Frontend Testing: Stepping in and Collaborating with Developers

Gil Tayar (@giltayar)
October 2018

This presentation: http://bit.ly/collaborating-with-developers
Github repo: https://github.com/giltayar/collaborating-with-developers

@giltayar applitools

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

@giltayar applitools

What I'm Going to Talk About

@giltayar

applitools

Stepping in and Collaborating with Developers

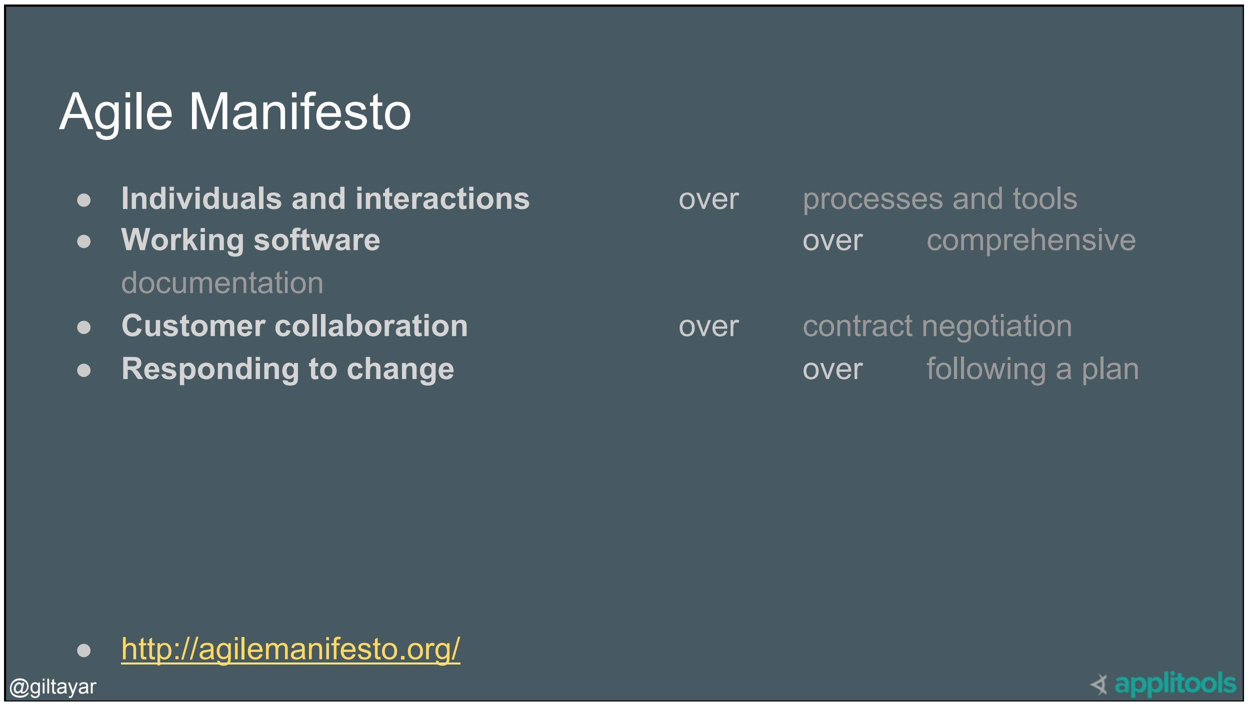
Stepping in and Collaborating with Developers

Why?

