Hardening

Diego Pacheco

About me...

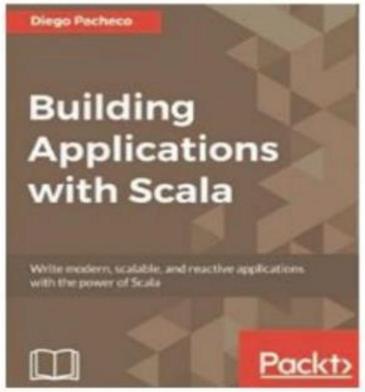
- ❑ Cats Father
- ❑ Software Architect
- ❑ *Agile Coach*
- ❑ *SOA/Microservices Expert*
- ❑ *DevOps Practitioner*
- ❑ *Author*
- ❑ *Speaker*

diegopacheco

@diego_pacheco

http://diego-pacheco.blogspot.com.br/

tinyurl.com/diegopacheco



Diego Pacheco
Building Applications with Scala
Write modern, scalable, and reactive applications with the power of Scala
Packt>

Diego Pacheco
Building Effective Microservices
Explore microservices and their implementation hands-on
Packt>

https://diegopacheco.github.io/

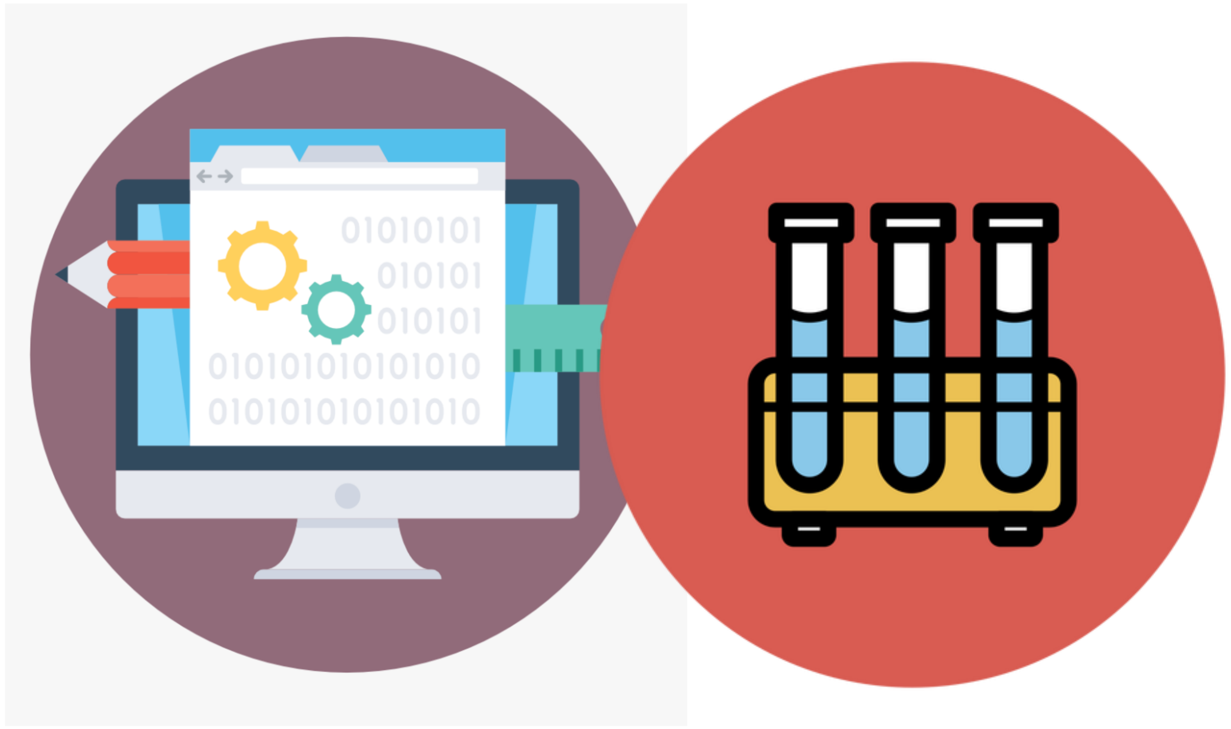
Why Test matters?

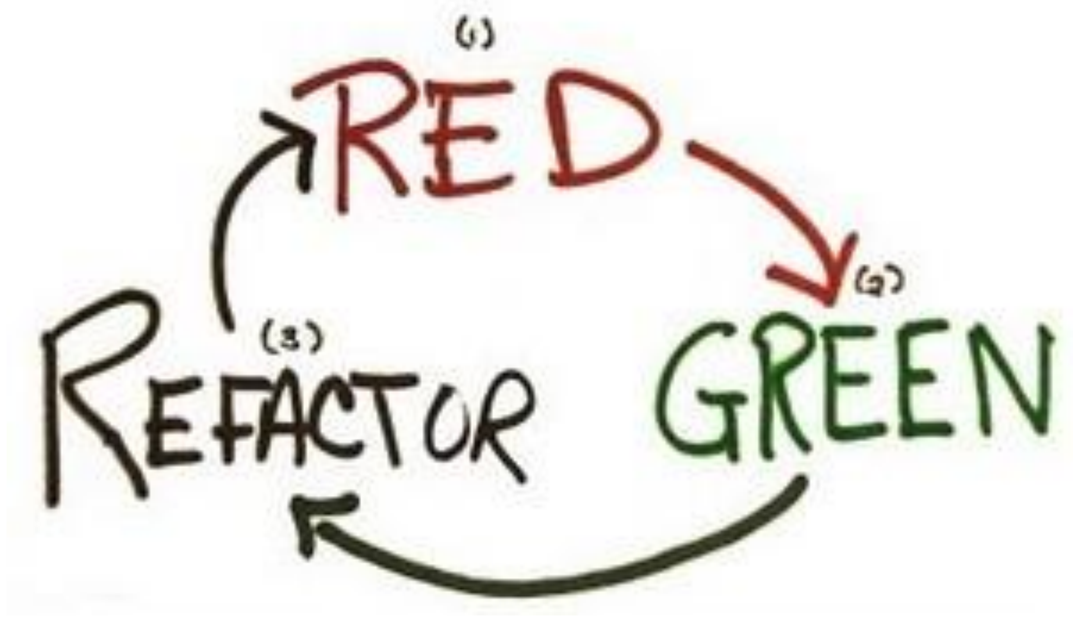
- How do we Know the solution works?
- How do we know still working ?
- How do we know if we are breaking consumers ?
- How do we know we can survive failures in prod ?
- How we do know we are building the right product?

Why Test matters?

- How do we Know the solution works (Confidence)?
- How do we know still working (Automation + Regression)?
- How do we know if we are breaking consumers (Contract Testing)?
- How do we know we can survive failures in prod (Chaos Testing) ?
- How we do know we are building the right product (A/B/n Testing)?

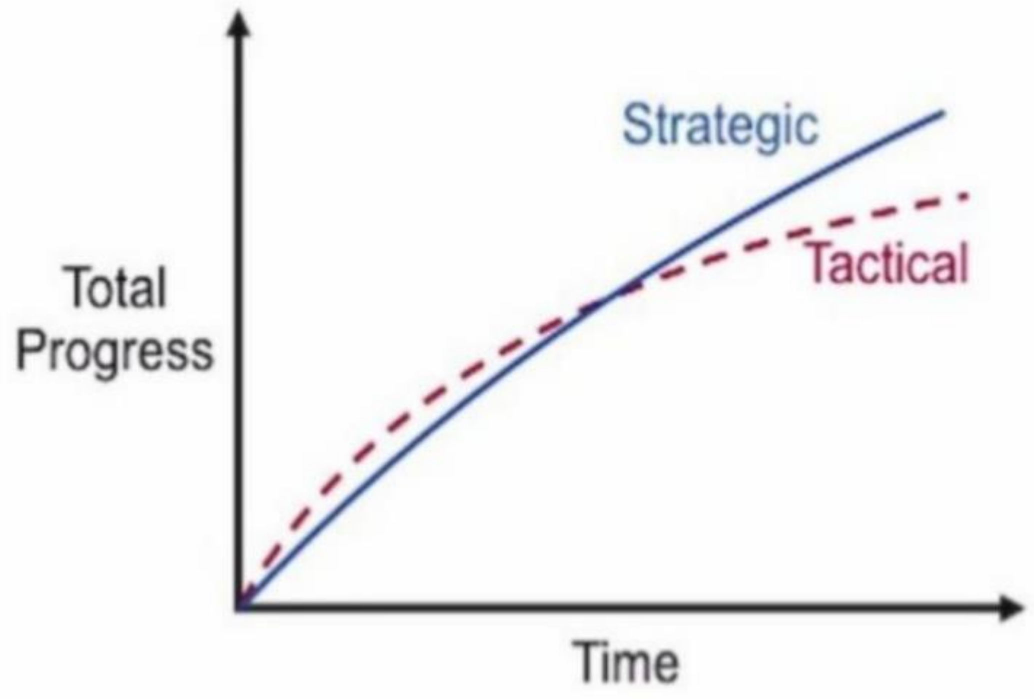
Test & Design Relationship



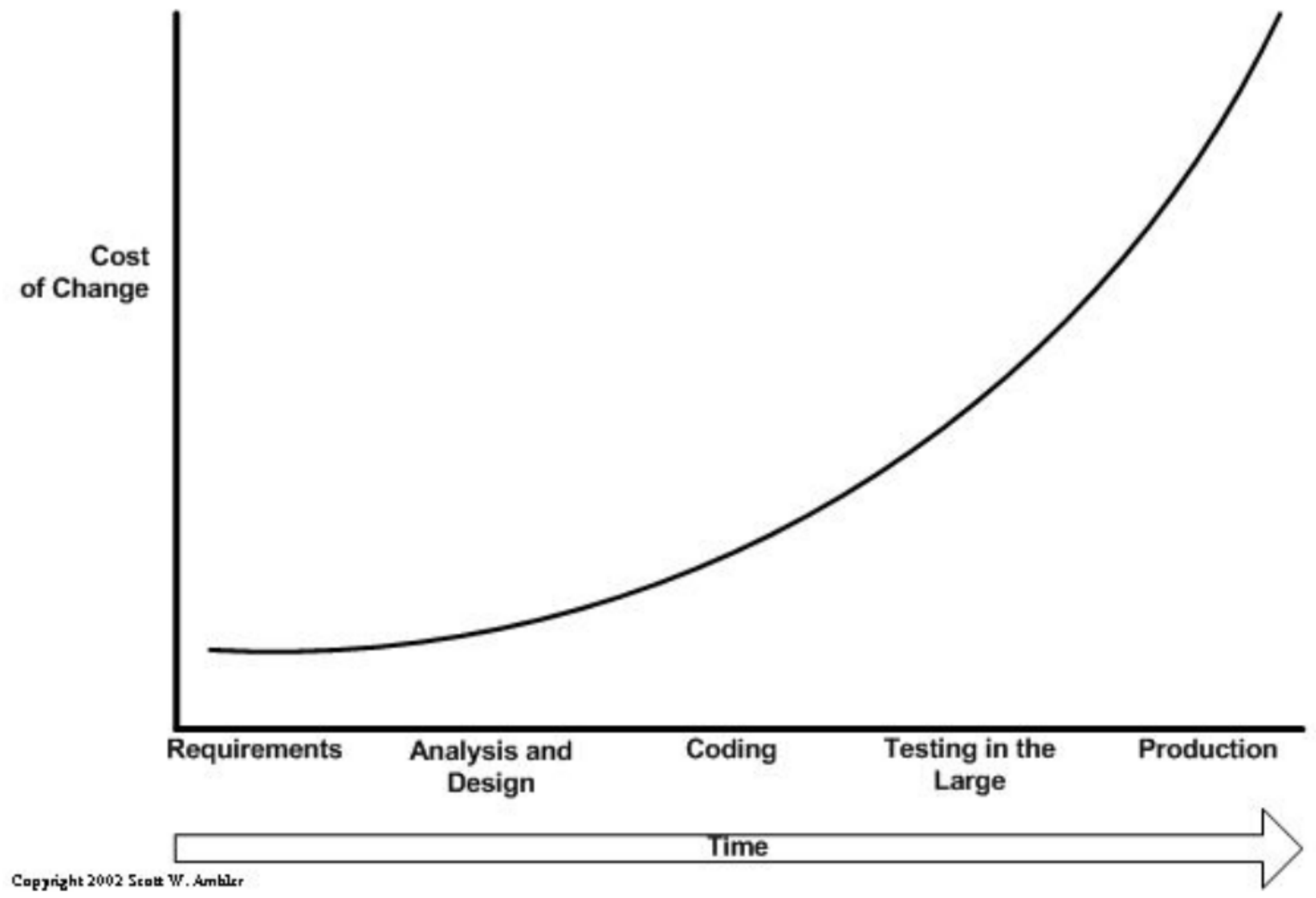


Design is strategic

- **Strategic programming**
  - Goal: produce a great design
  - Simplify future development
  - Minimize complexity
  - Must sweat the small stuff
- **Investment mindset**
  - Take extra time today
  - Pays back in the long run



Why it matters?

