Microsoft

February 28, 2019 — 8:30 AM - 5:00 PM | Detroit, Michigan

How Microsoft Engineering uses Azure DevOps

Learn.

Connect.

Explore.

How Microsoft Engineering uses Azure DevOps

MS Engineering uses 1ES to build Azure DevOps, Windows 10, Visual Studio, Bing…etc.

…using Azure DevOps

…building 146,000 times per day

…deploying 85,000 times per day

… for millions of users + most of Microsoft.

Before Azure DevOps


1ES

One Engineering System ("1ES")

There cannot be a more important thing for an engineer, for a product team, than to work on the systems that drive our productivity.

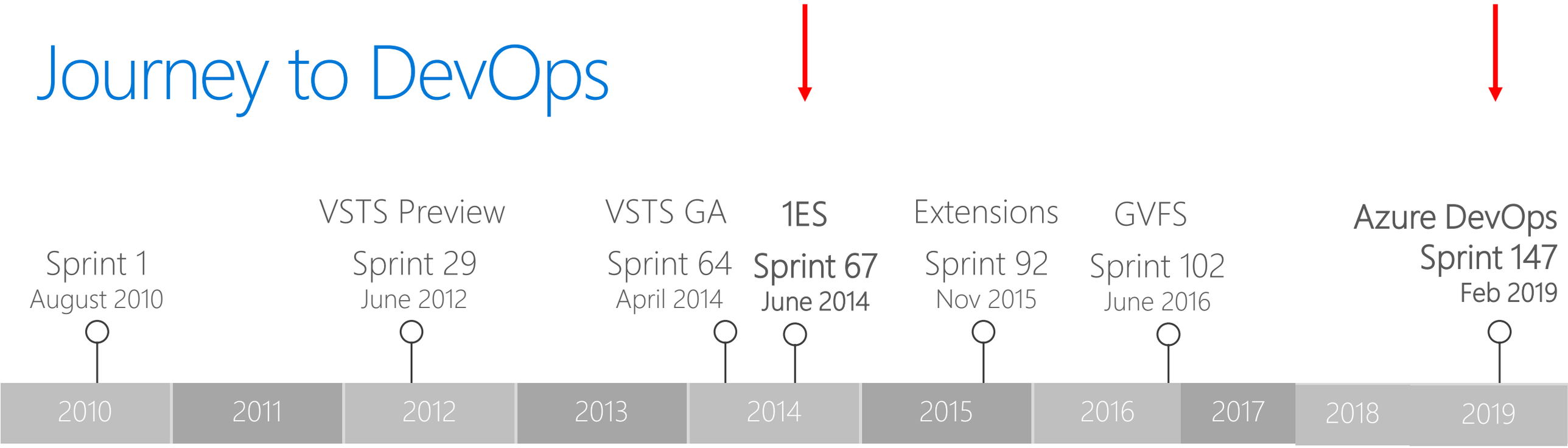
So I would, any day of the week, trade off features for our own productivity.

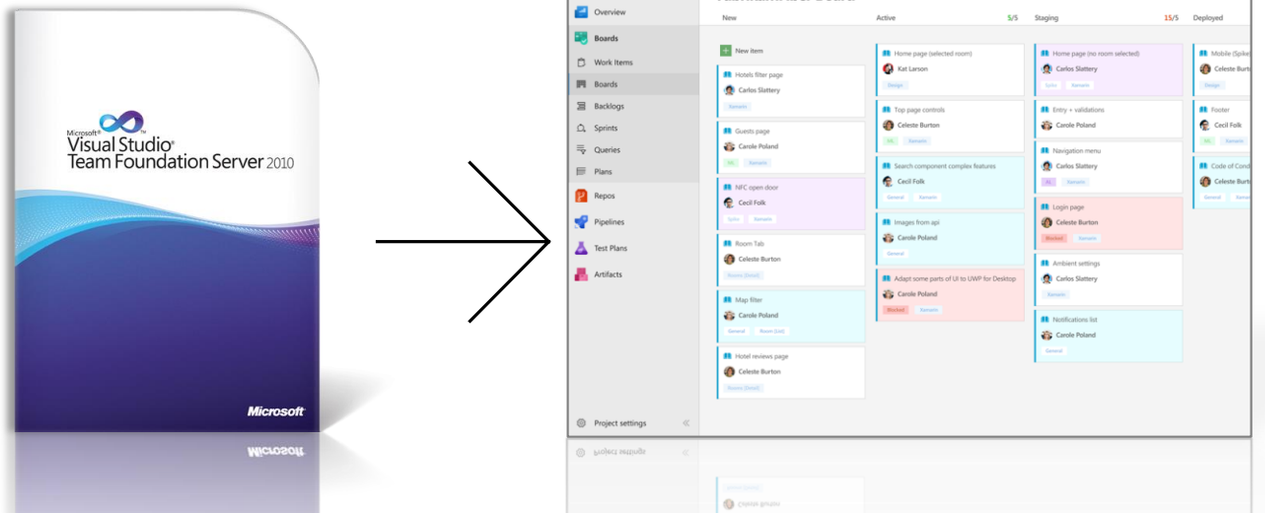
I want our best engineers to work on our engineering systems, so that we can later on come back and build all of the new concepts we want.

- Satya Nadella

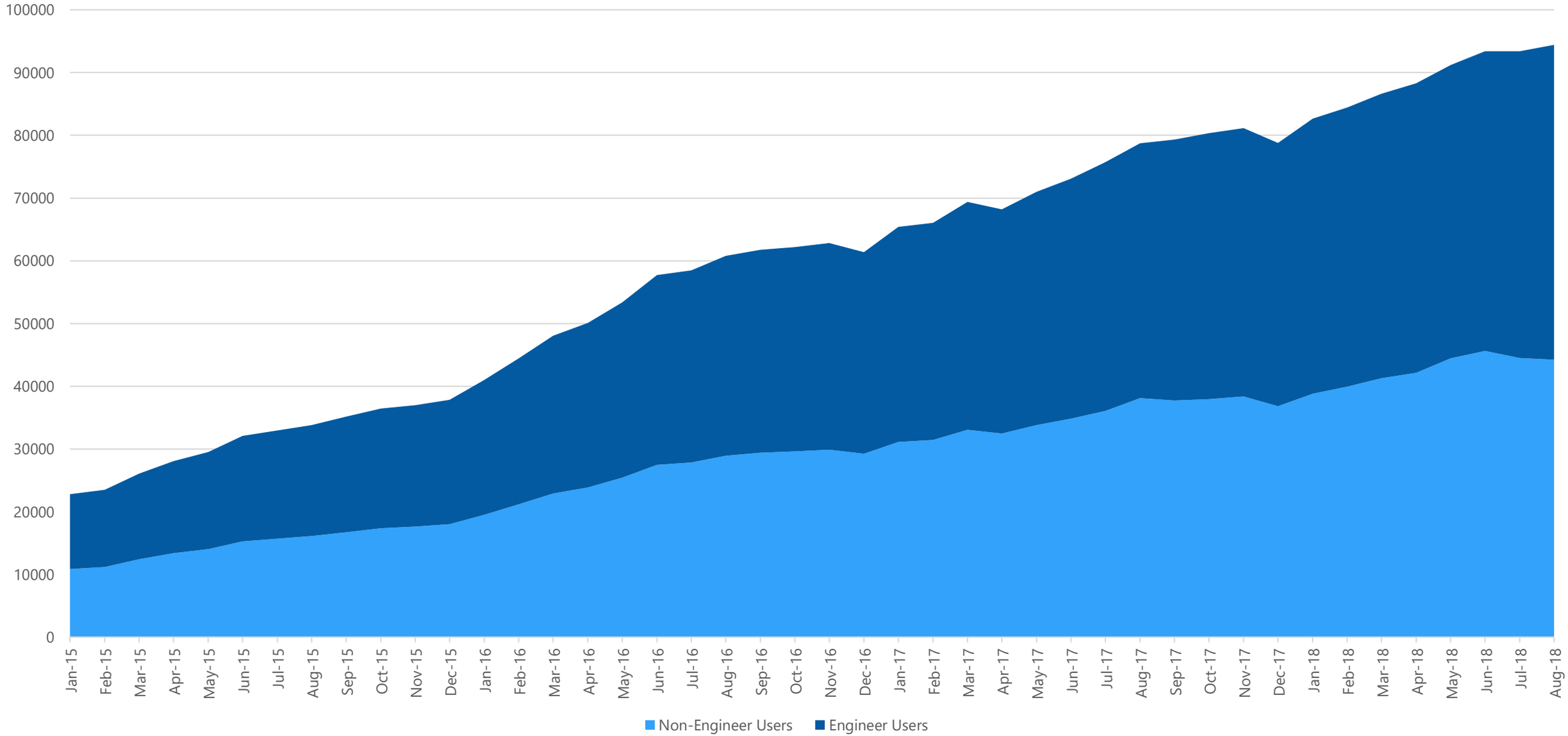
One Engineering System with Azure DevOps

Journey to DevOps





Azure DevOps: Millions of users + Most of Microsoft



Data: Internal Microsoft engineering system activity, September 2018

DevOps in Microsoft

Azure DevOps is the toolchain of choice for Microsoft engineering with over 96,000 internal users

