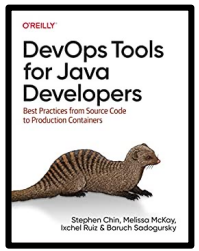
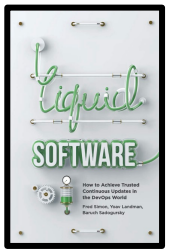
WE SOLVED DEVOPS. WHAT'S NEXT?

Baruch Sadogursky - @jbaruch

- Developer Productivity Advocate
- Gradle Inc
- Development -> DevOps -> #DPE





SHOWNOTES

- speaking.jbaru.ch
- Slides
- Video
- All the links!

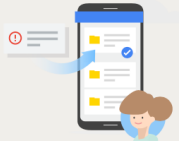
A DECADE OF DORA

- x Deployment frequency
- x Change lead time
- x Change fail rate
- x MTTR

Change lead time:
the time it takes for a code commit or change to be successfully deployed to production.

Deployment frequency:
how often application changes are deployed to production.

Change fail rate:
the percentage of deployments that cause failures in production,[1] requiring hotfixes or rollbacks.



Failed deployment recovery time:
the time it takes to recover from a failed deployment.

PUPPET DORA REPORT 2015

Figure 2
Distribution of deployment frequency
by performance cluster

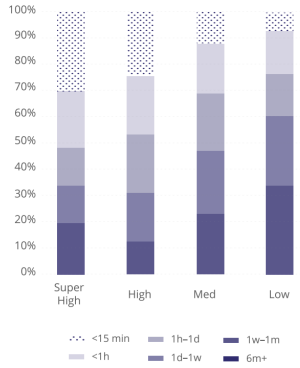


Figure 3
Distribution of deployment lead time
by performance cluster

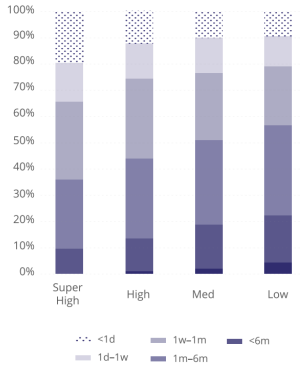
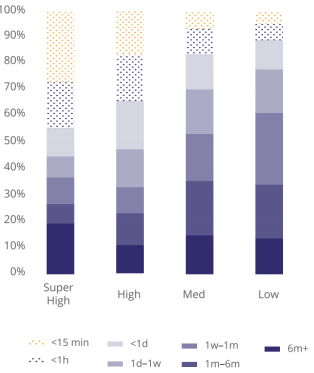
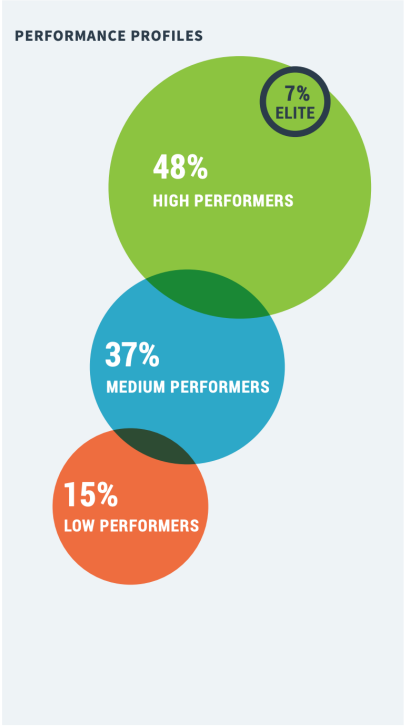


Figure 4
Distribution of mean time to recover (MTTR)
by performance cluster



ACCELERATE DORA REPORT 2018

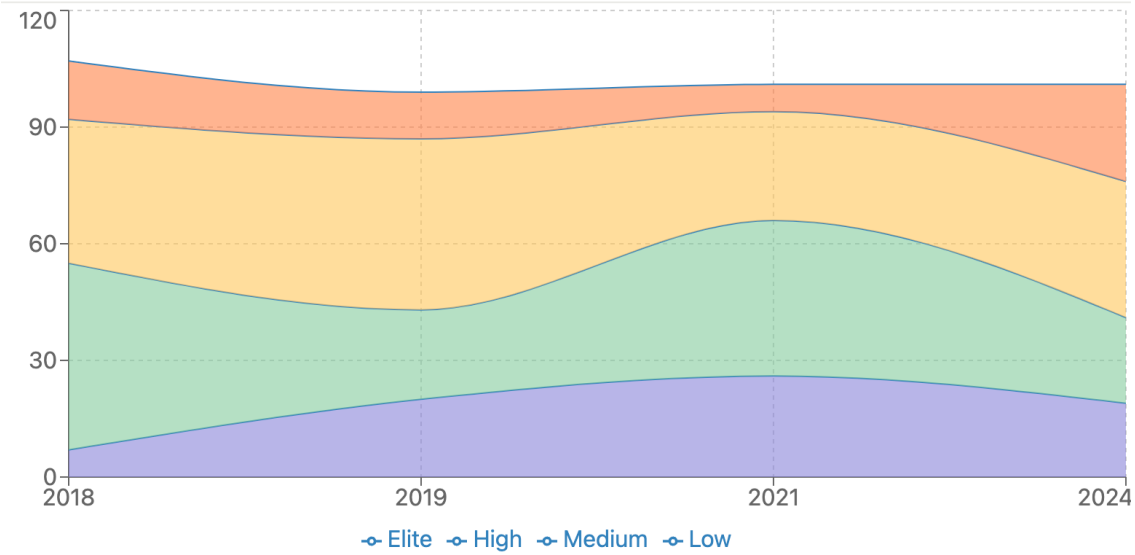
Aspect of Software Delivery Performance	Elite[a]	High	Medium	Low
Deployment frequency For the primary application or service you work on, how often does your organization deploy code?	On-demand (multiple deploys per day)	Between once per hour and once per day	Between once per week and once per month	Between once per week and once per month
Lead time for changes For the primary application or service you work on, what is your lead time for changes (i.e., how long does it take to go from code commit to code successfully running in production)?	Less than one hour	Between one day and one week	Between one week and one month[b]	Between one month and six months[b]
Time to restore service For the primary application or service you work on, how long does it generally take to restore service when a service incident occurs (e.g., unplanned outage, service impairment)?	Less than one hour	Less than one day	Less than one day	Between one week and one month
Change failure rate For the primary application or service you work on, what percentage of changes results either in degraded service or subsequently requires remediation (e.g., leads to service impairment, service outage, requires a hotfix, rollback, fix forward, patch)?	0-15%	0-15%	0-15%	46-60%



GOOGLE CLOUD DORA REPORT 2024

Performance level	Change lead time	Deployment frequency	Change fail rate	Failed deployment recovery time	Percentage of respondents*
Elite	Less than one day	On demand (multiple deploys per day)	5%	Less than one hour	19% (18-20%)
High	Between one day and one week	Between once per day and once per week	20%	Less than one day	22% (21-23%)
Medium	Between one week and one month	Between once per week and once per month	10%	Less than one day	35% (33-36%)
Low	Between one month and six months	Between once per month and once every six months	40%	Between one week and one month	25% (23-26%)

