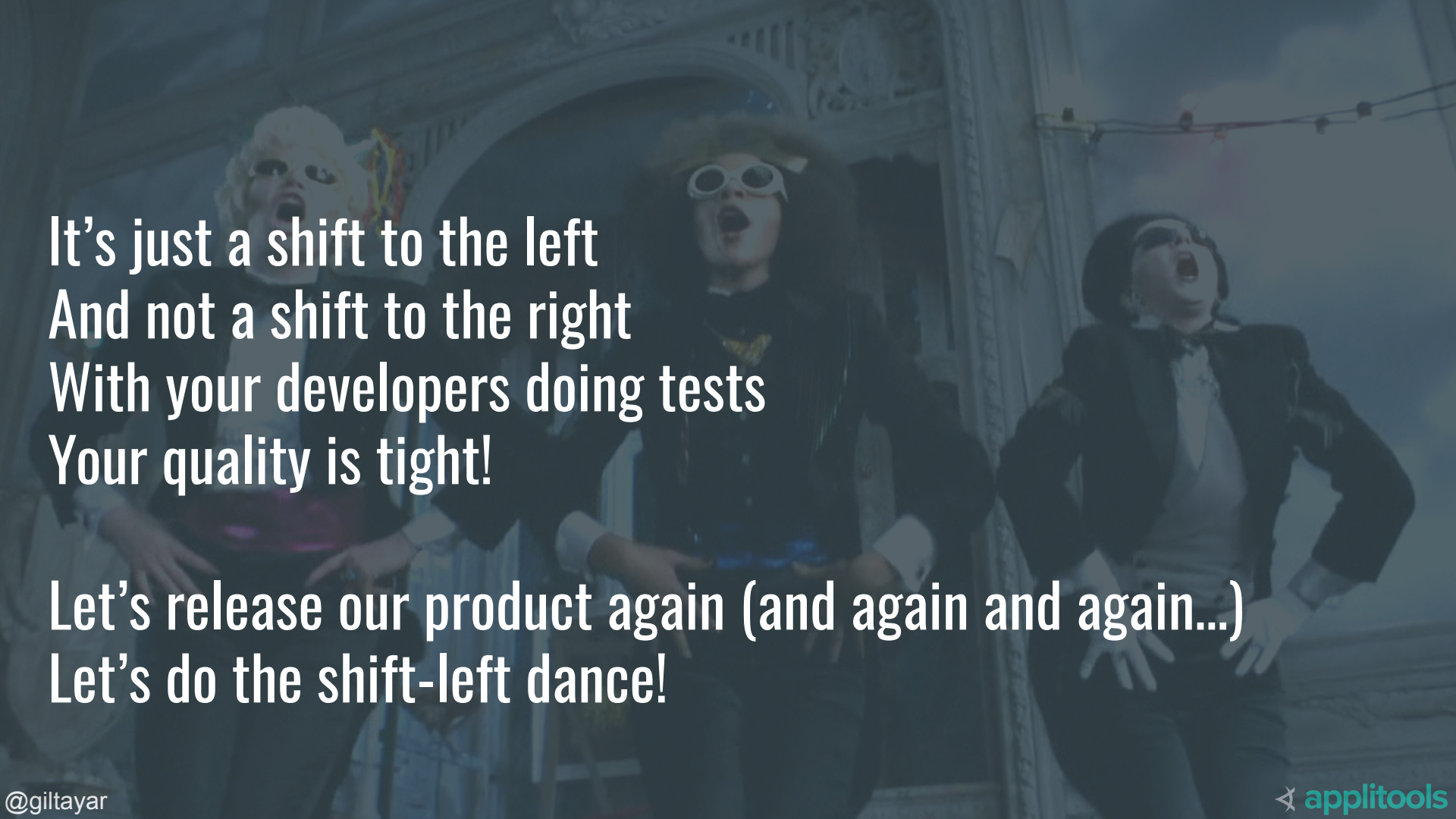
How To Manage Quality in Shift Left Environments

...

Gil Tayar (@giltayar)
October 2020 (Corona Times!)

This presentation: http://bit.ly/manage-quality-shift-left-testcon

Or...


It’s just a shift to the left
And not a shift to the right
With your developers doing tests
Your quality is tight!
Let’s release our product again (and again and again...)
Let’s do the shift-left dance!

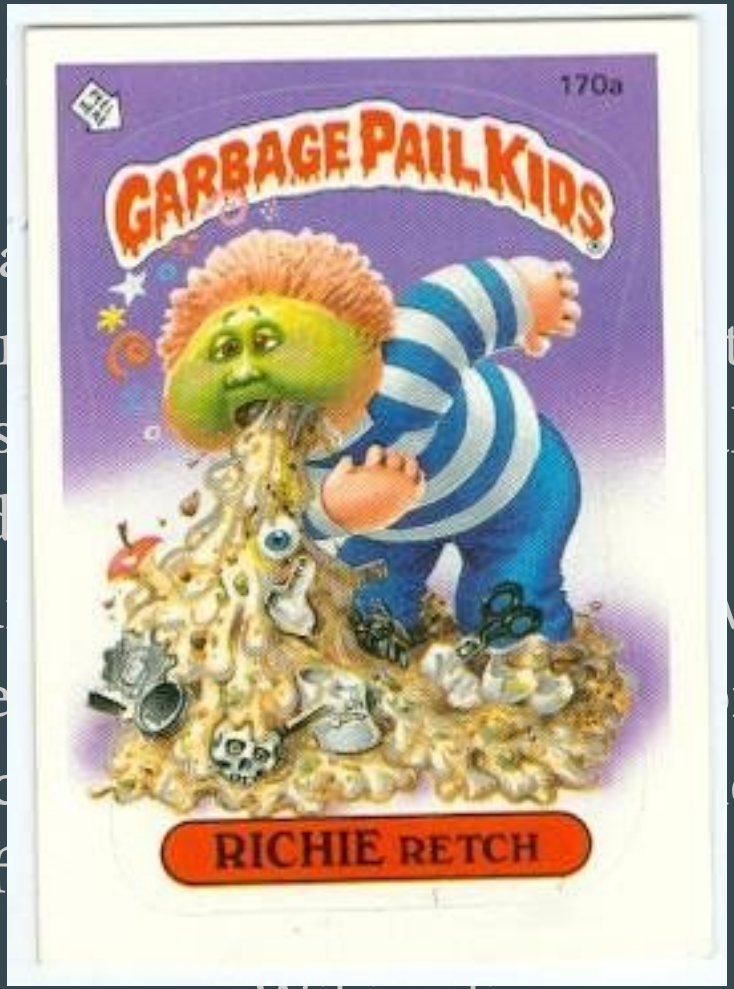
About Me

- My developer experience goes all the way back to the ‘80s.
- Am, was, and always will be a developer
- Testing the code I write is my passion
- Currently evangelist and architect @ **Applitools**
- We deliver Visual Testing tools:
  If you’re serious about testing, checkout Applitools Eyes

- Sometimes my arms bend back
- But the gum I like is coming back in style

What is Software

Software testing is a to provide stakeholders with i ty of the product or service under tes lso provide an objective, independ allow the business to appreciate and u vare implementation. Te rocess of executing a program or applic ding software bugs (errors or other def



