60,000 tests in six minutes
Create a reliable pipeline, eliminate flaky tests, and deploy safely but quickly

@SamGuckenheimer
Azure DevOps
Continuous Delivery for Every Team, Every App, Every Platform

**Azure Boards**
Deliver value to your users faster using proven agile tools to plan, track, and discuss work across your teams.

**Azure Pipelines**
Build, test, and deploy with CI/CD that works with any language, platform, and cloud. Connect to GitHub or any other Git provider and deploy continuously.

**Azure Repos**
Get unlimited, cloud-hosted private Git repos and collaborate to build better code with pull requests and advanced file management.

**Azure Test Plans**
Test and ship with confidence using manual and exploratory testing tools.

**Azure Artifacts**
Create, host, and share packages with your team, and add artifacts to your CI/CD pipelines with a single click.

[https://azure.com/devops](https://azure.com/devops)
My Journey to DevOps

- **Sprint 1**
  - August 2010

- **VSTS Preview**
  - Sprint 29
  - June 2012

- **VSTS GA**
  - Sprint 64
  - April 2014

- **1ES**
  - Sprint 67
  - June 2014

- **Extensions**
  - Sprint 92
  - Nov 2015

- **GVFS**
  - Sprint 102
  - June 2016

- **Azure DevOps**
  - Sprint 140
  - Sep 2018

Timeline:
- 2010
- 2011
- 2012
- 2013
- 2014
- 2015
- 2016
- 2017
- 2018
Azure DevOps: Millions of users + Most of Microsoft

Data: Internal Microsoft engineering system activity, September 2018
DevOps at Scale

Azure DevOps is the toolchain of choice for Microsoft engineering with over 90,000 internal users

https://aka.ms/DevOpsAtMicrosoft

372k Pull Requests per month
4.4m Builds per month
5m Work items viewed per day
2m Git commits per month
500m Test executions per day
500k Work items updated per day
78,000 Deployments per day

Data: Internal Microsoft engineering system activity, September 2018
Comparing the elite group against the low performers, we find that elite performers have...

- 46 times more frequent code deployments
- 2,555 times faster lead time from commit to deploy
- 7 times lower change failure rate (changes are 1/7 as likely to fail)
- 2,604 times faster time to recover from incidents
What do you do about testing?

https://screeningfilm.com/events/shh-at-the-movies/
Some antipatterns from our past....

- Developers threw code over the wall to test automation engineers
- Test automation engineers threw test automation over the wall to testers
- Outsourced testing as a functional silo to vendors
- Split career tracks for devs and testers
- Maintained test automation outside the code base
- Relied on long-running, complex tests through the UI
You get what you measure (Don’t measure what you don’t want)

**Usage**
- Acquisition
- Engagement
- Satisfaction
- Churn
- Feature Usage

**Velocity**
- Time to Build
- Time to Test
- Time to Deploy
- Time to Improve
- Failed and flaky automation

**Live Site Health**
- Time to Detect
- Time to Communicate
- Time to Mitigate
- Which customers affected
- Incident Prevention Items
- Aging Prevention Items
- SLA per Customer
- Customer Support Metrics

**Things we don’t watch**
- Original estimate
- Completed hours
- Lines of Code
- Team capacity
- Team burndown
- Team velocity
- # of bugs found
- % code coverage
- SLA per Customer
- Customer Support Metrics
- Live Site Health
- Failed and flaky automation
Azure DevOps

Definition of Done

Live in production, collecting telemetry, supporting or diminishing the starting hypothesis.
Remember:

Shipping is a feature
Our reality in Sept 2014

Tests took too long

Tests failed frequently

Quality signal unreliable in Master

Huge quality cliff to climb at the end of sprint
Lesson learned:

Say Nightly Automation Run and your tests will take all night.
Everyone has an opinion

**Unit Tests? Bah!**

Devs won’t take time to write them

Unit tests replace functional tests? That isn’t right.

**Philosophical divide: “classical” and “mockist”**

What do you mean?

