How Improving the Testing Experience Goes Beyond Quality: A Dev Productivity Point of View

Roberto Perez Alcolea & Aubrey Chipman
Developer Productivity Team
Netflix
Agenda

- Why improving the Testing Experience?
- Day in life of a developer
- Netflix JVM Build Landscape
- Common problems in the software testing experience
- Improving the testing experience for developers
- Learnings along the way
- Q&A
Why focus on improving the Testing Experience?
Flow state

Dev Productivity with No Interruptions

Flow State Productivity

Image from https://about.sourcegraph.com/blog/developer-productivity-thoughts
Mental context switches = out of Flow state

Image from https://about.sourcegraph.com/blog/developer-productivity-thoughts
Developer Inner Loop

Commit → Search → Read → Write → Build → Test → Commit
Testing is crucial in the inner loop and suboptimal experience leads to context switching and losing flow state.
Day in the life of a developer

(Low productive environment...)
Tuesday Afternoon
Wednesday Morning

GOOD morning

SUNSHINE
The developer:
The developer doesn’t achieve much, is frustrated and unmotivated
Netflix JVM Build Landscape
Builds executing tests

14 Million test cases executed in 28 days
Let’s focus on testing... at the project level
Common problems in the software testing experience
Testing becomes difficult

- Hard to write tests with provided tools
- Lack of actionable feedback from outputs
- Test suites evolution
- Lack of documentation or examples
There are tests, but they are flaky

Flaky tests fail to produce the same outcome with each individual test run.

Potential reasons:

- Asynchronous waits, concurrency
- Test Order Dependency
- Poorly modeled tests
Results of having flaky tests

This can and will frustrate developers. In order to ship a change, folks might:

— Delete the test
— Ignore the test and might never revisit it

... versus

— Identify the flaky test and fix it in the moment

In any case, they might start losing confidence in the test suite
There are tests, but they are slow

Variety in

- Test setup time
- Shared resources
- Parallelization
There are tests, but they are inconsistent

Variety in

- Local Mac machines
- CI Linux machines
- Network and security access
Are these issues causing problems in my organization?
Most likely, yes! 😱 😱 😱
Unwanted situations

Developers will be frustrated and it is possible to fall into the following traps:

— Avoid writing tests

— Ignore or remove tests in order to verify their changes

— Avoid running tests locally and wait for CI job executions to provide feedback for every single small change
Developer Inner Loop

Commit → Search → Read → Build → Write → Search → Read → Build → Write

Dev Productivity with No Interruptions

Flow State Productivity

Images from https://about.sourcegraph.com/blog/developer-productivity-thoughts
And...

“It is staggering how tolerant engineers are of toil and frustration and friction.”
At Netflix, we are not immune to that
And we have invested on it...
Improving the testing experience for developers
Faster test startup
Let’s look at a real world example 🕵️‍♀️
Project setup
Great tools!!
but...
Running a single test class was slow

<table>
<thead>
<tr>
<th>Test</th>
<th>Outcome</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>contextLoads</td>
<td>PASSED</td>
<td>1.728s</td>
</tr>
</tbody>
</table>

The test method is not the problem.
The actual bottlenecks

49 seconds applying flyway migrations!

13 seconds waiting for crdb to be ready in Docker on M1!
The tools were not the problem, but how we used them!
Why so slow?

- Many migration files with multiple SQL statements
- Docker in ARM based Mac
- Not reusing containers
Faster tests -> faster builds

Execution 1 of 1  PASSED  Total / own / serial time: 48.600s / 48.000s / 48.600s  Started May 19 2023 at 15:22:12 PDT

No test class setup or cleanup failures occurred during this test class execution

Output
Test output is not captured for successful test executions, except for the first successful retry after a prior failure.

Test executions
<table>
<thead>
<tr>
<th>Test</th>
<th>Outcome</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>contextLoads</td>
<td>PASSED</td>
<td>0.600s</td>
</tr>
</tbody>
</table>
What did we change?

- Database migrations baselines
- Testcontainers singleton pattern
- Testcontainers Cloud
Other suggestions

- Test slicing (reduced application context and data)
- Context initialization
- Invest on application startup time
  - Better modularization
  - Trim dependencies
You might find opportunities to standardize tools across teams!
Flaky Test Detection & remediation
Flaky Test Detection

When a test fails, how do we determine if it’s flaky or not?
Why is this important?

From https://medium.com/contino-engineering/know-the-testing-pyramid-42a4b3573988
Flaky Test Visualization

We use Gradle’s Test Retry Gradle plugin with Gradle Enterprise to catch and visualize flakiness.
Flaky Test Detection

5.60K builds (1% of 469K builds that executed tests)

In a month, 5.6k Builds (1%) have flaky tests, not good!
Flaky Test Detection by class

We know which tests are constantly reported as flaky
Flaky Test Detection by class

And with how long they usually take to run, this test used over 17 hours for the 296 retries.
Detection != Solving the problem 😞
What can we do about it?