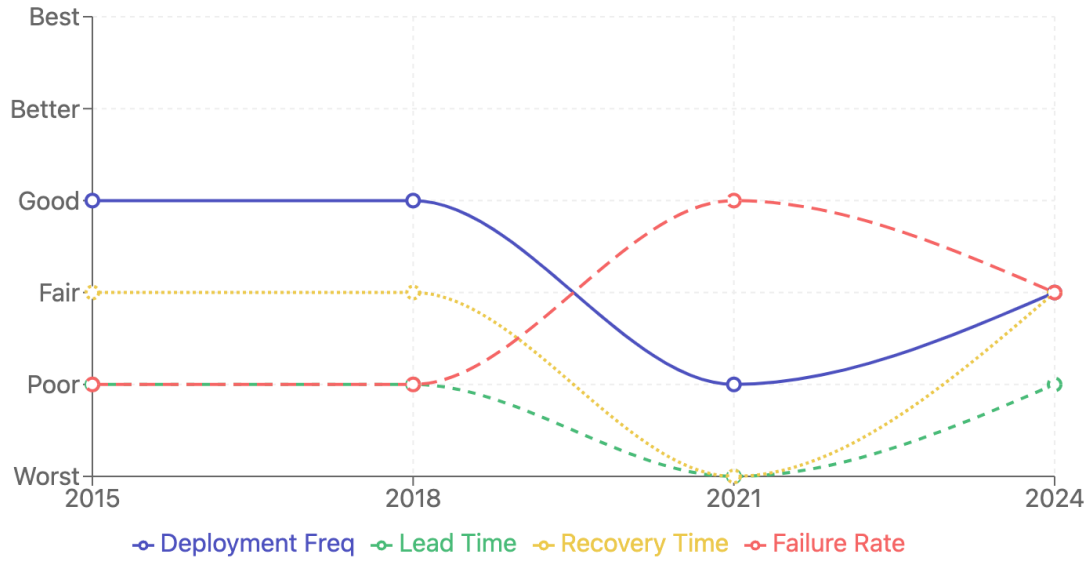
CLUSTER CHANGES OVER TIME



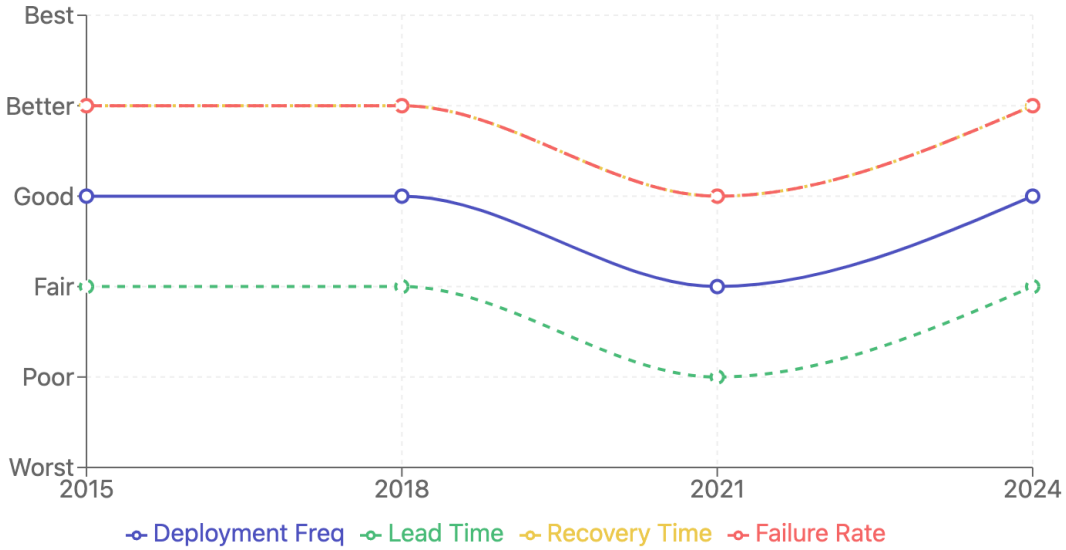
HOT TAKE:

WE FIGURED THE "DEVOPS" SHIT OUT. DONE.