**Skip the partisan debate and move to the desired outcome**

- Unit tests are a design tool to achieve testability
- You are responsible for your own quality
- You get lightning fast, rock solid reliability wired into Pull Request flow
- Acknowledge we need both unit *and* functional tests
- [Fowler: Mocks aren’t stubs](#) provides balance between mockists and purists
Created a “Non-Denominational” Test Taxonomy

- **L0**: In-memory unit tests with no dependencies
  - In Pull Request: < 60 ms
- **L1**: Unit tests with dependencies e.g. SQL
  - < 2000 ms (avg 400 ms)
- **L2**: Functional tests against APIs
  - In pre-prod deployment, with mocked auth
- **L3**: PROD
  - In production rings
Principles

- Tests should be written at the lowest level possible
- Write once, run anywhere including production system
- Product is designed for testability
- Test code is product code, only reliable tests survive
- Testing infrastructure is a shared Service
- Test ownership follows product ownership
A journey of a thousand miles begins with a single sprint
Shifting the test portfolio over time

- Started with L0 / L1 tests
  - Made it easy to author and execute high quality L0/L1 tests
  - Stopped creating new TRA tests as much as possible

- Analyzed legacy TRA tests
  - Tests that can be deleted
  - Tests that can move to L0/L1
  - Tests that will move to VSSF Test SDK
  - On-prem tests we expect to maintain in lights-on mode

- TRA to L2 conversion
  - Test Arch v-team re-wrote L2 test framework
  - Top-down push from management with org wide scorecard

- L3 tests
  - Use telemetry pipe
  - Carefully curated set
Sprint over sprint progress creating new and retiring old tests

L0 – Requires only built binaries, no dependencies

L1 – Adds ability to use SQL and file system

Run L0 & L1 in the pull request builds

L2 – Test a service via REST APIs
Make them fast: progress by Sept 2017

Level 0

Execution: 6:00 mins
Total Tests: 55,411
Number of Assemblies: 2104

Level 1

Execution: 3:30 mins
Total Tests: 4,647
Number of Assemblies: 41

Source: Internal Test Execution Time, September 2017
Demo

Pull Requests ➔ CI ➔ Pre-Production

Test runs at each level
Left uncontrolled, non-deterministic tests can completely destroy the value of an automated regression suite.

Martin Fowler

https://martinfowler.com/articles/nonDeterminism.html
Finding flaky tests and fixing them

Reliability Run Workflow

1. Execute Reliability Runs on Green Builds
2. Tag Failed Tests as Flaky & file Reliability bugs
3. Aggregate Flaky tags over cumulative runs
4. Un-tag Flakiness when reliability bug is resolved

Reliability Bugs surfaced in Team scorecards

Official Run Workflow

1. Execute Runs
2. Carry Flakiness from Latest Reliability run to current run
3. Flaky tests excluded from Test Run Summary

Reliability Bugs surfaced in Team scorecards

Flakiness data automatically flows to official runs in real time

Flaky test are not quarantined; simply removed from the results

Reliability bugs fixed faster with clear visibility
Demo

Flaky test detection
There’s no place like production
Your aim won’t be perfect.

Control the blast radius.
Demo

Realworld Progressive Deployment
Continuous improvement, not continuous investigation.

- 2016: 200x Deploy frequency, 2555x Lead time for changes, 24x Time to restore service, 3x Change fail rate
- 2017: 46x Deploy frequency, 440x Lead time for changes, 96x Time to restore service, 5x Change fail rate
- 2018: 46x Deploy frequency, 2555x Lead time for changes, 2604x Time to restore service, 7x Change fail rate
Progress follows a J-curve

Technical debt and increased complexity cause additional manual controls and layers of process around changes, slowing work.

Automation helps low performers progress to medium performers.

Relentless improvement work leads to excellence and high performance! High and elite performers leverage expertise and learn from their environments to see jumps in productivity.

Teams begin transformation and identify quick wins.

Automation increases test requirements, which are dealt with manually. A mountain of technical debt blocks progress.
Lessons we learned

- Accountability can motivate culture change
- Trunk-based development requires fast, reliable testing
- Tests are code, are delivered with code and treated like code
- Green means green, red means red
- You’re not done until telemetry confirms you’re done
- Change takes time – but you won’t get there unless you start
Thank you!

Sam Guckenheimer

@SamGuckenheimer

https://aka.ms/DevOps/