https://aka.ms/DevOpsAtMicrosoft

372k
Pull Requests per month

4.4m
Builds per month

5m
Work items viewed per day

85,000
Deployments per day

2m
Git commits per month

500m
Test executions per day

500k
Work items updated per day

Data: Internal Microsoft engineering system activity, November 2018

One Engineering System with Azure DevOps

Azure Networking

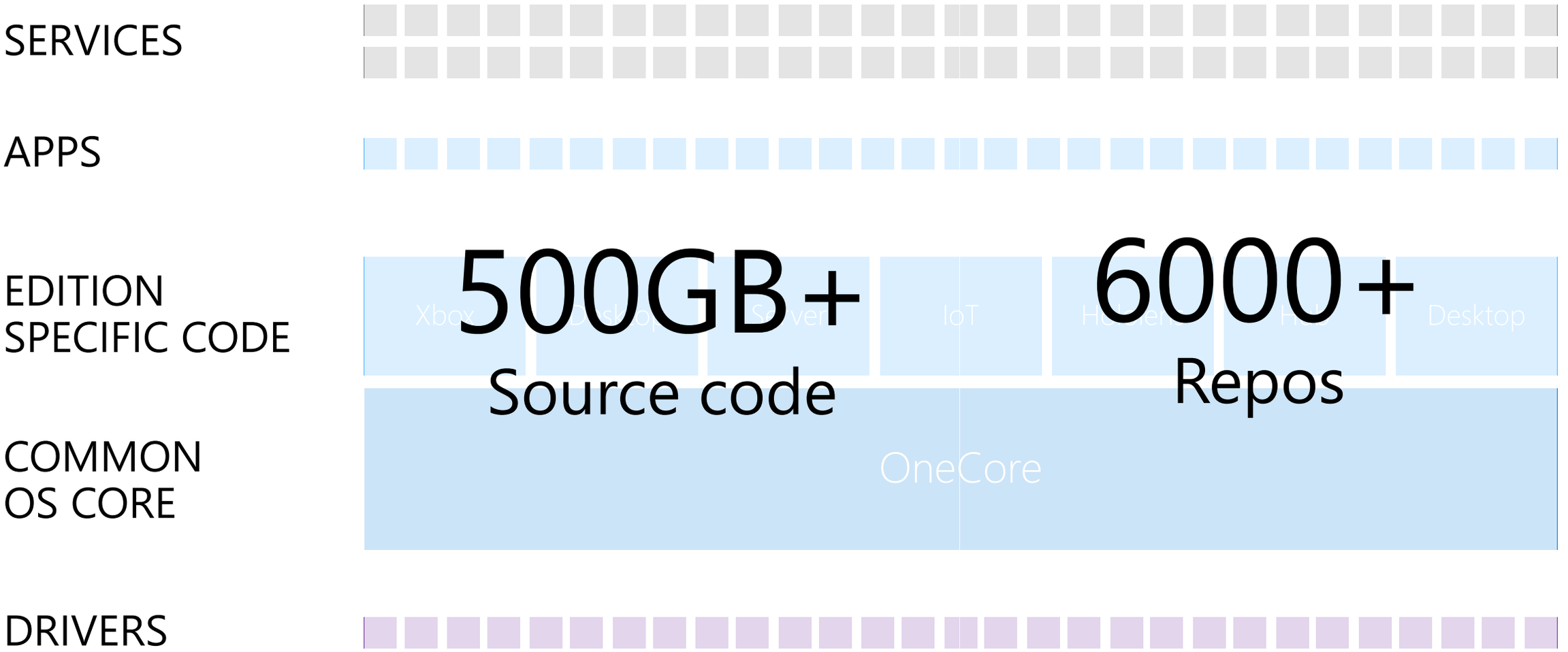
Visual Studio Code

Bing


Windows

Windows 10

Software Products



Five habits we've learned so far

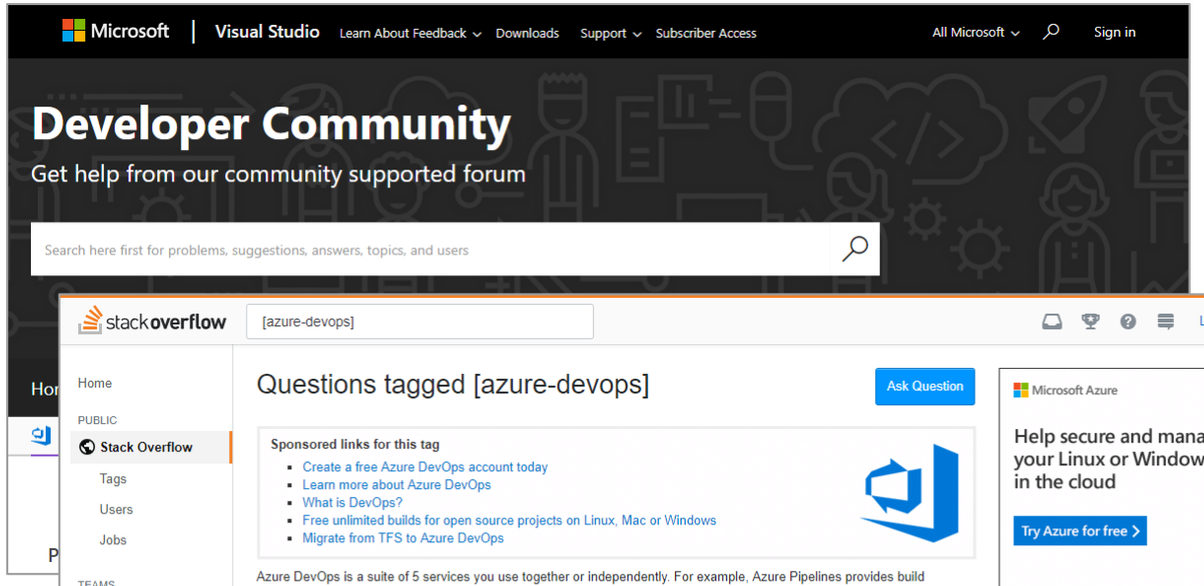
- Customer Obsessed
- Production First Mindset
- Team Autonomy + Enterprise Alignment
- Shift Left Quality
- Infrastructure as Flexible Resource

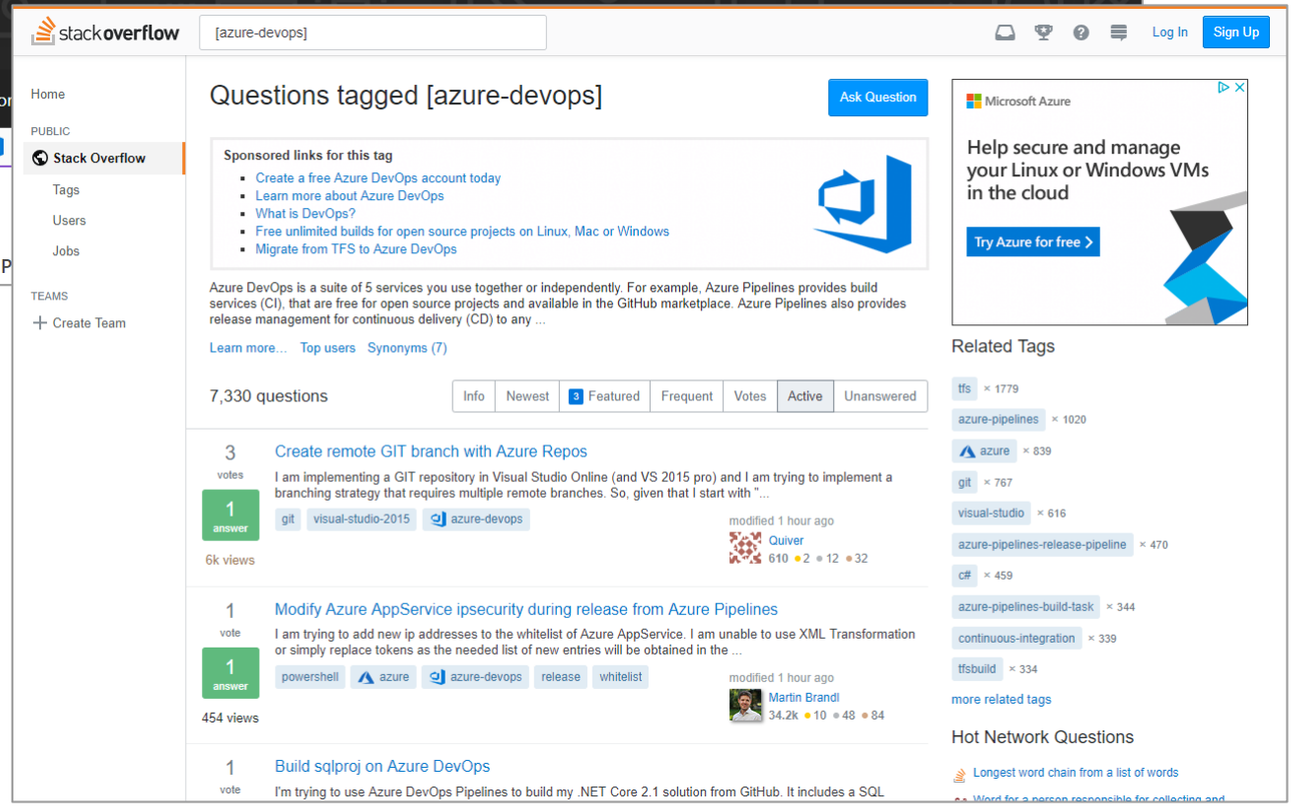
Five habits we've learned so far

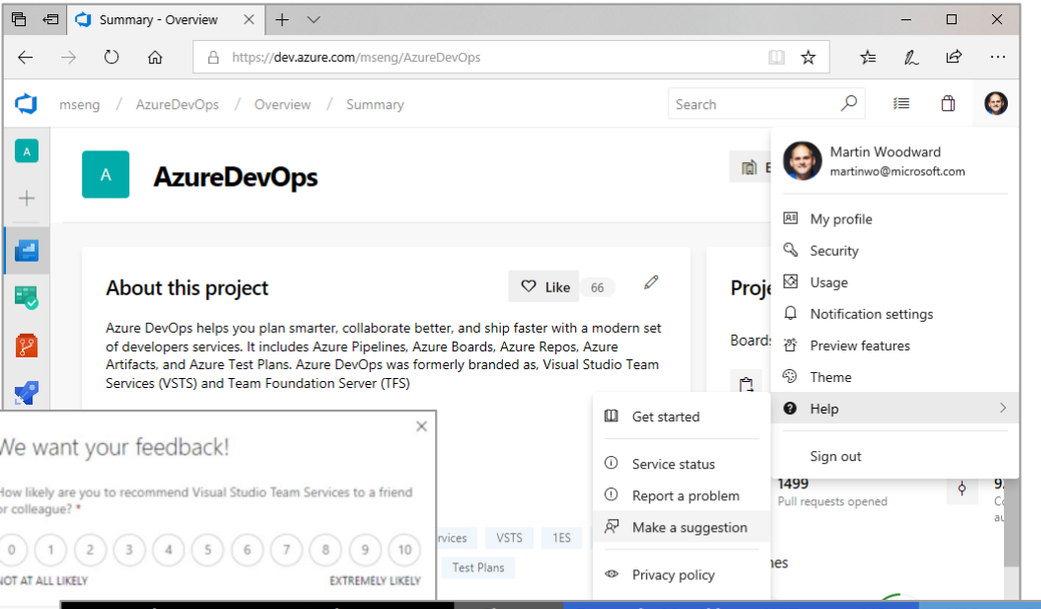
- Customer Obsessed
- Production First Mindset
- Team Autonomy + Enterprise Alignment
- Shift Left Quality
- Infrastructure as Flexible Resource

Listen to our customers

Quantitively & Qualitatively



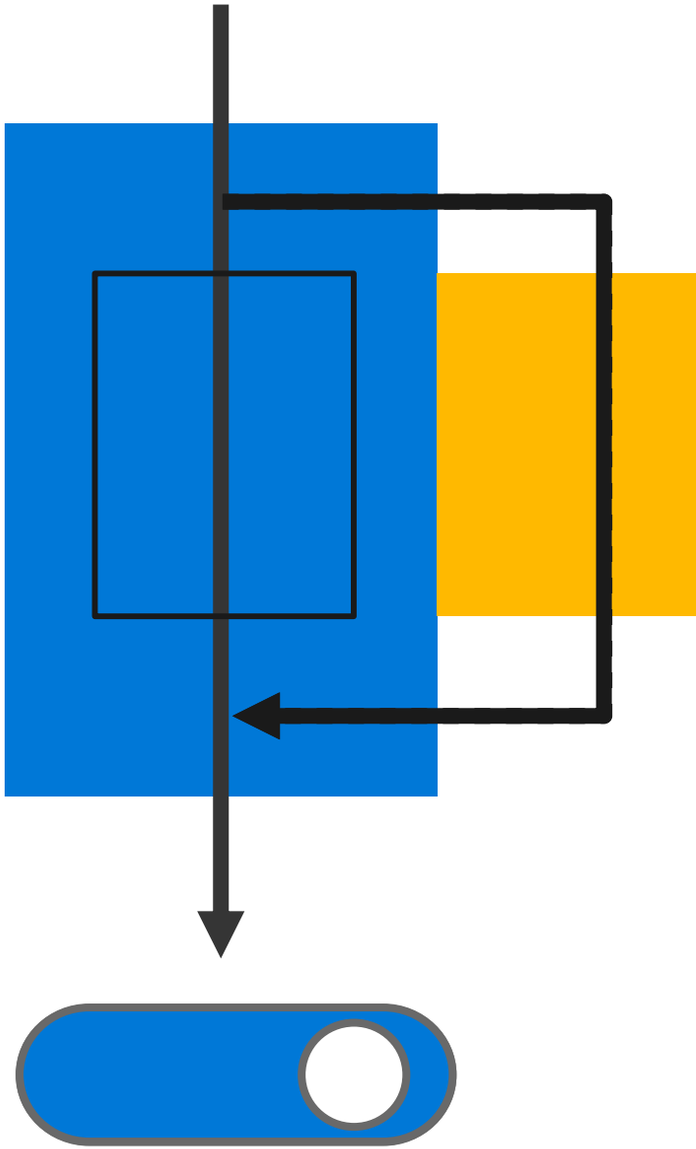




Location	Business	Champ	Engaged Users	Monthly Δ	WIT	VC	NPS	Resp.
Dublin, Ireland	Professional services	aaronha	7303	335	56%	56%	32%	204
London, England	Professional services	midenn	5216	363	62%	46%	28%	164
Norway, NL, Houstor	Oilfield services	samgu	4242	195	64%	54%	2%	203
New York, NY	Professional services	trevorc	3973	423	60%	55%	17%	195
Amsterdam, NL	Oil & gas	jeffbe	3848	94	56%	35%	8%	167
New York, NY	Financial information / ana	amitgup	3197	409	72%	32%	20%	154
Medellin, Colombia	Commercial banking	mariorod	2682	24	80%	31%	-11%	90
Peoria, IL	Heavy equipment manufac	abarr	2644	137	57%	42%	23%	137
London, England	Oil & gas	rajr	2197	209	54%	34%	-8%	62
Princeton, NJ	Reinsurance	hdixon	1768	97	61%	27%	-20%	81
London, England	Professional services	jsharma	1668	376	48%	45%	40%	45
San Ramon, CA	Oil & gas	puagarw	1667	342	47%	42%	40%	48
Utrecht, NL	Insurance	saumyav	1554	66	73%	23%	-38%	16
Gothenberg, Swede	Automotive	sadar	1455	97	61%	41%	20%	46
Seattle, WA	IT Consulting	aaronha	1408	136	35%	43%	55%	47
Raleigh, NC	Computer assisted legal res	roferg	1392	11	38%	61%	21%	47
Liberty Lake, WA	Energy and water resource	buckh	1377	107	68%	49%	-2%	66
Seattle, WA	Freight forwarding service	midenn	1327	11	59%	0%	-52%	27
Amsterdam, NL	Financial services	shasb	1316	298	50%	30%	3%	38
Bentonville, AR	Retail	buckh	1304	18	47%	52%	-3%	34
Amsterdam, NL	Information Services	fraga	1251	35	56%	61%	10%	40
Auckland, NZ	Telecommunications	atinb	1249	-8	73%	15%	19%	37
Palo Alto, CA	Computer hardware & soft	mariorod	1242	78	64%	17%	5%	59
Lowell, AR	Trucking & transport	gauravsi	1226	57	77%	74%	40%	5
Charleston, SC	Software publishing	chrispat	1165	31	63%	60%	-28%	47