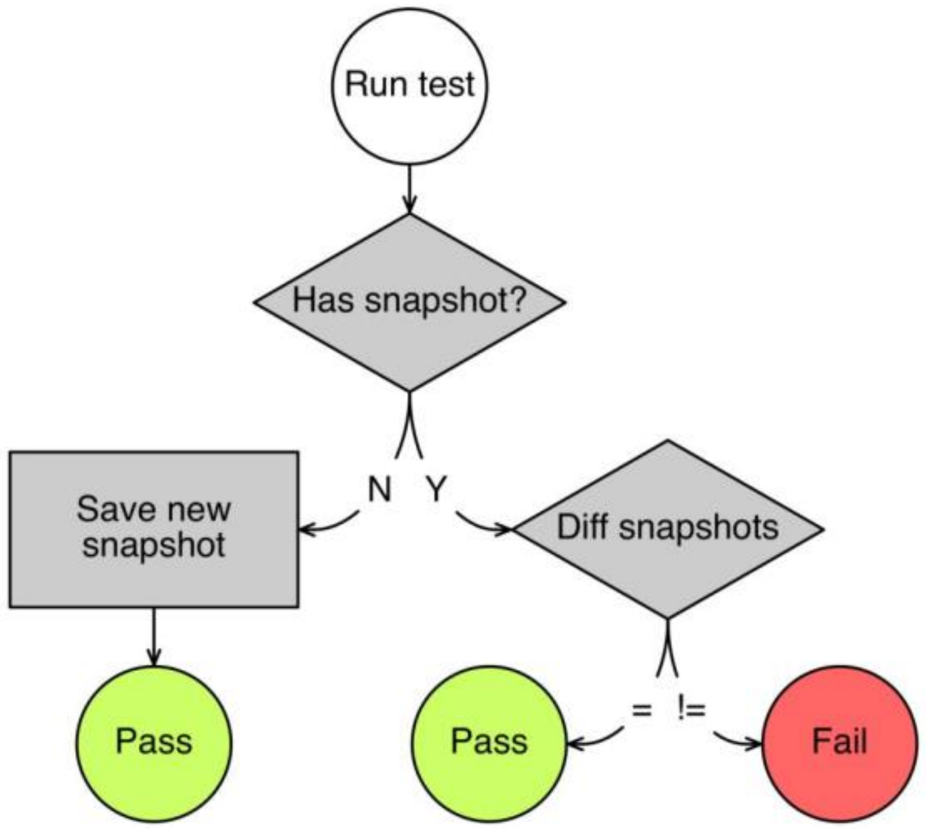


Tooling & Test Diversity

VS

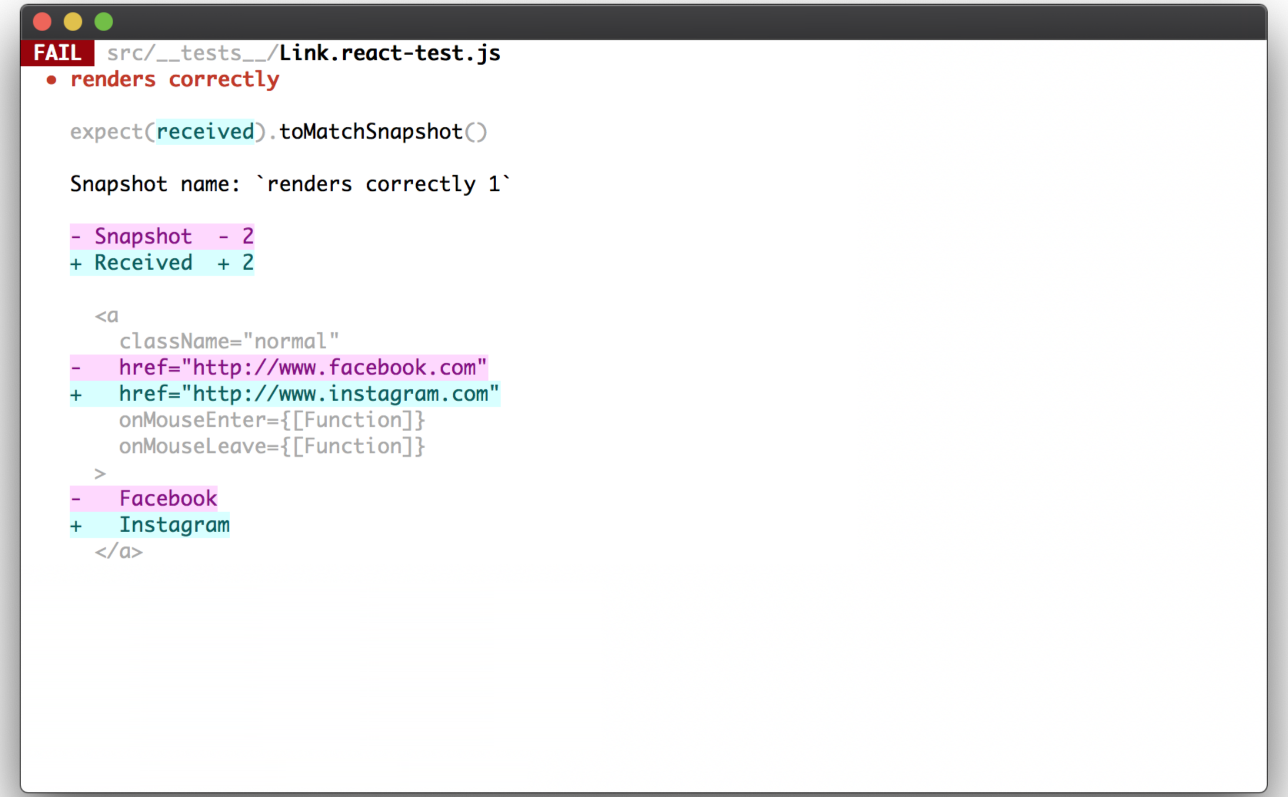
WAIT! HAMMER NO!
QUIET NAIL.

Snapshot Testing



- Enhance Unit Testing for some scenarios.
  - Reduce typing
  - Increase coverage
- Perfect for Design systems / Components.
- Requires good tooling
- Could be great or useless.
- For Frontend: Jest(React) is super awesome.

Snapshot Testing

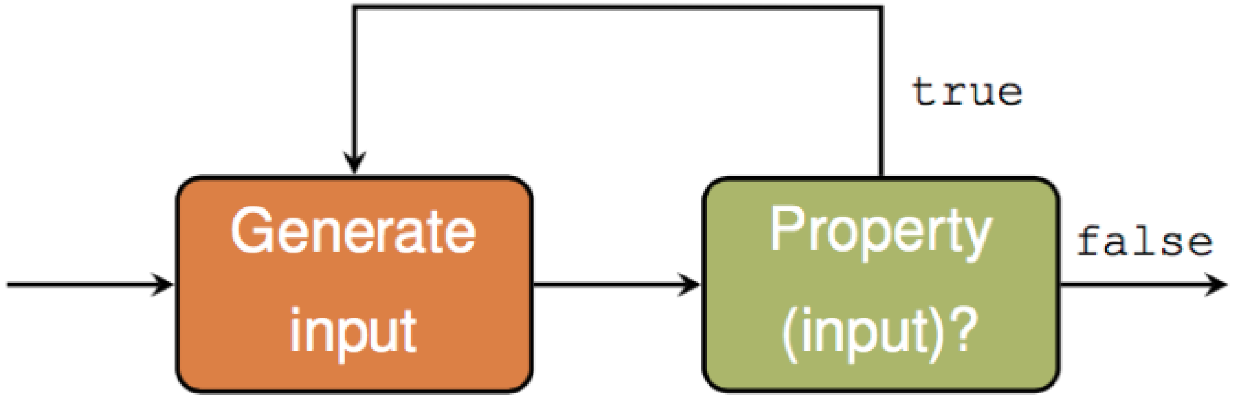


```
FAIL  src/__tests__/Link.react-test.js
 • renders correctly

   expect(received).toMatchSnapshot()

   Snapshot name: `renders correctly 1`

   - Snapshot  - 2
   + Received  + 2

     <a
       className="normal"
   -   href="http://www.facebook.com"
   +   href="http://www.instagram.com"
       onMouseEnter={[Function]}
       onMouseLeave={[Function]}
     >
   -   Facebook
   +   Instagram
     </a>
```

```
it('renders correctly', () => {
  const tree = renderer
    .create(<Link page="https://prettier.io">Prettier</Link>)
    .toJSON();
  expect(tree).toMatchInlineSnapshot();
});
```