Agile

@giltayar

applitools



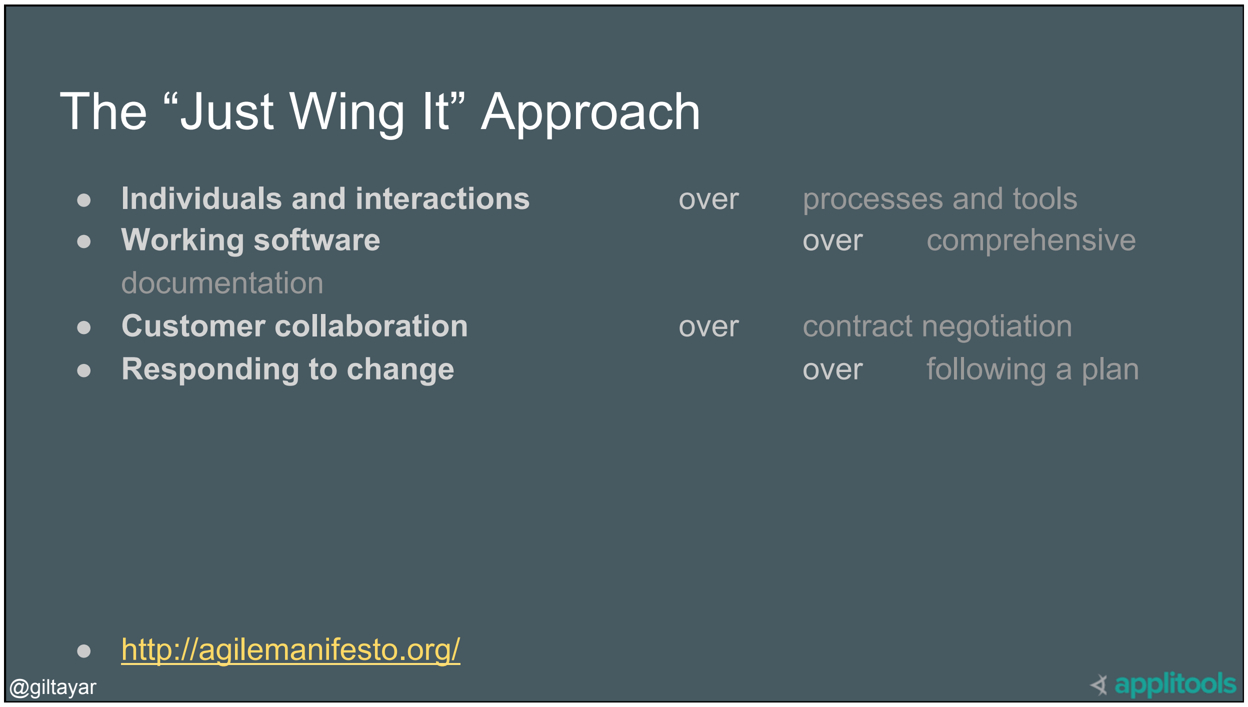
Agile Manifesto

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

- http://agilemanifesto.org/

@giltayar

applitools



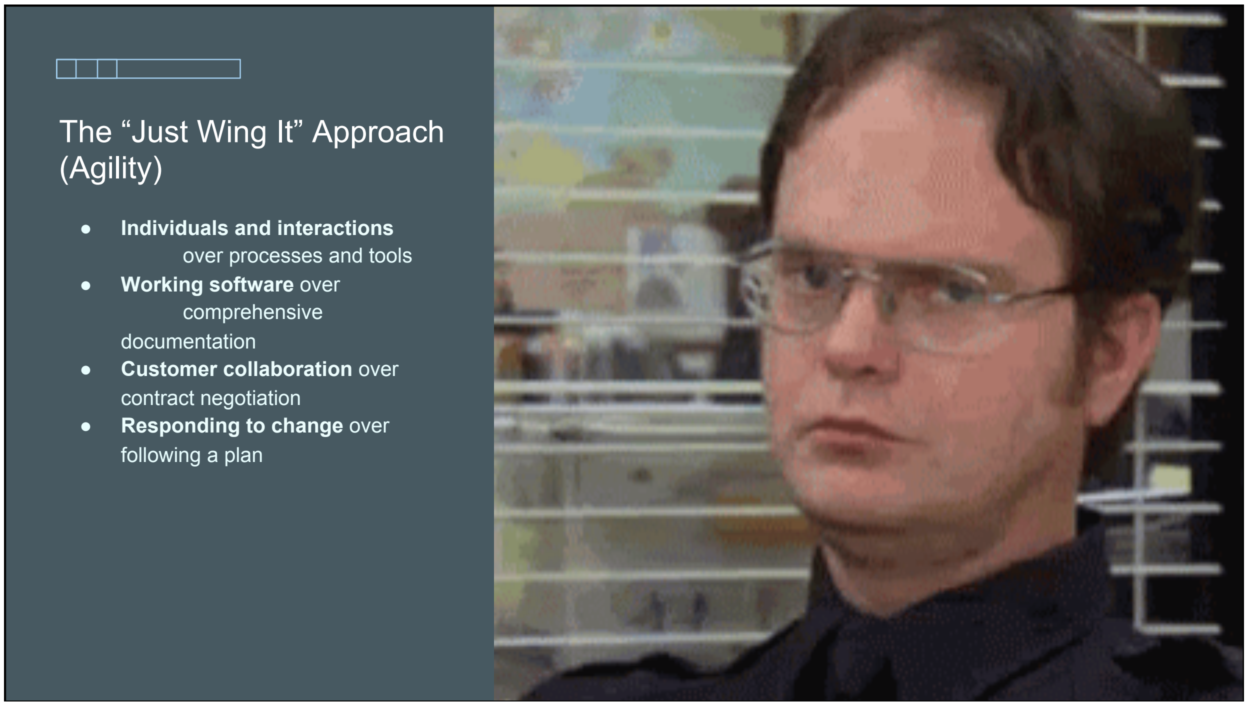
