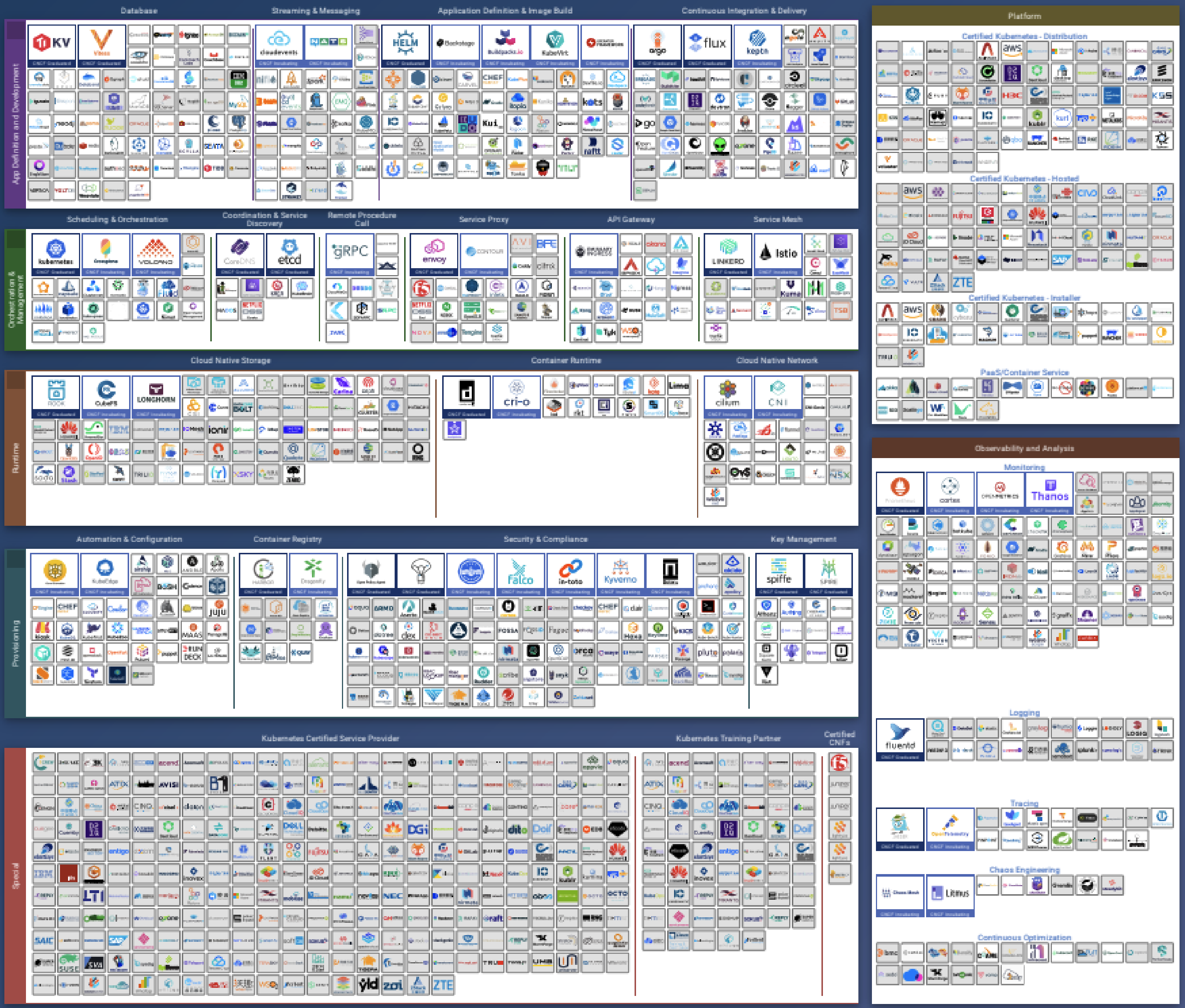
High-performing engineering teams and the Holy Grail

{ }
HTML
</>

CNCF Cloud Native Landscape

1.0

Overwhelmed? Please see the CNCF Trail Map. That and the interactive landscape are at l.cncf.io


Database
Streaming & Messaging
Application Definition & Image Build
Continuous Integration & Delivery
App Definition and Development
Scheduling & Orchestration
Coordination & Service Discovery
Remote Procedure Call
Service Proxy
API Gateway
Service Mesh
Orchestration & Management
Cloud Native Storage
Container Runtime
Cloud Native Network
Runtime
Automation & Configuration
Container Registry
Security & Compliance
Key Management
Provisioning
Kubernetes Certified Service Provider
Kubernetes Training Partner
Certified CNFs
Special
Platform
Certified Kubernetes - Distribution
Certified Kubernetes - Hosted
Certified Kubernetes - Installer
PaaS/Container Service
Observability and Analysis
Monitoring
Logging
Tracing
Chaos Engineering
Continuous Optimization
CNCF Graduated
CNCF Incubating

Jeremy Meiss

circleci

Director, DevRel & Community

🐦 @IAmJerdog

So back to the tech industry....

YOU SEEK THE HOLY GRAIL.

Forrester 2021 Total Economic Empact study

Using best-in-class CI/CD platforms can provide:

- $7.8 million saved from shorter software development cycles.
- $4.3 million recuperated in lost developer productivity.
- 50% decrease in annual infrastructure spend.
- $1.7 million estimated value of improved code quality.

puppet labs
201
STATE
DevOps
REPORT
2016
State of DevOps Report
Presented by:
puppet
2017
State of DevOps Report
Presented by:
puppet + DORA
DEVOPS RESEARCH & ASSESSMENT
Sponsored by:
Atlassian
amazon web services
splunk>
WAVEFRONT by vmware
Hewlett Packard Enterprise
Deloitte.
Electric Cloud
2018
State of DevOps Report
Presented by:
puppet + splunk>
Sponsored by:
aws
cloudability
CYBERARK
Cognizant
diaxion
2019
State of DevOps Report
Presented by
puppet
circleci
splunk>
Sponsored by
ANITIAN
servicenow