https://jestjs.io/

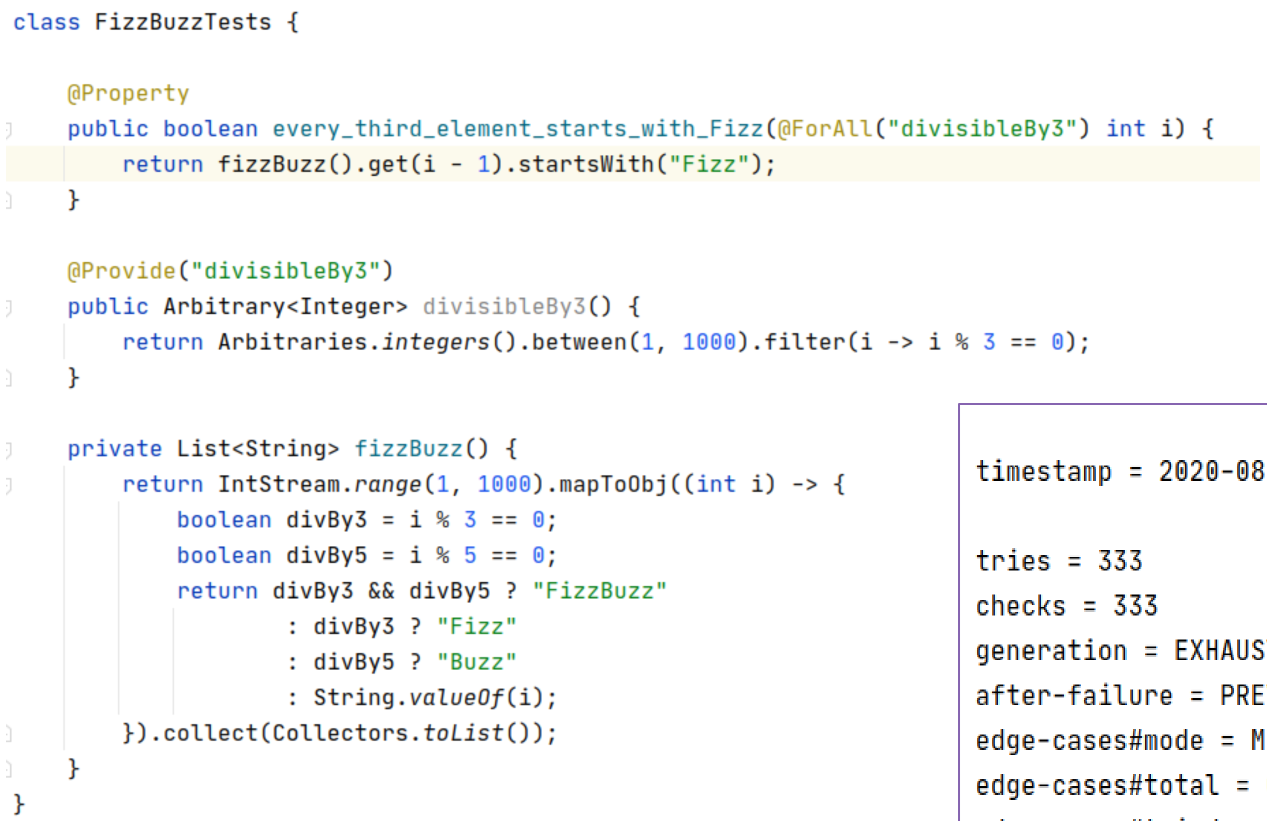
Property-Based Testing



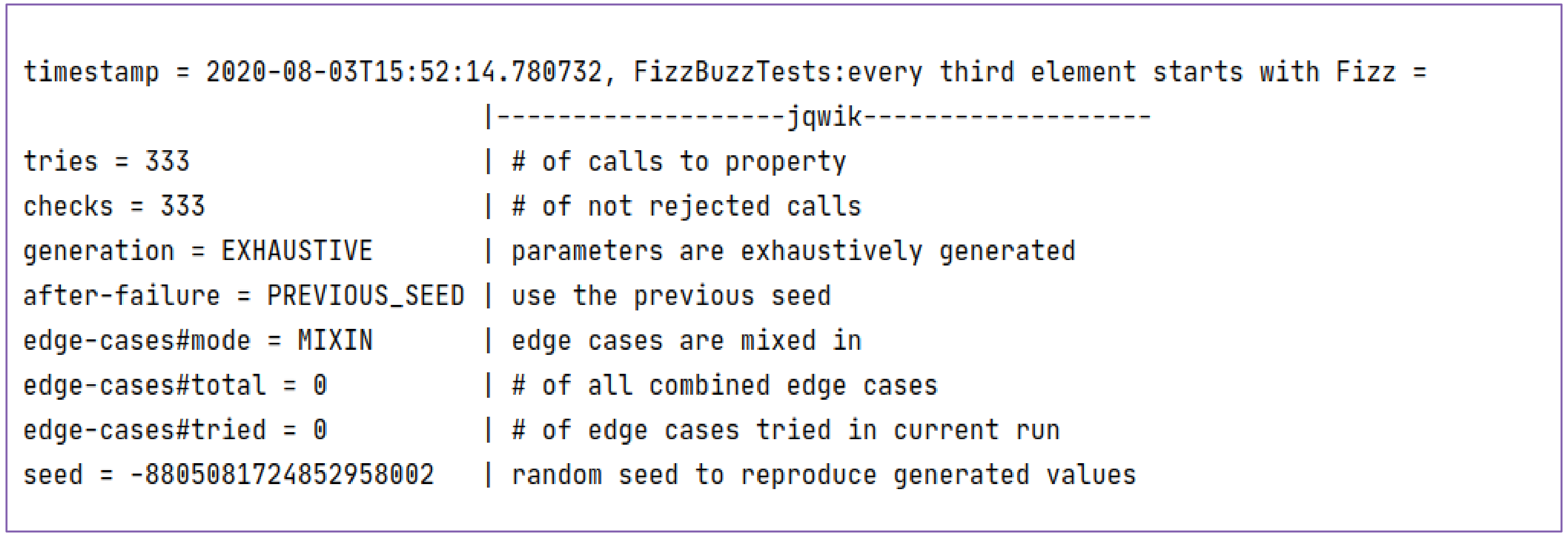
- Enhance Unit Testing for some scenarios.
  - Reduce typing
  - Increase coverage
- Focus on Properties.
- Inputs got Generated.
- From Functional Programing
  - Quick Check Haskell

https://jqwik.net/

Property-Based Testing



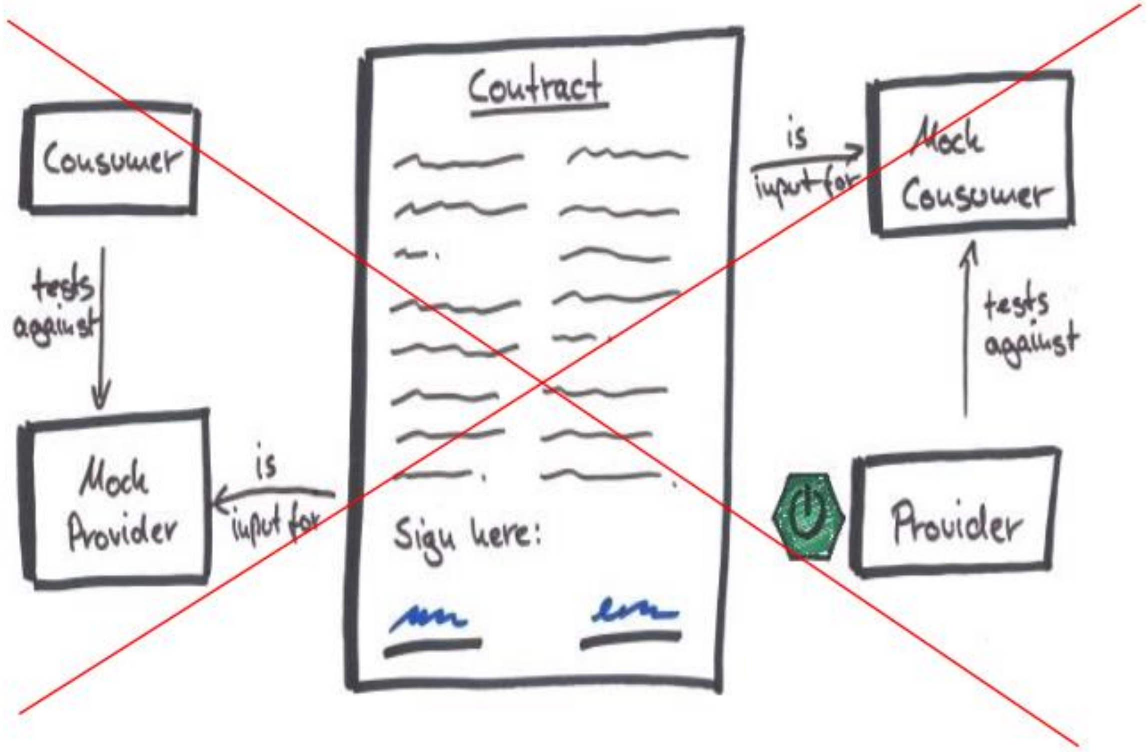
```java
class FizzBuzzTests {

    @Property
    public boolean every_third_element_starts_with_Fizz(@ForAll("divisibleBy3") int i) {
        return fizzBuzz().get(i - 1).startsWith("Fizz");
    }

    @Provide("divisibleBy3")
    public Arbitrary<Integer> divisibleBy3() {
        return Arbitraries.integers().between(1, 1000).filter(i -> i % 3 == 0);
    }

    private List<String> fizzBuzz() {
        return IntStream.range(1, 1000).mapToObj((int i) -> {
            boolean divBy3 = i % 3 == 0;
            boolean divBy5 = i % 5 == 0;
            return divBy3 && divBy5 ? "FizzBuzz"
                    : divBy3 ? "Fizz"
                    : divBy5 ? "Buzz"
                    : String.valueOf(i);
        }).collect(Collectors.toList());
    }
}
```



```
timestamp = 2020-08-03T15:52:14.780732, FizzBuzzTests:every third element starts with Fizz =
                              |-------------------jqwik-------------------
tries = 333                   | # of calls to property
checks = 333                  | # of not rejected calls
generation = EXHAUSTIVE       | parameters are exhaustively generated
after-failure = PREVIOUS_SEED | use the previous seed
edge-cases#mode = MIXIN       | edge cases are mixed in
edge-cases#total = 0          | # of all combined edge cases
edge-cases#tried = 0          | # of edge cases tried in current run
seed = -8805081724852958002   | random seed to reproduce generated values
```

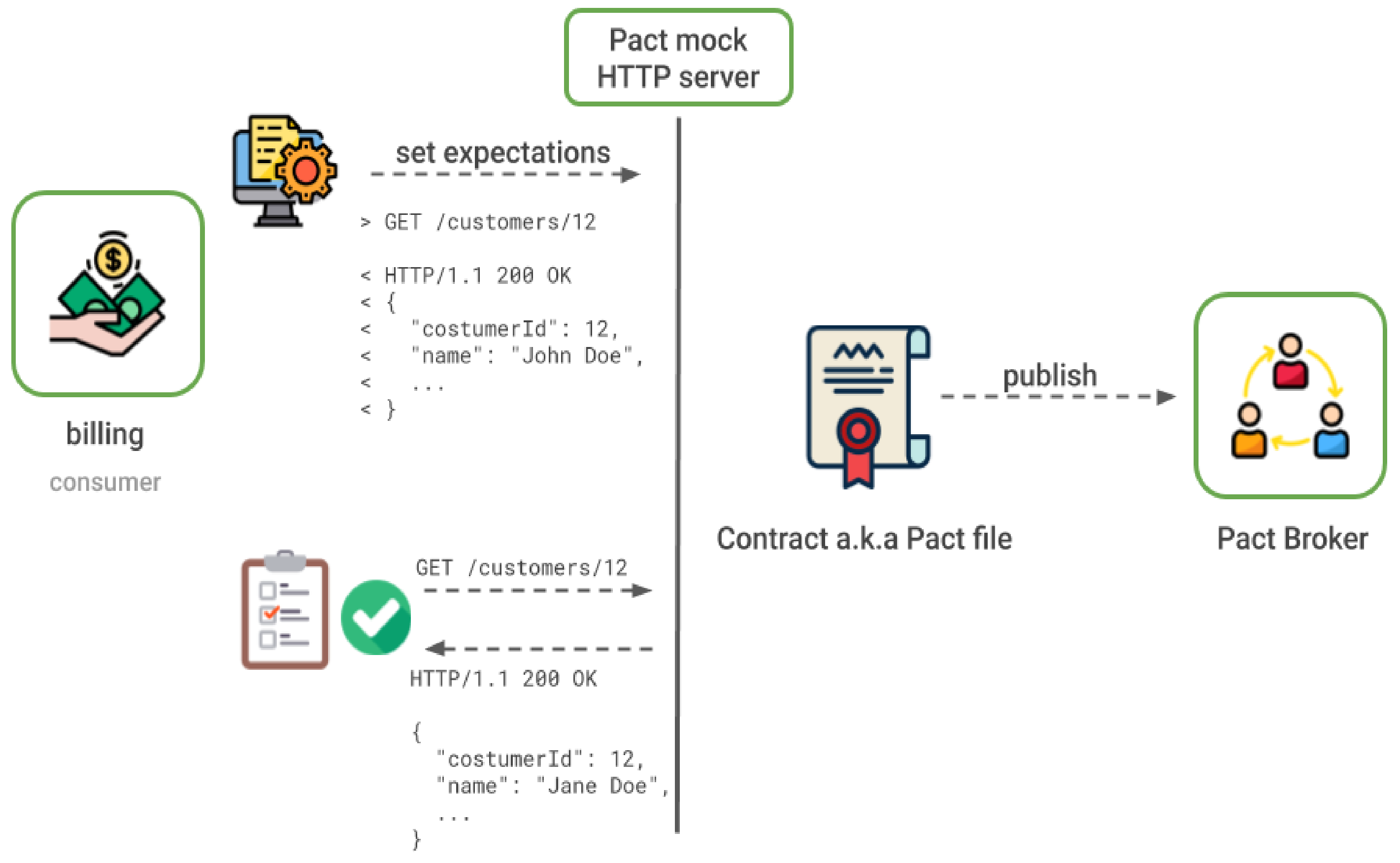
https://jqwik.net/

Contract Testing



- Perfect for SOA / Microservices
- Consumer Driven Tests
  - Provider
  - Consumer
  - Re-use
- Does not depend on specific tool
- Junit + mocks (provider)