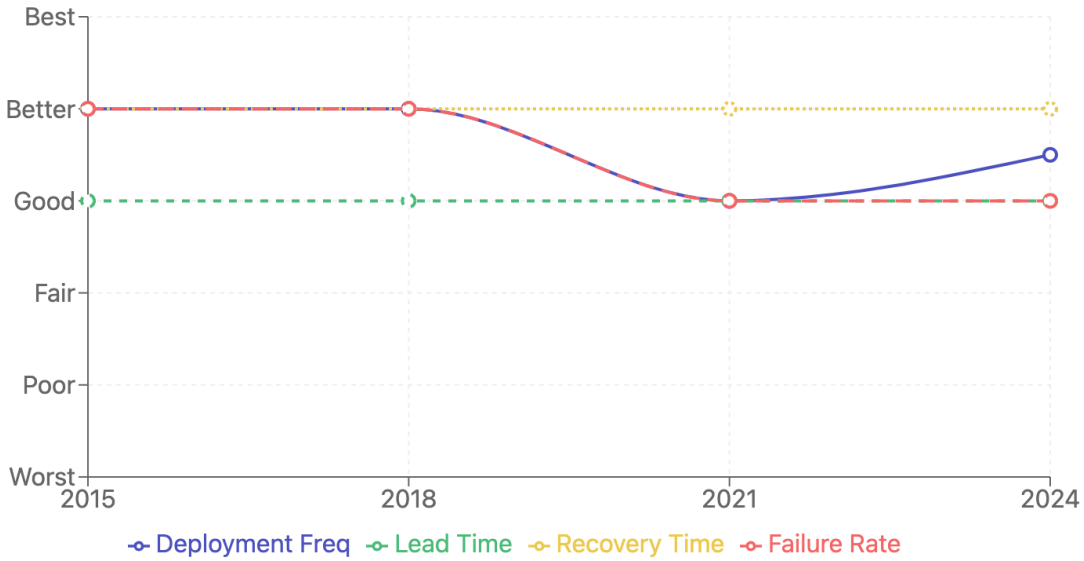
LOW PERFORMERS OVER TIME



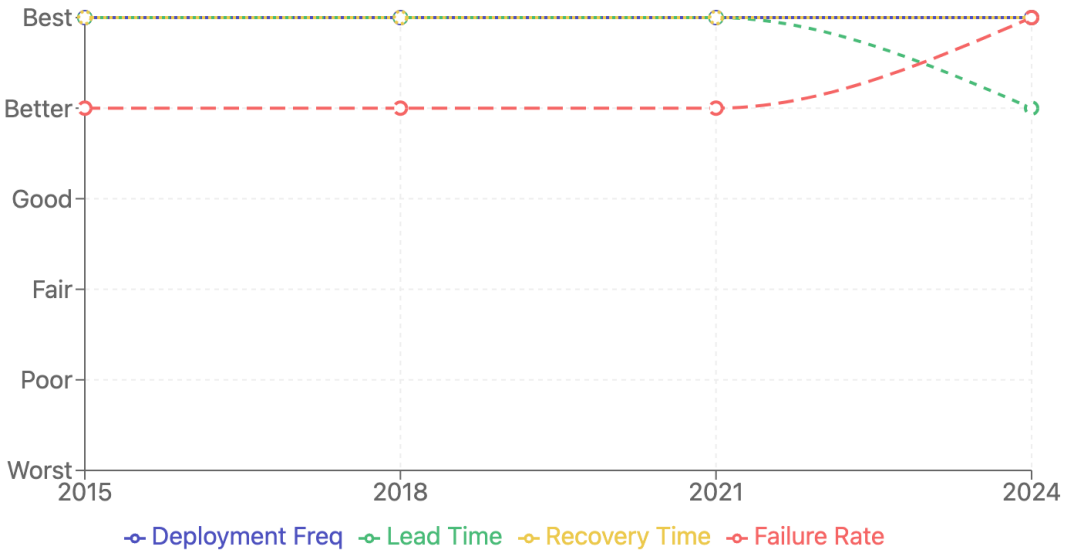
Medium Performers Over Time



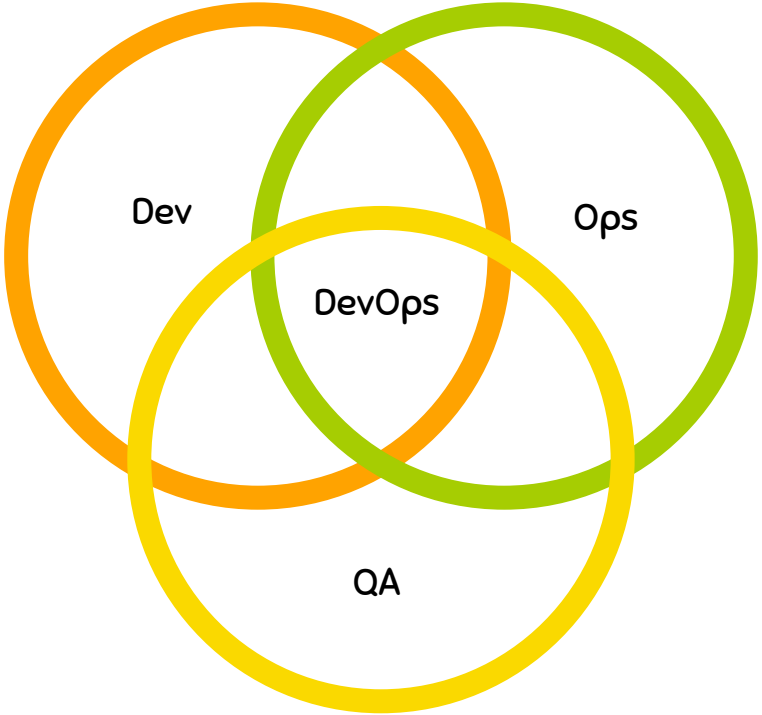
HIGH PERFORMERS OVER TIME

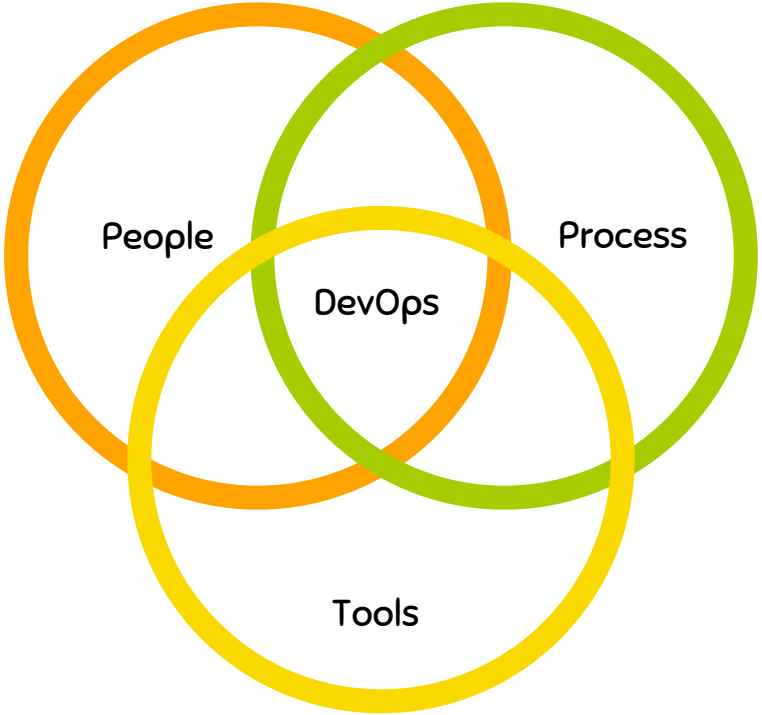


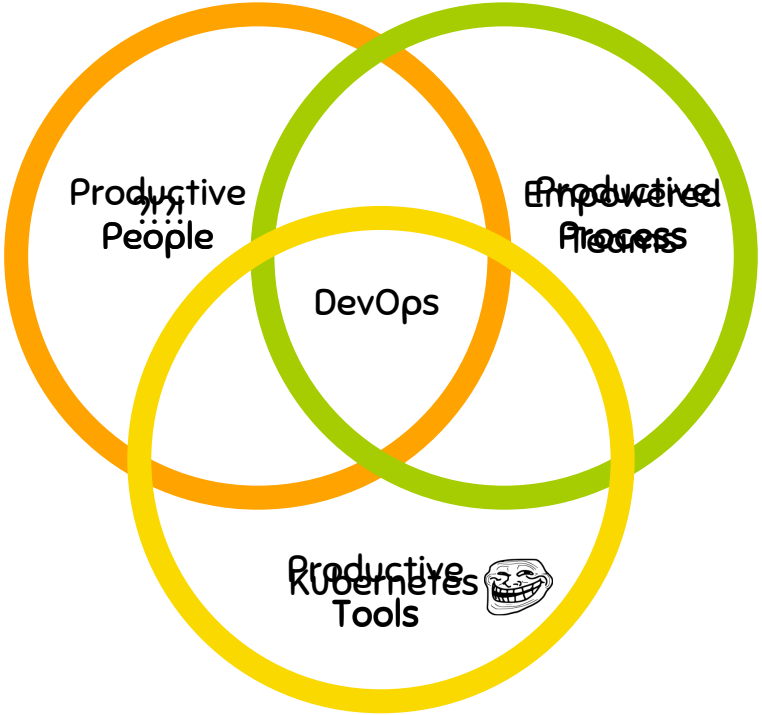
ELITE PERFORMERS OVER TIME



WHY DOES LEAD TIME DECLINE?!


Dev
Ops
DevOps
QA


People
Process
DevOps
Tools


Productive People
Empowered Teams
Productive Process
DevOps
Productive Kubernetes Tools

THE NEXT FRONTIER:

PRODUCTION SYSTEM OF YOUR PRODUCTION SYSTEMS

THE PAIN IS REAL

Builds and Tests are Slow!

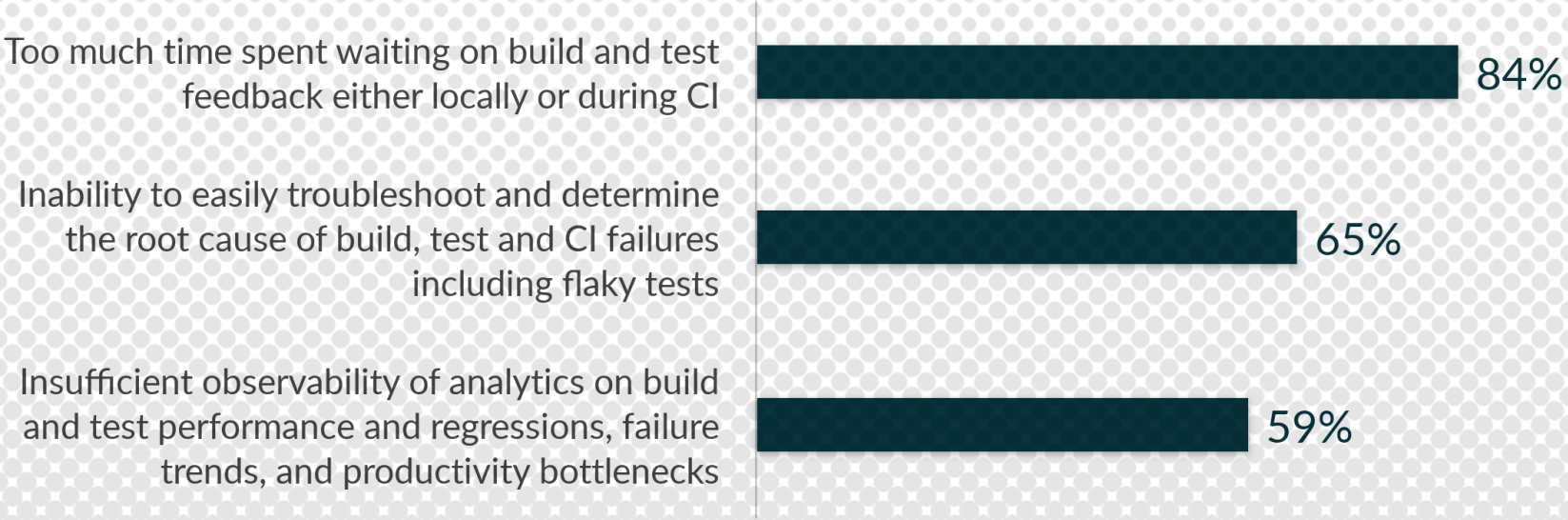
90% of surveyed IT organizations experienced challenges with too much time spent waiting on build and test feedback.