Feature Flags

- All code is deployed, but feature flags control exposure
  - Reduces integration debt
- Flags provide runtime control down to individual user
- Users can be added or removed with no redeployment
- Mechanism for progressive experimentation & refinement
- Enables dark launch

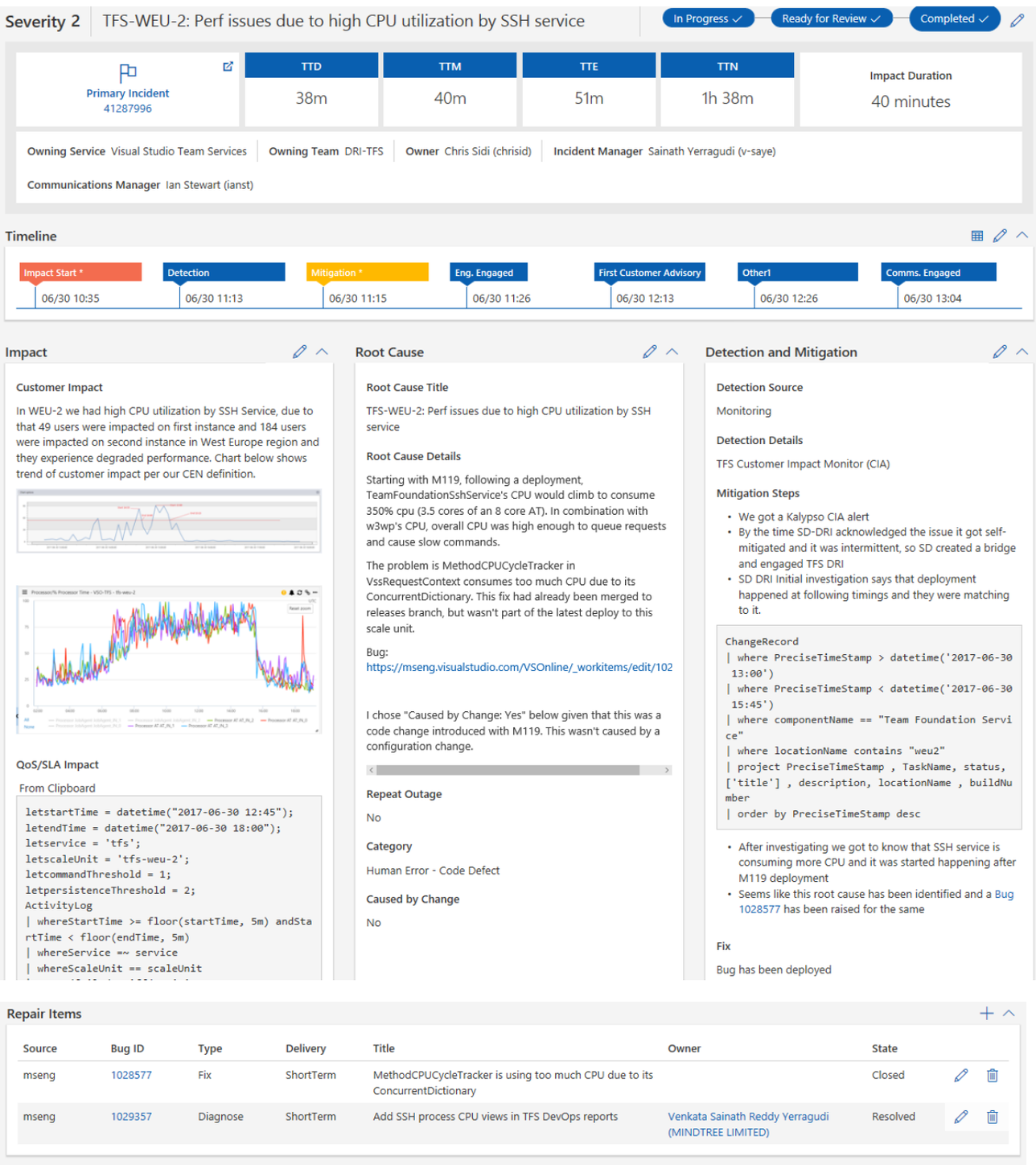


Five habits we've learned so far

- Customer Obsessed
- **Production First Mindset**
- Team Autonomy + Enterprise Alignment
- Shift Left Quality
- Infrastructure as Flexible Resource

Live Site Incidents

- On detection, LSI conference bridge created
- DRI's brought in to call
- Communication externally and internally
- Gather data for repair items & mitigate for customers
- Every action recorded
- Plan to rotate people during long running LSI's
- Create & track repair Items to prevent reoccurrence and improve detection time



Severity 2 | TFS-WEU-2: Perf issues due to high CPU utilization by SSH service

In Progress ✓ | Ready for Review ✓ | Completed ✓

Primary Incident 41287996

TTD	TTM	TTE	TTN	Impact Duration
38m	40m	51m	1h 38m	40 minutes

Owning Service Visual Studio Team Services | **Owning Team** DRI-TFS | **Owner** Chris Sidi (chrisid) | **Incident Manager** Sainath Yerragudi (v-saye)

Communications Manager Ian Stewart (ianst)

Timeline

Impact Start *	Detection	Mitigation *	Eng. Engaged	First Customer Advisory	Other1	Comms. Engaged
06/30 10:35	06/30 11:13	06/30 11:15	06/30 11:26	06/30 12:13	06/30 12:26	06/30 13:04

Impact

Customer Impact

In WEU-2 we had high CPU utilization by SSH Service, due to that 49 users were impacted on first instance and 184 users were impacted on second instance in West Europe region and they experience degraded performance. Chart below shows trend of customer impact per our CEN definition.

QoS/SLA Impact

From Clipboard

```
letstartTime = datetime("2017-06-30 12:45");
letendTime = datetime("2017-06-30 18:00");
letservice = 'tfs';
letscaleUnit = 'tfs-weu-2';
letcommandThreshold = 1;
letpersistenceThreshold = 2;
ActivityLog
| whereStartTime >= floor(startTime, 5m) andSta
rtTime < floor(endTime, 5m)
| whereService =~ service
| whereScaleUnit == scaleUnit
```

Root Cause

Root Cause Title

TFS-WEU-2: Perf issues due to high CPU utilization by SSH service

Root Cause Details

Starting with M119, following a deployment, TeamFoundationSshService's CPU would climb to consume 350% cpu (3.5 cores of an 8 core AT). In combination with w3wp's CPU, overall CPU was high enough to queue requests and cause slow commands.

The problem is MethodCPUCycleTracker in VssRequestContext consumes too much CPU due to its ConcurrentDictionary. This fix had already been merged to releases branch, but wasn't part of the latest deploy to this scale unit.

Bug: https://mseng.visualstudio.com/VSOnline/_workitems/edit/102

I chose "Caused by Change: Yes" below given that this was a code change introduced with M119. This wasn't caused by a configuration change.

Repeat Outage

No

Category

Human Error - Code Defect

Caused by Change

No

Detection and Mitigation

Detection Source

Monitoring

Detection Details

TFS Customer Impact Monitor (CIA)

Mitigation Steps

- We got a Kalypso CIA alert
- By the time SD-DRI acknowledged the issue it got self-mitigated and it was intermittent, so SD created a bridge and engaged TFS DRI
- SD DRI Initial investigation says that deployment happened at following timings and they were matching to it.

```
ChangeRecord
| where PreciseTimeStamp > datetime('2017-06-30
 13:00')
| where PreciseTimeStamp < datetime('2017-06-30
 15:45')
| where componentName == "Team Foundation Servi
ce"
| where locationName contains "weu2"
| project PreciseTimeStamp , TaskName, status,
['title'] , description, locationName , buildNu
mber
| order by PreciseTimeStamp desc
```

- After investigating we got to know that SSH service is consuming more CPU and it was started happening after M119 deployment
- Seems like this root cause has been identified and a Bug 1028577 has been raised for the same

Fix

Bug has been deployed

Repair Items

Source	Bug ID	Type	Delivery	Title	Owner	State
mseng	1028577	Fix	ShortTerm	MethodCPUCycleTracker is using too much CPU due to its ConcurrentDictionary		Closed
mseng	1029357	Diagnose	ShortTerm	Add SSH process CPU views in TFS DevOps reports	Venkata Sainath Reddy Yerragudi (MINDTREE LIMITED)	Resolved

Live Site Culture

- Live site status is always the top priority
- Weekly live site review
- Closing LSI requires listing repair items
- LSI repair items in backlog (2 sprint rule)
- Actionable alerts
- Monthly service review
- On-call Designated Responsible Individual (DRI)
- Customer Obsessed Availability model (SLA)
- Per team / service health reports

Be Transparent

Visual Studio Team Services is up and running

✔ Everything is looking good

View all Team Services support options Visit our service blog for details and histon

A Rough Patch

Brian Harry MS 25 Nov 2013 3:06 PM 10

Either I'm going to get increasingly good at apologizing to fewer and fewer people or we're going to get better at this. I vote for the later.

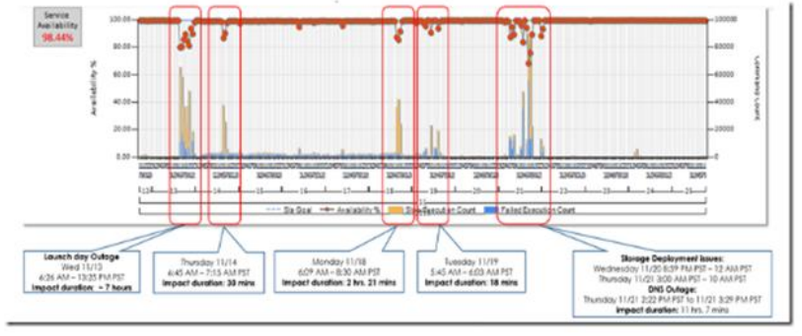
We've had some issues with the service over the past week and a half. I feel terrible about it and I can't apologize enough. It's the biggest incident we've had since the instability created by our service refactoring in the March/April timeframe. I know it's not much consolation but I can assure you that we have taken the issue very seriously and there are a fair number of people on my team who haven't gotten much sleep recently.

The incident started the morning of the Visual Studio 2013 launch when we introduced some significant performance issues with the changes we made. You may not have noticed it by my presentation but for the couple of hours before I was frantically working with the team to restore the service.

At launch, we introduced the commercial terms for the service and enabled people to start paying for usage over the free level. To follow that with a couple of rough weeks is leaving a bad taste in my mouth (and yours too, I'm sure). Although the service is still officially in preview, I think it's reasonable to expect us to do better. So, rather than start off on such a sour note, we are going to extend the "early adopter" program for 1 month giving all existing early adopters an extra month at no charge. We will also add all new paying customers to the early adopter program for the month of December – giving them a full month of use at no charge. Meanwhile we'll be working hard to ensure things run more smoothly.

Hopefully that, at least, demonstrates that we're committed to offering a very reliable service. For the rest of this post, I'm going to walk through all the things that happened and what we learned from them. It's a long read and it's up to you how much of it you want to know.

Here's a picture of our availability graph to save 1,000 words:



Explanation of July 18th outage

Brian Harry MS 31 Jul 2014 5:58 AM 6

RATE THIS

Sorry it took me a week and a half to get to this.

We had the most significant VS Online outage we've had in a while on Friday July 18th. The entire service was unavailable for about 90 minutes. Fortunately it happened during non-peak hours so the number of affected customers was fewer than it might have been but I know that's small consolation to those who *were* affected.