Contract Testing

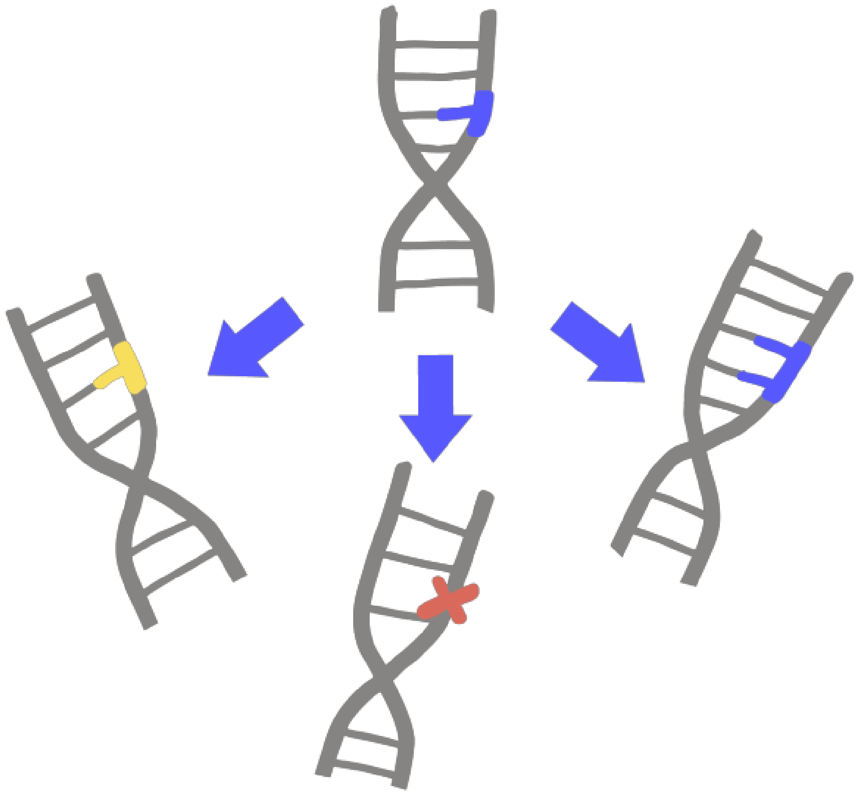


```
> GET /customers/12

< HTTP/1.1 200 OK
< {
<    "costumerId": 12,
<    "name": "John Doe",
<    ...
< }
```

```
GET /customers/12

HTTP/1.1 200 OK

{
  "costumerId": 12,
  "name": "Jane Doe",
  ...
}
```

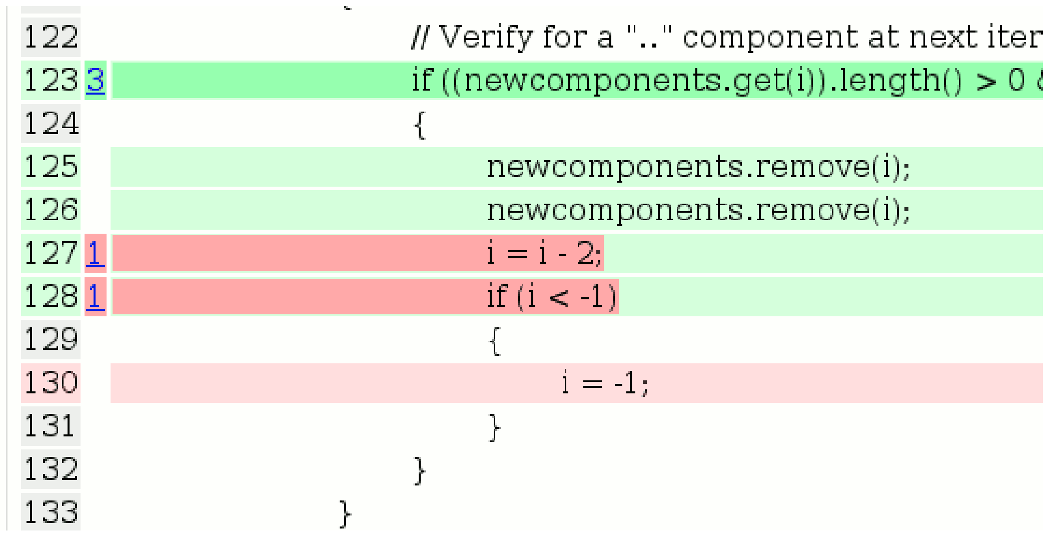
- Service Provider should know they consumers
  - How do you the right design?
  - How we do proper gov?
- Other options: Swagger / OpenAPI

Mutation Testing



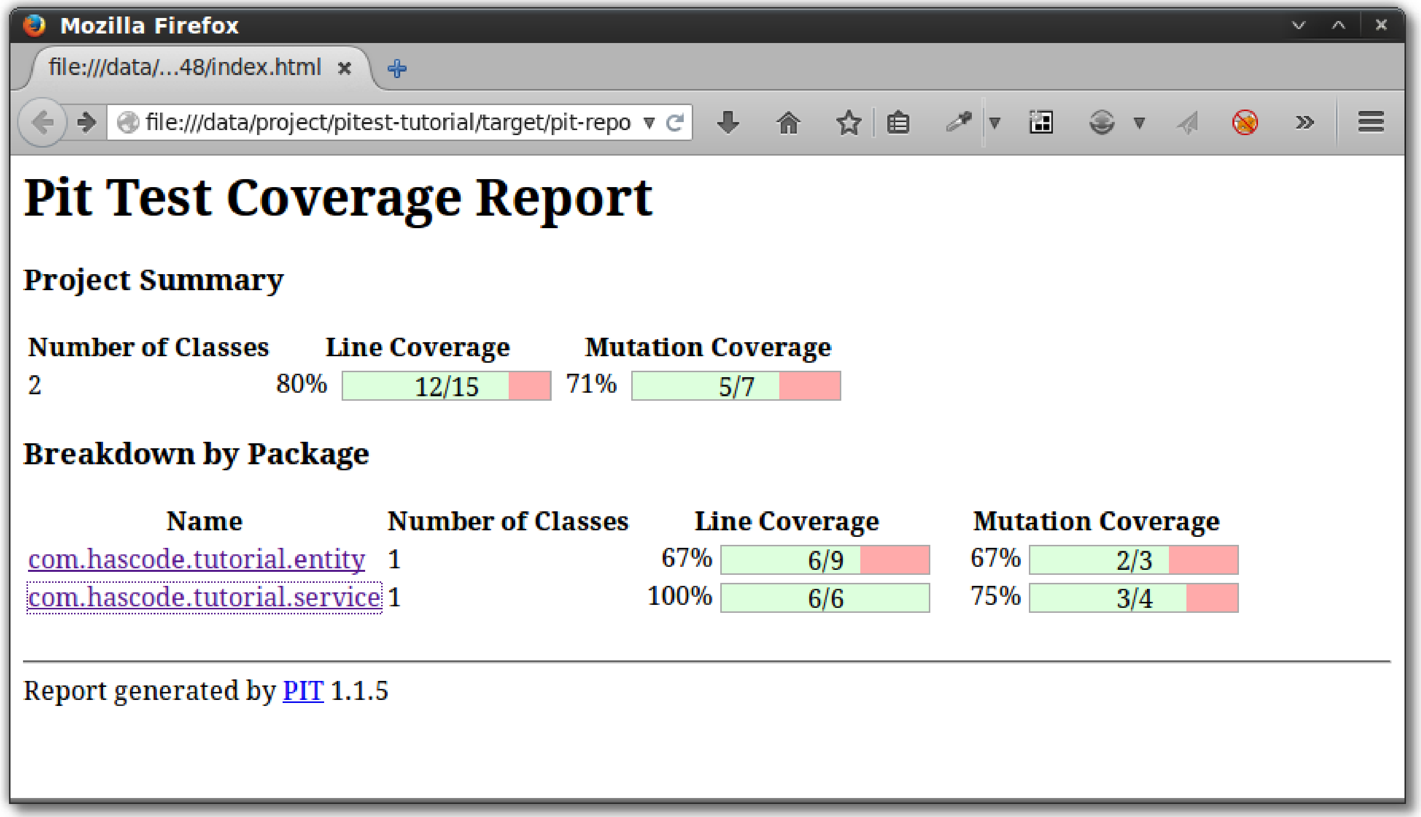
https://pitest.org/

- Line coverage has limitations
- Uncover bugs | improve the tests.



```
                    // Verify for a ".." component at next iter
3                   if ((newcomponents.get(i)).length() > 0
                    {
                        newcomponents.remove(i);
                        newcomponents.remove(i);
1                       i = i - 2;
1                       if (i < -1)
                        {
                            i = -1;
                        }
                    }
                }
```

Mutation Testing


Mozilla Firefox
file:///data/...48/index.html
file:///data/project/pitest-tutorial/target/pit-repo

