on very formal quality measurements. That gave us good confidence in declaring a product ready to release. The test discipline built deep expertise in testing techniques and tooling. One of the benefits of this model was that when we were ready to do a product sign-off, we'd have the test discipline bring a very formal sign-off criteria and formal measurements in quality. They also developed deep expertise in testing because the test discipline was solely focused on testing, thinking about this day-in day-out.

What Didn't Work

Did it really work though? In a single word – NO. We started seeing some problems but were somewhat masked by the commercial success of the products. By late 90s the problems boiled over. Developers threw the code over the wall to SDETs. SDETs threw the test automation over the wall to STEs. We responded by continuously growing the STEs, particularly the vendors. STEs had limited career opportunities to advance within the test discipline. It was very expensive to maintain this setup. Testing became a bottleneck and caused product delays, but again we couldn't see through it.

First Transformation, circa 2000: Collapsing SDET and STE

By 2000, the whole company recognized this problem. We did our first major transformation in the test discipline. A companywide decision was made to get rid of the STE roles and make SDETs accountable for not only developing tests but also running and maintaining automation. It was a painful transition for the company. We worked hard to move STEs to other roles, and some did, but a good number of them didn't.

This improved the model. It improved accountability and incentive on SDETs to write better and more automation. But the core problem remained. The SDETs couldn't keep up with the code thrown over the wall.

We compensated by extending the validation phases in the release cycle. Another response to this continued problem was introduction of a notion called MQ (Quality Milestone). It was a special phase added at the beginning of a new release cycle to catch up on all the test debt accrued in the last release. This idea had good intention, but it didn't work in practice for a couple of reasons. It created a known period of time to work on quality that would cause people to defer test debt. Since there was a milestone dedicated to quality, it would unleash the team to work on random quality initiatives. MQ often created priority inversions. It took fixed number of people to work on quality, sometimes without good sense of priority or wrong people working on things that in the grand scheme of things were just low priority.

Enter Cloud Cadence

Things changed dramatically with the arrival of the cloud cadence around mid-2000s. It brought new pressure on the engineering system to go faster and even more faster. Long stabilization phases are gone. We don't get customer validations through Beta milestones or dogfooding (deploying code internally inside Microsoft). Microservices are deployed independently and constantly. The services need to stay up with high availability and must support no-downtime deployments.

Our initial response to these new challenges was to keep doing what we knew how to do, but do it faster. There was a big push to make existing test automation run faster and smarter by running only tests impacted by the changes.

This initial approach didn't work. Testing became a major bottleneck – we reached a breaking point. In theory we were operating in cloud cadence but the trains didn't run on time. In Azure DevOps, at times we would spend 3 weeks after the end of sprint to stabilize and deploy – effectively killing the next sprint.

Quality in the Cloud Cadence – What Changed?

It became very apparent the model wasn't working; we were doing it wrong. We were not the first team to recognize the problem. There were services before us, like Bing, that saw this. And we started observing the best practices of some of the companies born in the cloud
How our approach to Quality changed in the Cloud cadence? We did three big things.

1. We redefined quality ownership, we fixed the accountability.
2. We understood that in order to ship frequently, Master branch must always remain as healthy as the release branch. We defined a core principle – Master is always shippable. The principle touches everything – source code management, code practices, build etc. From testing perspective, we pushed

two things: shift-left testing (i.e. greater emphasis on unit testing) and eliminating flaky tests.

3. We also understood there is no place like production. This is the shift-right part of the strategy. it's a set of practices about both safeguarding the production as well as ensuring quality in production.

In other words, we pushed testing left, we pushed testing right and got rid of most of the testing in the middle. This is a departure from the past where most of the testing that was happening in the middle – integration style testing in the lab. The rest of the document describes #1 in a little bit more detail.

The Org Change for Quality Ownership

We did 'combined engineering' – a term used at Microsoft to indicate merging of responsibilities for dev and test in a single engineering role. It is not just an organization change where you bring the Dev and Test teams together. It is an actual discipline merge, with single Engineer role that has qualifications and responsibilities of the SDE and SDET disciplines of the past.

Everyone has a new role, everyone needs to learn new skills. This is a very important point. When we talk about combined engineering, a common question we get is how we trained the former testers. We had to train both ways. A former developer had to learn how to be a good tester and a former tester had to pick up good design skills. Managers had to learn how to manage end to end feature delivery.

In this model, there is no handoff to another person or team for testing. Each engineer owns E2E quality of the feature they build – from unit testing, to integration testing, to performance testing, to deployment, to monitoring live site and so on. Partnership with other engineers is still valued, even more so. There is now greater emphasis on peer reviews, design reviews, code reviews, test reviews etc. But the accountability for delivering a high quality feature is not diluted across multiple disciplines.

This was a big cultural shift across the company. This change happened first in one org, but then over a few years, every team across Microsoft moved to this model. There are some variations to this model but at this point there are no separate dev and test teams at Microsoft. They are just engineering teams with the combined engineer roles.

Making the change

How did we make this transition? In short, very carefully! We meticulously inventoried everything the test team did and refactored how all those activities would get done in the new world. We were particularly concerned about the test "dark matter" – things test team did that the developers either didn't know exist and didn't know how to do.

The second thing we were very clear about was we were going to have to change the way we test. If we continued to test the way we did before, we would fail in this new model. This is the shift-left, shift-right part of the test strategy.

We gave ourselves about 12 months to go through this transition. As mentioned earlier, both SDE and SDETs had the same basic qualification when they were hired. However, their skills had drifted over the years doing specialized work. The drift narrowed sprint by sprint, with each engineer picking up the skills of the other discipline, starting with a small feature and then growing in scope. Eventually one could not recognize an SDET or an SDE among the Engineers. That said, some engineers didn't quite make the transition and had to find other roles.

Specialization

A core principle of the Combined Engineering is elimination of tasks passing from one team to another.

Handoff introduce delays and dilutes accountability. There are no specialized central teams that do certain tasks. Each feature team has end to end accountability of delivering features.

At the same time, we understood the importance of specialized skills. Performance testing can be a specialized skill. Security testing can be a specialized skill. How do we accomplish such specialized tasks in the Combined Engineering model? We do this by forming virtual teams (v-teams) with specific engineers from each feature team asked to take on special roles in addition to the normal feature development role. In other words, we kept the specialization needed for some tasks but distributed that work into the feature teams from the central teams.

We created an Test Arch v-team and put some of our most senior engineers on it. They were responsible for building the new test frameworks and championing the changes through the org. We created another v-team with subject matter experts in quality tenets like Security and Accessibility. We created Performance v-team which identified common performance bottlenecks in the product and drove changes. Another v-team was responsible for monitoring the health of the CI pipeline and driving quick actions. Engineers in these special v-teams share expertise but their accountability is aligned to the feature team they work in.

Higher Velocity

On the surface it would appear that every engineer is now doing twice or more amount of work than before and therefore feature velocity would slow down. But total capacity in feature teams hasn't changed because of merging of dev and test teams. The move to combined engineering definitely required increased investment in training and development of new skills for each engineer, but it was more than offset by the efficiency gain from the reduced handoff and new testing practices. We have seen continuous and fairly noticeable gain in feature velocity since implementing this change.

As Partner Director of Engineering in Microsoft's Cloud and Enterprise division, Munil Shah leads engineering for Azure DevOps and TFS products. Munil has over 20 years of experience building large scale software and distributed services. Prior to his current role, he held various engineering leadership positions Bing Advertising and Windows groups at Microsoft. He is passionate about leading engineering teams through significant transformation to deliver successful solutions to customers.