The “Just Wing It” Approach

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

- http://agilemanifesto.org/

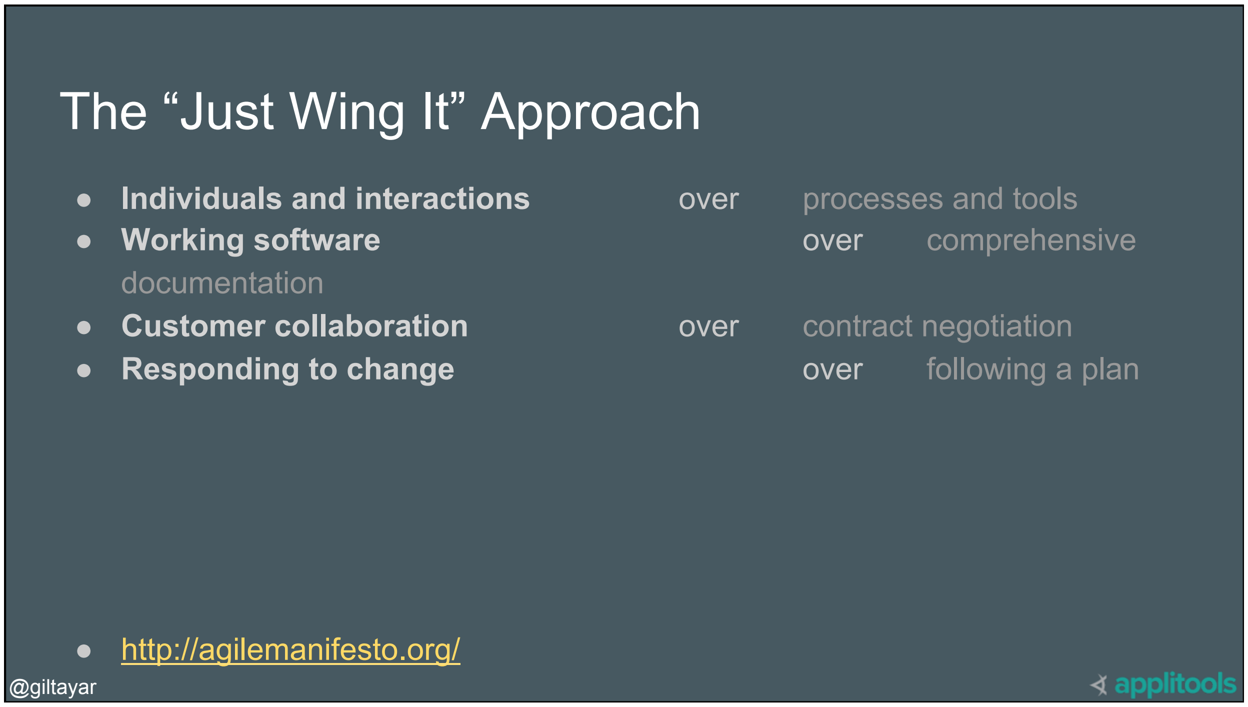
@giltayar

applitools



The “Just Wing It” Approach (Agility)