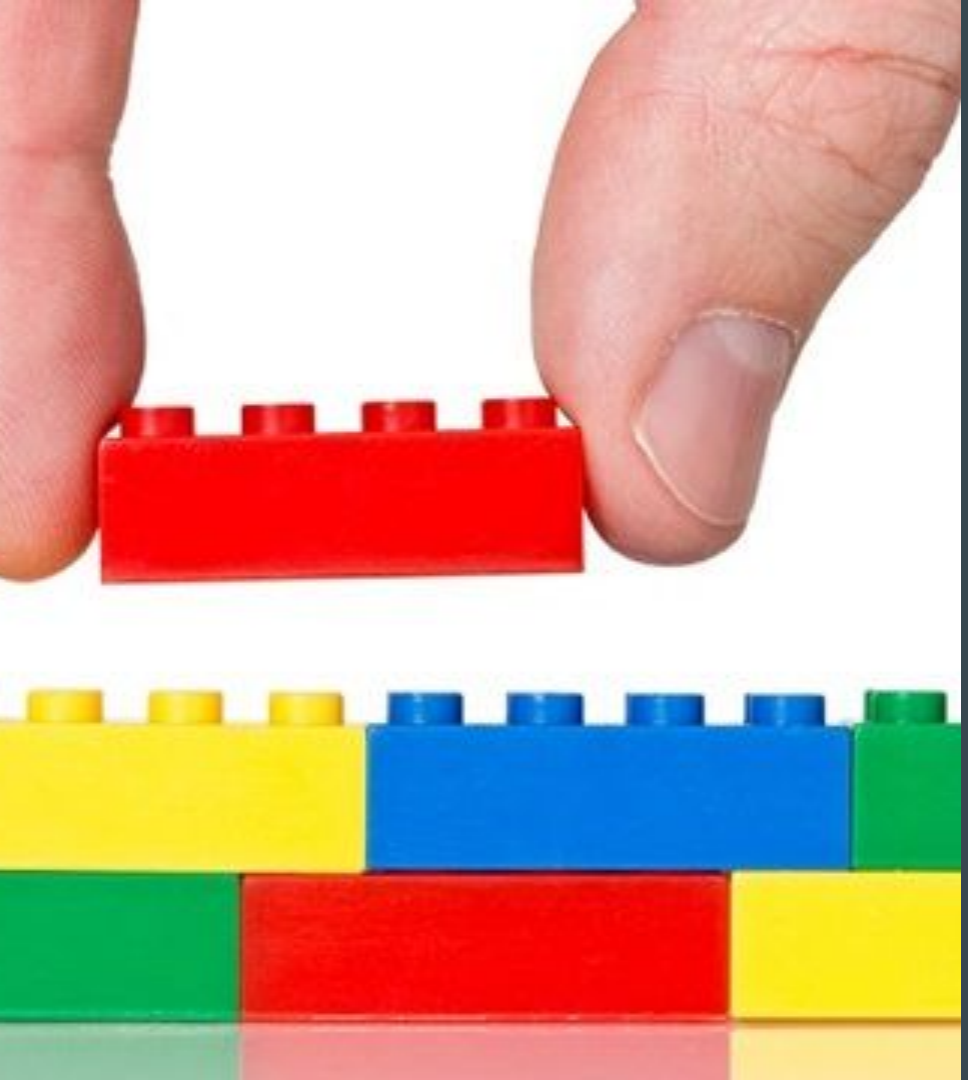
-- Wikipedia

What is Software Testing?

Testing is the art of ensuring that changes in the code will not cause new bugs to appear

Let’s talk about changes to code...
and fearing them

Egoprogramophobia



Adding Code

Fear of Adding Code

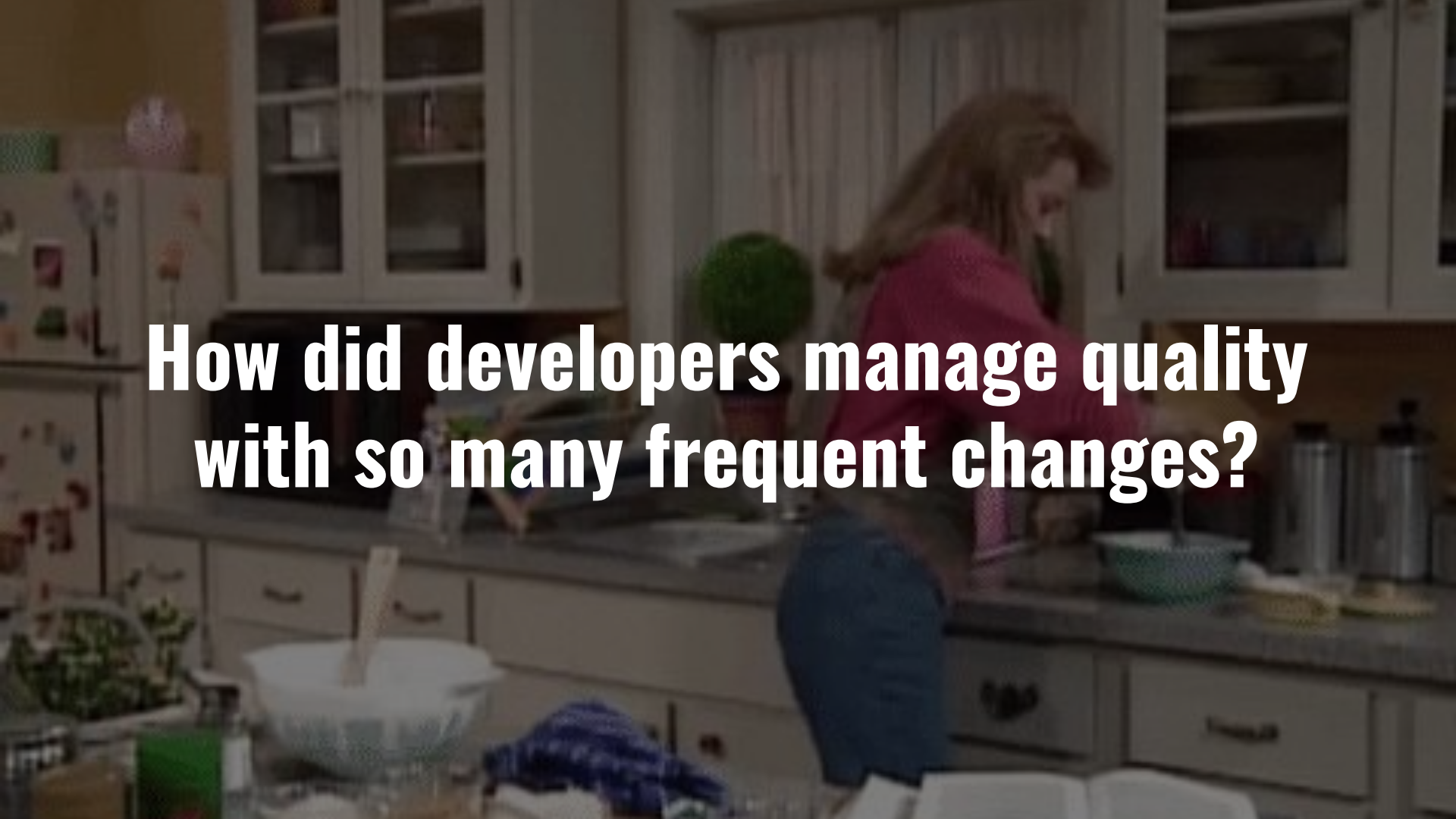
Removing Code

Fear of Removing Code

Refactoring Code

Fear of Refactoring

And developers do these things
day in, day out


How did developers manage quality with so many frequent changes?