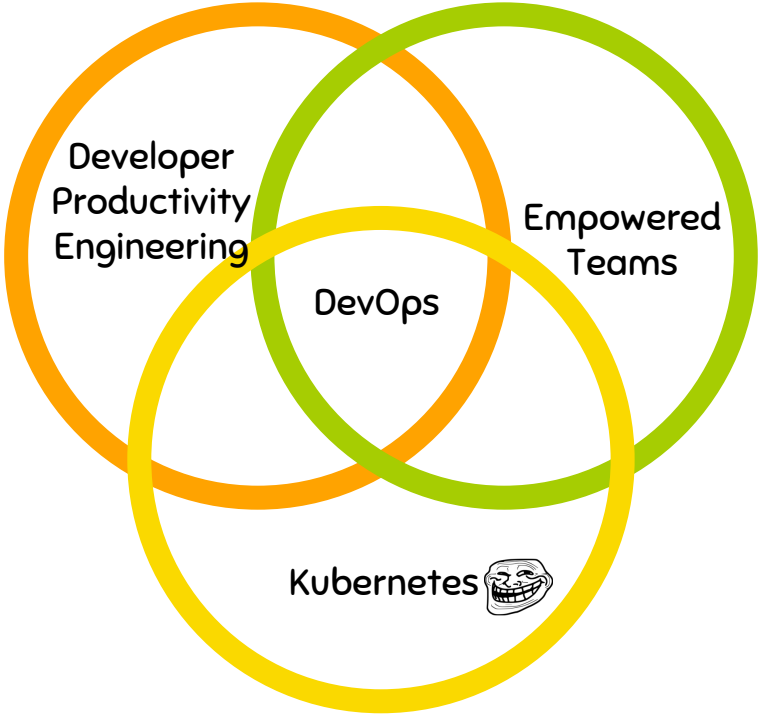
TechValidate
by SurveyMonkey

Validated Published: Sep. 25, 2023 TVID: AE3-632-6FE

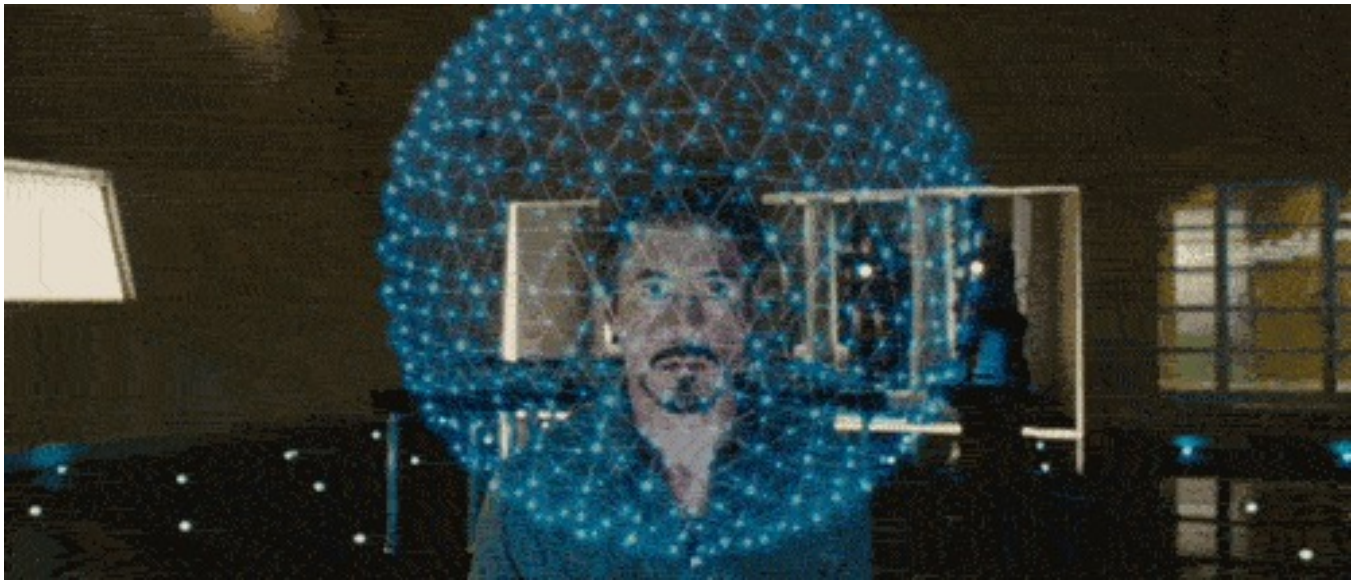
Development Pains are Widespread

Which of the following challenges or pain points did your organization experience prior to implementing Developer Productivity Engineering?




Developer
Productivity
Engineering
Empowered
Teams
DevOps
Kubernetes

DEVELOPER PRODUCTIVITY ENGINEERING!



DEVELOPER PRODUCTIVITY ENGINEERING

Foster Faster Feedback

Collaborate through Effective Tooling

Embrace Rigorous Observability for Proactive Improvement

Eliminate Toil for Developers

Prioritize Automation and Eliminate Bottlenecks

Dedicated Organizational Mindset

Outcomes Over Output

TALK IS CHEAP, SHOW ME THE GOODS!

Small DPE improvements make a huge difference

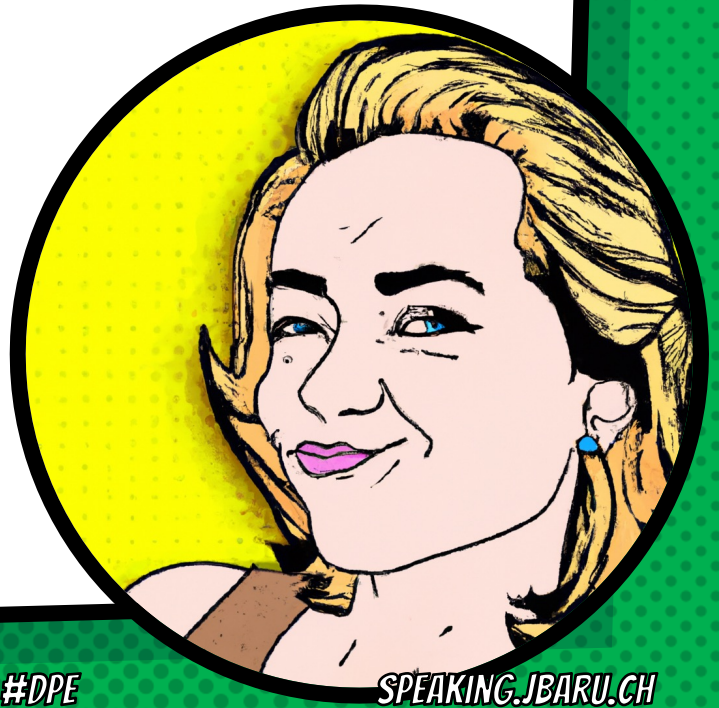
- Generate code faster: Better IDE
- Test better: Testcontainers
- Enforce better code: Sonar
- Test more reliably: Flaky test detection
- Foster Faster Feedback:

Feedback efficiency

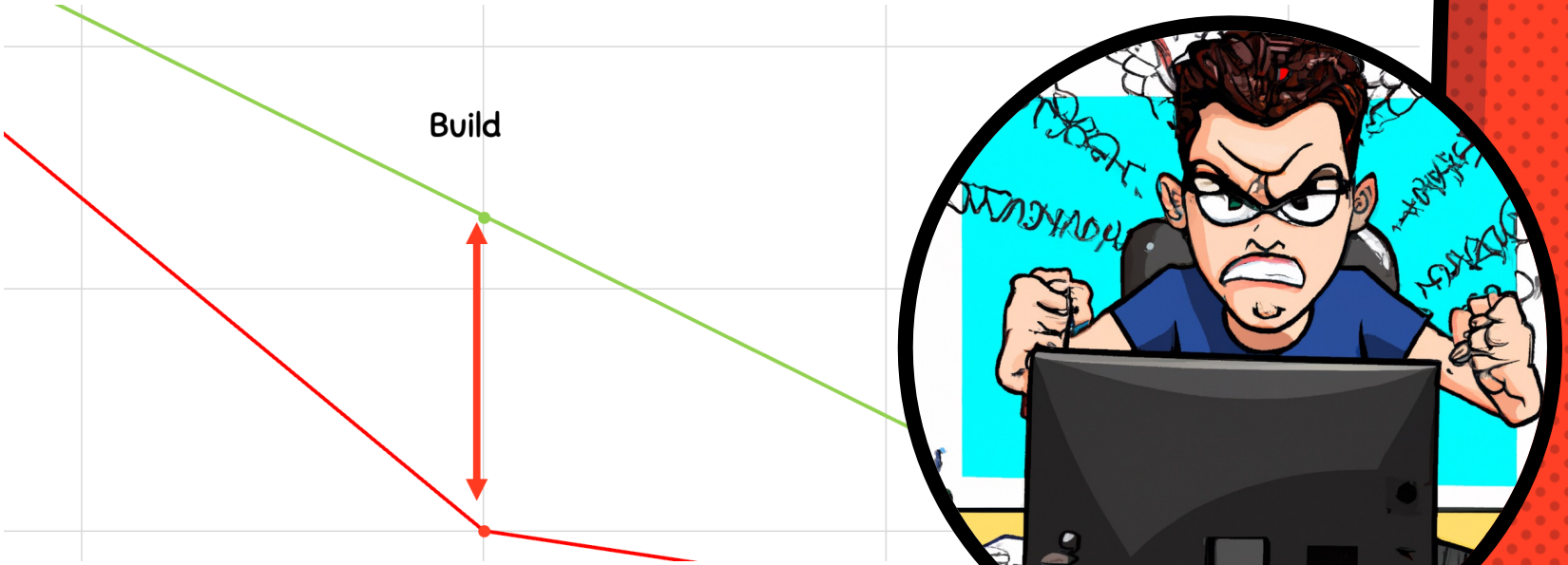
- IDE: Sub-seconds (I type, it marks it red)
- Build: Seconds
- CI: Minutes
- Production: Hours/Days

REVERSE DEPENDENCY ON DISTANCE FROM DEVELOPERS


THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:
"MY CODE'S COMPILING."
HEY! GET BACK TO WORK!
COMPILING!
OH. CARRY ON.



IT IS SLOW!

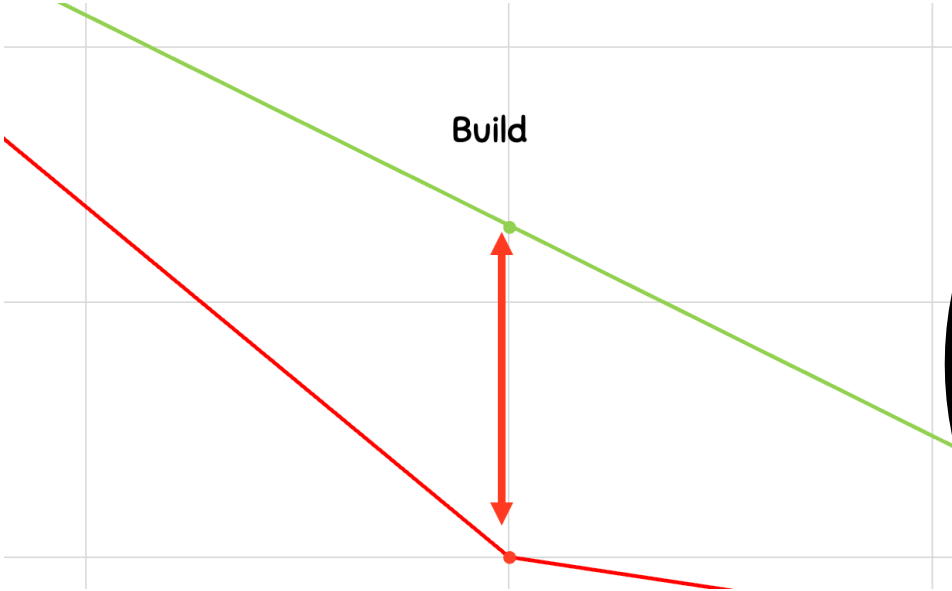


https://www.bruceeckel.com/2021/01/02/the-problem-with-gradle/

stymied me. This is the problem I had with Gradle:

> ***To do anything you have to know everything***

IT IS SLOW AND THE DEVELOPERS HAVE NO IDEA WHY!



What is build?

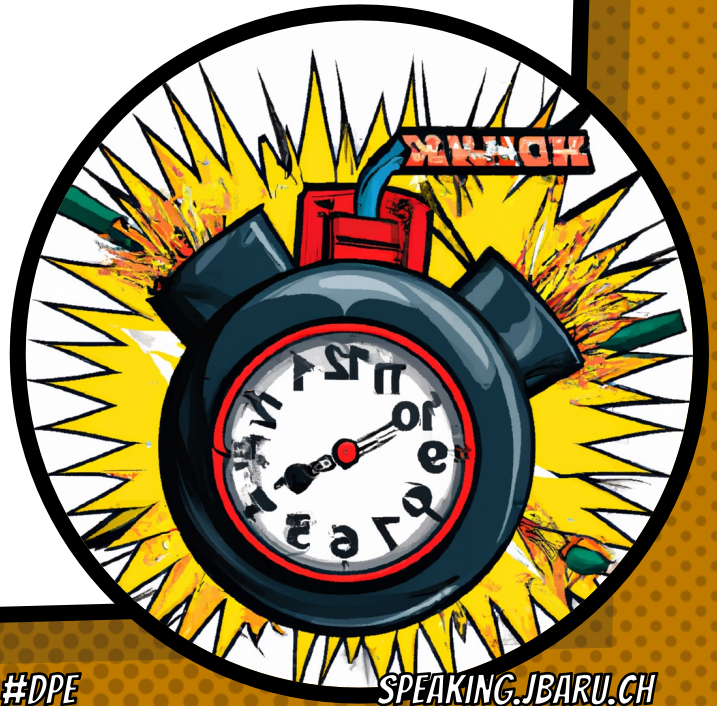
- Project setup
- Downloading the Internet
- Artifact generation: Compilation, packaging, etc
- Tests
- Artifact deployment

What can go wrong?

- Project setup
- Downloading the Internet
- Artifact generation: Compilation, packaging, etc
- Tests
- Artifact deployment

When can it go wrong?

At any time.



THE BUILD FRUSTRATES THE DEVELOPERS

LET'S ASK CHATGPT

What techniques can I use to speed up a Maven build?

Sure, here are some quick tips to speed up a Maven build:

1. **Skip Tests**: Use `-DskipTests` to skip running tests.
2. **Parallel E[illegible]**: Use `-T` flag to run [illegible] E.g., `-T 4` for 4 threads.
3. **Offline Mode**: Use `--offline` to avoid checking for updates.
4. **Incremental Build**: Use `mvn -am -pl` to build only changed modules and their dependencies.
5. **Profile**: Use Maven profiles to conditionally enable plugins.
6. **Resource Filtering**: Limit resource filtering to only necessary files.
7. **Use a Mirror**: Set up a local Nexus or Artifactory mirror.

