How Microsoft does Quality Assurance (QA)

The Redmond Big Tech giant pioneered the SDET role in the 90s. It then retired it 2014. What happened and why?

GERGELY OROSZ
SEP 19, 2023

163 2 Share

Hi, this is Gergely with a bonus, free issue of the Pragmatic Engineer Newsletter. In every issue, I cover topics related to Big Tech and startups through the lens of engineering managers and senior engineers. In this article, we cover one out of sev topics from today's subscriber-only issue on How Big Tech does QA. To get full issue twice a week, subscribe:

***Programming note**: I'll be traveling for 3 weeks in November, heading over to the SF and NYC –, with less than usual time to write. I'm looking for guest writers to he out during this time. If you are an engineer or engineering manager with hands-on expertise to share, please indicate your interest here.*

Microsoft has played an outsized role in the development and importance of qu assurance across the industry. Microsoft was the first major company to come u with a specialized testing role which went well beyond manual testing. In this iss we cover:

- The SDET role
- Retiring of the SDET role

The SDET role

The SDET (Software Development Engineer in Test) role was one that Microsoft pioneered across the tech industry. They are *software engineers* who focused on writing automated tests; building and maintaining testing systems. The only difference between an SDET and a software development engineer (SDE) is that

SDETs didn't generally write production code: they wrote test code, working in t
same team as SDEs.

I could not trace the *exact* introduction of the role, but it was most likely in the 1990s. For example, here's a post from 2004 from a member of the Microsoft Exchange team explaining what it means to be an SDET in their organization:

> "An SDET is a developer who works in a test team and not a development tea
> The SDET has a keen sense of a tester yet loves writing code, and lots of it.
>
> The SDET enables the test team with the tools and processes that need to be
> place for the testers to do what they do best… test the living daylights out o
> product and find as many bugs as possible before it goes to the market.
>
> An SDET has the ability to analyze the functionality and architecture of a Proc
> and thus design and implement tools that help testers test it.
>
> The SDET enjoys short project life cycles, designs and implements many tools
> test frameworks in a single year, uses the latest technology and has plenty of
> room for innovation.
>
> Though product quality is a prime concern, the SDET doesn't have the stressf
> days that developers have during the end of a product life cycle. In layman's
> terms… an SDET back-side is rarely on the line :)"

Microsoft had a formal career path for the SDET role. From the same post:

> "There's plenty of room for growth in [the SDET position.] If you love doing w
> you are doing as an SDET then you can grow to become a Test Architect. If yo
> want to get involved in management, then you can progress towards becomi
> SDET Lead and then Test Manager.
>
> If you want to just code and not be involved with testing then you can take th
> path of becoming a developer. Many people take this path. If you realize that
> heart belongs to testing, then you can become a tester."

A 2:1 SDE:SDET ratio was common across Microsoft until around 2014. It was
case on my team at Skype for Xbox One, in 2012, when the team was formed. He
how our team was composed, based on headcount:

- 12x SDEs (software development engineers)
- 6x SDETs (software development engineer in test)
- 2x PMs (product managers)
- 1x EM (engineering manager)
- 1x SDET lead

On our project, the SDET team owned all parts of testing:

- Manually verifying that features devs built, worked as expected, including e cases we might have not considered
- Building integration and end-to-end tests to automate checks
- Creating manual tests plans, and executing before major milestones
- Being involved during the planning phase of features, bringing ideas on edg cases and how the work would be validated
- Building solutions for tricky problems, such as how to do reliable performan benchmarking for our Skype product on the Xbox hardware

Unit tests were a source of tension, early on. Who should write them? Several experienced developers came from gaming, where developers typically don't wr automated tests, and their view was that any automation – including unit tests – should be done by the SDET teams. *We cover more on how games are built in Gan development basics, and go very hands-on in the issue, Building a simple game.*

Those of us who'd previously built applications, or did test driven development (TDD,) felt that this approach was wrong, and that developers should write their unit tests because unit tests and the code are tightly coupled. I was in this camp.