My main goal from any outage that we have is to learn from it. With that learning, I want to make our service better and also share it so, maybe, other people can avoid similar errors.

What happened?

The root cause was that a single database in SQL Azure became very slow. I actually don't know why, so I guess it's not really the root cause but, for my purposes, it's close enough. I trust the SQL Azure team chased that part of the root cause – certainly did loop them in on the incident. Databases will, from time to time, get slow and SQL Azure has been pretty good about that over the past year or so.

The scenario was that Visual Studio (the IDE) was calling our "Shared Platform Services" (a common service instance managing things like identity, user profiles, licensing, etc.) to establish a connection to get notified about updates to roaming settings. The Shared Platform Services were calling Azure Service Bus and it was calling the ailing SQL Azure database.

The slow Azure database caused calls to the Shard Platform Services (SPS) to pile up until all threads in the SPS thread pool were consumed, at which point, all calls to TFS eventually got blocked due to dependencies on SPS. The ultimate result was VS Online being down until we manually disabled our connection to Azure Service Bus an the log jam cleared itself up.

There was a lot to learn from this. Some of it I already knew, some I hadn't thought about but, regardless of which category it was in, it was a damn interesting/enlightening failure.

UPDATE Within the first 10 minutes I've been pinged by a couple of people on my team pointing out that people may interpret this as saying the root cause was Azure DB. Actually, the point of my post is that it doesn't matter what the root cause was. Transient failures will happen in a complex service. The interesting thing is that you react to them appropriately. So regardless of what the trigger was, the "root cause" of the outage was that we did not handle a transient failure in a secondary service properly and allowed it to cascade into a total service outage. I'm also told that I may be wrong about what happened in SB/Azure DB. I try to stay away from saying too much about what happens in other services because it's a dangerous thing to do from afar. I'm not going to take the time to go double check and correct any error because, again, it's not relevant to the discussion. The post isn't about the trigger. The post is about how we reacted to the trigger and what we are going to do to handle such situations better in the future.

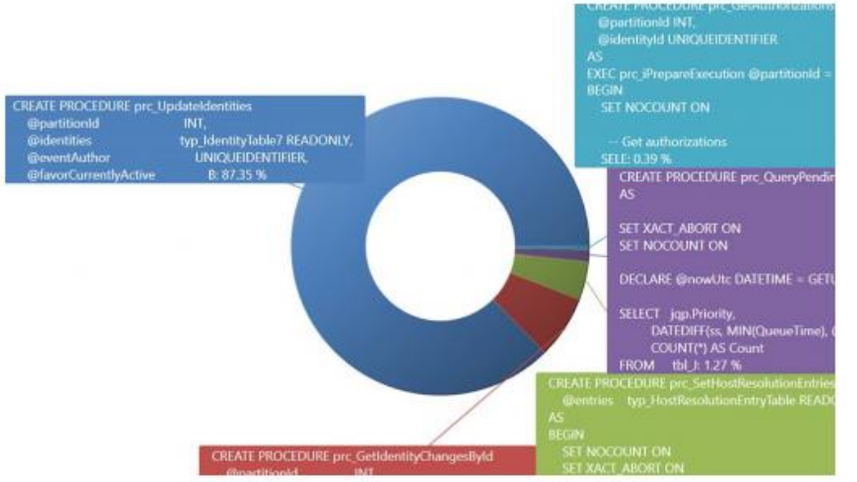
Don't let a 'nice to have' feature take down your mission critical ones

I'd say the first and foremost lesson is "Don't let a 'nice to have' feature take down your mission critical ones." There's a notion in services that all services should be loosely coupled and failure tolerant. One service going down should not cause a cascading failure, causing other services to fail but rather only the portion of functionality that absolutely depends on the failing component is unavailable. Services like Google and Bing are great at this. They are composed of dozens or hundreds of services and any single service might be down and you never even notice because most of the experience looks like it always does.

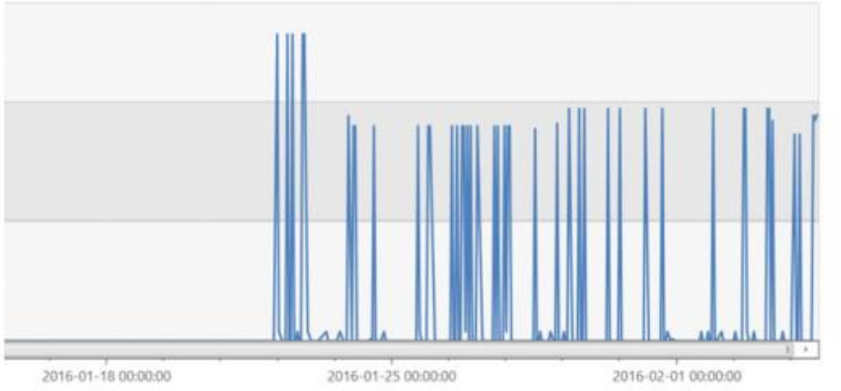
A bit more on the Feb 3 and 4 incidents

02/06/2016 by Brian Harry MS // 15 Comments

Drilling further by looking at what sprocs are waiting on RESOURCE_SEMAPHORE, we see that prc_UpdateIdentities dominates. Guess what... That's the sproc that caused this incident.

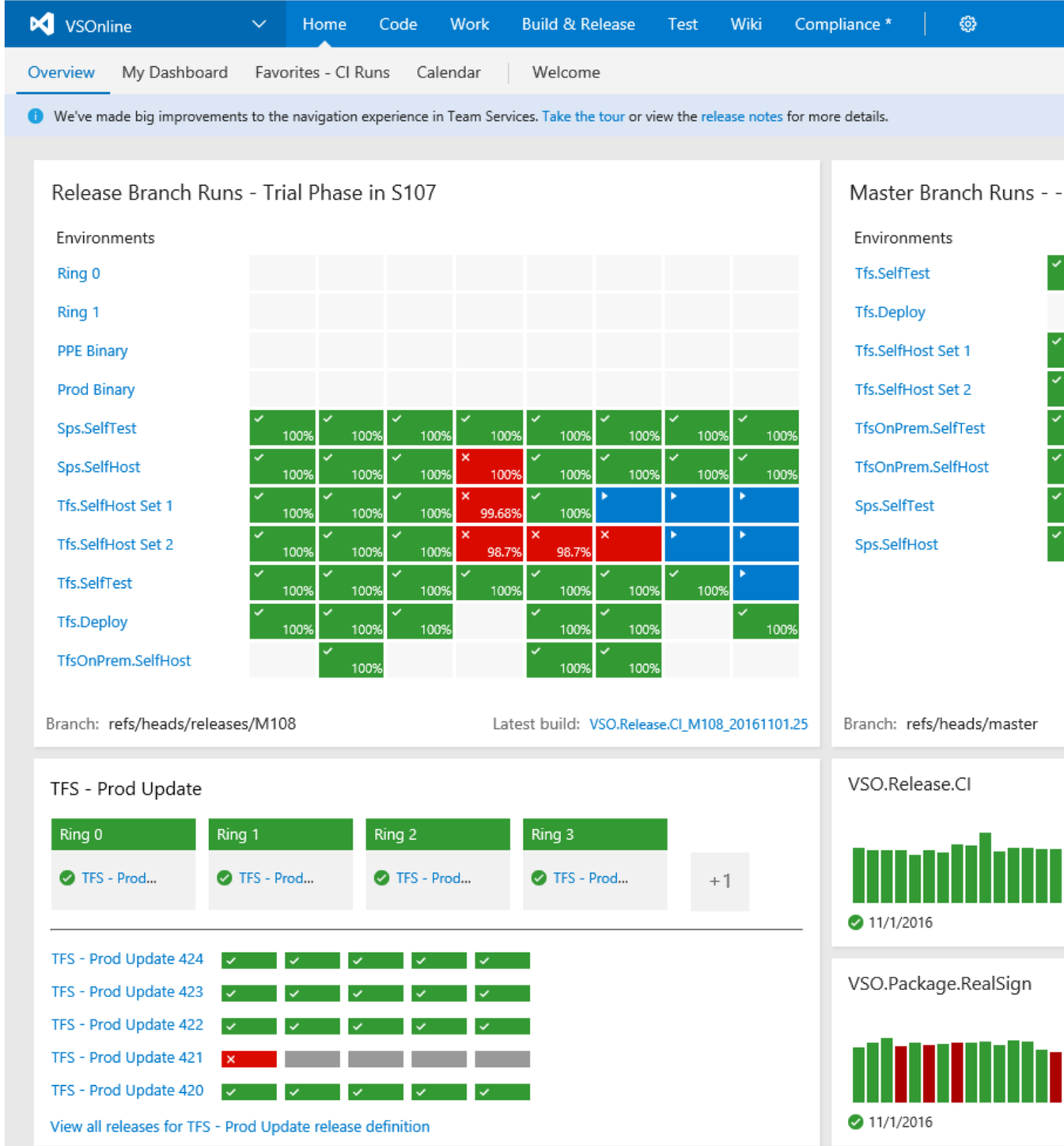


And now, let's look at a time chart of memory grant requests for this sproc. The huge spikes begin the moment we introduced the change to SQL compat level. This is a fantastic opportunity for automated anomaly detection. There's no reason we can't find this kind of thing long before it creates any actual incident. Getting all of the technology hooked up to make this possible and know which KPIs to watch isn't easy and will take some tuning but all the data is here.



Automate completely

- No more “one time” commands run manually
- Every command goes in PowerShell scripts that are checked in
- Deployment to pre-production & canary is the same as deployment to production **every time**
- All orchestrated with Release Management in VSTS



Security Mindset - Assume Breach

Started with war games to the learn attacks and practice response

VS.

- Initially double-blind test
- Over time, eliminated blue team

Our defenders need to be our defenders

Shifted left to prevent top risks

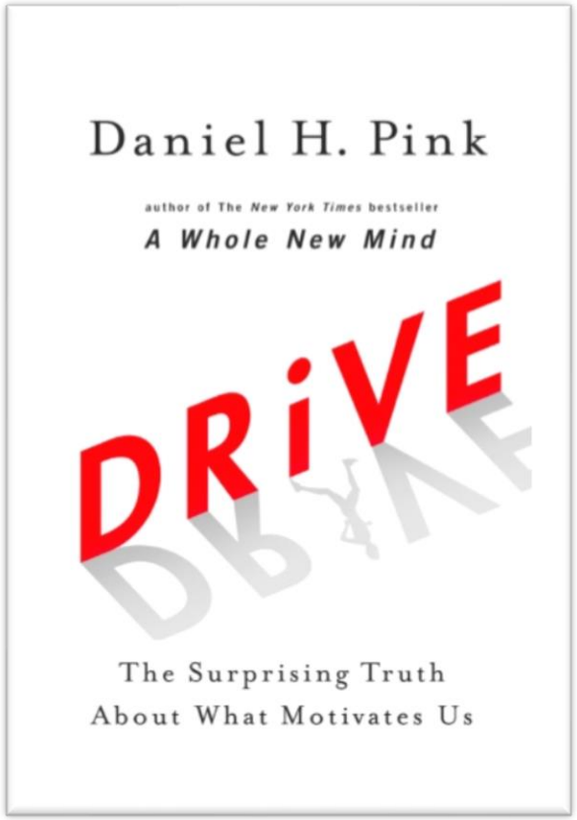
- Credential theft
- Secret leakage
- OSS vulnerabilities

Five habits we've learned so far

- Customer Obsessed
- Production First Mindset
- **Team Autonomy + Enterprise Alignment**
- Shift Left Quality
- Infrastructure as Flexible Resource

Agile at Scale with Aligned Autonomy

"Let's try to give our teams three things….
Autonomy, Mastery, Purpose"