Answer: they didn’t


They threw it over the wall to QA

Throwing over the wall begat...

- Long release times
- Integration time (aka “crunch time”)
- Automated testing

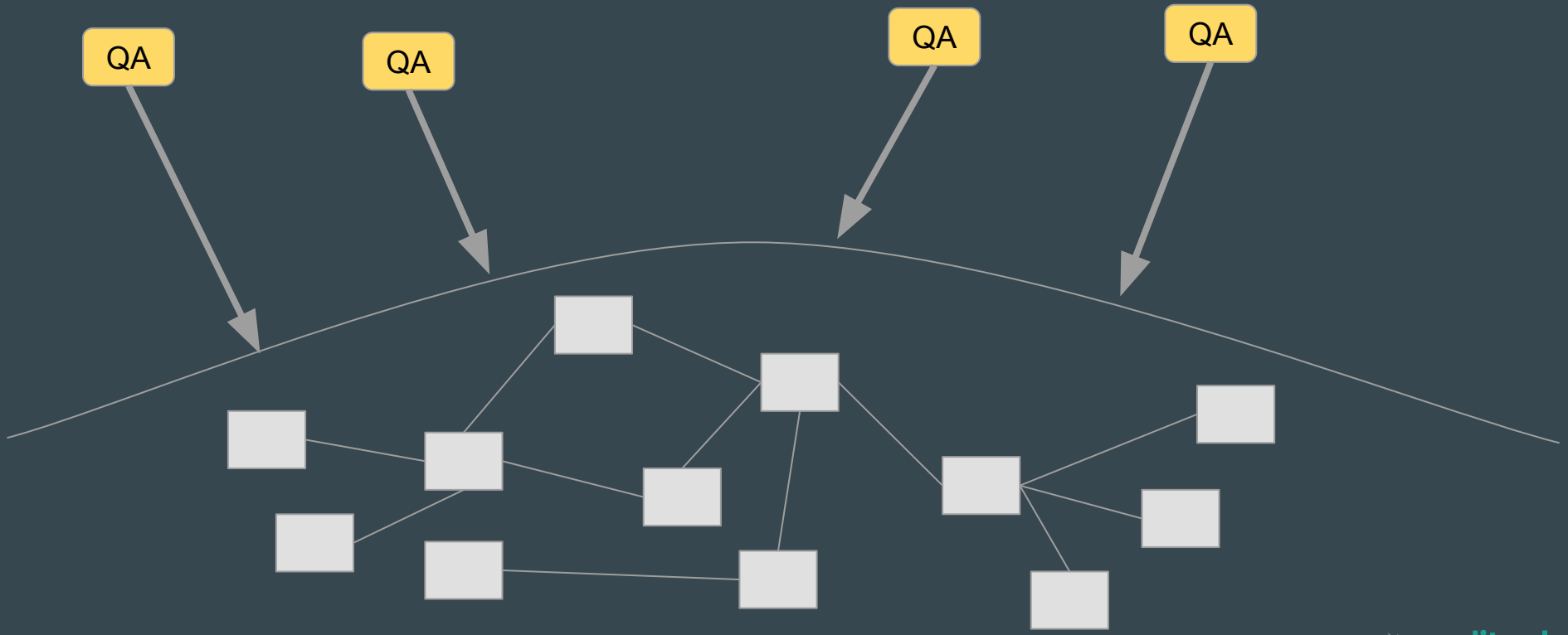
Let’s zoom in on

- Long release times
- Integration time (aka “crunch time”)
- Automated testing

Automated Testing

Testing was done “over the wall” ⇒ done by testers

Testers Have Access Only to the Final Product



These kinds of tests are E2E Tests

E2E Tests

- Extremely slow (seconds per test)
  - Think testing for many permutations (email validation)
- Difficult to test extreme conditions
  - Errors, load, etc.
- Need a staging environment
  - Expensive
  - Difficult to parallelize
- Flaky
  - Results are always “statistical”

But for a while it worked

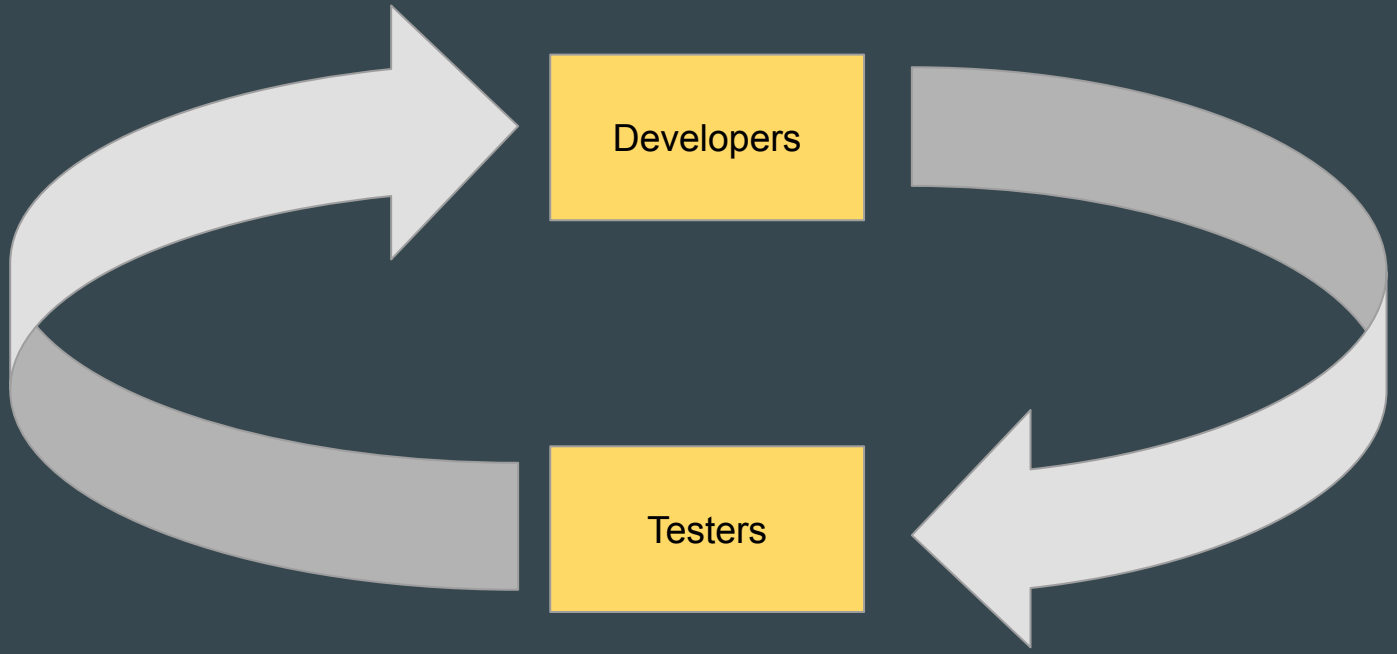

Internet

We need to be agile

We need a short release cycle

We need to get rid of *friction*

And that QA <-> Dev cycle isn't helping




what
do we
do now?