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

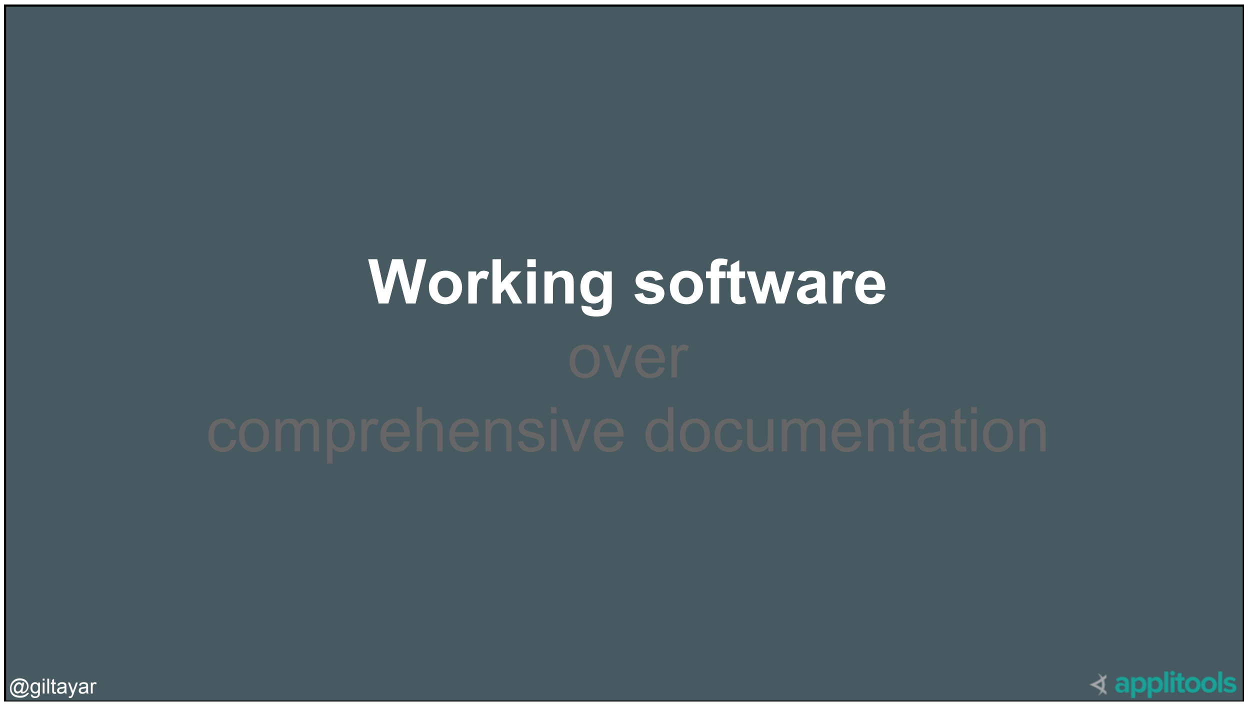


The “Just Wing It” Approach

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan
- http://agilemanifesto.org/