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage	
2	80%	12/15	71%	5/7

Breakdown by Package

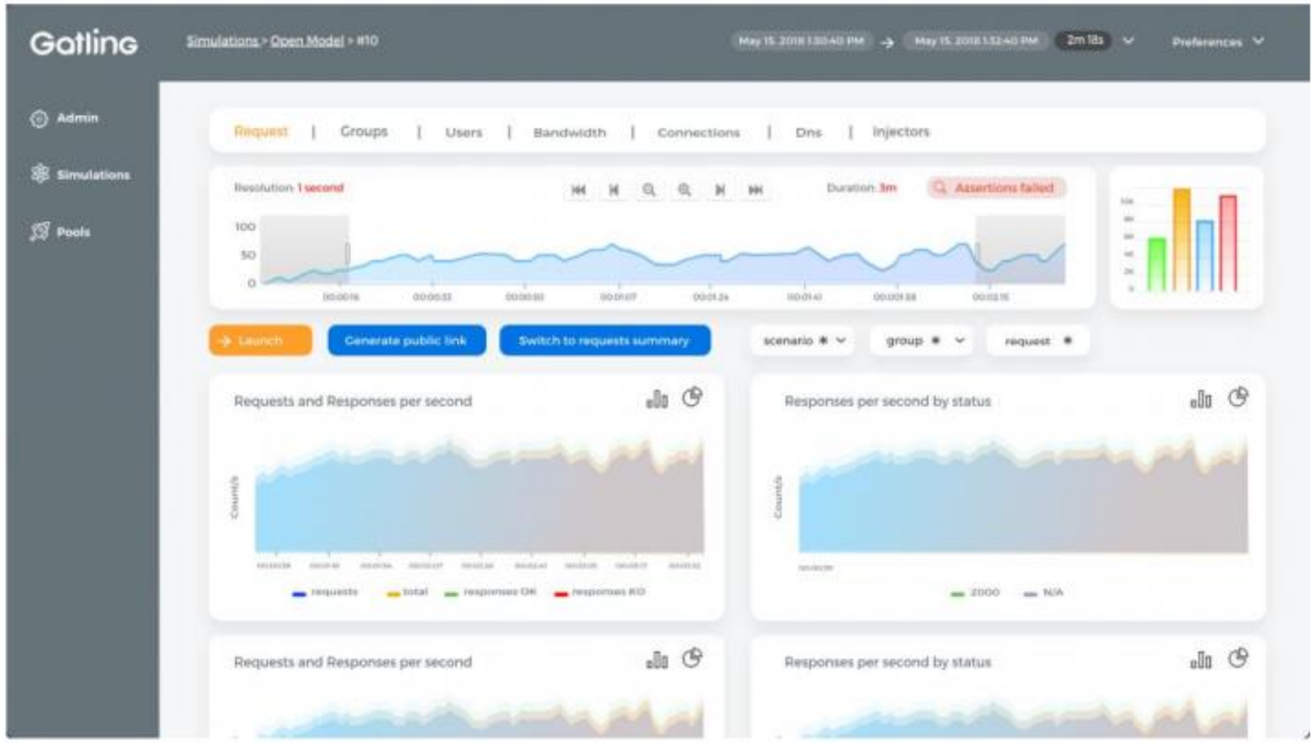
Name	Number of Classes	Line Coverage		Mutation Coverage	
com.hascode.tutorial.entity	1	67%	6/9	67%	2/3
com.hascode.tutorial.service	1	100%	6/6	75%	3/4

Report generated by PIT 1.1.5

pitest.org

https://pitest.org/

Stress/Load Testing


Gatling
Simulations > Open Model > #10
Preferences
Admin
Simulations
Pools
Request | Groups | Users | Bandwidth | Connections | Dns | Injectors
Resolution 1 second
Duration 3m
Assertions failed
Launch
Generate public link
Switch to requests summary
scenario
group
request
Requests and Responses per second
Responses per second by status
requests
total
responses OK
responses KO
2000
N/A

- Cost Effective
- Uncover Leaks | Inefficiencies
- More with Less
- Improve User Experience
- Important for CI/CD
- Can also help you with chaos testing (Assertions).

https://gatling.io/docs/current/

Stress/Load Testing

```
package computerdatabase // 1

import io.gatling.core.Predef._ // 2
import io.gatling.http.Predef._
import scala.concurrent.duration._

class BasicSimulation extends Simulation { // 3

  val httpProtocol = http // 4
        .baseUrl("http://computer-database.gatling.io") // 5
        .acceptHeader("text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8") // 6
        .doNotTrackHeader("1")
        .acceptLanguageHeader("en-US,en;q=0.5")
        .acceptEncodingHeader("gzip, deflate")
        .userAgentHeader("Mozilla/5.0 (Windows NT 5.1; rv:31.0) Gecko/20100101 Firefox/31.0")

  val scn = scenario("BasicSimulation") // 7
        .exec(http("request_1") // 8
        .get("/")) // 9
```

Gatling

https://gatling.io/docs/current/

Chaos Testing

Caos
Engineering

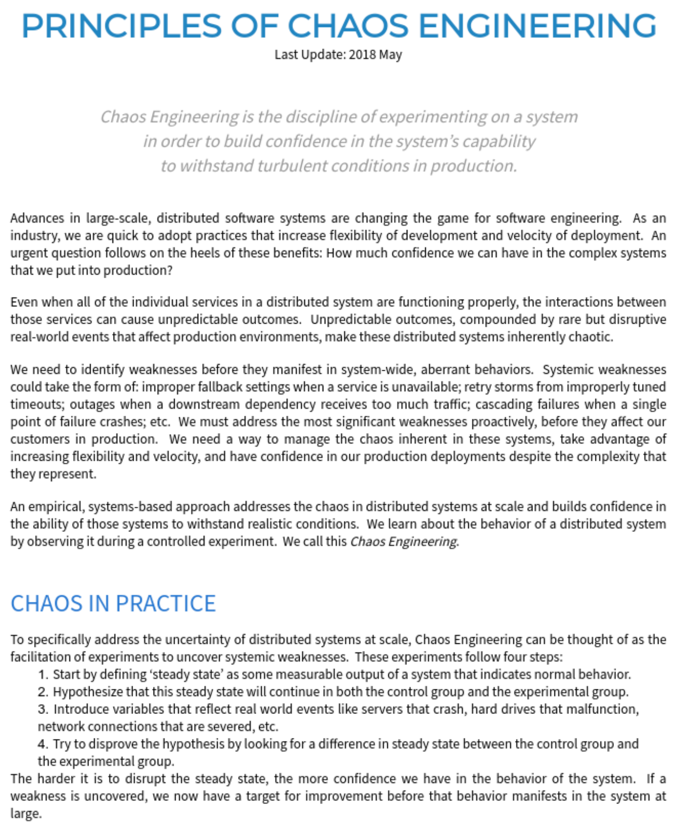
Simian Army Projects

- Chaos Monkey
- Chaos Gorilla
- Chaos Kong
- Janitor Monkey
- Doctor Monkey
- Compliance Monkey
- Latency Monkey
- Security Monkey

https://github.com/Netflix/SimianArmy

- Everything fails all the time.
- Make sure things keep working in production (cloud).
- Simian Army | Gremlin
- Simple Linux Tools
- Theories and Assertions (ST)

Chaos Testing



PRINCIPLES OF CHAOS ENGINEERING

Last Update: 2018 May

Chaos Engineering is the discipline of experimenting on a system in order to build confidence in the system's capability to withstand turbulent conditions in production.

Advances in large-scale, distributed software systems are changing the game for software engineering. As an industry, we are quick to adopt practices that increase flexibility of development and velocity of deployment. An urgent question follows on the heels of these benefits: How much confidence we can have in the complex systems that we put into production?

Even when all of the individual services in a distributed system are functioning properly, the interactions between those services can cause unpredictable outcomes. Unpredictable outcomes, compounded by rare but disruptive real-world events that affect production environments, make these distributed systems inherently chaotic.

We need to identify weaknesses before they manifest in system-wide, aberrant behaviors. Systemic weaknesses could take the form of: improper fallback settings when a service is unavailable; retry storms from improperly tuned timeouts; outages when a downstream dependency receives too much traffic; cascading failures when a single point of failure crashes; etc. We must address the most significant weaknesses proactively, before they affect our customers in production. We need a way to manage the chaos inherent in these systems, take advantage of increasing flexibility and velocity, and have confidence in our production deployments despite the complexity that they represent.

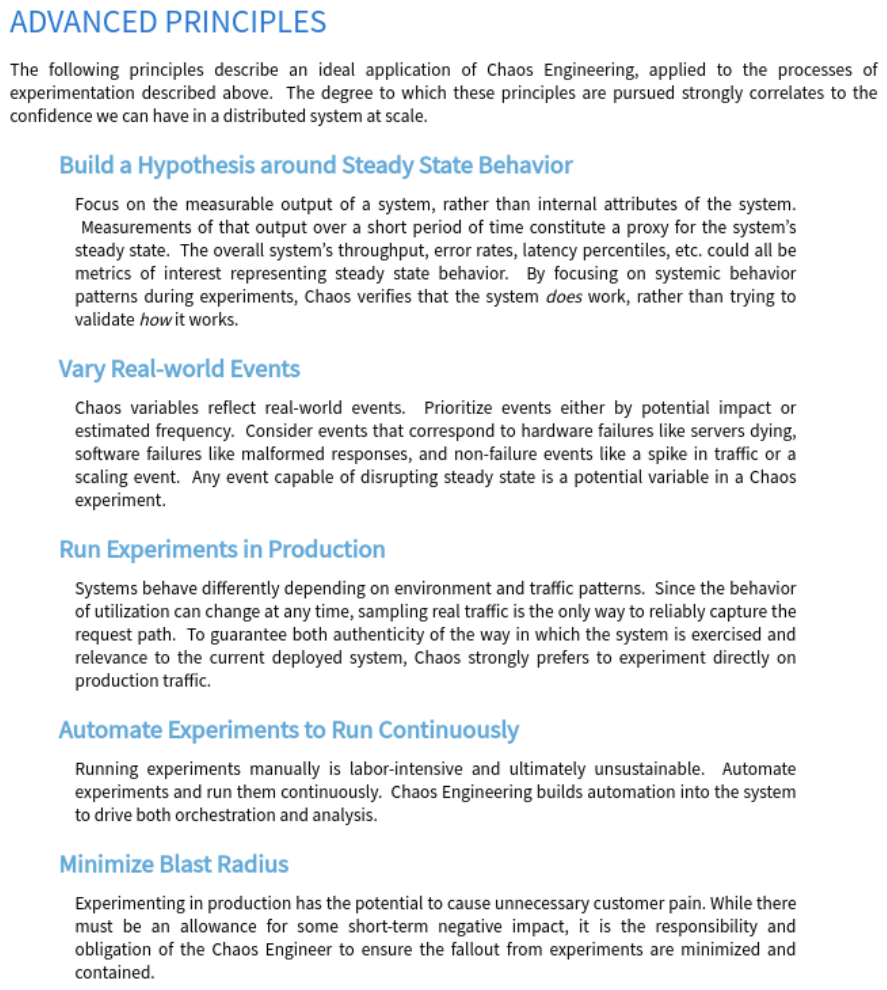
An empirical, systems-based approach addresses the chaos in distributed systems at scale and builds confidence in the ability of those systems to withstand realistic conditions. We learn about the behavior of a distributed system by observing it during a controlled experiment. We call this *Chaos Engineering.*

CHAOS IN PRACTICE

To specifically address the uncertainty of distributed systems at scale, Chaos Engineering can be thought of as the facilitation of experiments to uncover systemic weaknesses. These experiments follow four steps:

1. Start by defining 'steady state' as some measurable output of a system that indicates normal behavior.
2. Hypothesize that this steady state will continue in both the control group and the experimental group.
3. Introduce variables that reflect real world events like servers that crash, hard drives that malfunction, network connections that are severed, etc.
4. Try to disprove the hypothesis by looking for a difference in steady state between the control group and the experimental group.

The harder it is to disrupt the steady state, the more confidence we have in the behavior of the system. If a weakness is uncovered, we now have a target for improvement before that behavior manifests in the system at large.



ADVANCED PRINCIPLES

The following principles describe an ideal application of Chaos Engineering, applied to the processes of experimentation described above. The degree to which these principles are pursued strongly correlates to the confidence we can have in a distributed system at scale.

Build a Hypothesis around Steady State Behavior

Focus on the measurable output of a system, rather than internal attributes of the system. Measurements of that output over a short period of time constitute a proxy for the system's steady state. The overall system's throughput, error rates, latency percentiles, etc. could all be metrics of interest representing steady state behavior. By focusing on systemic behavior patterns during experiments, Chaos verifies that the system *does* work, rather than trying to validate *how* it works.