Having dedicated SDETs made it a tempting option for developers to "outsource" the writing of unit tests. I'm just going to say it now: without an SI team, the question of who writes unit tests would have not been up for debate: developers would have *had* to write them. This is a recurring debate I've seen in every team with assigned SDETs. Surprisingly, even this year I heard of a Silicon Valley-based company where a developer team has the test team write unit test

In our case, we settled on developers writing unit tests, with the SDET team doir everything else. This approach worked well enough, but there were a few memorable features of this setup:

- **An "us and them" dynamic that created division.** When us developers finis a feature, we handed it over to an SDET, who usually found issues, so the fea came back to developers to fix. This felt annoying to devs, as it created work might have not accounted for. Over time, it started to feel like there were tv teams, with different goals, which didn't always row in the same direction.
- **Ticket ping-pong between dev and test.** I finish work on my feature and se over to test (ping). The tester finds a bug and sends it back to me the next d (pong). I fix the bug, and send it over again in a few days (ping). The tester fi another bug and sends it back to me… and this back-and-forth happened m times than I'd be willing to admit!
- **Devs were good at building complex systems, and could help SDET a lot.** SDET team had been building an integration testing system for months, and progress was slow. Our team really needed this system, as manual tests wer taking too long. Finally, one of the senior engineers proposed that devs join to help build this system, *as one team*. Two weeks later, with the lead of experienced devs, a system was up and running. This got me thinking; woulc team not work better without the dev/test division? We had just proved it di
- **The elephant in the room: some devs looked down on the SDET role.** Although not everybody did, it was clear that many devs regarded SDET wor less challenging than their own. SDETs also knew they could have better car options by switching to a dev role.

Well, it turns out they didn't have to wait that long for advancement.

Retiring the SDET role

Early 2014, I joined a new team called Skype for Web. This team was different fr most teams at Skype, as they shipped a new version of the software every day, r every month.

The team consisted of 6 SDEs and 3 SDETs, on paper. In reality, we were 9 SDEs, thanks to the engineering manager and the test lead who quietly decided it mac zero sense to have a dedicated test role, when we shipped new features every d write about this change that the team's leadership didn't broadcast, in the issue [Big Tech runs tech projects and the curious absence of Scrum](https://newsletter.pragmaticengineer.com):

> "When I joined the Skype for Web team, we initially did two-week sprints, anc followed the usual Scrum processes. We also had a split of software engineer and QA engineers. However, our shipping pace was every two weeks, but we wanted to ship more frequently.
>
> The first thing we did was make QA a part of engineering. In the "old world", a engineer would finish their work, check into their branch, update a ticket, anc the QA know it's ready for review. The QA would take this ticket a day or two later, review it, and reopen the ticket if they found issues. This was a long del
>
> We made a quiet, unofficial, change where all SDETs built production softwar well, and all software engineers became responsible for testing their own coc Now, we no longer had to wait days for feedback before shipping code to production. However, the bi-weekly sprints and the numerous Scrum rituals became the next problem."

We became a lot more productive by removing the SDET role from our team! SDETs still focused mainly on testing-related work, but also picked up developm tasks. Just as importantly, we paired a lot! I remember pairing with SDETs to bui feature. I was good at thinking about how to make something work, and the SDE was really good at pointing out edge cases I hadn't considered. When it came to debugging, the resourcefulness of SDETs surprised me.

On most teams across Microsoft, SDETs spent a lot of time manually testing thir
and writing integration tests. But on our team there was very little manual testi
and we all built integration testing *infrastructure,* and monitoring *infrastructure*.
When a developer or an SDET picked up a piece of work, they wrote all tests – ur
and integration – which made sense.

The best part of this change was that there was no more "us vs them." Argumen
ceased about whether to fix a bug which an SDET had discovered, as now we did
own testing and fixed bugs which we discovered, before shipping to production.

Web teams across Microsoft started to quietly remove the SDET role. Back ir
2014, our web team in the London Skype office felt 'special,' as the only other te
to merge the SDET function were web-based teams, of which there were not ma
On every other team, SDETs kept working the way they always had.

However, it wasn't only web teams within the Skype division which merged these
roles for better efficiency. Web teams independently came to the realization tha
merging SDET and dev roles made them move faster, and so this happened acro
of Microsoft!

**In the middle of 2014, Microsoft formally retired the SDET role and introduc
the SE role.** The inspiration was apparently a larger web team at Microsoft, Bing
[From Ars Technica](), in 2014:

> "At Bing, the task of creating programmatic tests was moved onto developers
> instead of dedicated testers. QA still exists and is still important, but it perfor
> end-user style "real world" testing, not programmatic automated testing. Thi
> testing has been successful for Bing, improving the team's ability to ship chan
> without harming overall software quality."