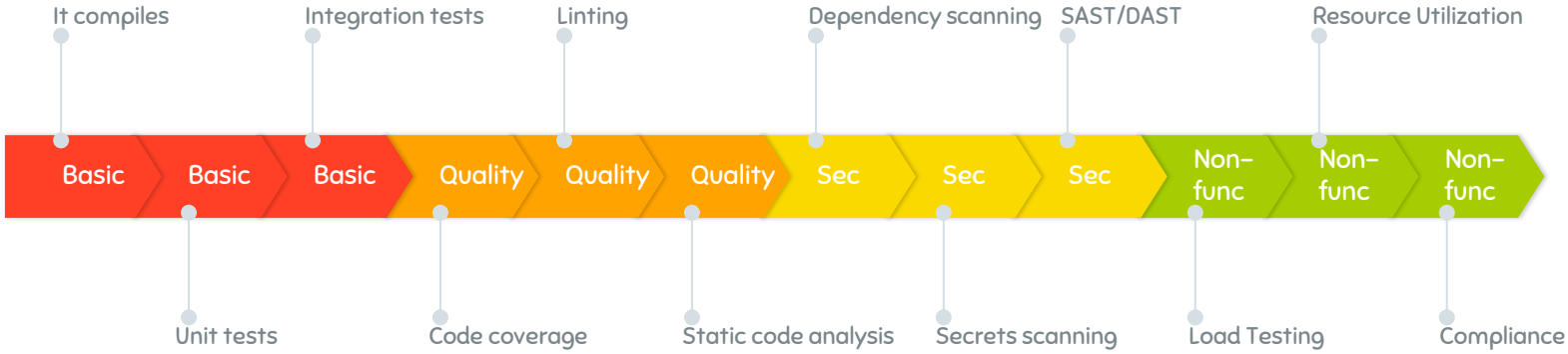
Hope that helps! Got a preference for Gradle, but these should give your Maven build a boost.

WHAT THE ACTUAL F*CK?!

- Skipping tests defeats the purpose of the build!
- How about skipping compilation?
- We want faster feedback, not less feedback

WHAT FEEDBACK DO WE WANT?

CI/CD PIPELINE QUALITY GATES

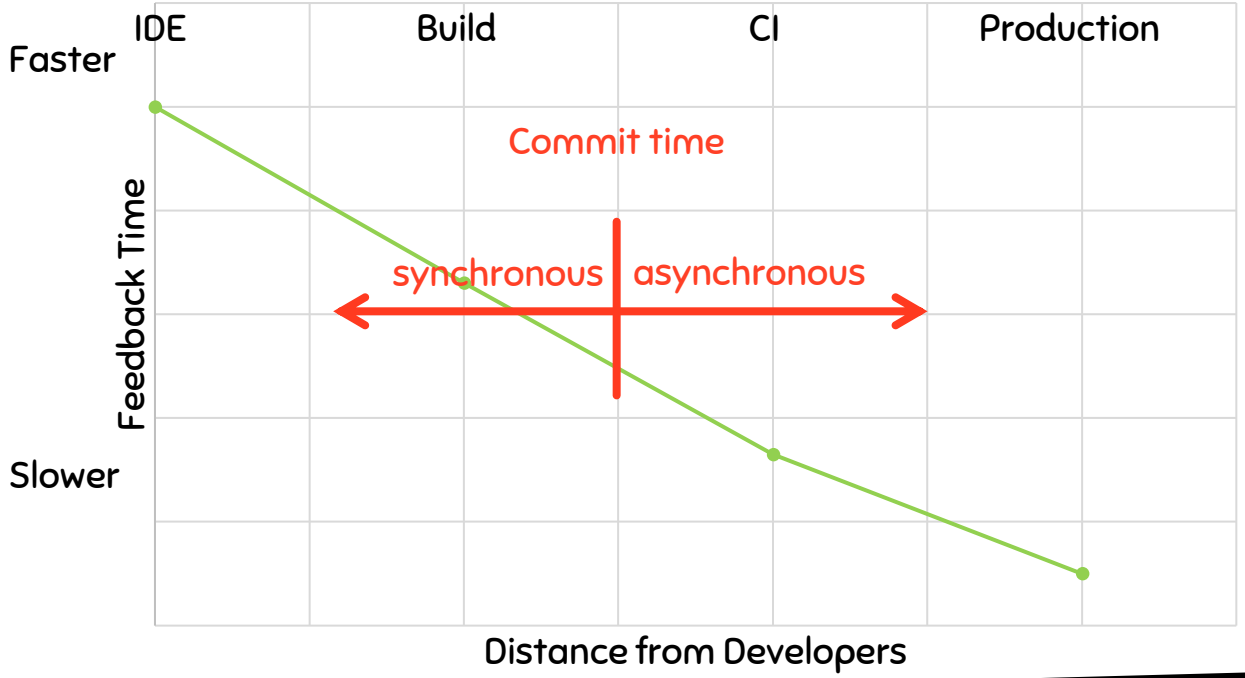



CI/CD? NOT GREAT, NOT TERRIBLE.

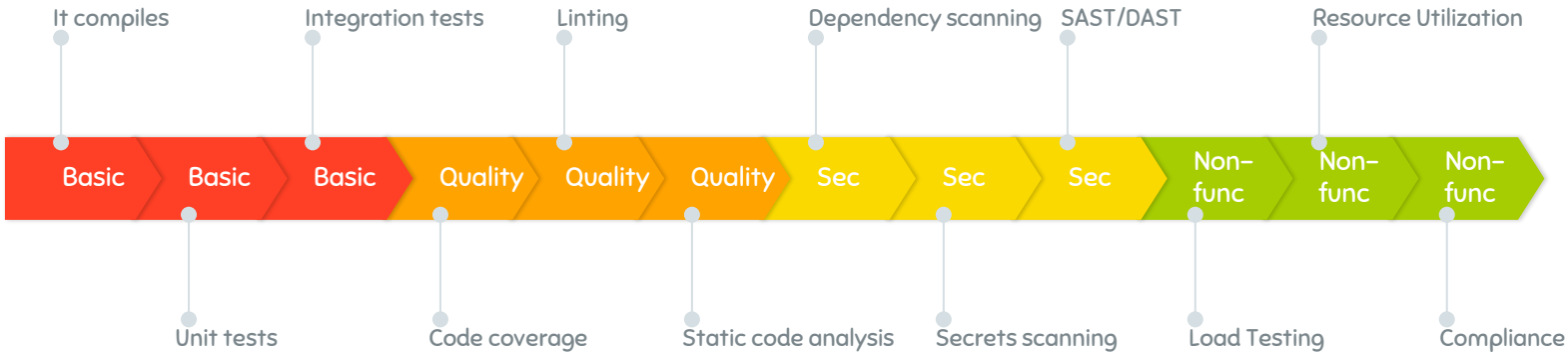
Two Types of Feedback

Asynchronous	x e.g., CI/CD x we never wait for it x results are distracting
Synchronous	x e.g., build x we'll wait for it in the flow x we'll be pissed off when it's slow

REVERSE DEPENDENCY ON DISTANCE FROM DEVELOPERS



IDEAL BUILD TIME FEEDBACK



BUT WON'T IT SLOW DOWN THE BUILD?!

Delightful Build (Pick Two):

- ☑ Provides Max Feedback
- ☑ Fast

SKIP WHAT CAN BE SKIPPED (BUT NO MORE!)