Organization
Roles
Teams
Cadence
Taxonomy

Alignment

Plan
Practices

Autonomy

ORG CHART

PROGRAM MANAGEMENT

DEVELOPMENT

TESTING

ORG CHART

PROGRAM MANAGEMENT

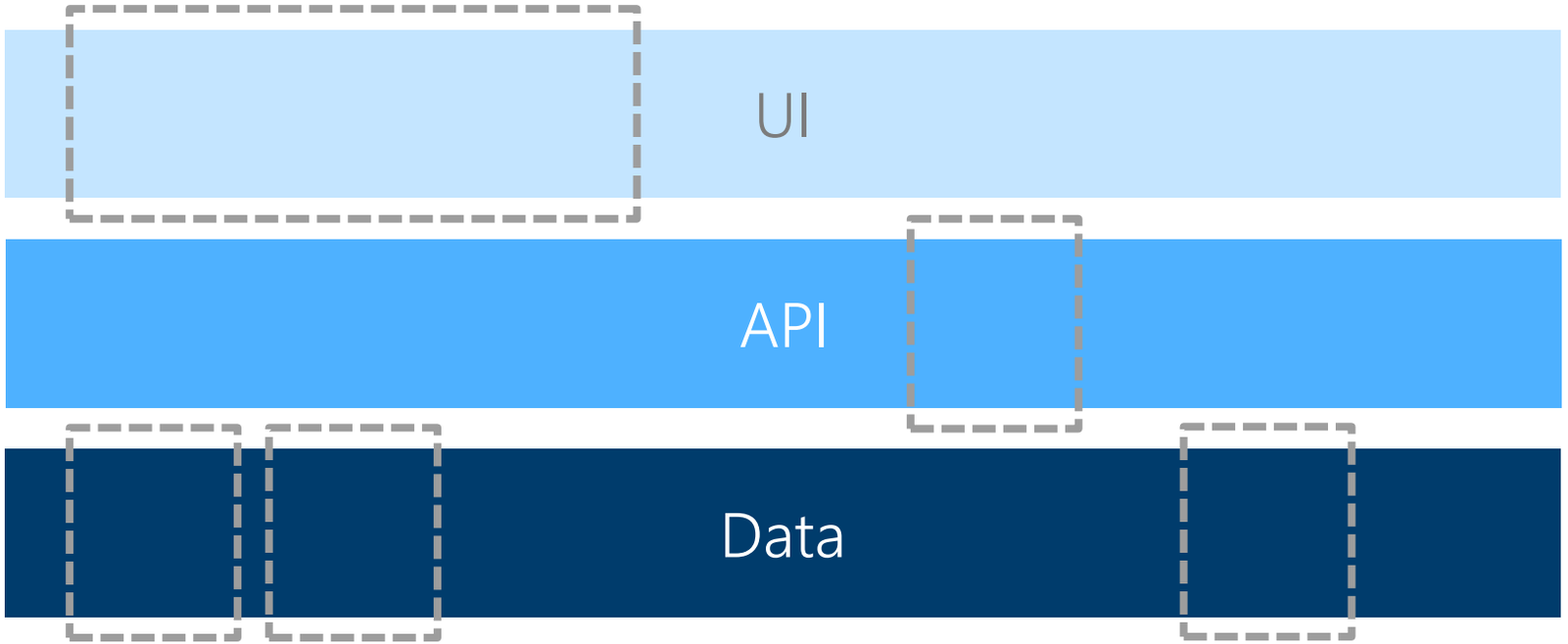
ENGINEERING

OPs

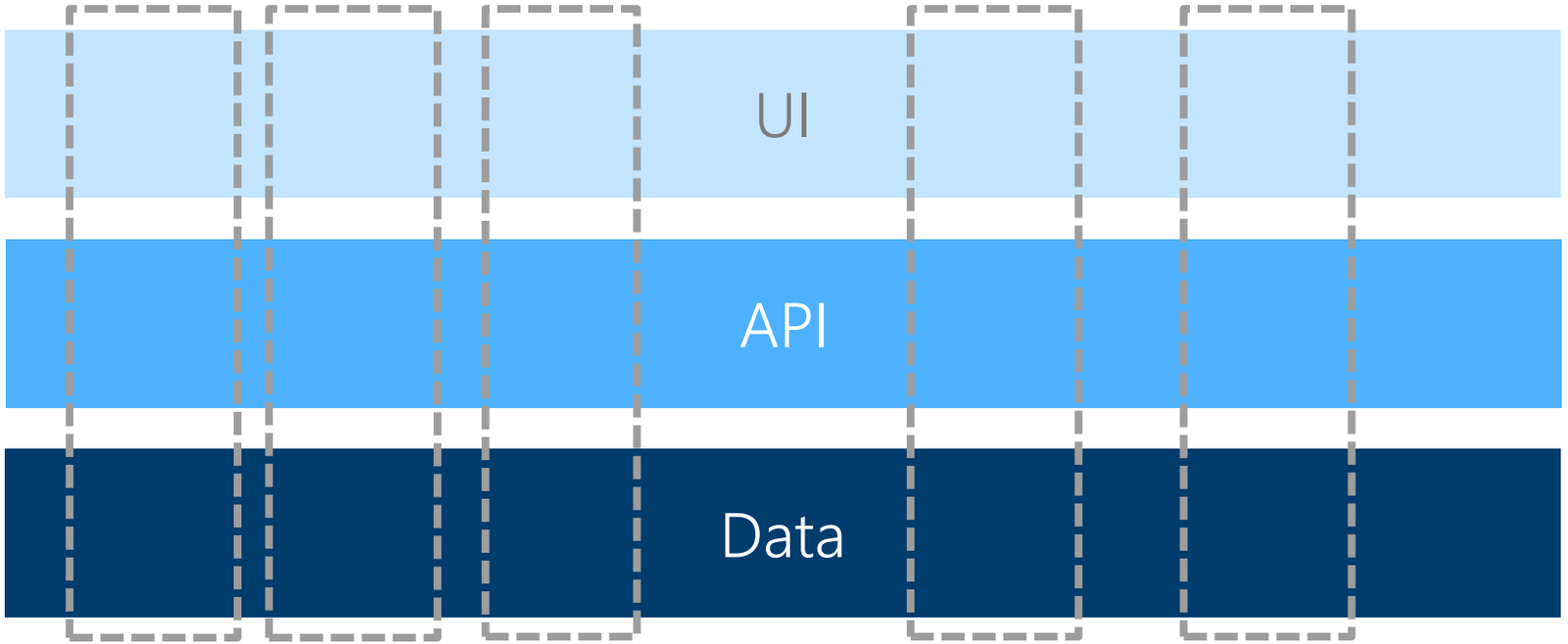
Teams

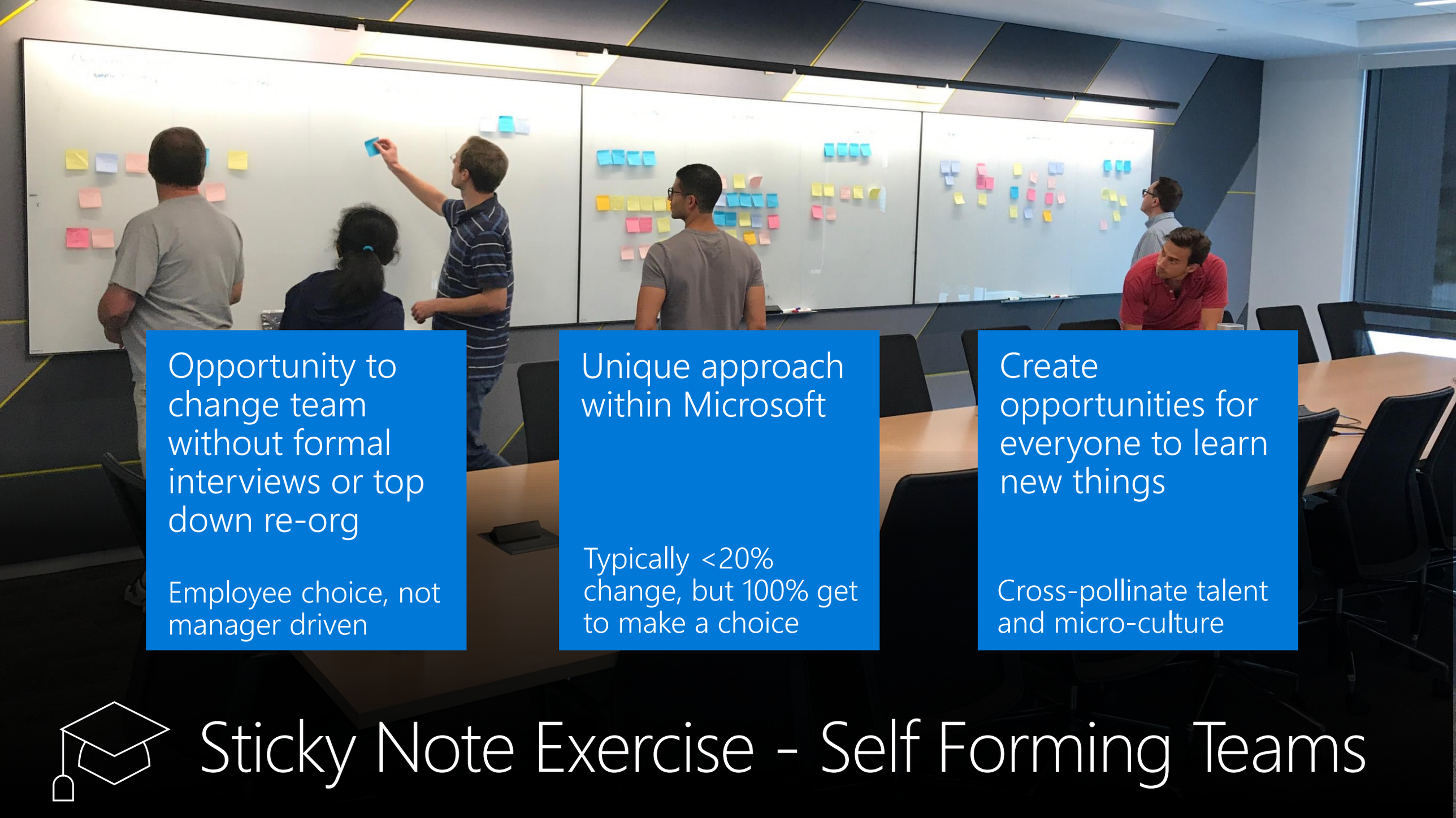
- Physical team rooms
- Cross discipline
- 10-12 people
- Self managing
- Clear charter and goals
- Intact for 12-18 months
- Own features in production
- Own deployment of features

Instead of Horizontal…



We strive for Vertical





Opportunity to change team without formal interviews or top down re-org

Employee choice, not manager driven

Unique approach within Microsoft

Typically <20% change, but 100% get to make a choice

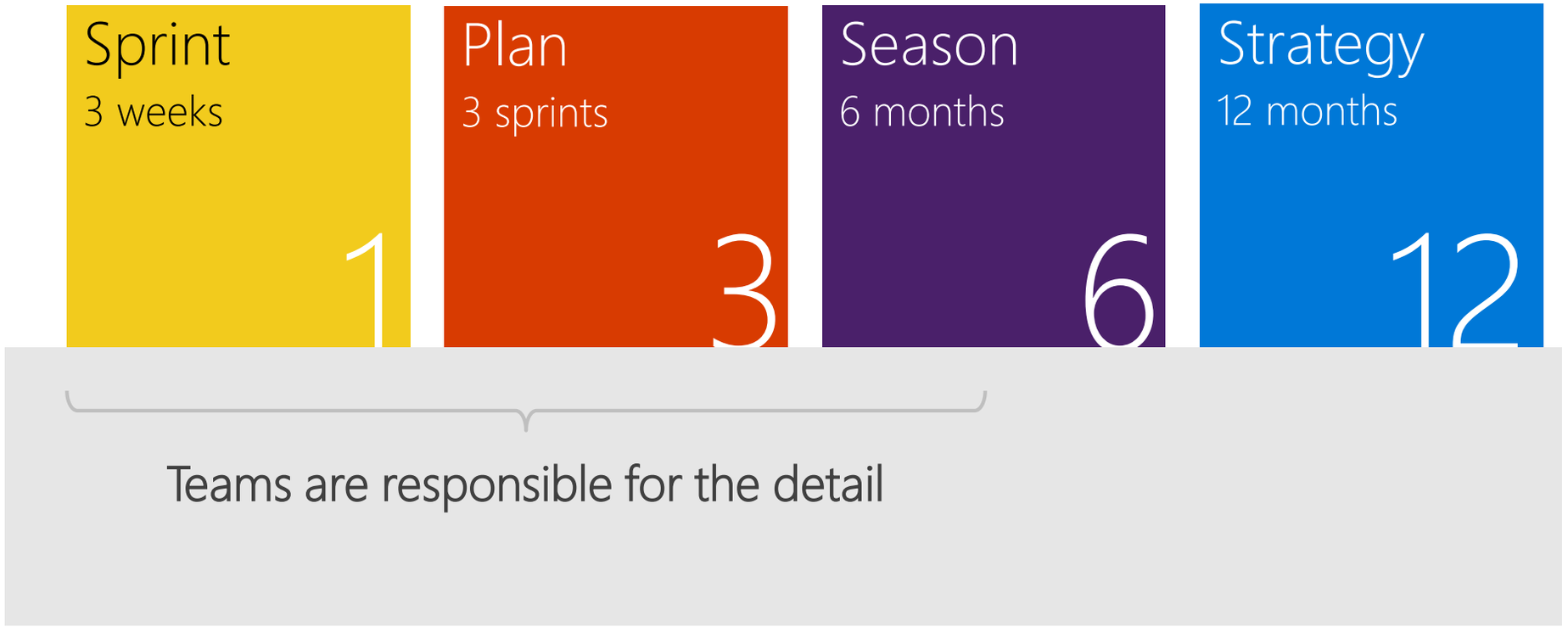
Create opportunities for everyone to learn new things