Image: *Risk Culture*

THE HOLY HAND GRENADE FOR HIGH-PERFORMING ENGINEERING TEAMS

CI/CD Benchmarks for high-performing teams

Duration

Mean time to recovery

Success rate

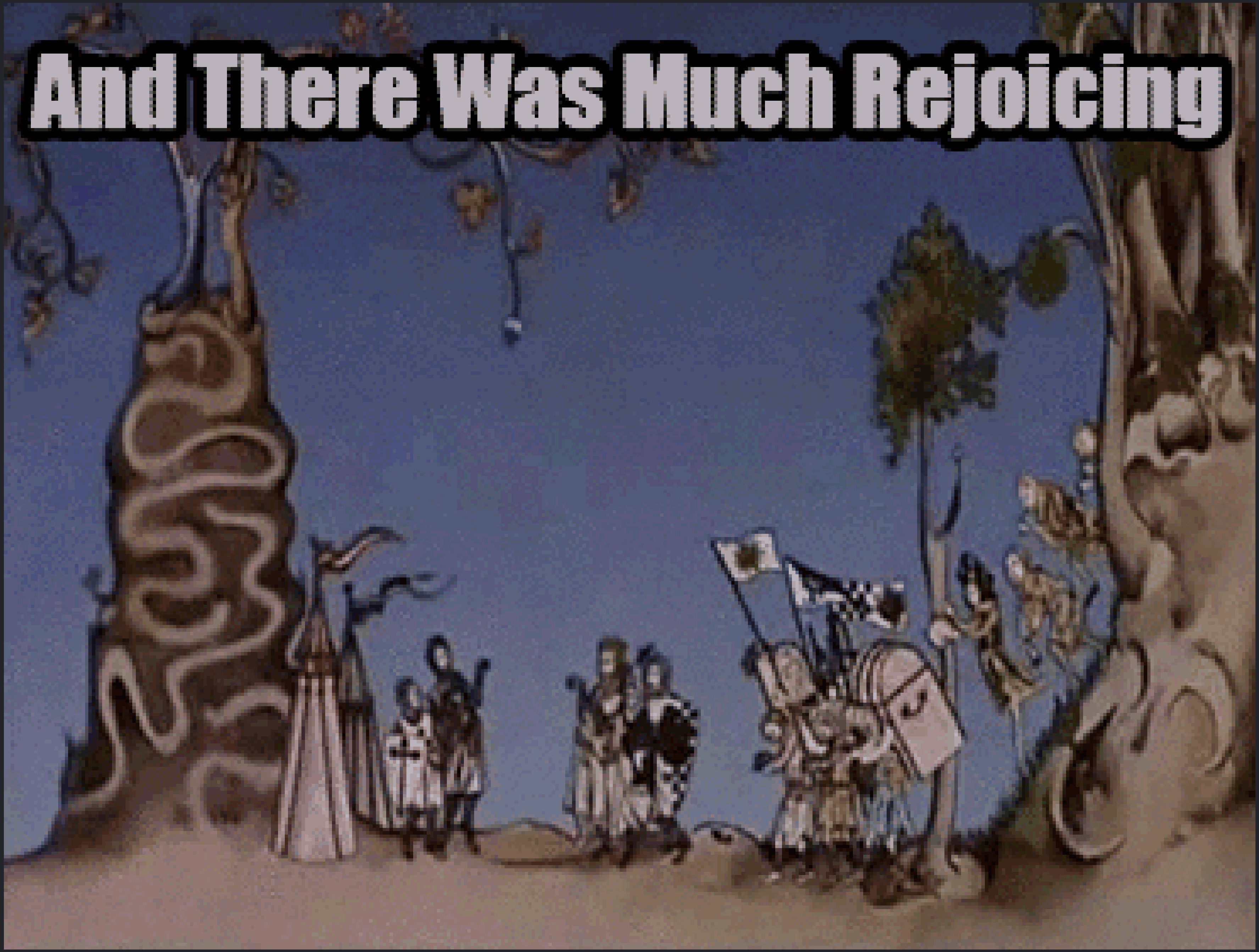
Throughput

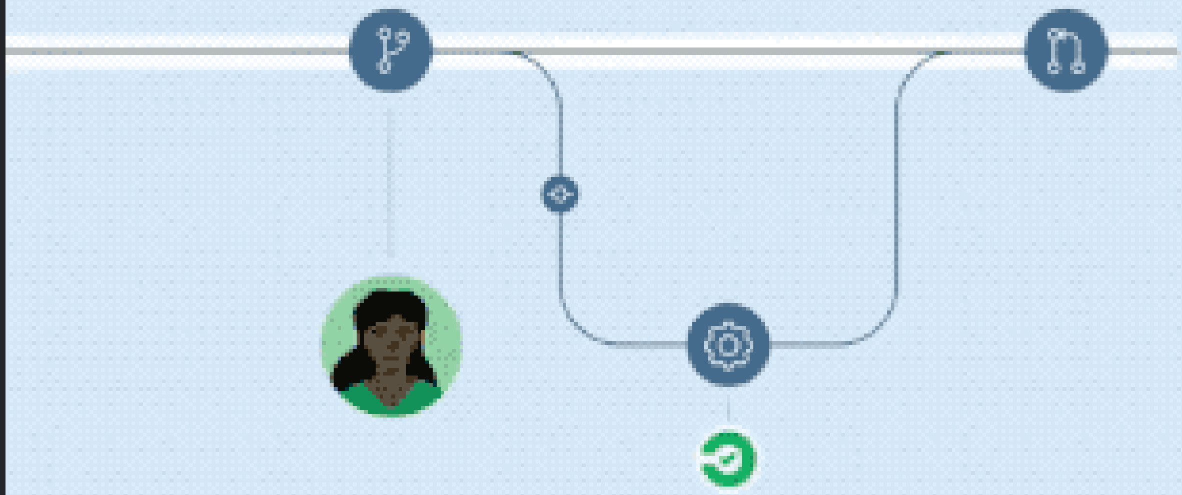
So what does the data say?