Avoidance: Incremental Build

- × Don't build what didn't changed
- × Don't build what isn't affected

Avoidance: Incremental Build Shortcomings

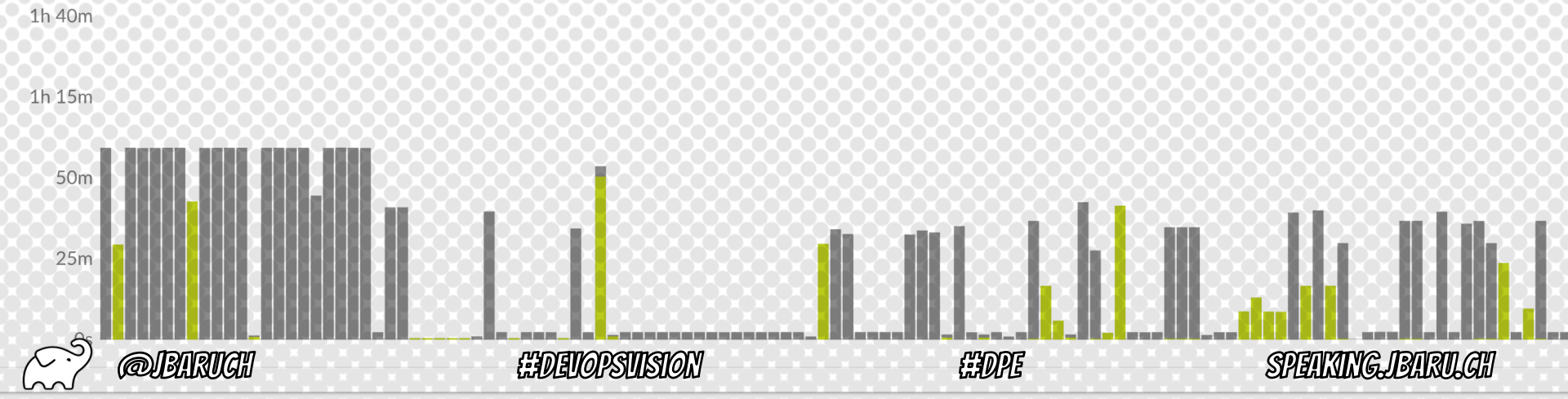
- × Relies on produced artifacts
- × Relies on architectural decisions

Avoidance: caching

- × Makes the build faster
- × Makes the build faster for everybody
- × Makes the build faster always
- × Makes all parts of the build faster

Build time	Serial execution	Avoidance savings	Build cache overhead	Dependency downloading
12 min 28 sec	15 min 13 sec (2.46x)	39 min 10 sec (73.62%)	18 sec	2.2 sec

Up to date	Local build cache	Remote build cache
19 min 33 sec	4.3 sec	19 min 33 sec



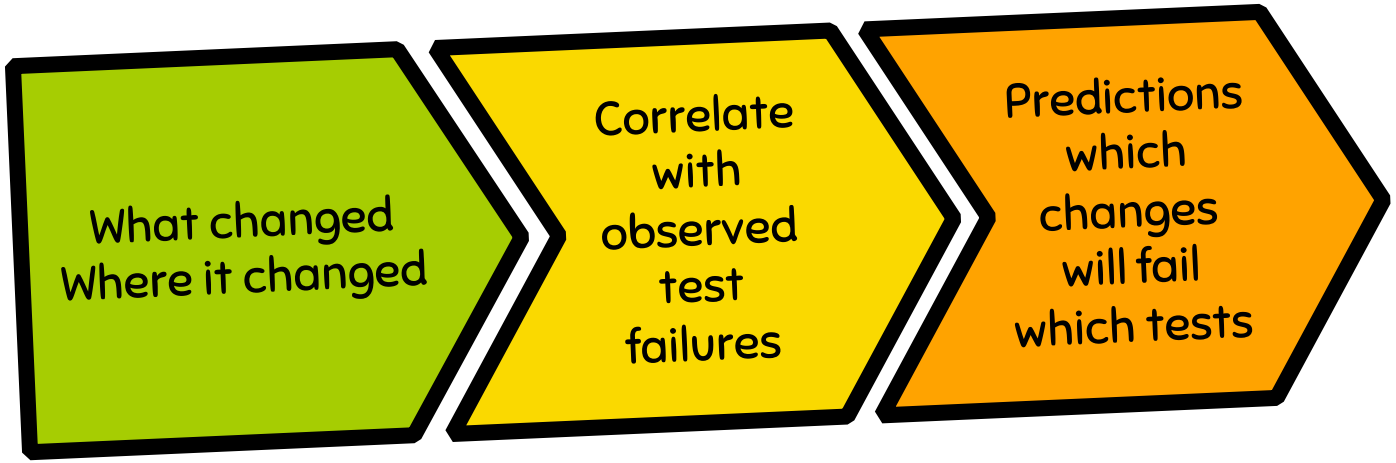
Avoidance: Predictive Test Selection

- × Learns code changes effects de-facto
- × Skips tests with high degree of confidence

How test prediction works

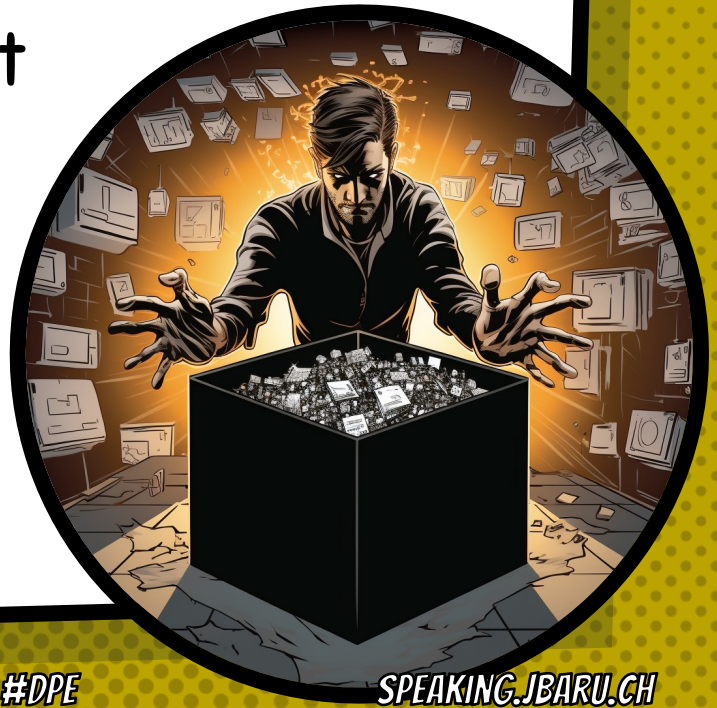
- Code changes and test results are thrown into learning model
- After a while, the model predicts which changes fail which tests

TEST PREDICTION



Black Magic in Action

- The more tests a project has, the less they break
- Refactorings in Java break tests less than in JavaScript



time | Custom values | Tags | Outcome

Last 90 days | Relative | Fixed | Refres

Predictive Test Selection › Find test task/goal

age (78 builds)
n build time: 11 min 18 sec

Simulations (51.3K builds)
Mean build time: 5 min 27 sec

Selection profile
Conservative | Standard | F

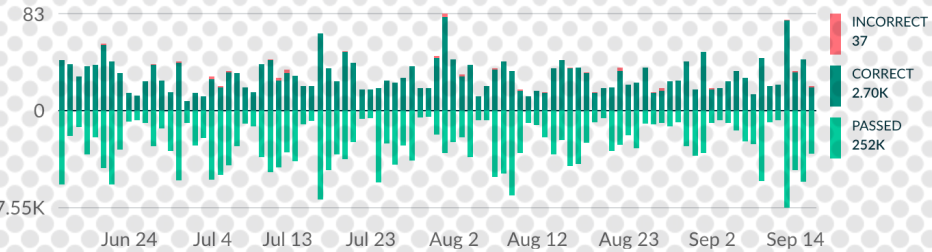
Task/Goal failures predicted
98.7% (2.70K of 2.74K total)

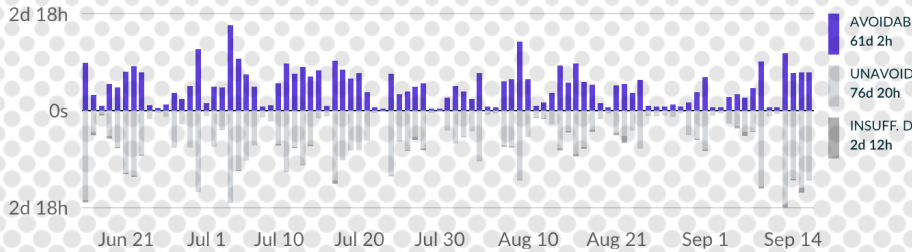
Test failures predicted
96.4% (3.55K of 3.68K total)