Cross-pollinate talent and micro-culture

Sticky Note Exercise - Self Forming Teams

Planning

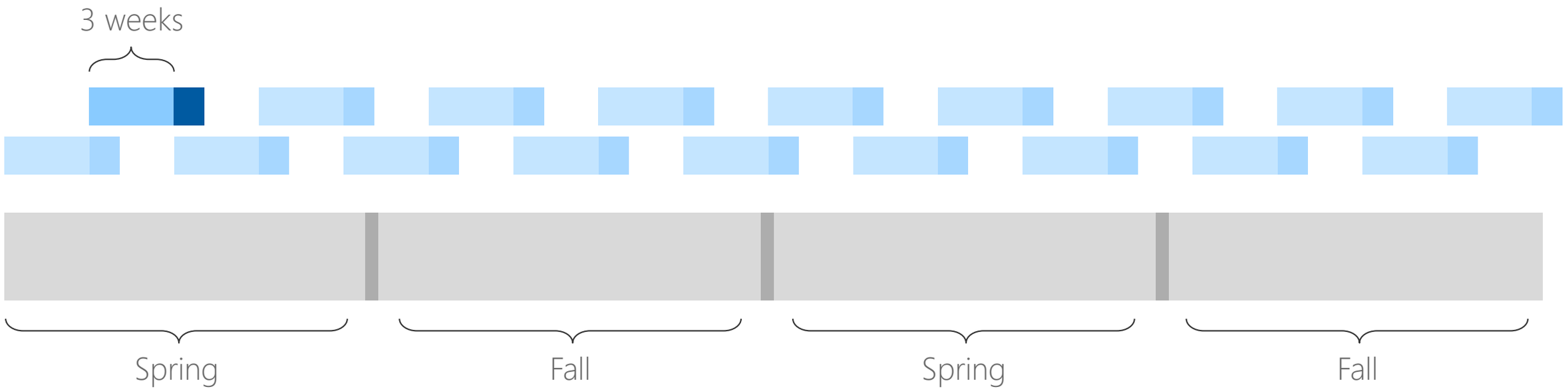
Leadership is responsible for driving the big picture



Microsoft

How do you stay in sync?

1. Sprint Mails
2. Team Chats
3. Experience Reviews



Five habits we've learned so far

- Customer Obsessed
- Production First Mindset
- Team Autonomy + Enterprise Alignment
- **Shift Left Quality**
- Infrastructure as Flexible Resource

Managing the pipeline:

How do you

go fast

and

not

break things?

Do Not Incur Debt

We all follow a simple rule we call the “Bug Cap”:

engineers on your team x 5 = ?

Bug Cap

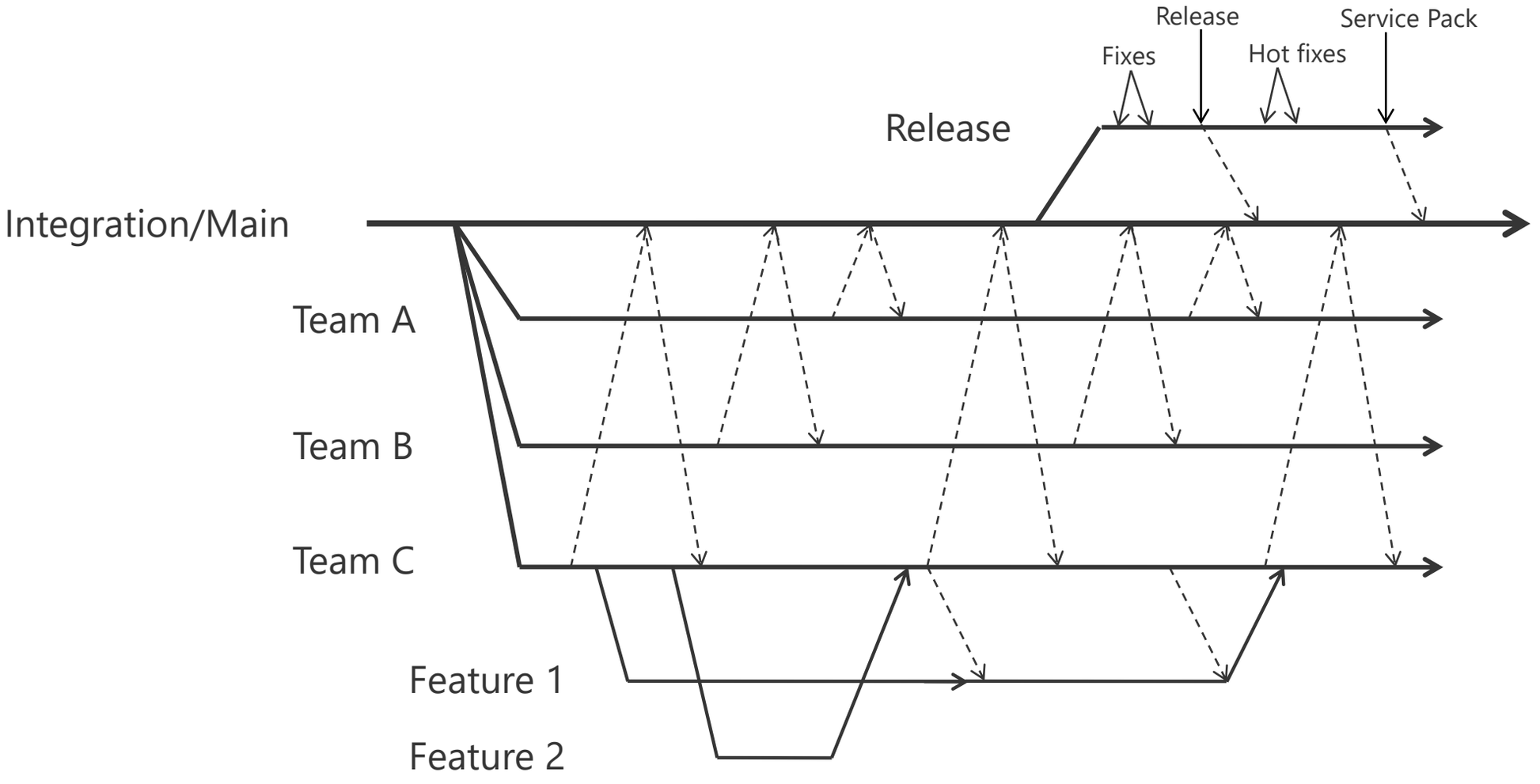
We all follow a simple rule we call the “Bug Cap”:

$$10 \times 5 = 50$$

Rule: If your bug count exceeds your bug cap… stop working on new features until you’re back under the cap.

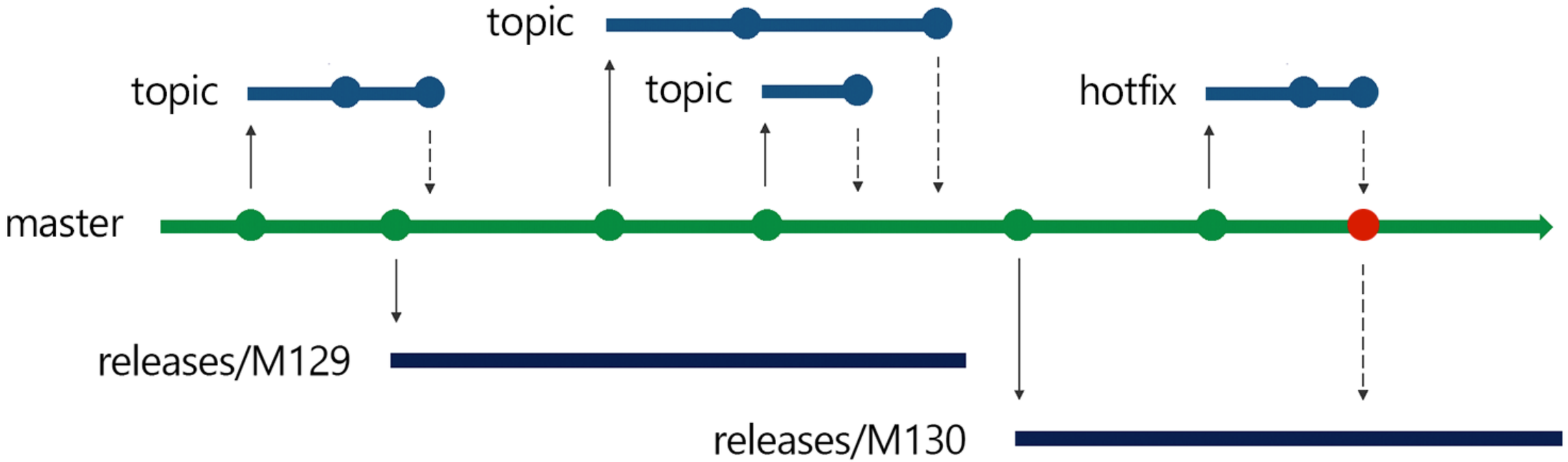
Old way of branching

Composing Isolation Mechanisms



Release Flow

Using Trunk Based Development to avoid Merge Debt



Pull Requests

PR's are point of code review

- L0+L1 Tests performed before merge
- Security tested before commit

Result:

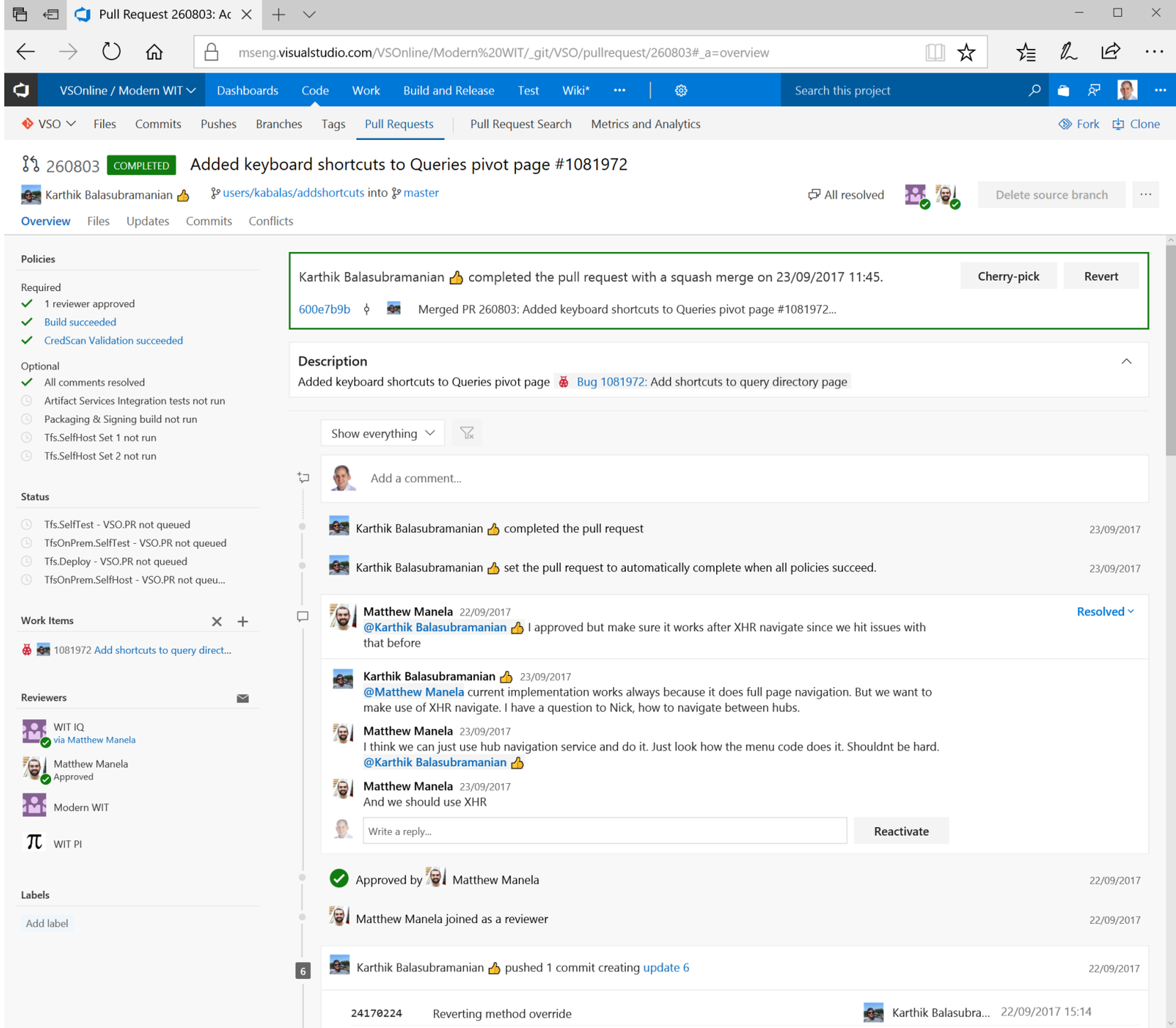
Shift-left testing to pre-merge

- 1/5 Pull requests fail

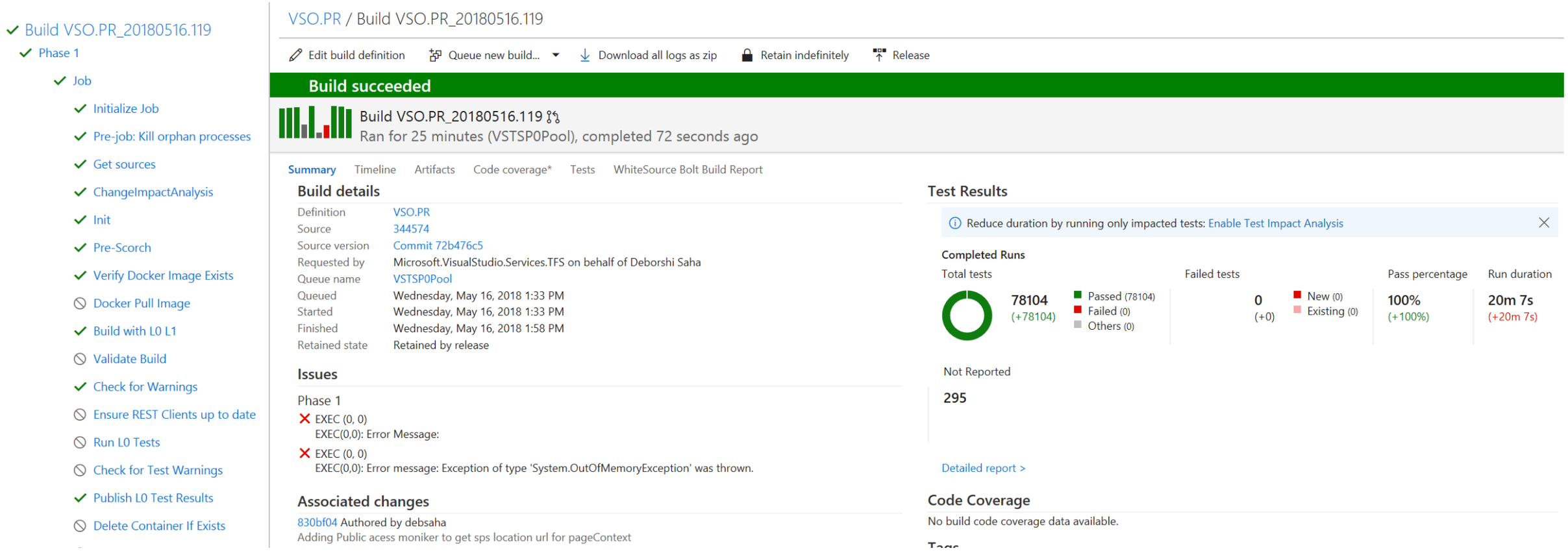
Makes CI build failures rare

- 1/100 CI builds fail

Accelerates the inner and outer loops



Tests Against the Pull Request



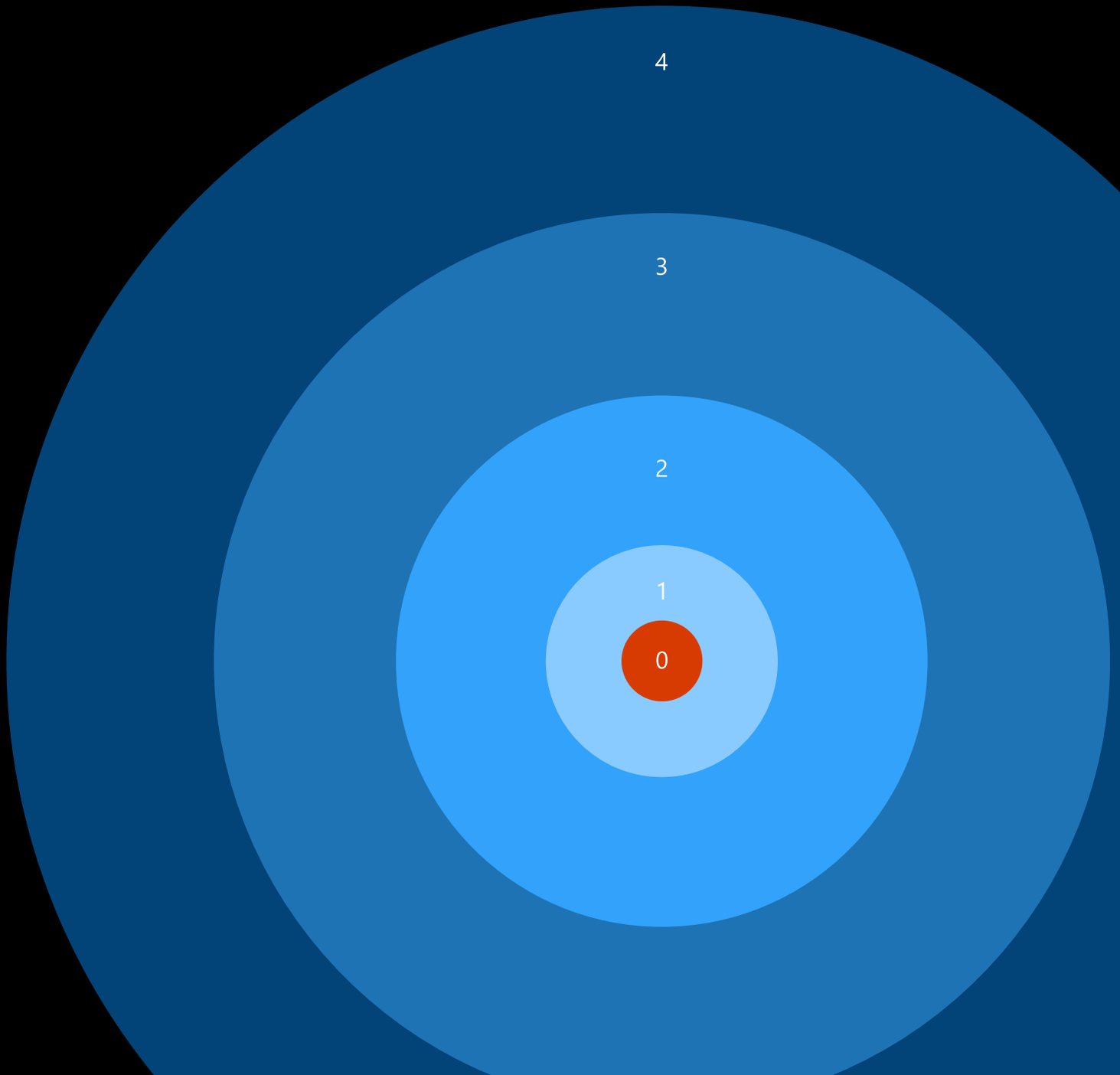
Feedback in minutes, before acceptance of PR

Green Means Green, Red Means Red

Only all-green builds get to release

Your aim won’t be perfect.

Control the blast radius.



Progressive Exposure

Deploy one stage at a time

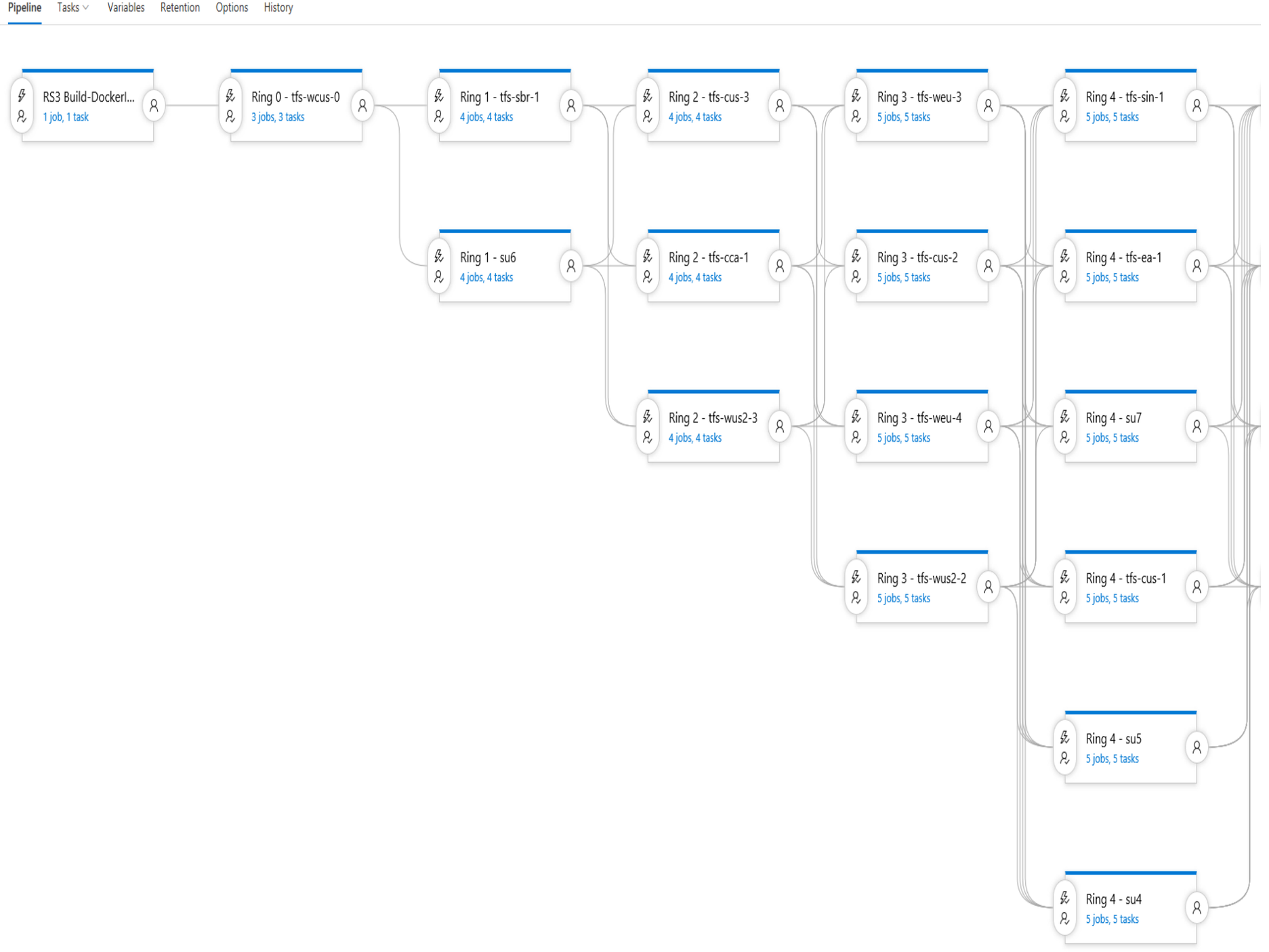
- Release Gates control progression
- Automated (L3) tests to check current ring
- Can slow down with explicit approval