Vary Real-world Events

Chaos variables reflect real-world events. Prioritize events either by potential impact or estimated frequency. Consider events that correspond to hardware failures like servers dying, software failures like malformed responses, and non-failure events like a spike in traffic or a scaling event. Any event capable of disrupting steady state is a potential variable in a Chaos experiment.

Run Experiments in Production

Systems behave differently depending on environment and traffic patterns. Since the behavior of utilization can change at any time, sampling real traffic is the only way to reliably capture the request path. To guarantee both authenticity of the way in which the system is exercised and relevance to the current deployed system, Chaos strongly prefers to experiment directly on production traffic.

Automate Experiments to Run Continuously

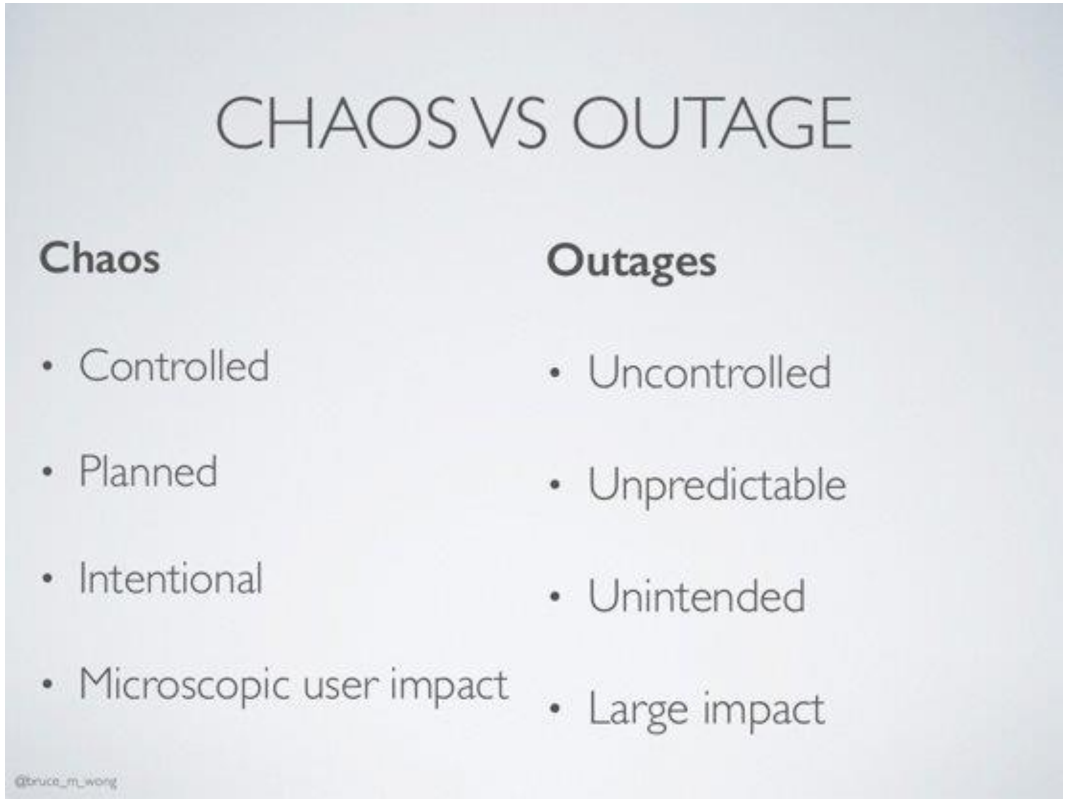
Running experiments manually is labor-intensive and ultimately unsustainable. Automate experiments and run them continuously. Chaos Engineering builds automation into the system to drive both orchestration and analysis.

Minimize Blast Radius

Experimenting in production has the potential to cause unnecessary customer pain. While there must be an allowance for some short-term negative impact, it is the responsibility and obligation of the Chaos Engineer to ensure the fallout from experiments are minimized and contained.

https://principlesofchaos.org/?lang=ENcontent

Chaos Testing



CHAOS VS OUTAGE

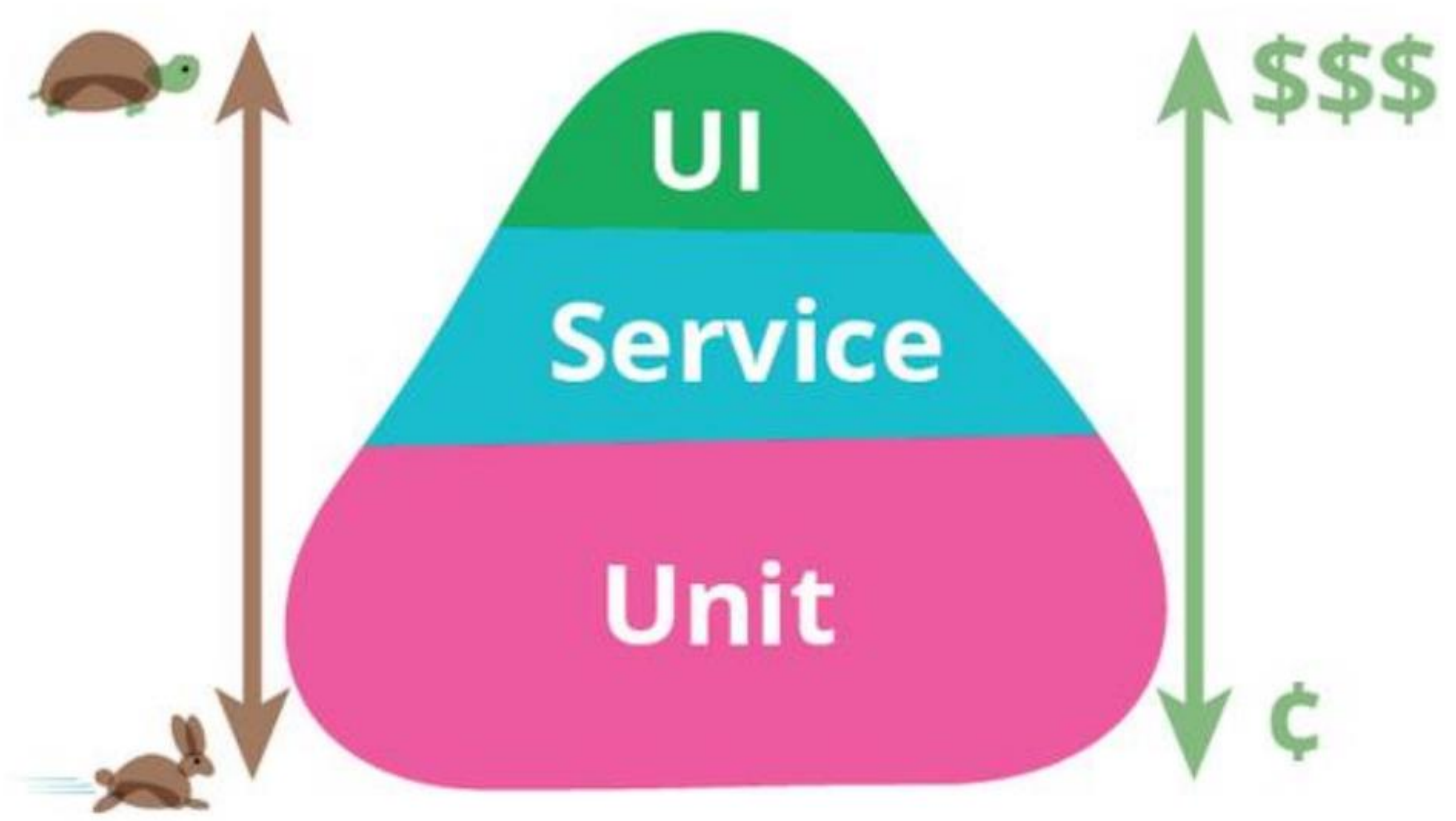
Chaos	Outages
Controlled	Uncontrolled
Planned	Unpredictable
Intentional	Unintended
Microscopic user impact	Large impact