Savings potential
61 d 2 hr (44%)

Avoidable tests
1.84M (30%)

Unavoidable tests
4.13M (68%)





s/Goals by mean duration (top 50)

All predictions | Only incorre

	Task/Goal failures predicted	Test failures predicted	Simulations	Mean duration	Total test time	Savings potential
g-boot-build › :spring-boot-project:spring-boot-tools:spring-b...	15 / 16 (93.8%)	15 / 17 (88.2%)	637 / 677	31 min 12 sec	13 d 14 hr	7 d 9 hr (54%)
g-boot-build › :spring-boot-project:spring-boot-tools:spring-b...	84 / 87 (96.6%)	86 / 89 (96.6%)	612 / 661	18 min 18 sec	7 d 10 hr	5 d 2 hr (69%)

SPEED UP WHAT CAN'T BE SKIPPED

Test Parallelization

- Use max power of local machine
- (Yes, your boss should buy you the bleeding edge)

Task path		Started after	Duration
:clean		0.499s	0.053s
:compileJava		0.553s	0.146s
:processResources	NO-SOURCE	0.699s	0.001s
:classes		0.700s	0.000s
:jar		0.701s	0.040s
:assemble		0.741s	0.000s
:compileTestJava		0.741s	0.242s
:processTestResources	NO-SOURCE	0.984s	0.000s
:testClasses		0.984s	0.001s
:test		0.985s	1m 59.135s
:check		2m 0.120s	0.001s
:build		2m 0.121s	0.001s

Task path		Started after	Duration
:clean		0.416s	0.048s
:compileJava		0.465s	0.085s
:processResources	NO-SOURCE	0.550s	0.000s
:classes		0.550s	0.000s
:jar		0.551s	0.040s
:assemble		0.591s	0.000s
:compileTestJava		0.592s	0.212s
:processTestResources	NO-SOURCE	0.804s	0.001s
:testClasses		0.805s	0.000s
:test		0.805s	10.553s
:check		11.359s	0.000s
:build		11.359s	0.000s

```
tasks.test { this: Test!
    onlyIf { true }
    useJUnitPlatform()
    maxParallelForks = Runtime.getRuntime().availableProcessors()
    testLogging { this: TestLoggingContainer
```

TEST DISTRIBUTION

- CI uses fan-out to speed-up tests
- Shouldn't you enjoy it for local tests?
- Use the cloud to distribute test load
- RUN ALL THE TESTS!

Why not just using CI fan-out?

- × Relying on shared CI infrastructure
- × CI infrastructure is not optimized for real-time feedback!
- × Are the agents as fast as they can be?

DON'T LET IT SLIDE

Observe and improve

- Measure local build times across time and environments
- Detect downfacing trends
- Find root causes and improve

Gradle Enterprise

Build Scans | Performance | Trends | Failures | Tests | Predictive Test Selection | Sign in

Basic search | Advanced search

User | Hostname | Project | Requested tasks/goals/targets | Build tool | Build tool version

Start time: Jan 16 2020 02:00 CDT — Feb 2 2020 23:59 CDT | Relative | Fixed

Custom values: git branch name=sam/performance-scenario

Tags | Outcome | Refresh

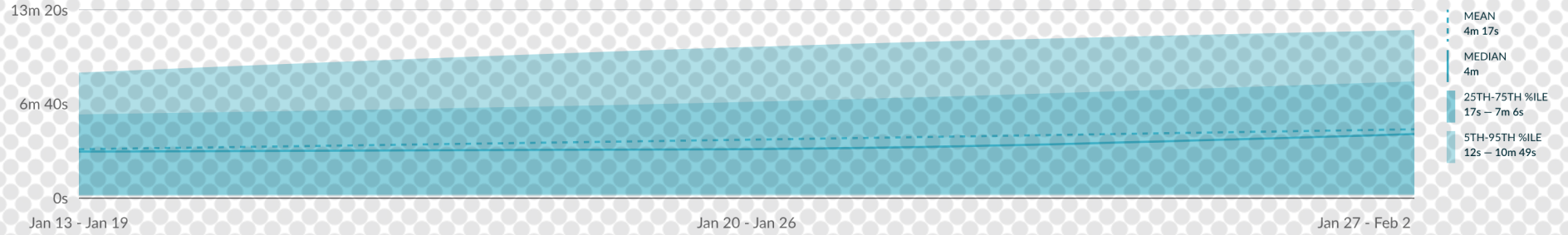
Overview | Build time | Serial execution | Avoidance savings | Build cache overhead | Dependency downloading

Day | Week | Month

Build time
4 min 17 sec

Non execution
28 sec

Execution
3 min 49 sec



Serial execution
35 min 41 sec (7.7x)

Non-cacheable
44 sec

Build cache miss
34 min 40 sec

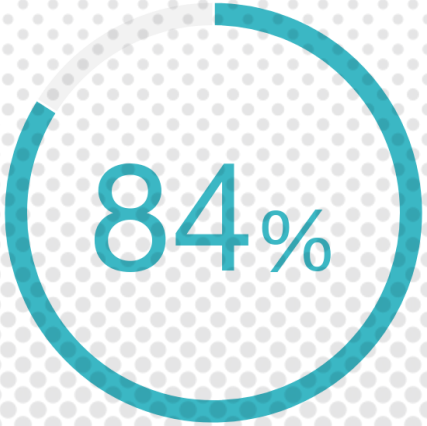
Build cache hit
4.2 sec

Up to date & non-actionable
13 sec

THE GAINS ARE REAL!

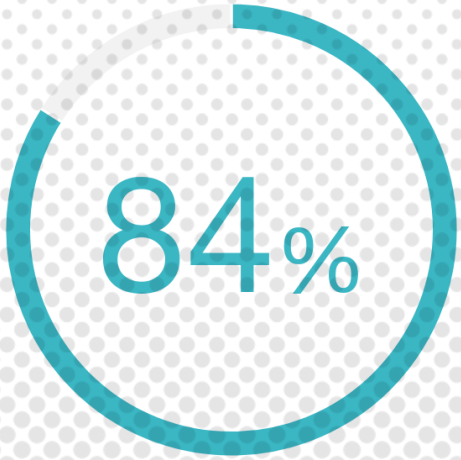
DPE Dramatically Improves Productivity

Almost every surveyed IT organization agreed that “Since integrating Developer Productivity Engineering into our development process, the time savings we experienced on build and test cycle times have dramatically improved developer productivity.”



DPE Fosters Developer Joy

84% of surveyed users agree that DPE's impact on their toolchain makes their job more enjoyable.



TechValidate
by SurveyMonkey

Validated Published: Sep. 25, 2023 TVID: 930-05A-A5F

Developer Productivity is the next frontier

- We figured out (most of) DevOps
- If you want to excel in your production environment, you know how
- But what about your path to production?
- There is work to be done there.
- DPE is the way to go!

LEARN MORE AND TRY IT TODAY!

- Take the Gradle/Maven Speed Challenge!
- Be DPE Agent of Change!
- Read the DPE Handbook!
- Watch the DPE Summit keynotes!

SPEAKING.JBARU.CH

$500 OFF THE EARLY BIRD PRICING! USE CODE DV5BARW

DEVOPS VISION

The latest insights, tools, and best practices in the DevOps world.

December 2 - 4, 2024 - Clearwater, FL

@JBARUCH #DEVOPSVISION Register #DPE SPEAKING.JBARU.CH

THANKS!

Q&A and ~~Twitter~~ X/Bsky/Mastodon/LinkedIn ads:

- x @jbaruch
- x #DevOpsVision
- x speaking.jbaru.ch