Duration

the foundation of software engineering velocity, measures the average time in minutes required to move a unit of work through your pipeline

GIPHY


And There Was Much Rejoicing



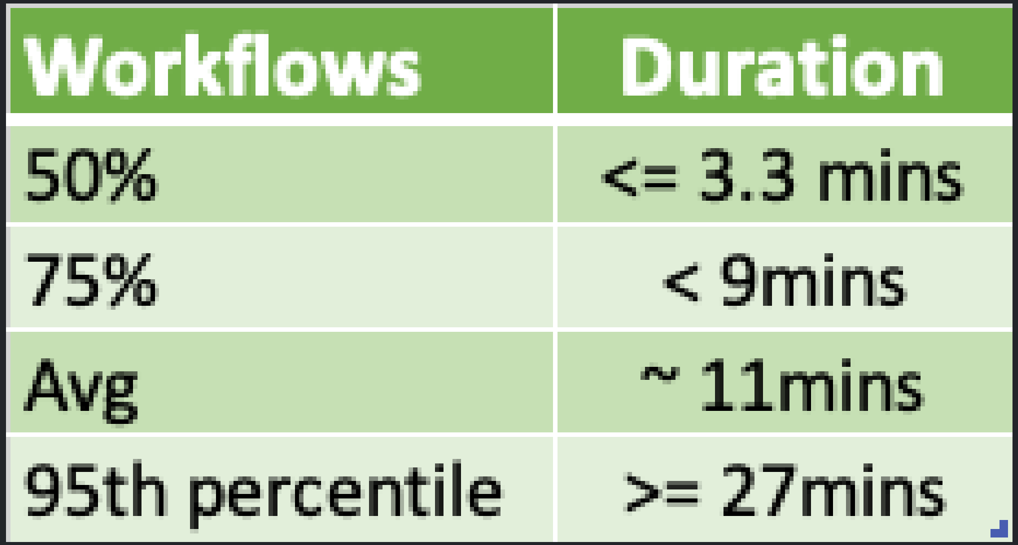
So what is an ideal Duration?

<=10 minute builds

"a good rule of thumb is to keep your builds to no more than ten minutes. Many developers who use CI follow the practice of not moving on to the next task until their most recent checkin integrates successfully. Therefore, builds taking longer than ten minutes can interrupt their flow."

-- Paul M. Duvall (2007). *Continuous Integration: Improving Software Quality and Reducing Risk*

Duration: What the data shows



Workflows	Duration
50%	<= 3.3 mins
75%	< 9mins
Avg	~ 11mins
95th percentile	>= 27mins

Benchmark: 5-10mins

”Why so much lower than the Duration benchmark?”

Improving test coverage

- Add unit, integration, UI, and end-to-end testing across all app layers
- Incorporate code coverage tools into pipelines to identify inadequate testing
- Include static and dynamic security scans to catch vulnerabilities
- Incorporate TDD practices by writing tests during design phase

Optimizing your pipelines

- Use **test splitting** and **parallelism** to execute multiple tests simultaneously
- Cache dependencies and other data to avoid rebuilding unchanged portions
- Use Docker images custom made for CI environments
- Choose the right machine size for your needs

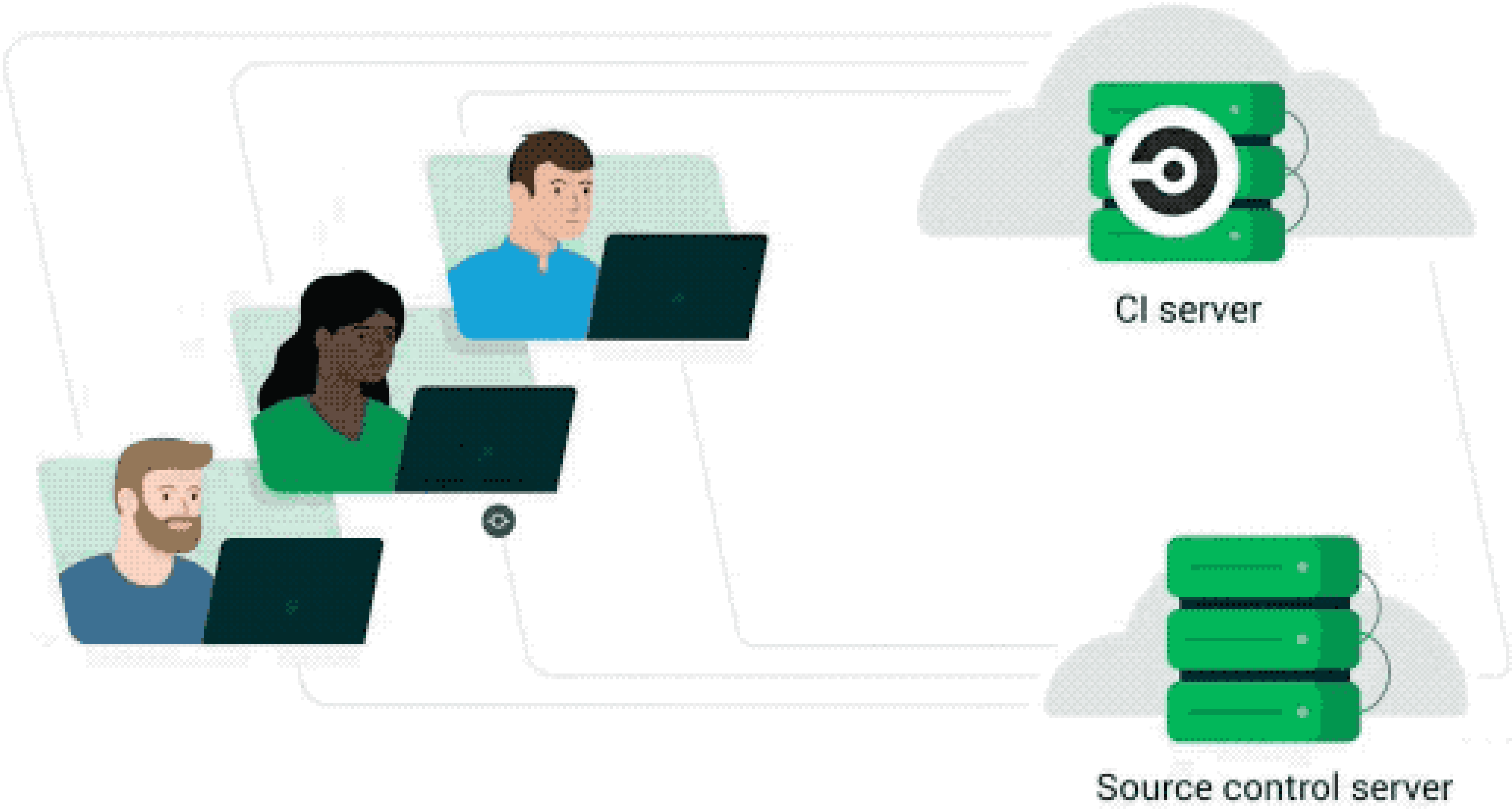
Duration and the Platform Team

- Identify and eliminate impediments to developer velocity
- Set guardrails and enforce quality standards across projects
- Standardize test suites and CI pipeline configs, i.e. shareable config templates and policies
- Welcome failed pipelines, i.e. fast failure
- Actively monitor, streamline, and parallelize pipelines across the org

Mean time to Recovery

the average time required to go from a failed build signal to a successful pipeline run

Mean time to recovery is indicative of resilience


CI server
Source control server

"A key part of doing a continuous build is that if the mainline build fails, it needs to be fixed right away. The whole point of working with CI is that you're always developing on a known stable base."