@bruce_m_wong

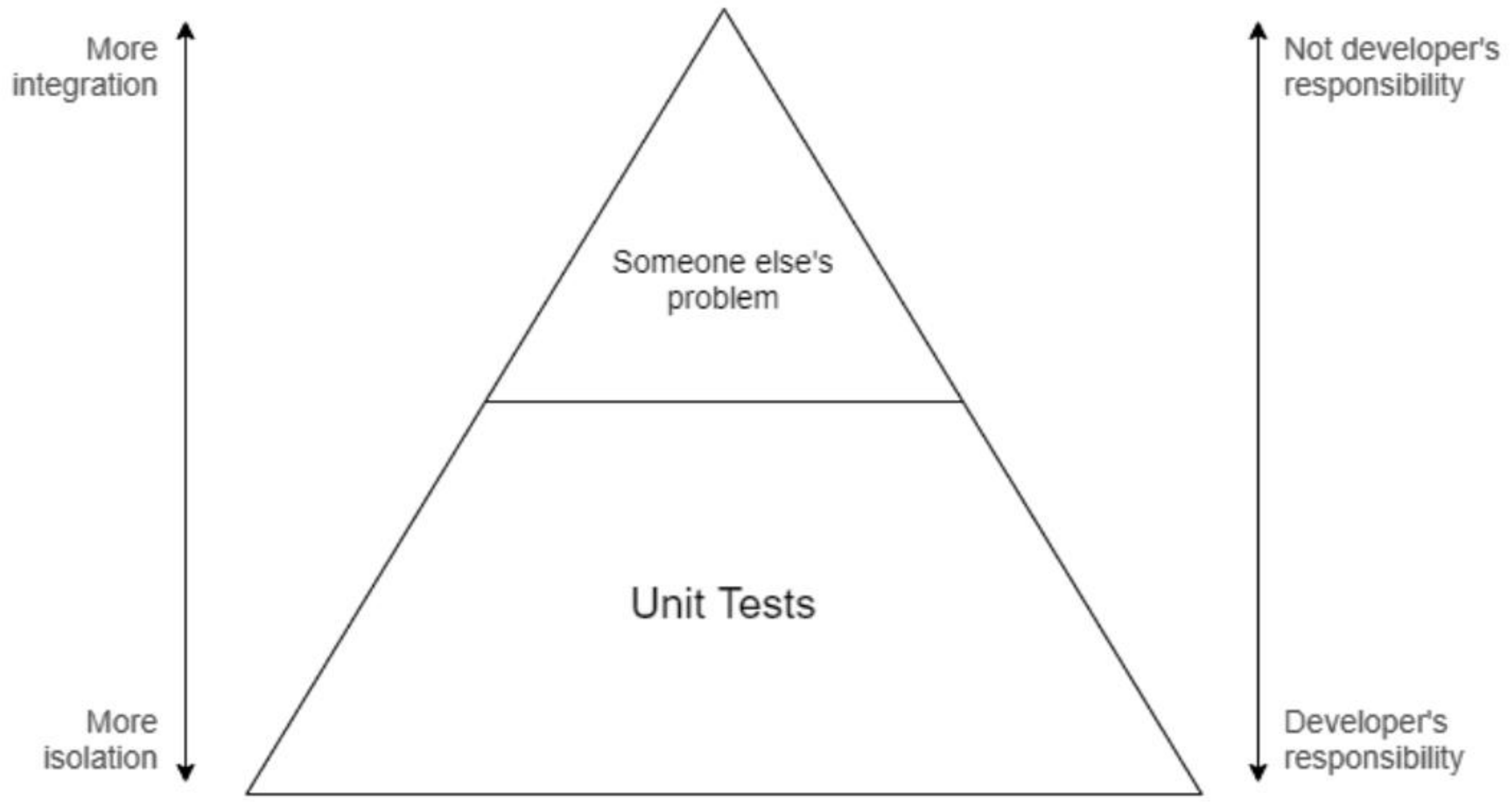
VS

https://principlesofchaos.org/?lang=ENcontent

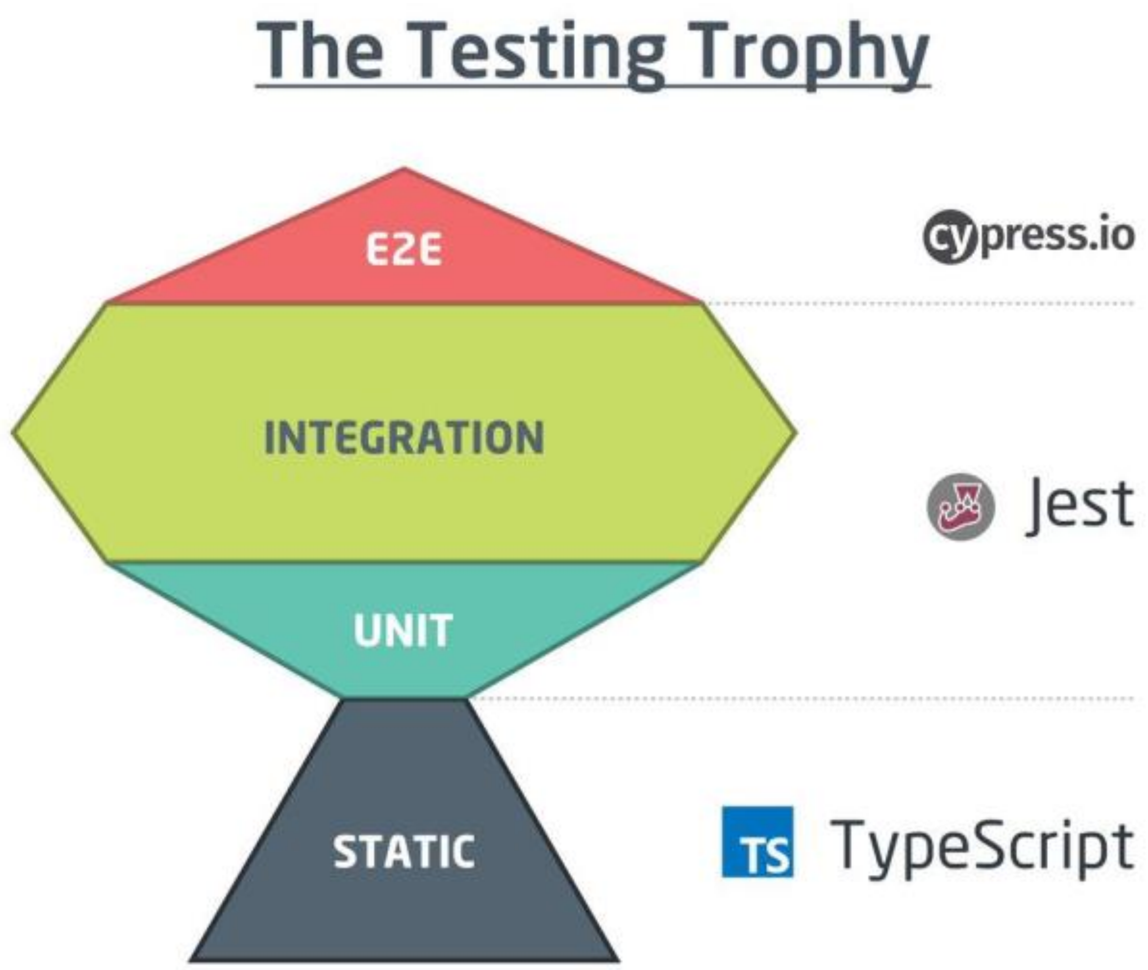
Testing Pyramid: Backend (Before Cloud)



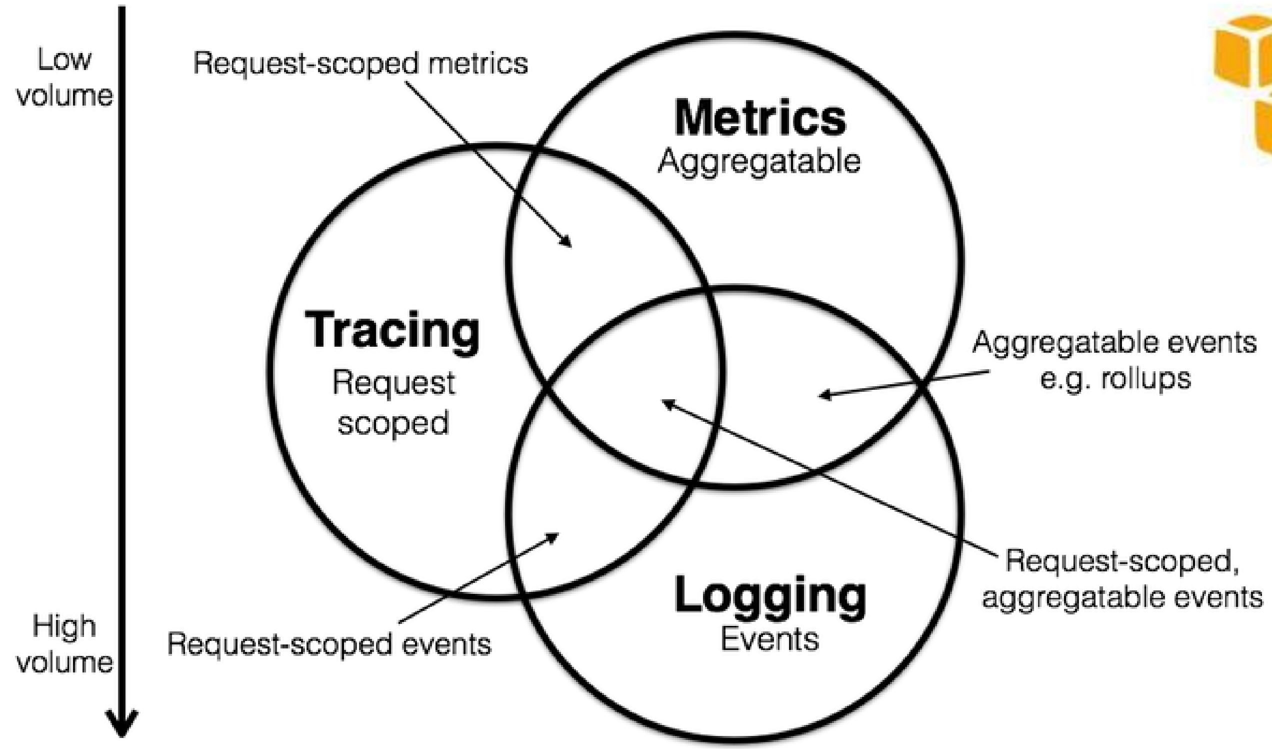
Testing Pyramid



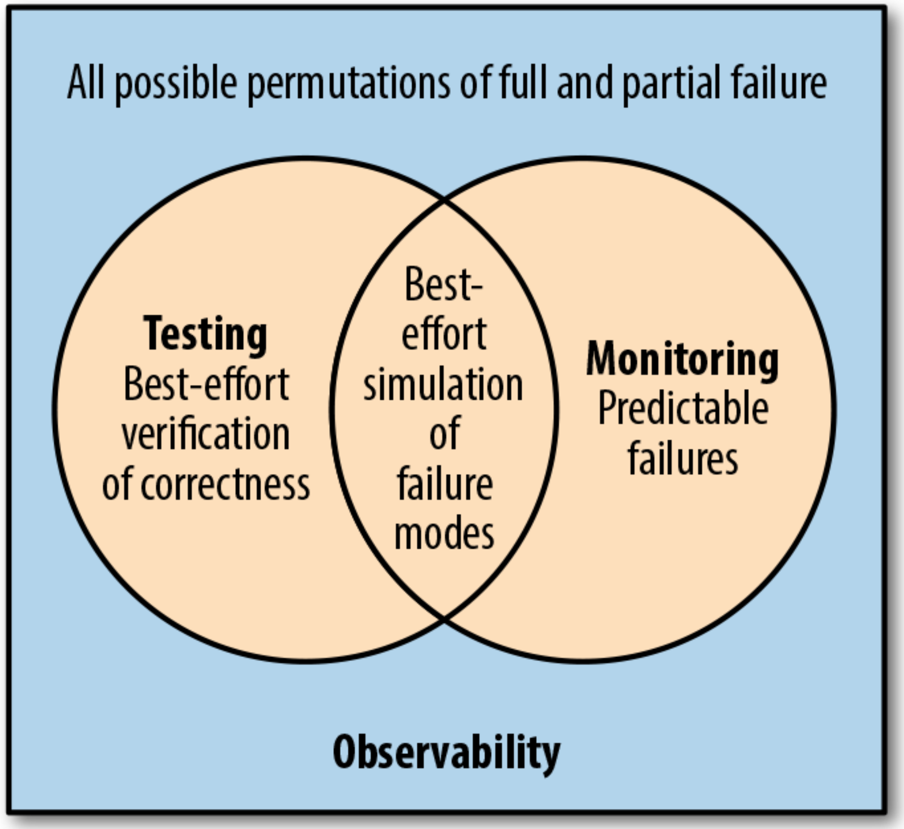
Trophy is all about Frontend!

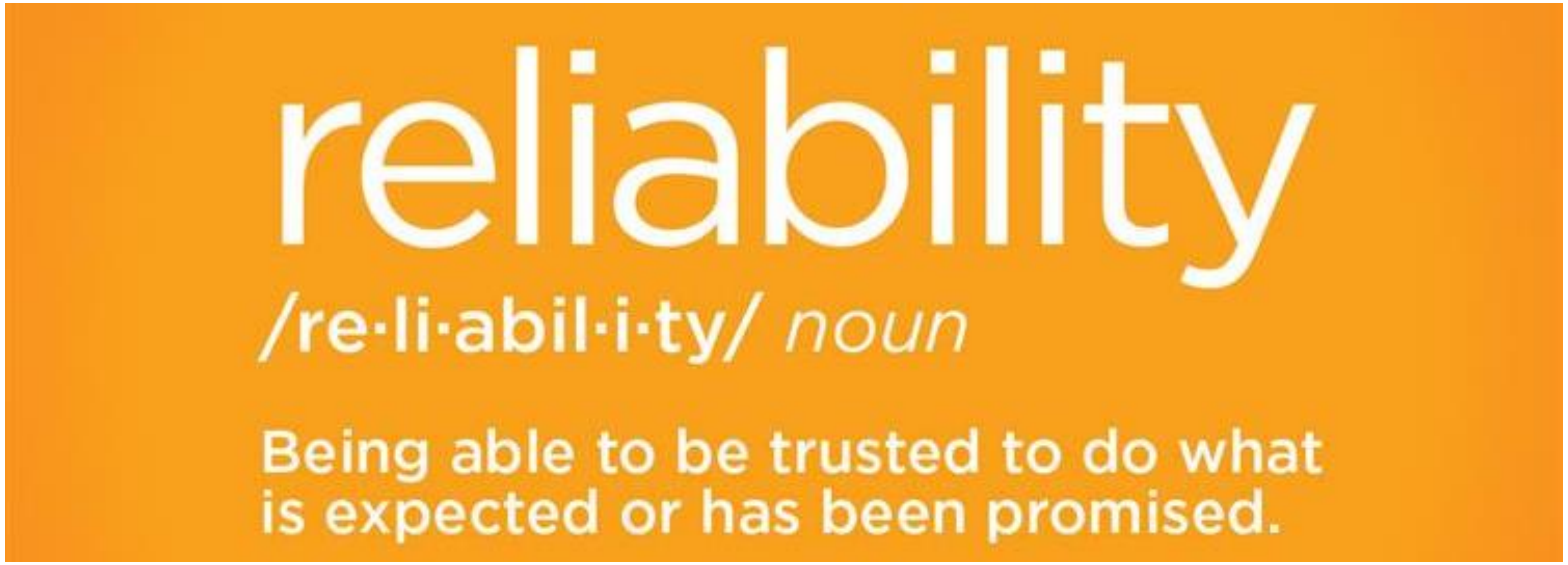


Enter the Cloud: Changes everything!