We let the developers test

Developers

Testers

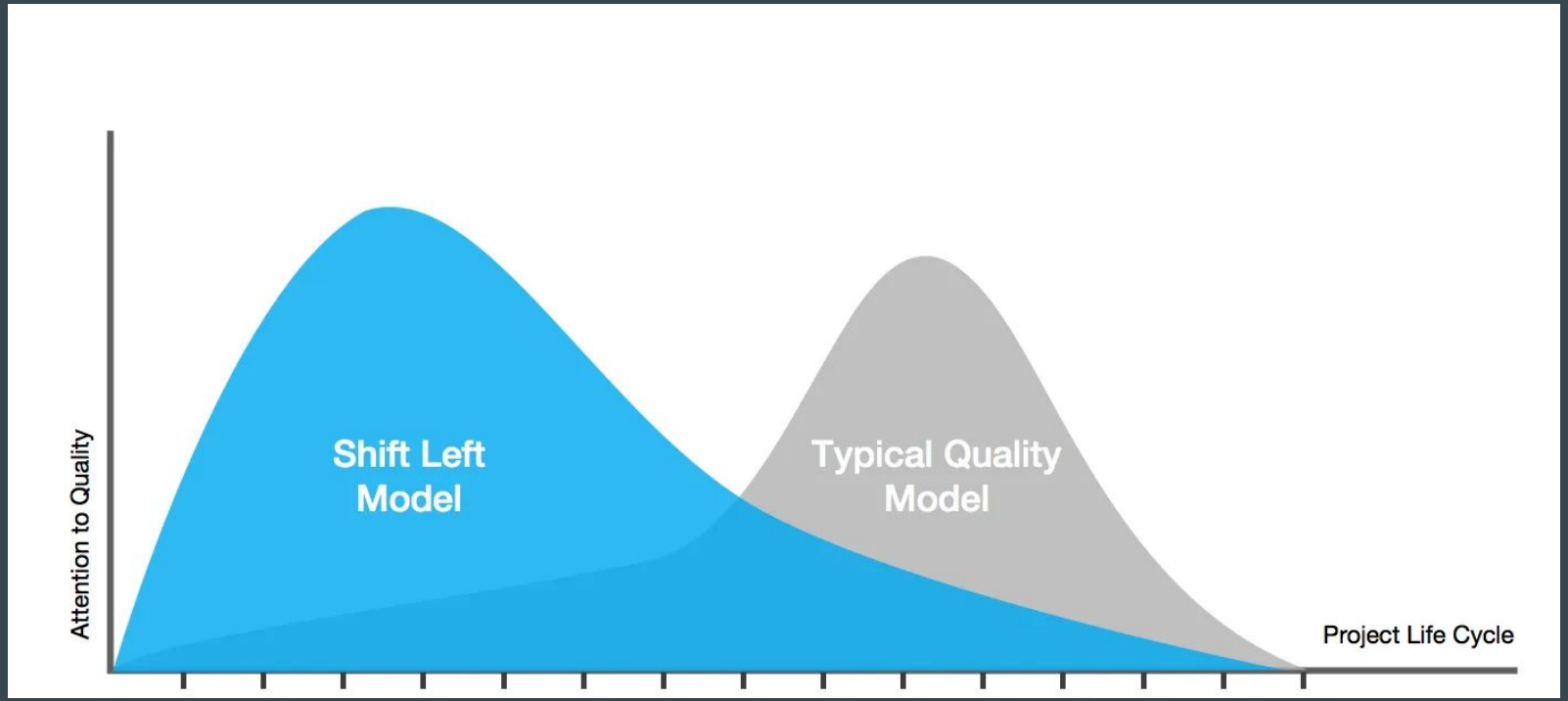
Testers

- We don't have access to the code
- We can test the application only from the outside,
- We're OK with long suites of tests
- So we MUST do E2E tests

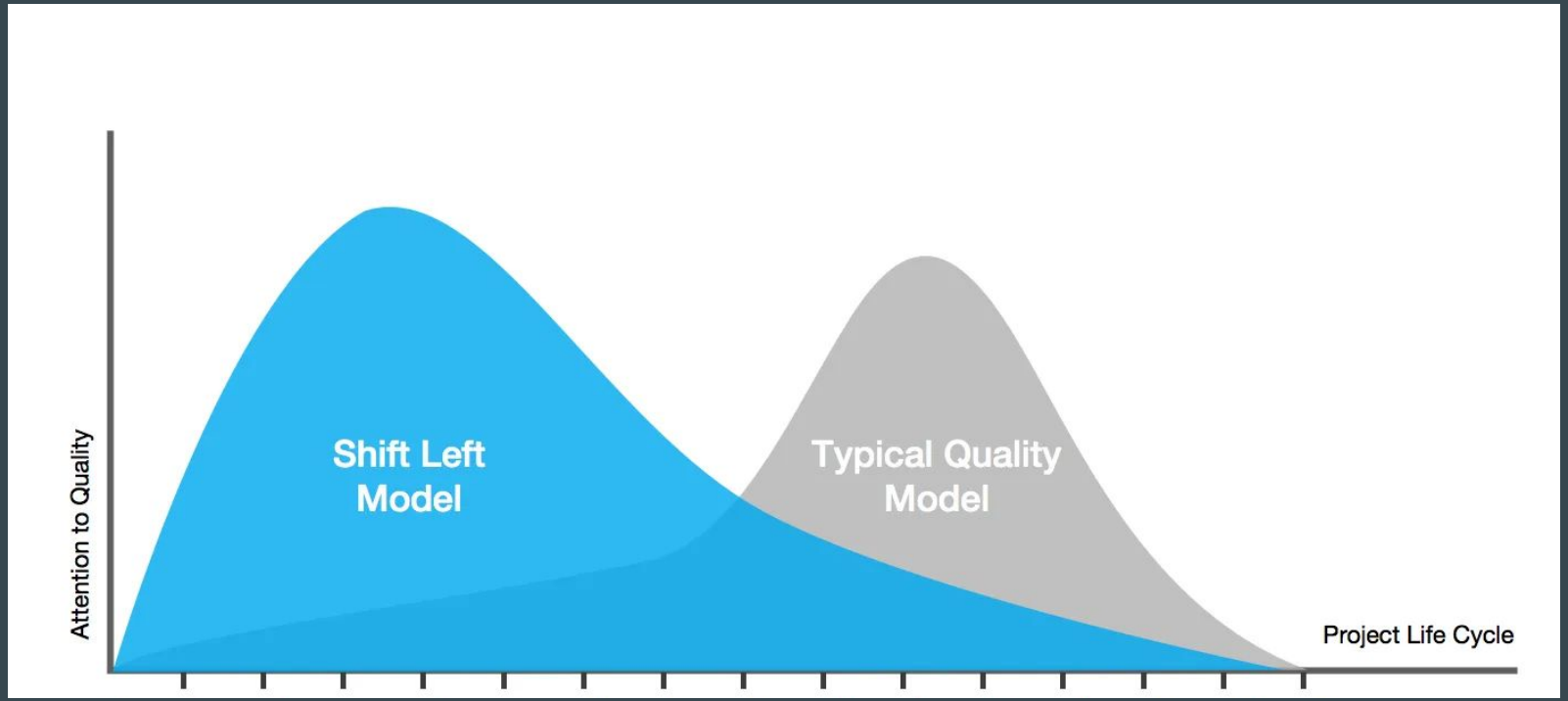
Developers

- We have access to the code
- We can test the application from the inside
- We like fast
- Prefer new kinds of tests: unit & integration tests

Shift Left!