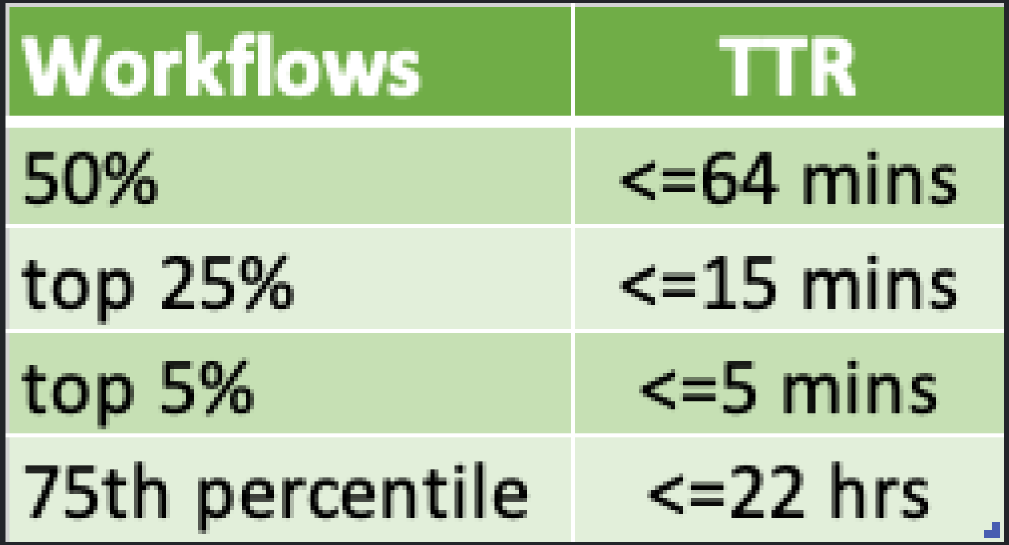
-- Fowler, Martin. "Continuous Integration." Web blog post. *MartinFowler.com*. 1 May 2006. Web.

Build begins....

So what MTTR is ideal?

<=60min MTTR on
default branches

MTTR: What the data shows



Workflows	TTR
50%	<=64 mins
top 25%	<=15 mins
top 5%	<=5 mins
75th percentile	<=22 hrs

Benchmark: 60mins

”10 minutes is a striking improvement - what happened?”

Two factors impacting reduced MTTR

- Economic pressures in the macro environment + rising competition in the micro environment, forcing teams to prioritize product stability and reliability over growth
- High performers increasingly rely on platform teams to achieve steadier and more resilient development pipelines with built-in recovery mechanisms.

Treat your default branch as the lifeblood of your project

Getting to faster recovery times

- Set up instant alerts for failed builds using services like Slack, Twilio, or Pagerduty.
- Write clear, informative error messages for your tests that allow you to quickly diagnose the problem and focus your efforts in the right place.
- SSH into the failed build machine to debug in the remote test environment. Doing so gives you access to valuable troubleshooting resources, including log files, running processes, and directory paths.

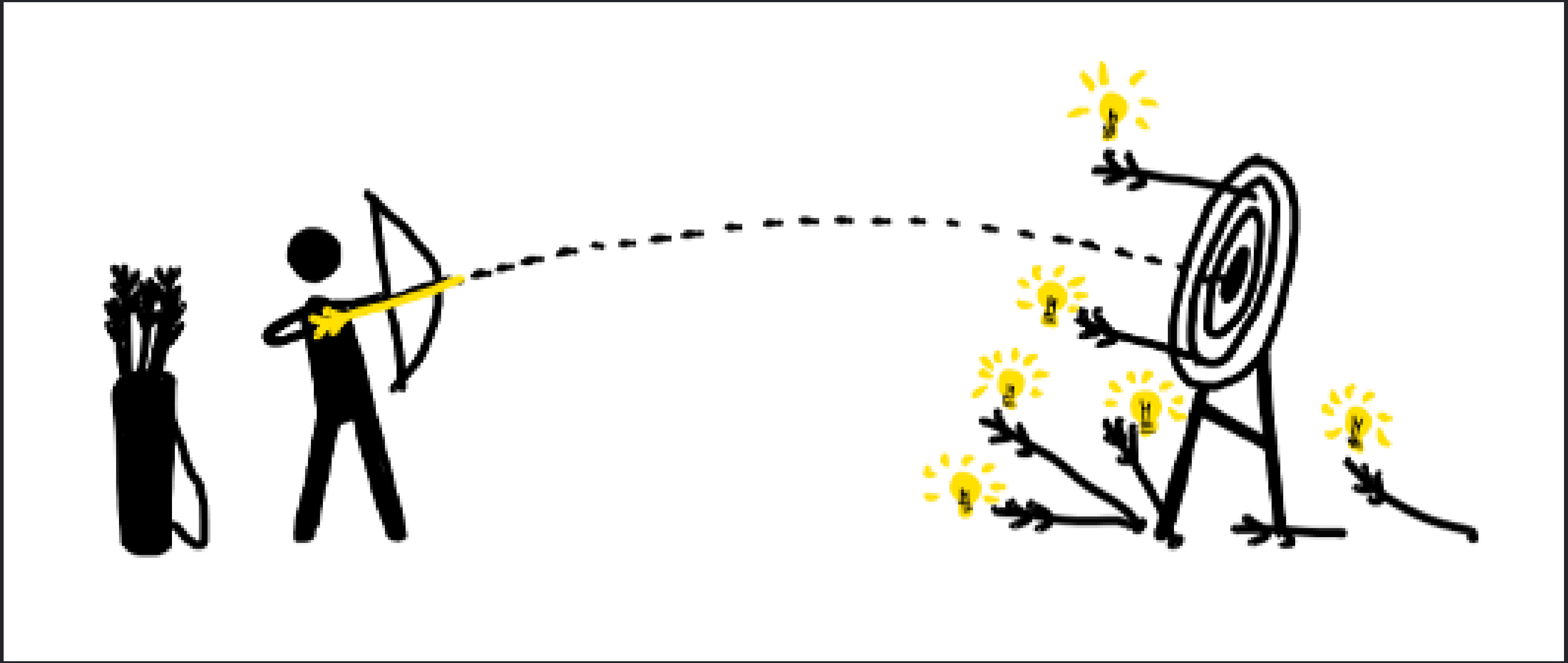
MTTR and the Platform Team

- Ephasise the value of deploy-ready, default branches, with clear processes & expectations for failure recovery across all projects
- Set up effective monitoring and alerting systems, and track recovery time
- Limit frequency and severity of broken builds with role-based AC and config policies
- Config- and Infrastructure-as-Code tools limit potential for misconfig errors
- Actively monitor, streamline, and parallelize pipelines across the org