Result:

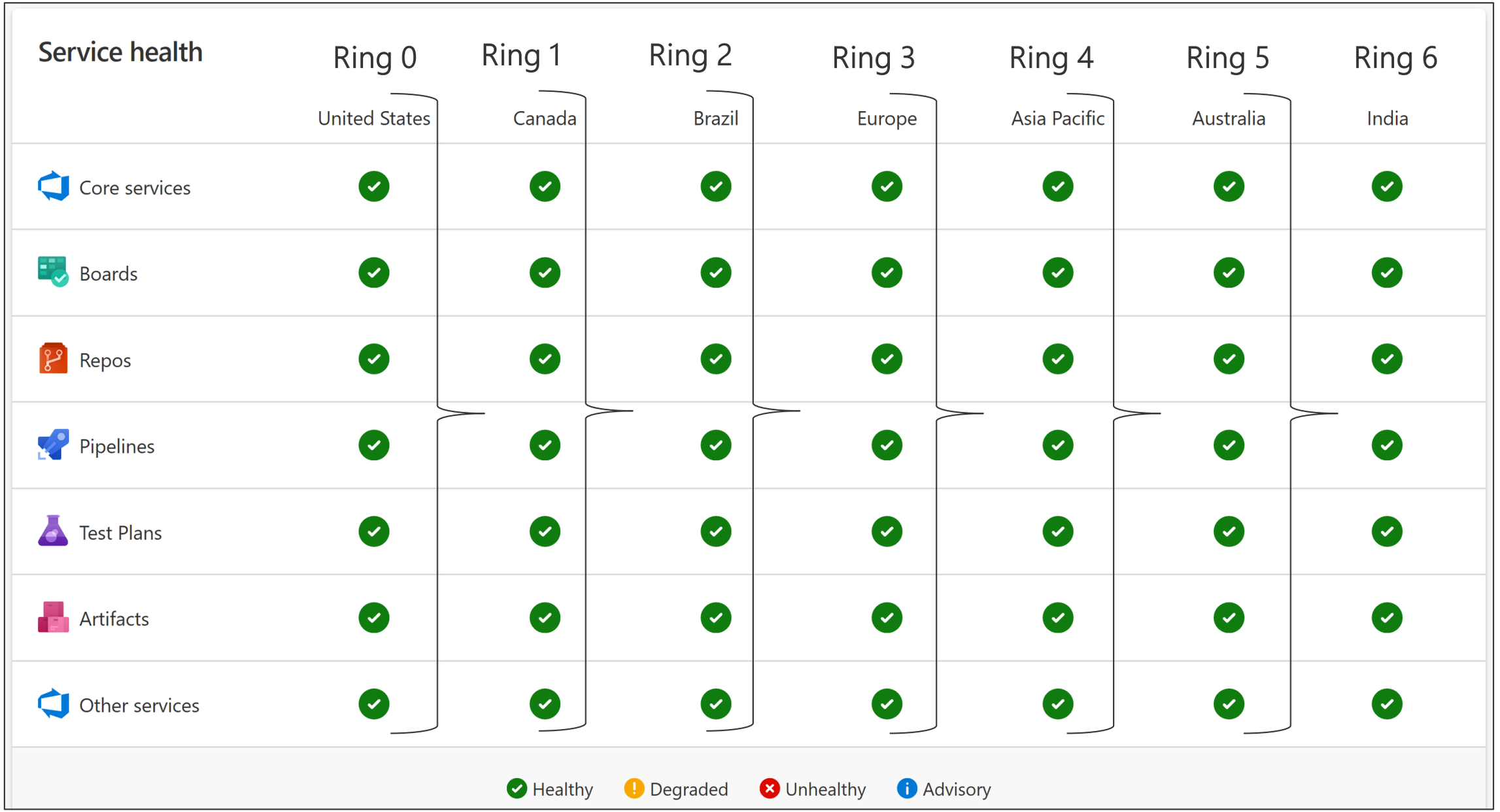
Visibility into impact of every deployment

Move across

1. Canary
2. Data centers with small user counts
3. Data center with large user count
4. Highest latency
5. The rest



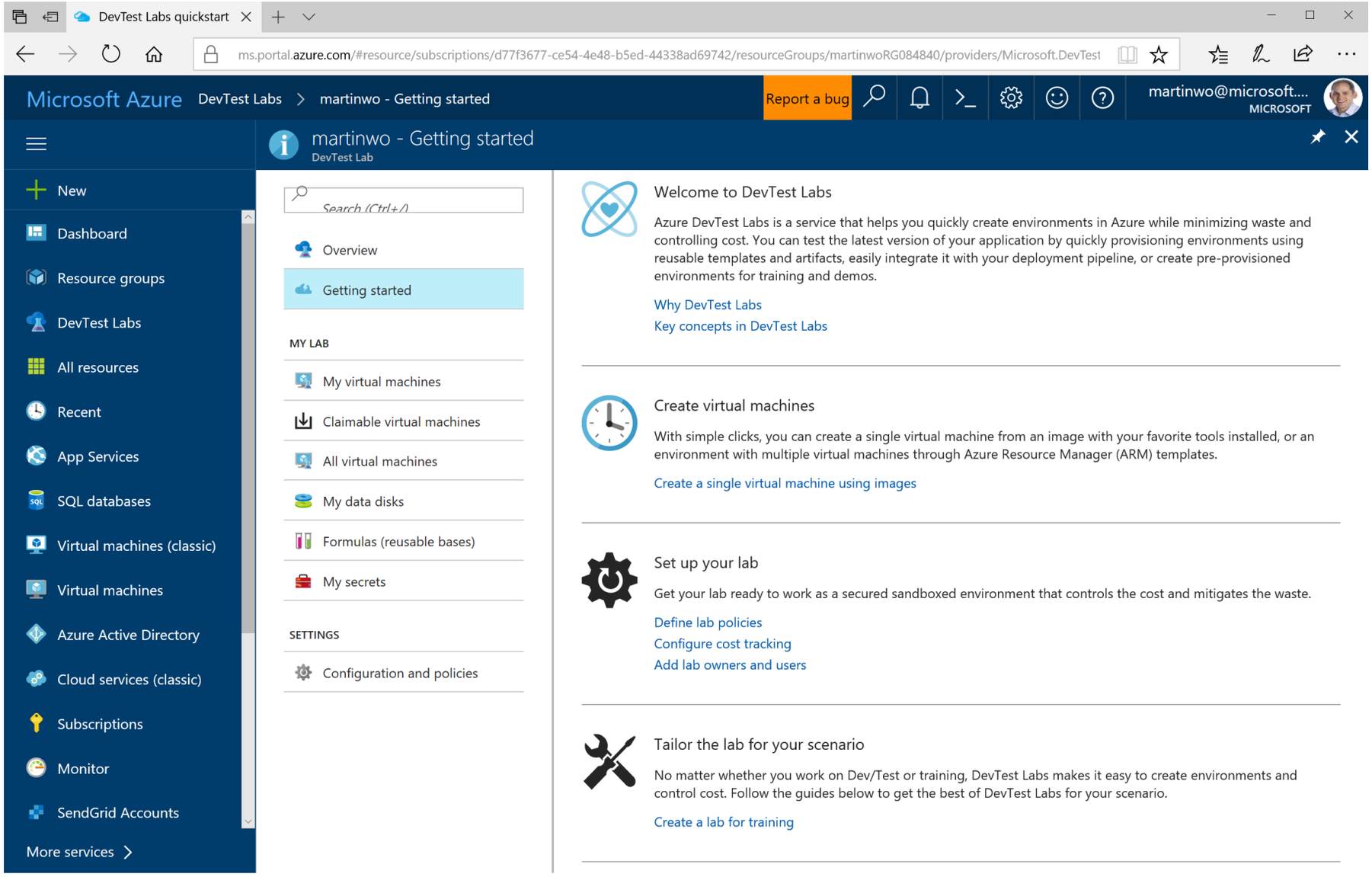
Progressive Exposure – Health Status



Five habits we've learned so far

- Customer Obsessed
- Production First Mindset
- Team Autonomy + Enterprise Alignment
- Shift Left Quality
- Infrastructure as Flexible Resource

From Labs to VM's to DevTest Labs

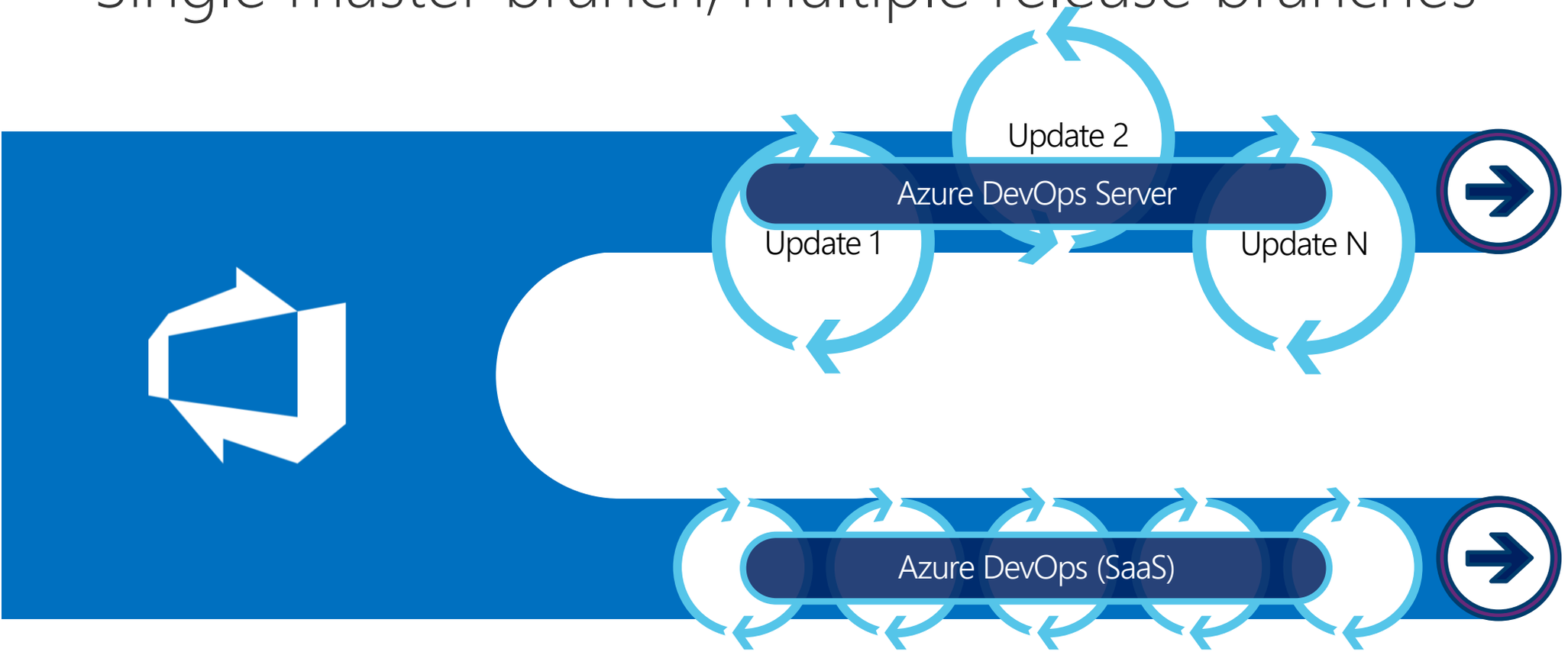


Code: Cloud first, then move on-premises

One code base with multiple delivery streams

Shared abstraction layer

Single master branch, multiple release branches



Our DevOps Transformation – the story so far

Before

- 4-6 month milestones
- Horizontal teams
- Personal offices
- Long planning cycles
- PM, Dev, Test
- Yearly customer engagement
- Feature branches
- 20+ person teams
- Secret roadmap
- Bug debt accumulated
- 100 page spec documents
- Private repositories
- Deep organizational hierarchy
- Success is a measure of install numbers
- Features shipped once a year

After

- 3-week sprints
- Vertical teams
- Team rooms
- Continual Planning & Learning
- PM & Engineering
- Continual customer engagement
- Everyone in master
- 8-12 person teams
- Publicly shared roadmap
- Debt paid as incurred
- Mockups in PPT
- Inner source
- Flattened organization hierarchy
- User satisfaction determines success
- Features shipped every sprint

Demo of MSENG

Building Azure DevOps
…with Azure DevOps

Thank you!