In July 2014, Microsoft announced that they will execute their largest layoffs to
date, letting go 18,000 staff of the 127,000 employees at the company. 12,500 c
the cuts were for the Nokia division. As part of this layoff, a large number of SDE
roles were also eliminated. This happened around the same time as the SDET rol
was announced to be retired, and existing SDETs needed to move over to the

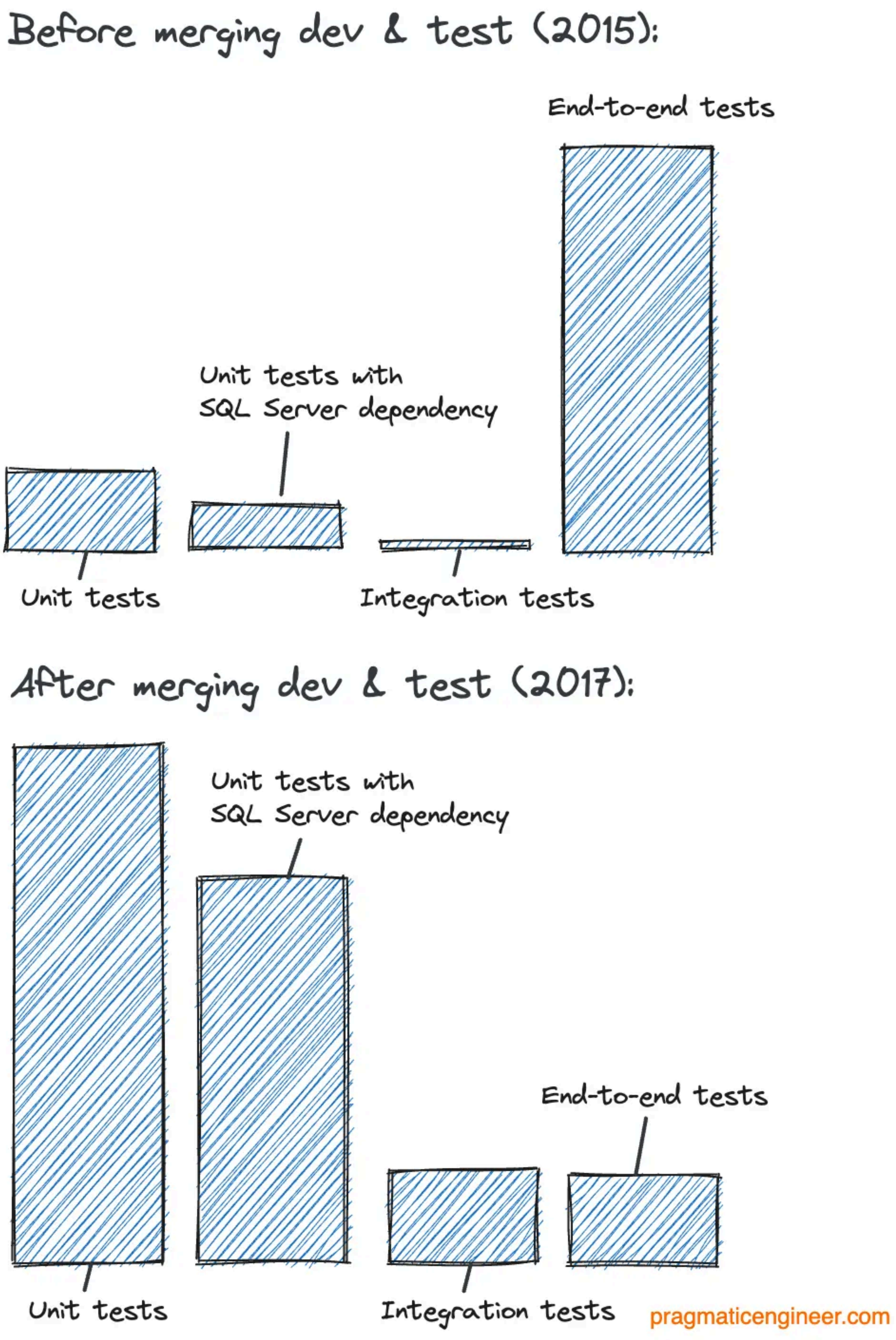
software development engineer (SDE) track over the next several months. The S
role was also renamed to SE – Software Engineer.

How did this transition work out? From what I gather, it went fine. The change
made a lot of sense for teams that ship on a daily basis. And teams within Micros
that ship weekly or monthly are increasingly rare, as Microsoft also leans into th
software-as-a-service (SaaS) model. Of course, Microsoft continues to be a vend
for the Windows operating system family, and the Surface tablet. These are botl
areas where the approach to quality needs to be different to that of SaaS produ

A good account of the change came from the Visual Studio Team Services team i
2017, three years after this change. Reflecting on it, Brian Harry — currently
Technical Fellow at Microsoft — wrote:

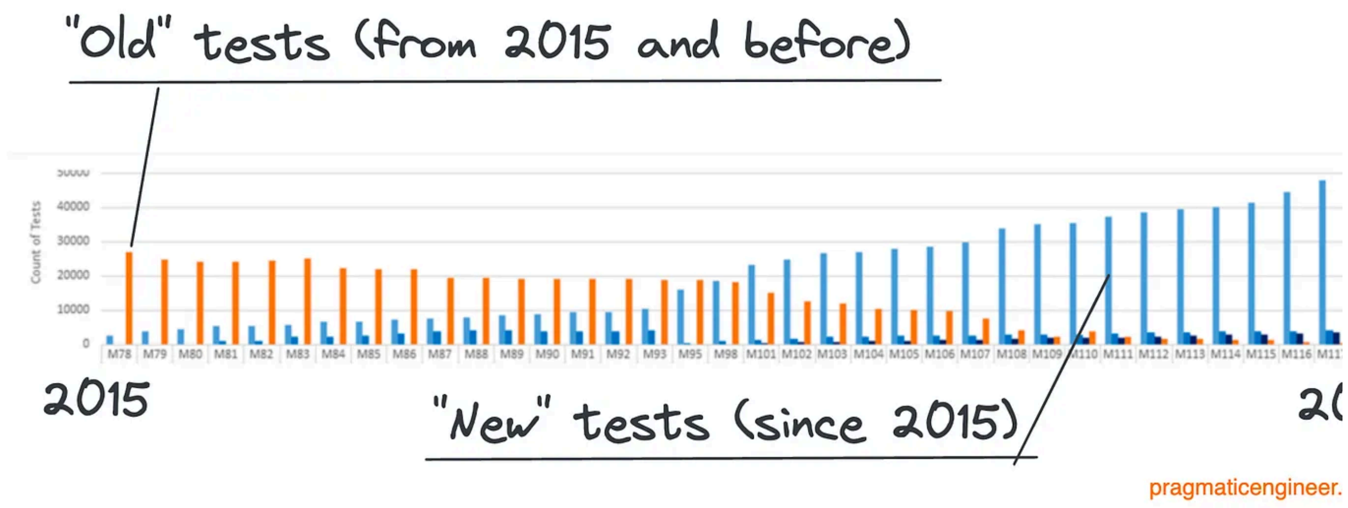
> "Two years ago [in 2015], we had 10's of thousands of tests. They were writte
> 'testers' to test code written by 'developers'. While there were some advanta
> of this model – like clearly measurable and controllable investment in test,
> expertise and career growth in the testing discipline, etc., there were also ma
> disadvantages – lack of accountability on the developers, slow feedback cycle
> (introduce bug, find bug, fix bug), developers had little motivation to make th
> code "testable", divergence between code architecture and test architecture
> made refactoring and pivoting very hard/expensive, and more. (...)
>
> Full testing would take the better part of a day to run, many more hours to
> "analyze the results" to identify false failures, and days or weeks to repair all
> tests that were broken due to some legitimate change in the product. (...) Two
> years ago, we started on a path to completely redo testing.
>
> **We combined the dev and test orgs into a consolidated 'engineering' org.**
> the most part, we eliminated the distinction between people who code and
> people who test. That's not to say every person does an identical amount of e
> but every person does some of everything and is accountable for the quality
> what they produce. We also set out to completely throw away our 10's of
> thousands of tests that took 8 years to create, and replace them with new tes
> that were done completely differently."

This team took stock of the type of tests they had in place and decided they didr
like that there were few small unit tests, but lots of complex and hard to mainta
end-to-end tests. So they changed this:



The change in the number and type of tests the Visual Studio Team Services experienced after merging the dev and the test teams. Before the merge: end-to-end tests dominated, but unit and integration tests were rare. This flipped after the merge. Data source: Microsoft Dev Blogs

Here's another visualization on how the team's tests changed over a two-year timeframe:



In 2 years, almost all "old" tests from when test was separate from dev, were gone. The new tests became more granular as well. Data source: [Microsoft Dev Blogs]

So, was all this work worth it in the end? According to Brian, yes it was. Writing a time, he said:

> "We are starting to reap the benefits in improved quality, agility and enginee satisfaction."

We covered the QA approach at one out of seven Big Tech companies from the article [How Big Tech does QA]. In the full edition, we cover additional details for:

1. Google
2. Meta
3. Apple
4. Amazon
5. Uber
6. Netflix

Read the full issue here.

Related issues, which overlap with testing:

- How Big Tech runs tech projects and the curious absence of Scrum. How pro are organized does have an impact on how straightforward – or not – it is to manual testing on a regular cadence
- Healthy oncall practices – while not all Big Tech companies have a dedicated function, engineers are almost always oncall, and this forces some more foc on quality
- Shipping to production
- Dealing with a low-quality engineering culture at Big Tech
- What is a senior software engineer at Big Tech?

Discussion about this post

Comments Restacks

Write a comment...