Success Rate

number of passing runs divided by the total number of runs over a period of time

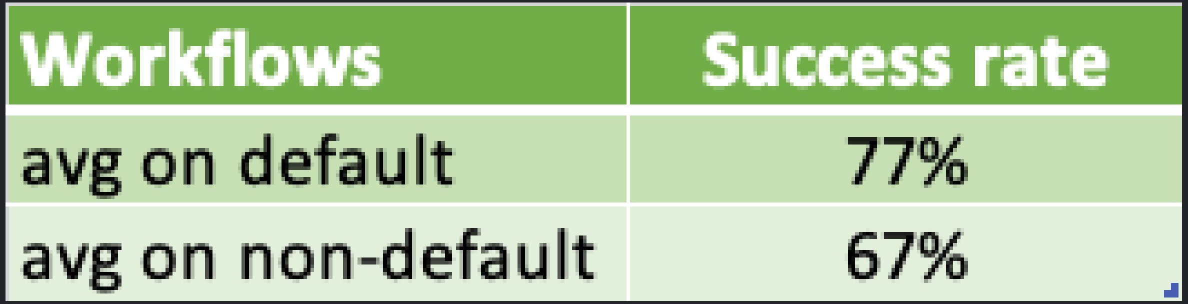
now go away...
...or i will
taunt you a
second time!



So what Success rate is ideal?

90%+ Success rate on default branches

Success rate: What the data shows



Workflows	Success rate
avg on default	77%
avg on non-default	67%

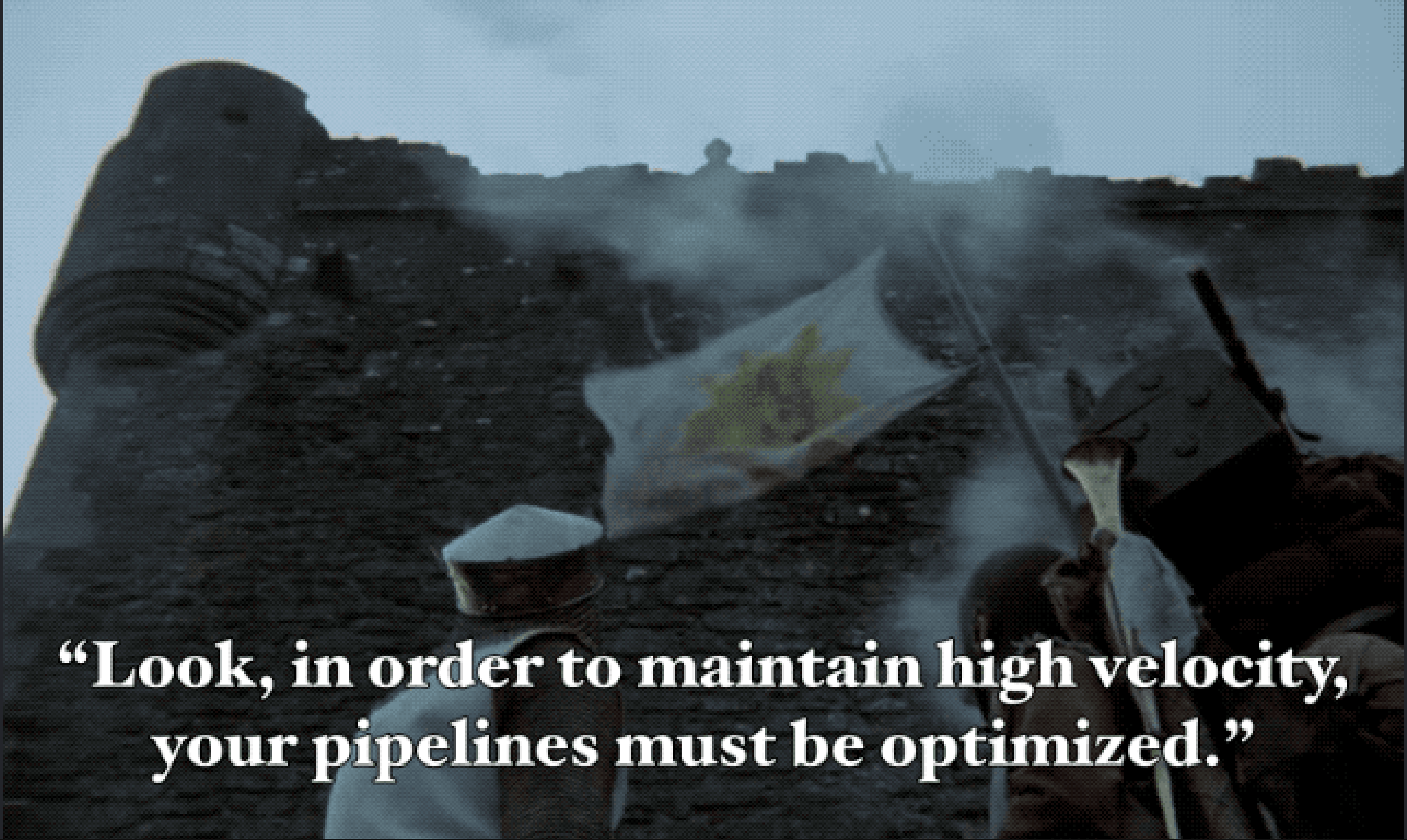
Benchmark: 90%+ on default

Success rate and the Platform Team

- With low success rates, look at your MTTR and shorten recovery time first
- Set a baseline success rate, then aim for continuous improvement, looking for flaky tests or gaps in test coverage
- Be mindful of patterns and influence of external factors, i.e. decline on Fridays, holidays, etc.