Attention to Quality
Shift Left Model
Typical Quality Model
Project Life Cycle

I’m not a big fan of this way of looking at “shift left”



Because a good shift should be qualitative,
not quantitative

“Shift left” should change the way development and deployment are done

“Shift left” means

Testing should become part of the development process

This means that

1. Developers should write tests
for every piece of code they write

And so

2. Every push
should be ready for deployment

1. Developers should write tests for all code they write

This is the quantitative part of “shift left”

Developers write unit tests!

- Code that tests
  - A class
  - A function
  - A module
  - A UI component

And that’s where most developers stop

But most of our code is glue

And many times the units work, but...

So they write integration tests!

- Testing a set of units functioning together
- Testing a whole frontend app, without the backend
- Testing a whole microservice, without the other microservices

What about the visual aspects?

Visual Testing

- Testing also means testing your CSS and HTML
- Functionality can fail from time to time, but if the visuals fail, the customer *notices*
- A frontend developer thing.
    - Can be tested separately from the backend

But what about the whole app?
Does it function well together?

End to End Tests (E2E)

1. Deploy the application to a testing environment
    a. As similar as possible to production
2. Write a test that automates a browser/mobile device
    a. Just like a user would

Is E2E Practical?

- Very slow and flaky
- Difficult to reproduce production
- Microservice environments, E2E is a fractal, and usually impossible in full
    - The only place to truly test a huge environment is in production! 😱