Enter the Cloud: Changes everything!


All possible permutations of full and partial failure
Testing
Best-effort verification of correctness
Best-effort simulation of failure modes
Monitoring
Predictable failures
Observability



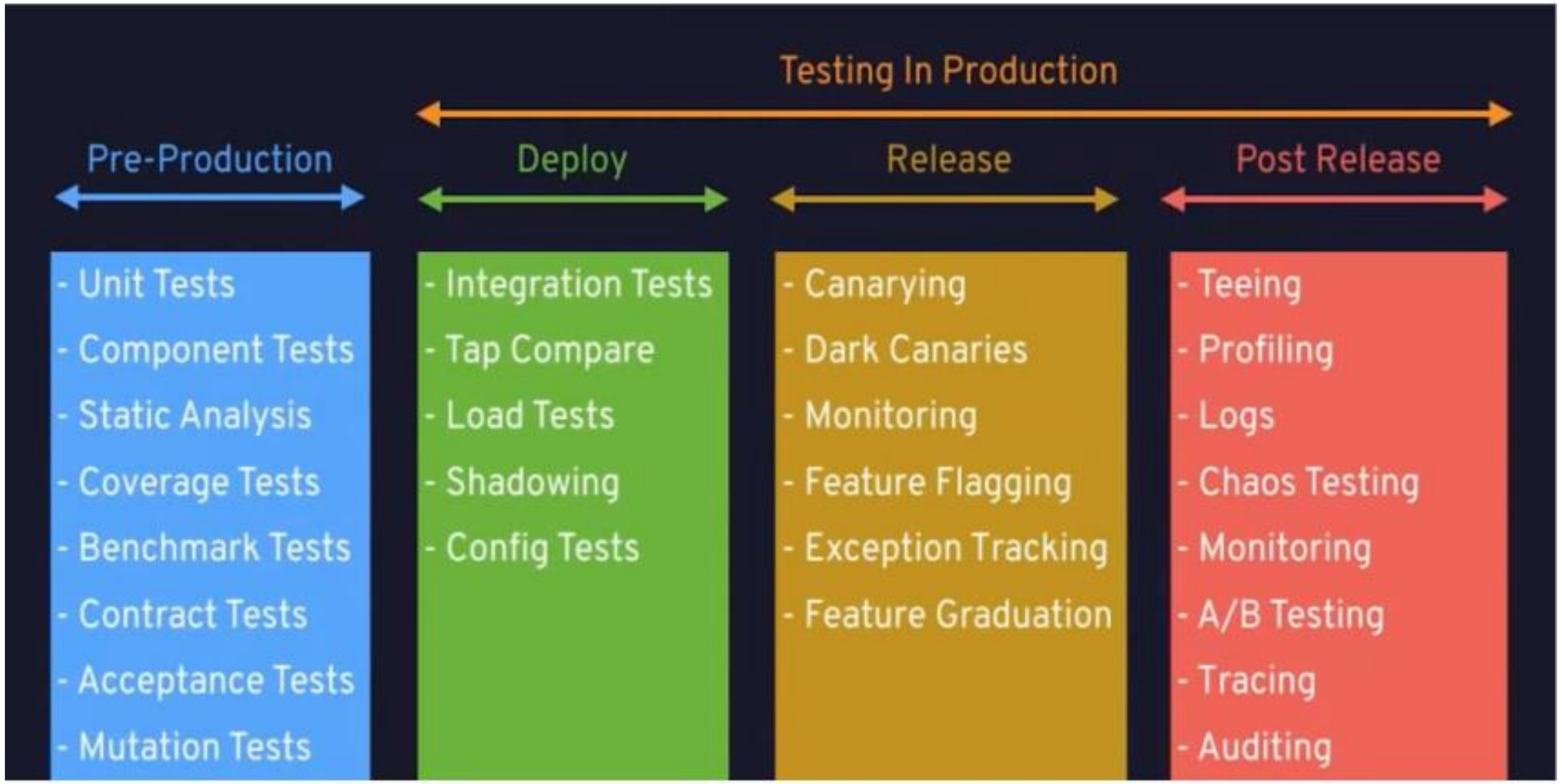
reliability

/re·li·abil·i·ty/ *noun*

Being able to be trusted to do what is expected or has been promised.

How is that different from testing?

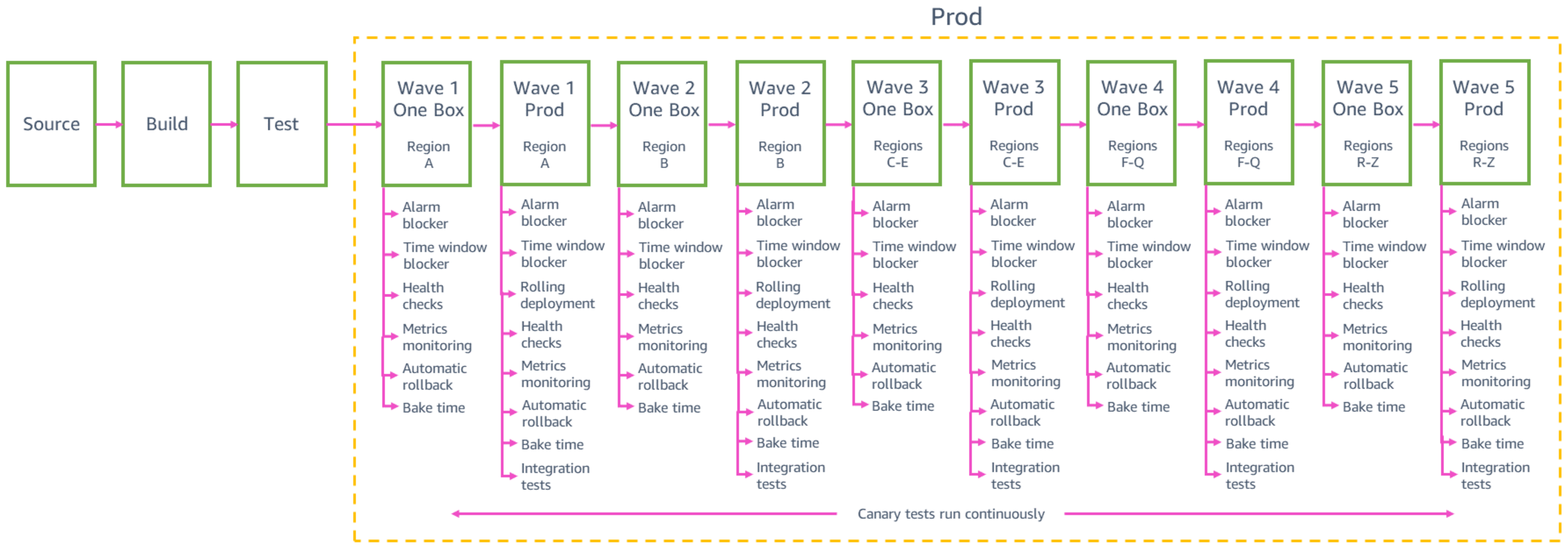
Spectrum: Hardening production.



Testing In Production

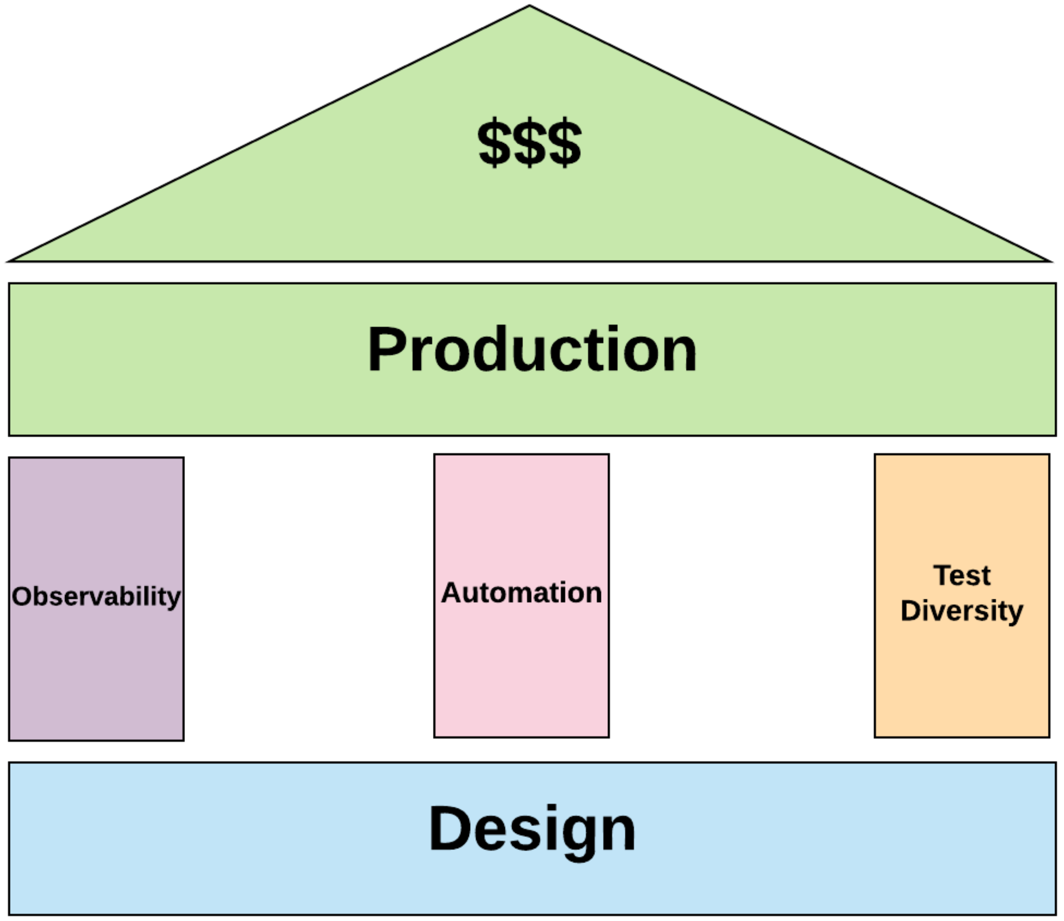
Pre-Production	Deploy	Release	Post Release
- Unit Tests	- Integration Tests	- Canarying	- Teeing
- Component Tests	- Tap Compare	- Dark Canaries	- Profiling
- Static Analysis	- Load Tests	- Monitoring	- Logs
- Coverage Tests	- Shadowing	- Feature Flagging	- Chaos Testing
- Benchmark Tests	- Config Tests	- Exception Tracking	- Monitoring
- Contract Tests		- Feature Graduation	- A/B Testing
- Acceptance Tests			- Tracing
- Mutation Tests			- Auditing

Isolation is key!



https://aws.amazon.com/builders-library/automating-safe-hands-off-deployments/

Hardening Production!



Production: Is for Everybody or...

Thank you

Diego Pacheco