@giltayar

applitools



Working software

over

comprehensive documentation

@giltayar

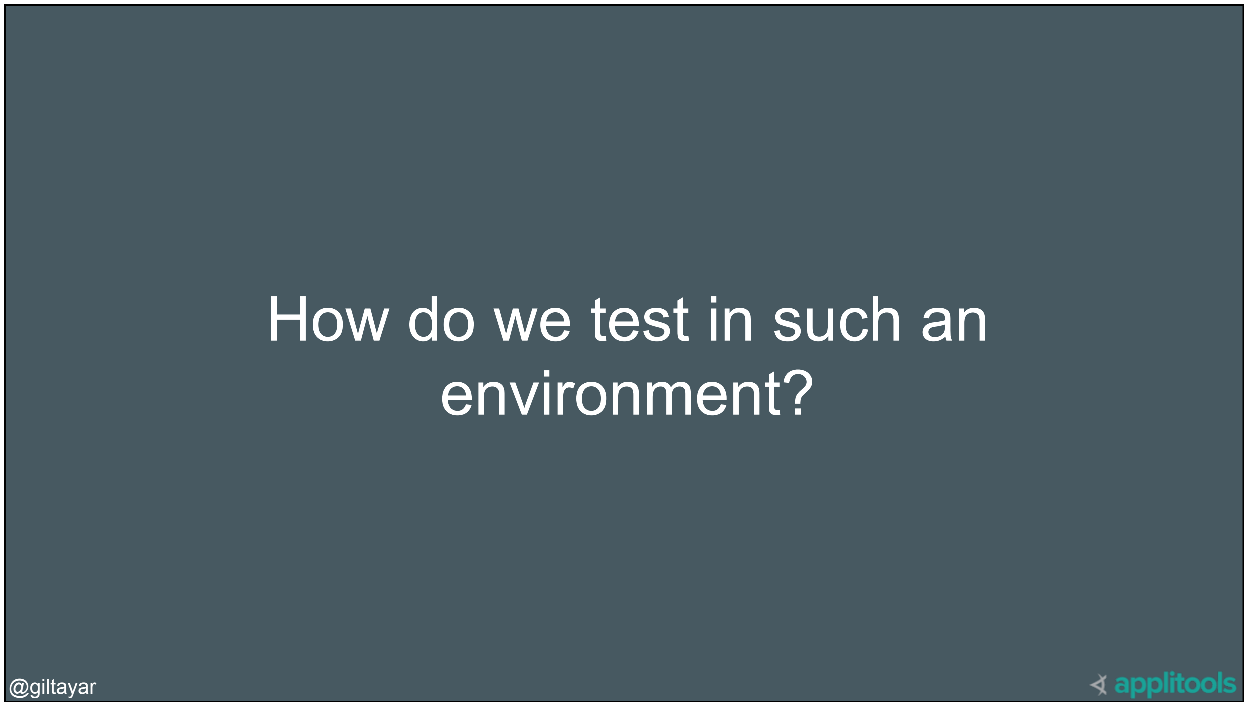
applitools

Trunk-based Development

@giltayar

applitools

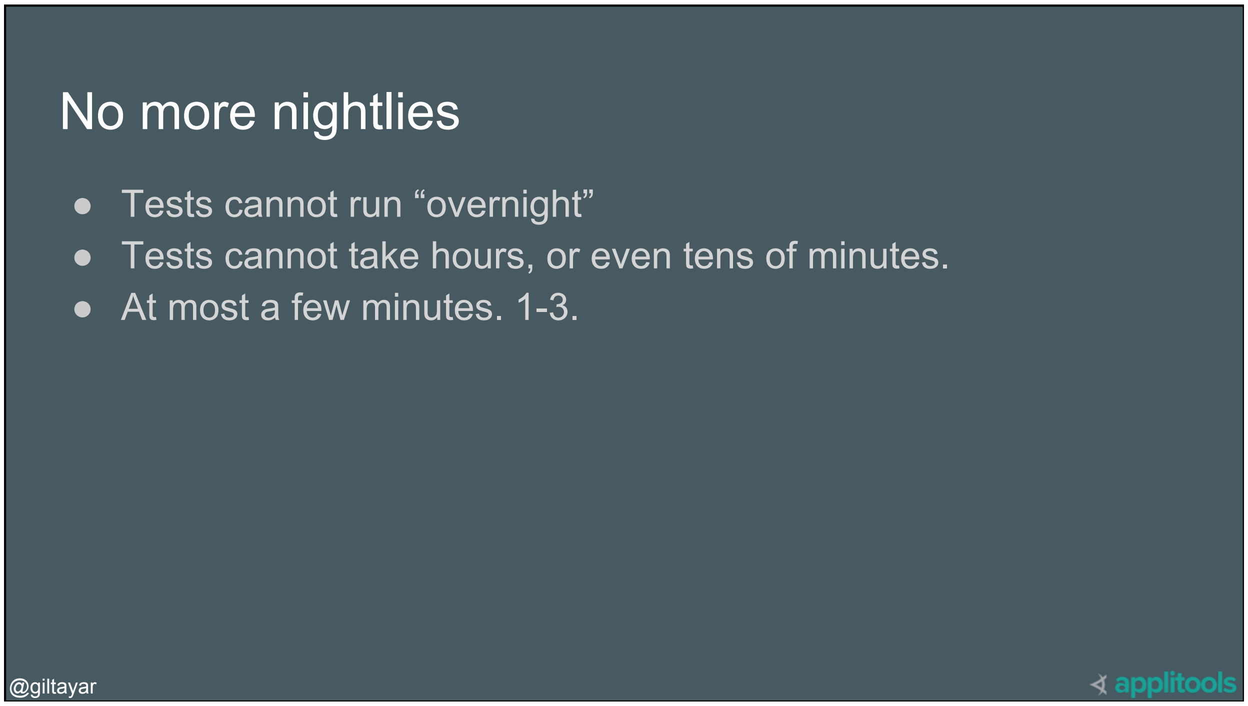
Trunk-based Development



How do we test in such an environment?

@giltayar

applitools



No more nightlies

- Tests cannot run “overnight”
- Tests cannot take hours, or even tens of minutes.
- At most a few minutes. 1-3.