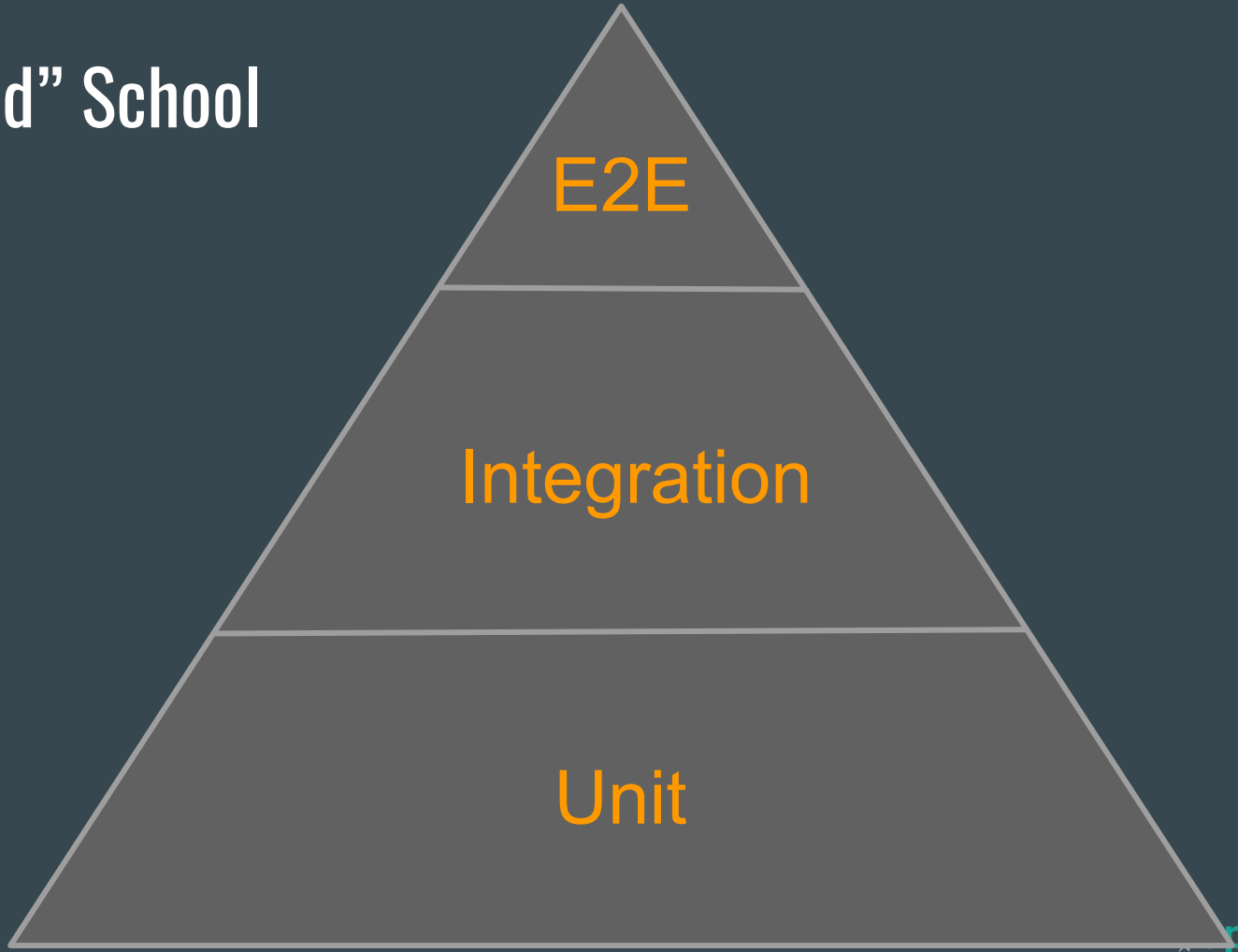
Guidelines for Developer Testing

Don’t Do Just Unit Testing

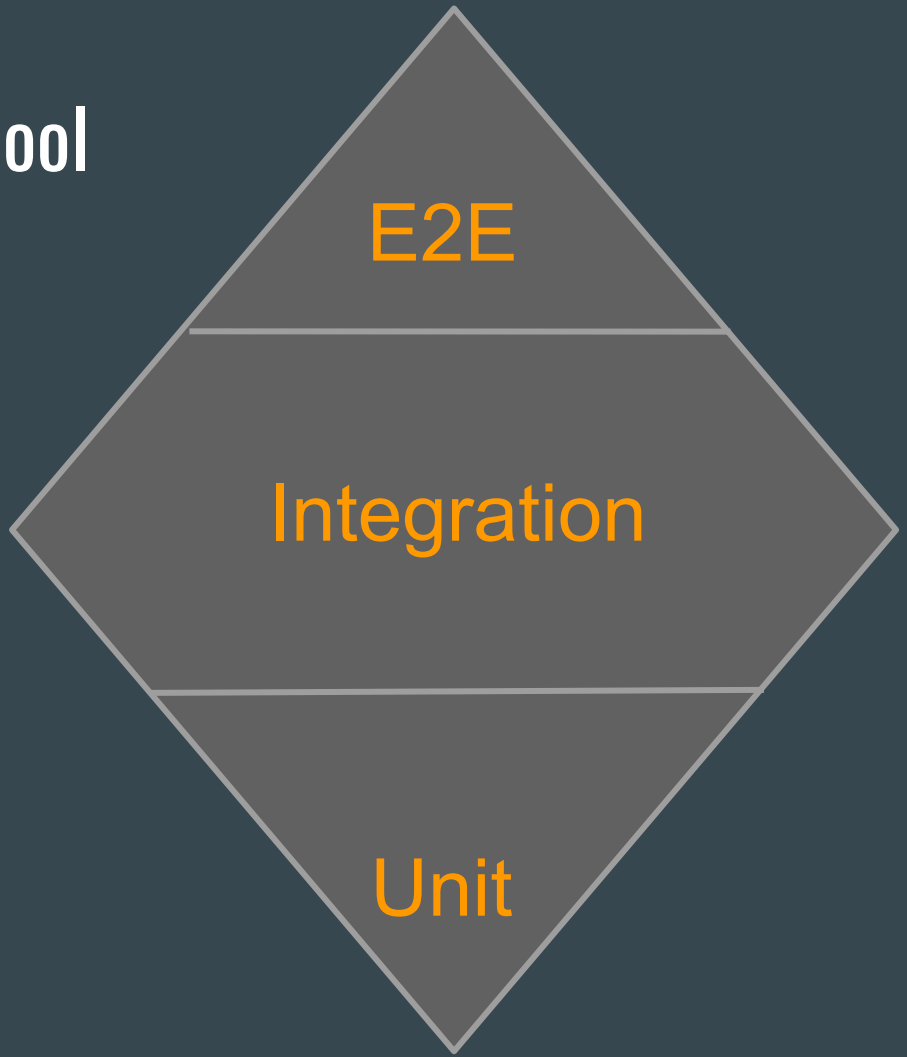
- Developers should own *all* of their tests
- Including the end to end
- Don’t leave it to QA
- If you have QA, ensure that they get a *working* product
- **Choose the testing school based on *your* likes**