- Surface this information to project owners on a friendly way
- Culture change
- Quarantine the tests
The goal...

Hopefully one day!
Predictive Test Selection
Predictive Test Selection

Increases developer productivity by automatically and intelligently selecting and executing the subset of tests that are most relevant to a code change, providing faster feedback.

Popularized by Meta. Read the paper!
How does PTS work?

- Predictive model by observing code changes and test outcomes from your Build Scan data
- Predictive Test Selection will not attempt to make predictions for test tasks or goals for which fewer than 14 days of code
- Tests will always be chosen if they are recently new, recently changed, recently failed, or recently flaky.

**NOTE:** We are trading testing comprehensiveness for faster feedback, making it worthwhile for many test executions where reducing feedback time is critical, such as local and pre-merge/pull-request builds.
Monthly PTS Simulations for local and CI builds

Local

Savings potential: 33 d 8 hr (19%)
Avoidable tests: 187K (27%)

CI

Savings potential: 73 d 5 hr (38%)
Avoidable tests: 1.11M (42%)
30,684 potential hour savings in a year
We rolled it out to all compatible projects 🚢 🚢 🚢 🚢
How did we roll out PTS?

Image from https://www.drawio.com/feature-flag-devops-whitepaper
PTS results the first month

Test tasks which enabled Predictive Test Selection

226K (87% of total)

107 d 15 hr (88% of 122 d 5 hr total savings potential)
But not everything goes as expected, unfortunately.
Learnings

- Developers might not like trading testing comprehensiveness for speed
- Missing inputs/outputs in test configurations
- Impact on code coverage tooling
Remote Test Execution
Remote Test Execution

Take existing test suites and distribute them across remote agents to execute them faster.

The tests and their supporting files are transferred to each agent and executed, with their logging and results streamed back to the build in real time.
Why Remote Test Execution?

Image from https://about.sourcegraph.com/blog/developer-productivity-thoughts
Build time chart from Netflix build data
Why Remote Test Execution?

- Consistent experience between local and CI
- Better compute resource usage
- Faster feedback
- Run more tests locally

Image from https://about.sourcegraph.com/blog/developer-productivity-thoughts
Build time chart from Netflix build data
Life before remote test execution
Life after remote test execution

33 tasks executed in 2 projects in 4m 59.676s, with 7 avoided tasks saving 16.369s.
Be aware of potential limitations

- Network or security access
- Different environment debugging
- Network traffic and slow connections
Current use as a Beta offering

- 6.4% Remote execution (builds)
  - 14.8k of 233k builds
- 16% Remote execution (repositories)
  - 544 of 3.4k repositories

We are working to close our gaps.
Learnings along the way
Sure, there are 😊😊!
Technological solutions
We improved other experiences, too

- Enabled CI jobs parallel executions
- Increase on testing against real datastores and AWS cloud resources
- Reduction on CI agents failures due to misconfigured Test Suites
- Reduce CI compute resource usage
Social solutions
Abstracting tooling where appropriate

- All the solutions discussed can be applied on a small scope of project-by-project to a large scope of all projects at once

- Use good judgement to determine if the solution should be targeted to a select set of repositories versus applied for all
Make it simple

— Be clear on what is changing, when and what to expect.

— Over-communicate the ways folks can find useful information.

— Prevent migration fatigue. Enable low-effort opt-in or opt-out.

— Judiciously decide when to require user changes.
Understandability, documentation, and tracking

— Provide actionable error messages that point to further documentation where needed

— Provide actionable Pull Requests with informative messages that are tracked in a campaign

— Spread awareness of the new features that save them time and energy! Newsletters and townhall feature reviews are great for this
Beta test this features with your team and/or close partners

- Find customers who would benefit the most from this to work with and help lead this effort
- Sharing with close partners helps uncover scenarios that you probably didn’t think of
- Communicate expected outcomes and risks
- Measure before and after a feature has been introduced
Request feedback, feedback is key
Last thoughts…

- Scale doesn’t matter
- Treat the testing process well!
- Invest on testing experience
- Enable fast testing cycles
- Treat test suites like production code
How?

Developer Inner Loop

Commit → Search → Read → Build → Write → Test → Commit
Questions?

Roberto Perez Alcolea
rperezalcolea@netflix.com

Aubrey Chipman
achipman@netflix.com
Thank you!

Roberto Perez Alcolea
rperezalcolea@netflix.com

Aubrey Chipman
achipman@netflix.com