Throughput

average number of workflow runs that an organization completes on a given project per day



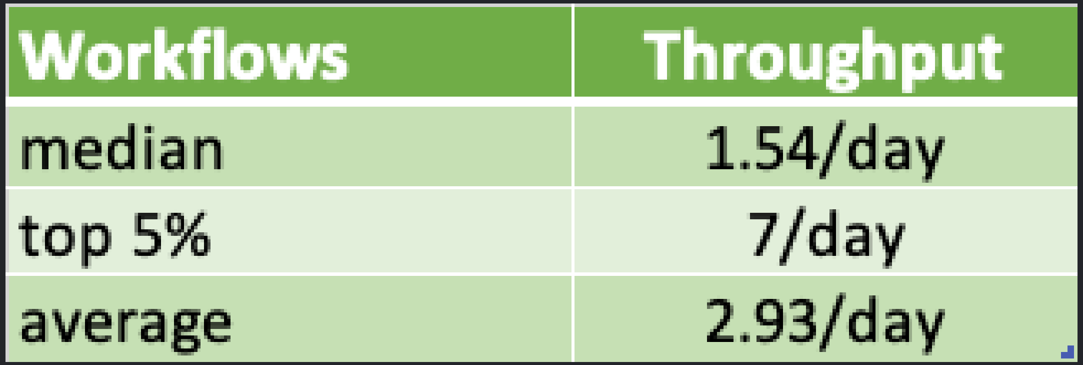
“Look, in order to maintain high velocity, your pipelines must be optimized.”

It's only a model.

So what Throughput is ideal?

It depends.

Throughput: What the data shows



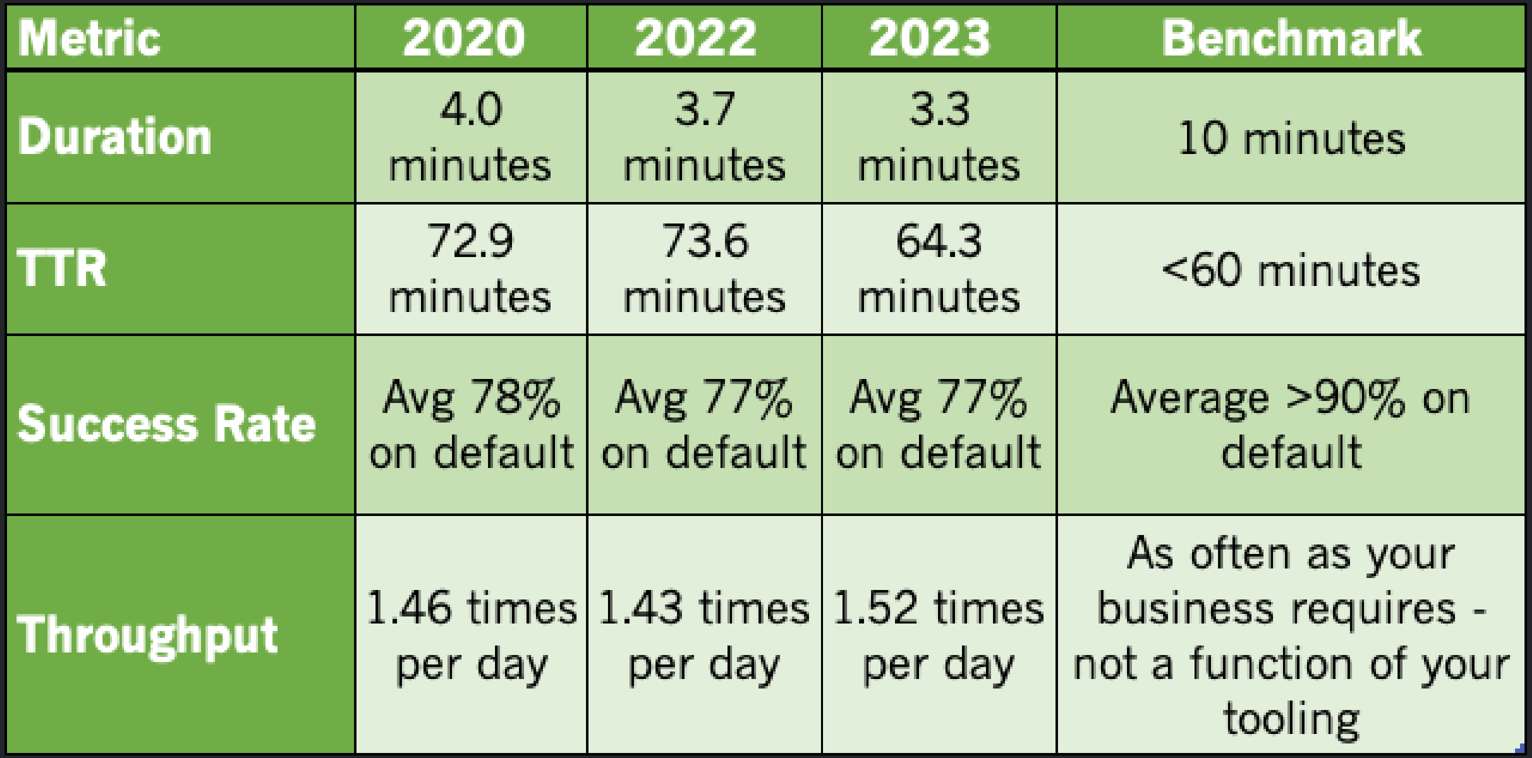
Workflows	Throughput
median	1.54/day
top 5%	7/day
average	2.93/day

Benchmark: at the speed of your business

Throughput and the Platform Team

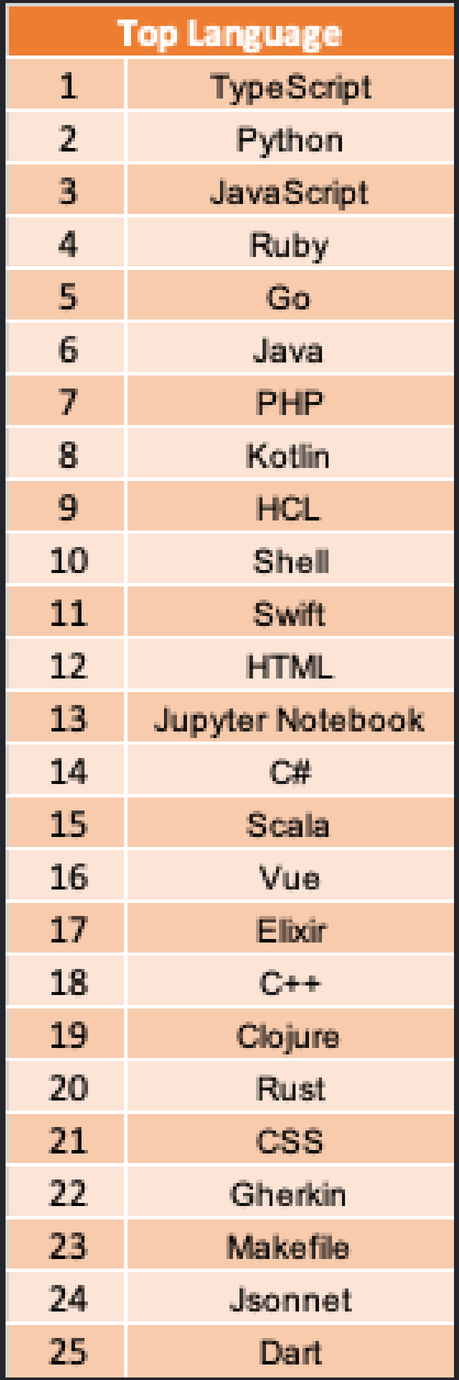
- Map goals to reality of internal and external business situations, i.e. customer expectations, competitive landscape, codebase complexity, etc.
- Capture a baseline, monitor for deviations
- Alleviate as much developer cognitive load from day-to-day work