There are Three Main Schools of Testing

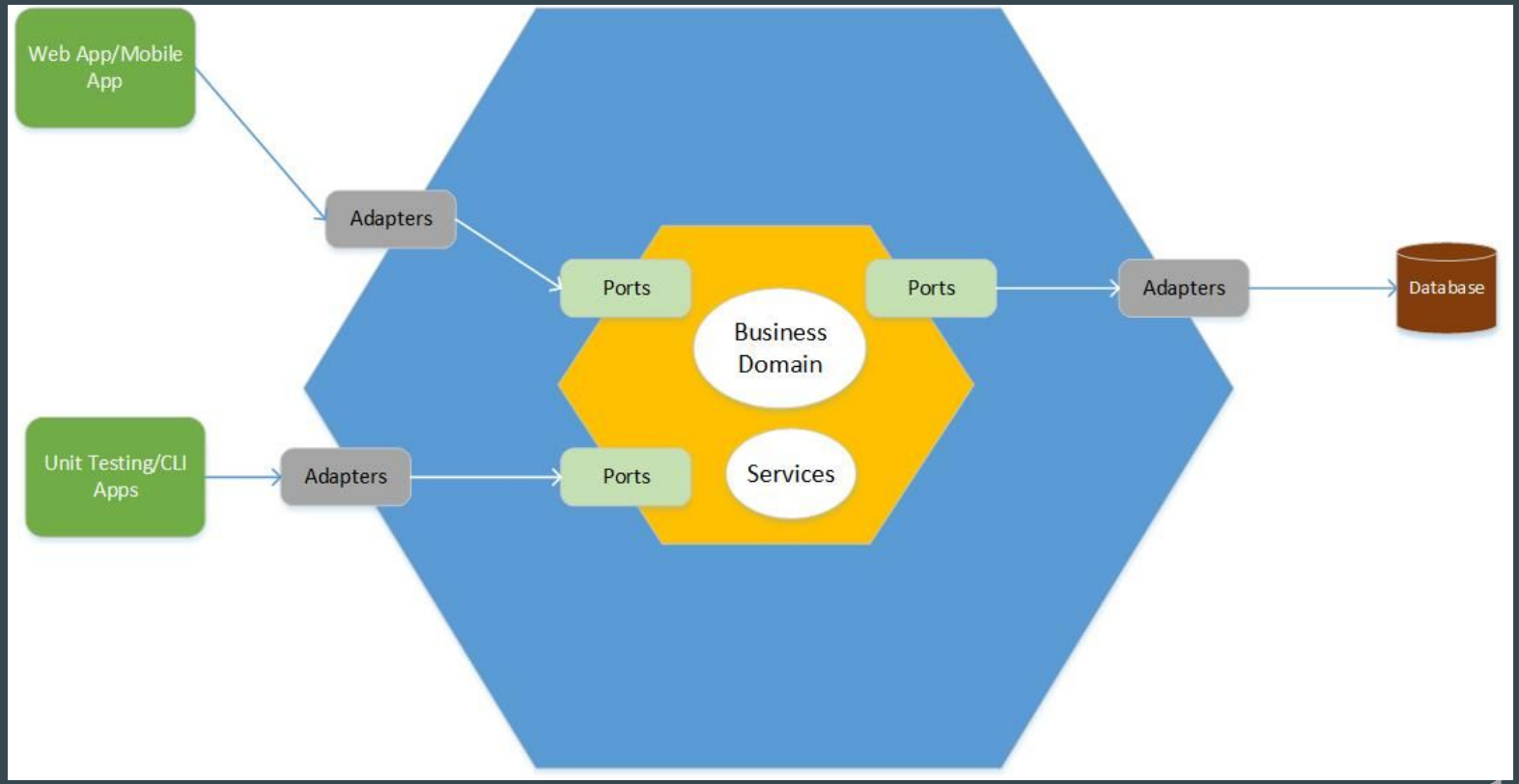
The “Pyramid” School



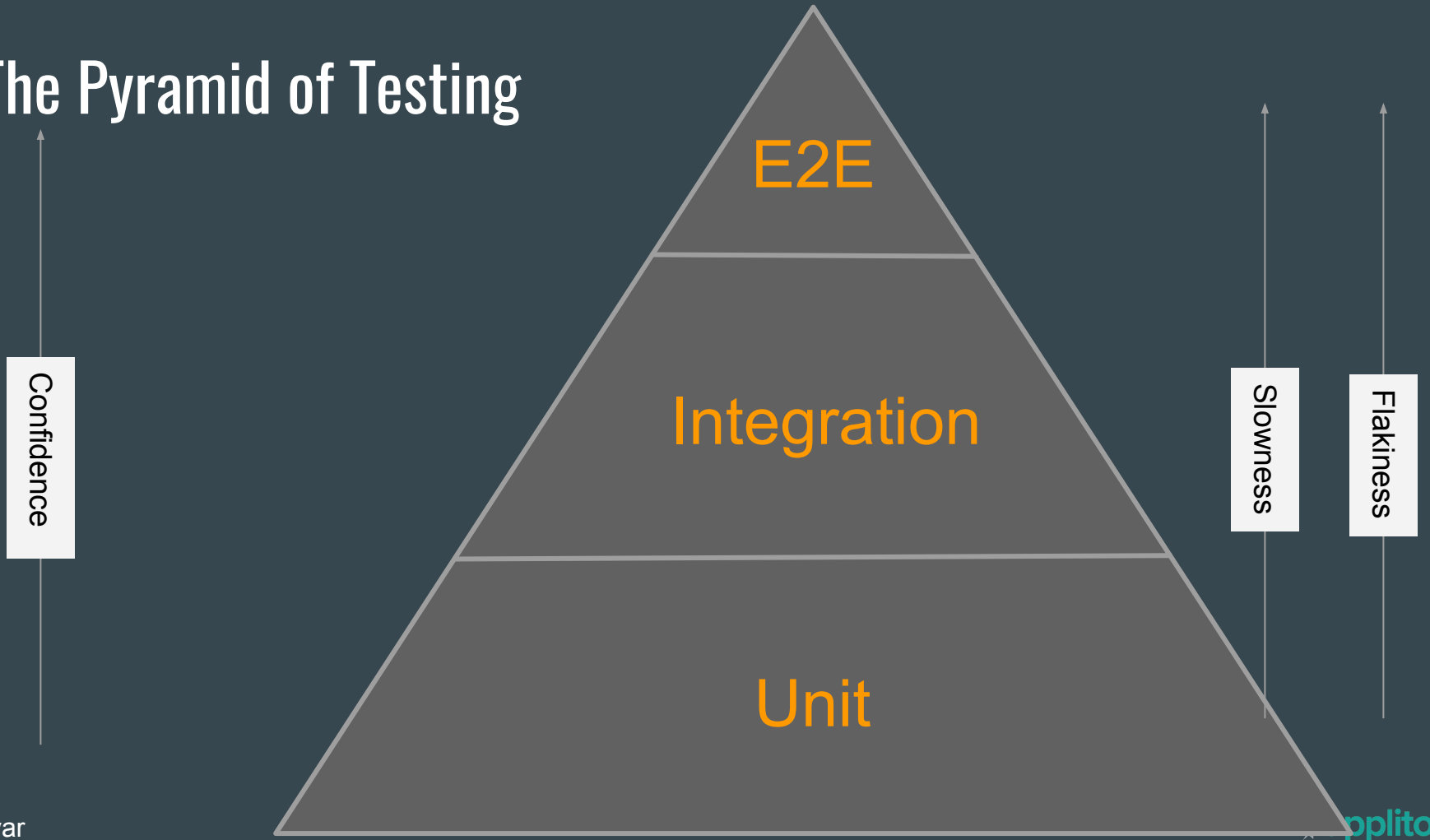
The “Diamond” School



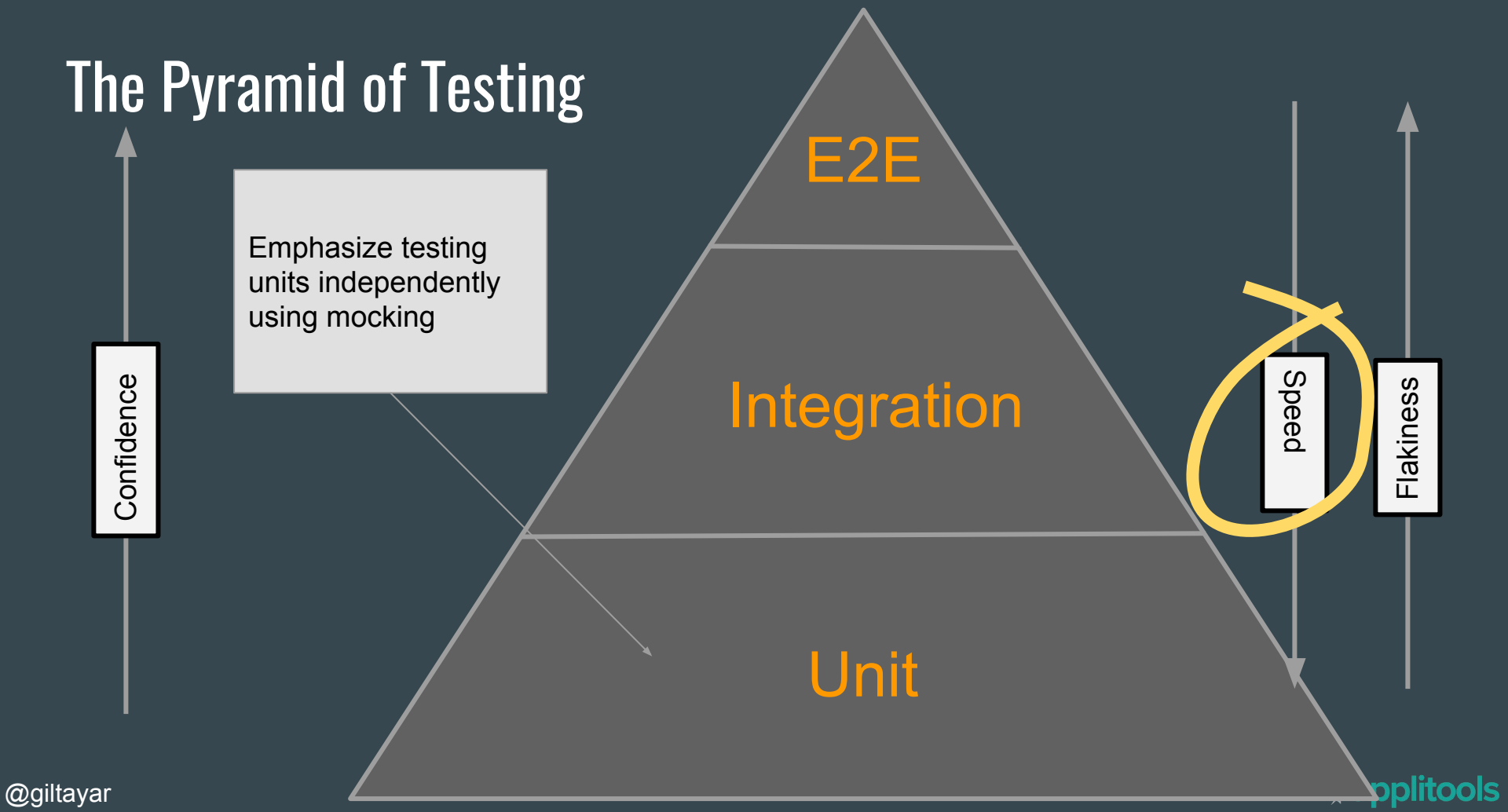
Hexagonal (Ports & Adapters) Architecture