High-Performing Teams in 2023

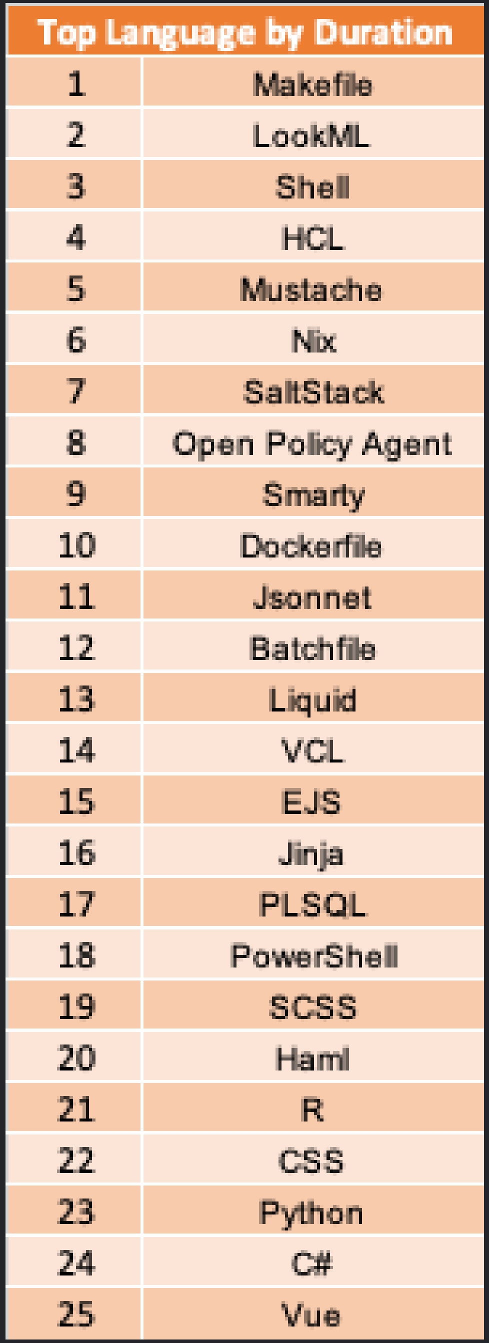


Metric	2020	2022	2023	Benchmark
Duration	4.0 minutes	3.7 minutes	3.3 minutes	10 minutes
TTR	72.9 minutes	73.6 minutes	64.3 minutes	<60 minutes
Success Rate	Avg 78% on default	Avg 77% on default	Avg 77% on default	Average >90% on default
Throughput	1.46 times per day	1.43 times per day	1.52 times per day	As often as your business requires - not a function of your tooling

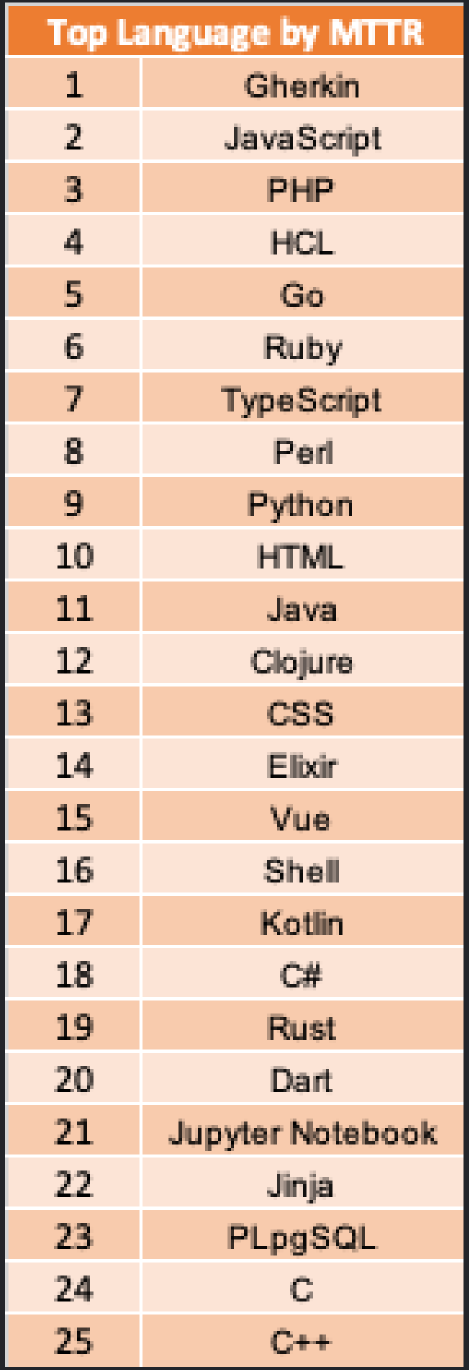
”Surely <insert programming language> helps me achieve the ”Holy Grail”!?”



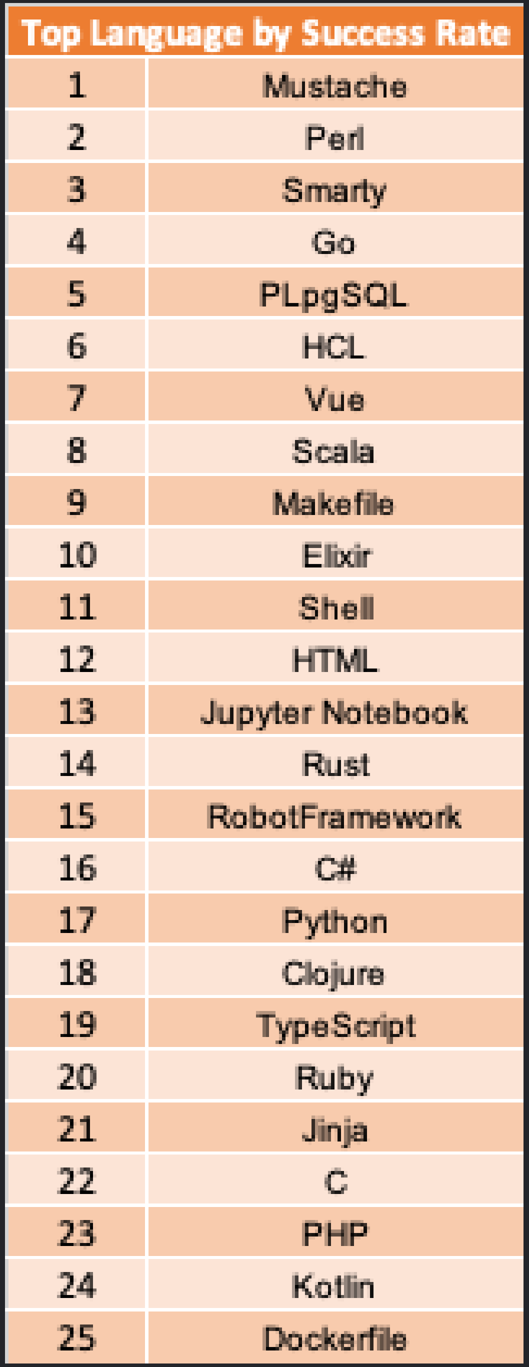
Top Language	
1	TypeScript
2	Python
3	JavaScript
4	Ruby
5	Go
6	Java
7	PHP
8	Kotlin
9	HCL
10	Shell
11	Swift
12	HTML
13	Jupyter Notebook
14	C#
15	Scala
16	Vue
17	Elixir
18	C++
19	Clojure
20	Rust
21	CSS
22	Gherkin
23	Makefile
24	Jsonnet
25	Dart



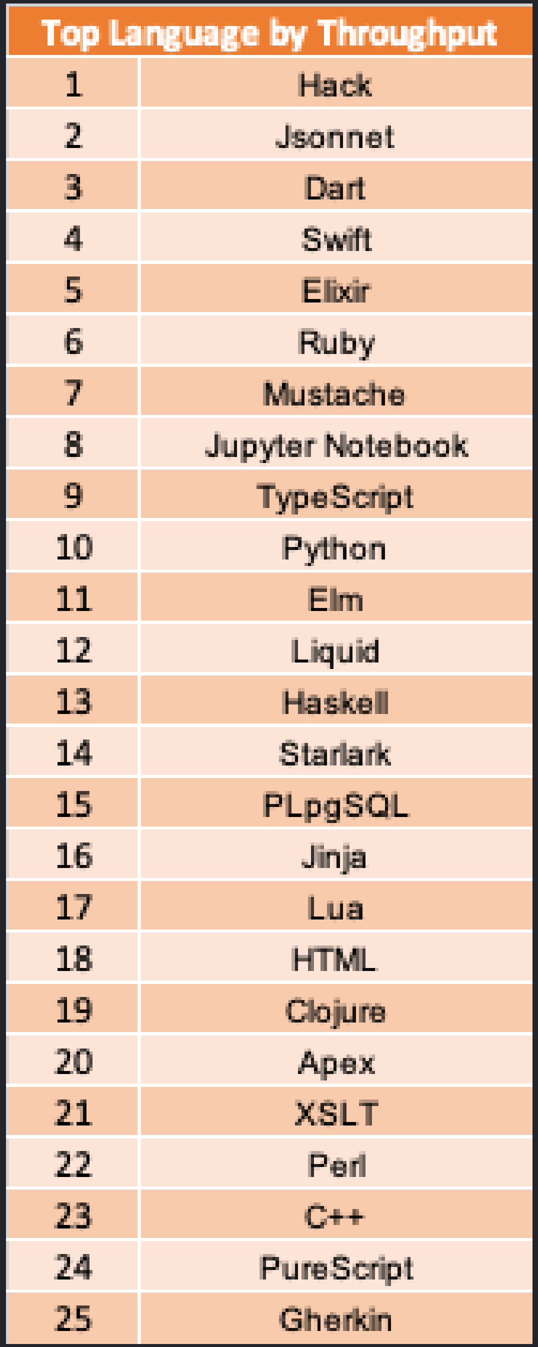
Top Language by Duration	
1	Makefile
2	LookML
3	Shell
4	HCL
5	Mustache
6	Nix
7	SaltStack
8	Open Policy Agent
9	Smarty
10	Dockerfile
11	Jsonnet
12	Batchfile
13	Liquid
14	VCL
15	EJS
16	Jinja
17	PLSQL
18	PowerShell
19	SCSS
20	Haml
21	R
22	CSS
23	Python
24	C#
25	Vue



Top Language by MTTR	
1	Gherkin
2	JavaScript
3	PHP
4	HCL
5	Go
6	Ruby
7	TypeScript
8	Perl
9	Python
10	HTML
11	Java
12	Clojure
13	CSS
14	Elixir
15	Vue
16	Shell
17	Kotlin
18	C#
19	Rust
20	Dart
21	Jupyter Notebook
22	Jinja
23	PLpgSQL
24	C
25	C++



Top Language by Success Rate	
1	Mustache
2	Perl
3	Smarty
4	Go
5	PLpgSQL
6	HCL
7	Vue
8	Scala
9	Makefile
10	Elixir
11	Shell
12	HTML
13	Jupyter Notebook
14	Rust
15	RobotFramework
16	C#
17	Python
18	Clojure
19	TypeScript
20	Ruby
21	Jinja
22	C
23	PHP
24	Kotlin
25	Dockerfile



Top Language by Throughput	
1	Hack
2	Jsonnet
3	Dart
4	Swift
5	Elixir
6	Ruby
7	Mustache
8	Jupyter Notebook
9	TypeScript
10	Python
11	Elm
12	Liquid
13	Haskell
14	Starlark
15	PLpgSQL
16	Jinja
17	Lua
18	HTML
19	Clojure
20	Apex
21	XSLT
22	Perl
23	C++
24	PureScript
25	Gherkin

2020 Report

https://circle.ci/ssd2020

Full 2022 Report

https://circle.ci/ssd2022

Thank You.

For feedback and swag: circle.ci/jeremy

timeline.jerdog.me

IAmJerdog

jerdog

/in/jeremymeiss

@jerdog@hachyderm.io