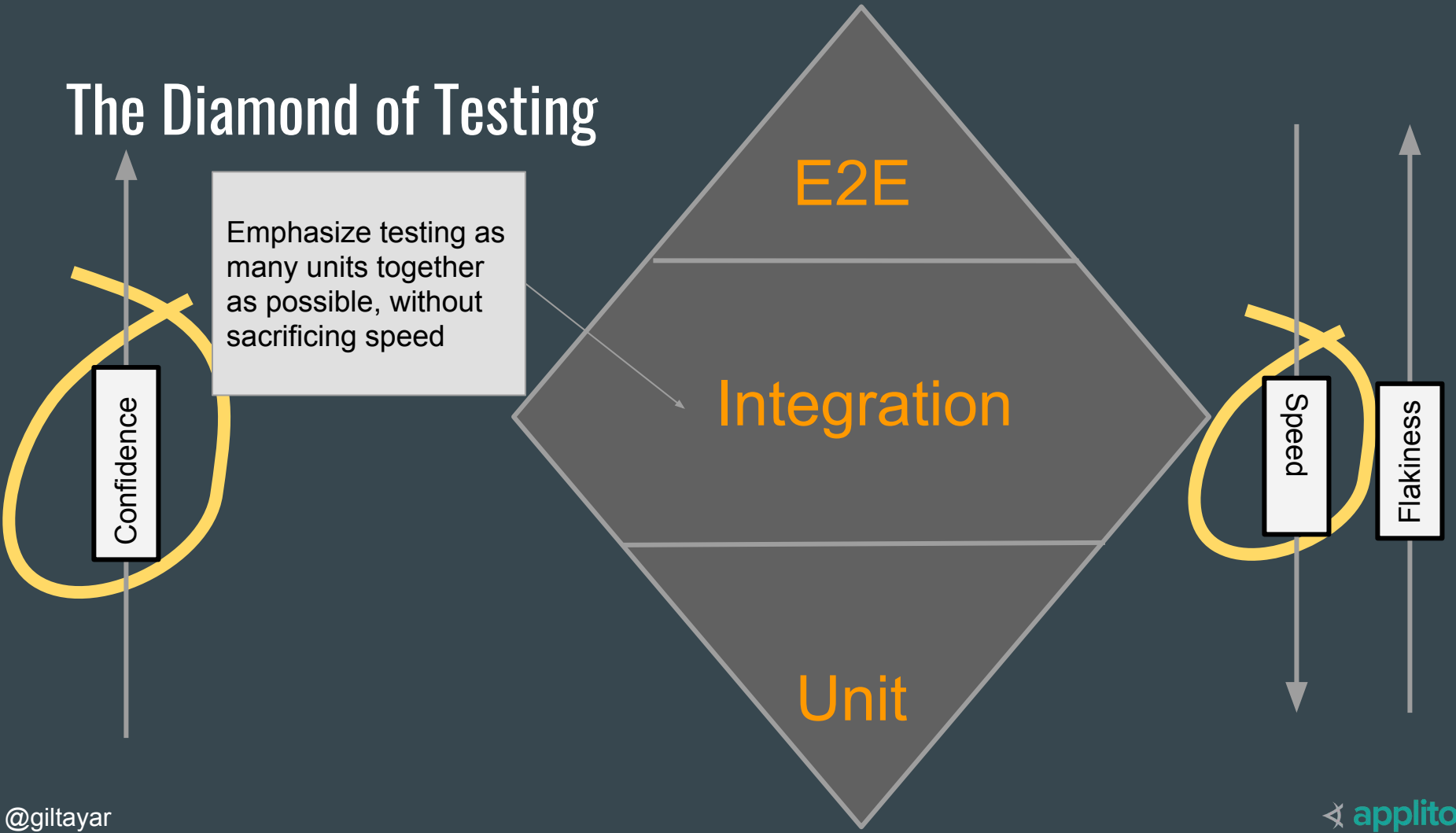
The Pyramid of Testing



The Pyramid of Testing



The Diamond of Testing



Which is better?

It's a Religion!

Commonalities

- **The need for speed**
- Few E2E tests
- **Test in all layers of the code**

Let’s continue with the guidelines...

There are fanatics out there

- Listen to them
- But don’t listen to their fanaticism
- Choose your own path
- **Choose your own testing strategy**

It doesn’t have to be TDD

- It could be, if you like it
- But it doesn’t have to be
- I don’t use it
- But some of the best developers I know do

Not too many

- Don’t go for 100% coverage. It’s a myth
- Too many tests are a problem
- **Enough tests to abolish your fear of deployment*, but no more!**

* Egoprogramophobia

How Do I Know I Wrote Enough Tests?

The Shakometer

Tests grow with time

- Don’t try to write all the tests at once
- **Let your production show you where to add tests**
- For every bug in production, add a test

Test manually as little as possible

- **Check the features through tests**
- Forces them to write tests
- More difficult with frontend code, but still possible

No “Big Project”

- “Yeah! We’re gonna work for a month and write all the tests!”
- No. Just No.
- Start small, and let it grow with time
- Ensure most *new* code is tested
- **It’s not a sprint. It’s a marathon**
- (Oh, and start with one big end to end test)

2. Every push should be ready for deployment


Realize the truth
There is no release
memeking.co.il

The QA Gateway Must Die

Treat every developer's push
as if it was meant for production

Safety Nets

How do we do that?

- Only one (“master”) branch
  - Or short-lived branches
- Small deltas
  - Big commits should be taboo
- Use feature flags
- Run a build+test (CI) process on every push
- Deploy constantly (CD) 😱

But to do that,
you need developers to write tests

This is where “shift left” ends
and where the qualitative change happens

Small deltas, coupled with *developer* tests

⇒ less bugs

Why?

- Developers don’t ignore their own test failures
- When a bug *does* pass through the tests, they can immediately figure out what caused it
    - Unit/Integration tests are much better than E2E
- Developers refactor more, and clean code means less bugs

Small deltas + *developer* tests
⇒ true agility

Why?

- Less friction ⇒
  more responsive to changes
- Less friction ⇒
  less MTTR ⇒
  less time developers spend on production bugs

The Role of Testers

In this wonderful world of “shift left”
what is the role of testers?

When not shifting left

- Developers that don't shift left are relying on testers to find their regression bugs for them
- Testers are so busy dealing with making a non-working product *work*, they don't have time to do their real job:

Ensuring the product is *good*

Or, to put it bluntly

When developers deliver a *working* product to testers, then testers can do the job they were meant to—Quality Assurance—and *not* covering the developer's a**es

This means

- Mentoring
- Pre-deployment smoke tests
- Manual testing in production
- Exploratory testing
- UX testing
- Really difficult E2E tests
- (and the list goes on and on...)

In Summary

Shift left quantitatively: shifting tests to developers

- Developers have the advantage of access to the code
- Test all layers: unit, integration, e2e
- As few e2e as possible
- The shakometer test

Shift left qualitatively: rapid, small-delta releases

- One master branch, and only short-lived branches
- Master branch should always be ready for production
- Test each push automatically using developer tests
- Less friction ⇒ more agile

Shift left: testers can do what they were supposed to do—make the product better

Thank You

...

Gil Tayar (@giltayar)

This presentation: http://bit.ly/manage-quality-shift-left-testcon