@giltayar

applitools

Developers MUST Test

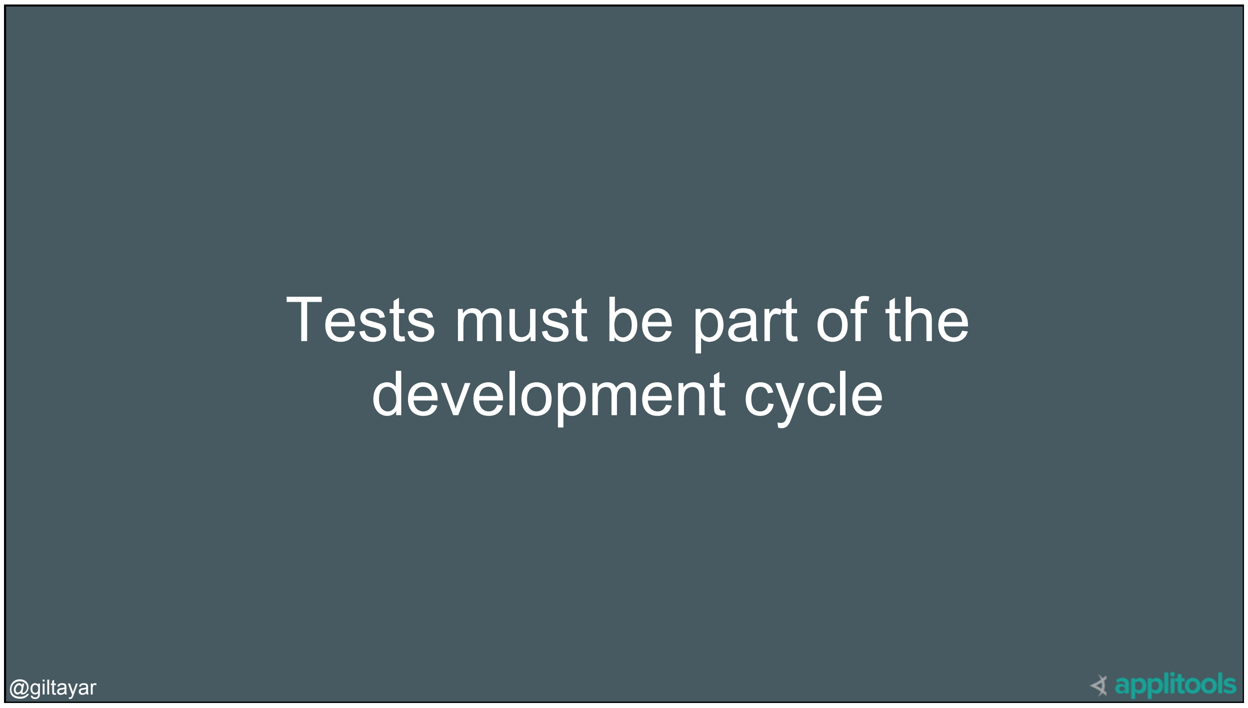
@giltayar

applitools

The QA Gateway Must Die

@giltayar

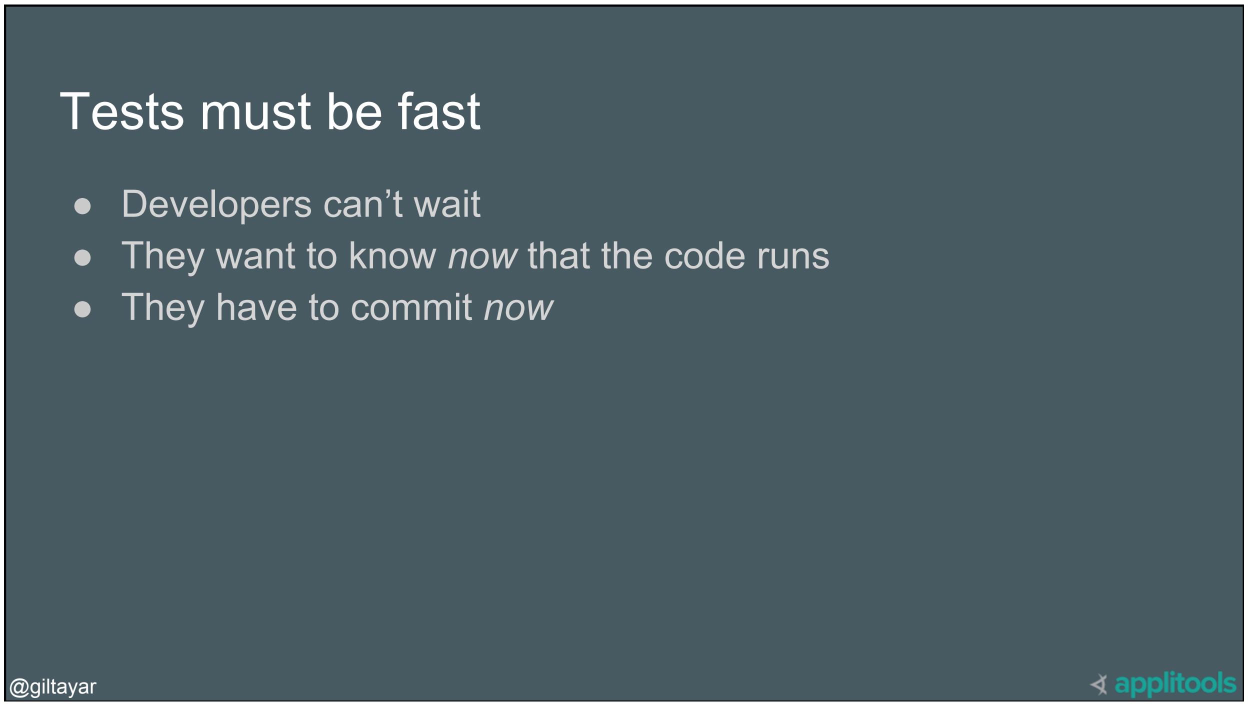
applitools



Tests must be part of the development cycle

@giltayar

applitools



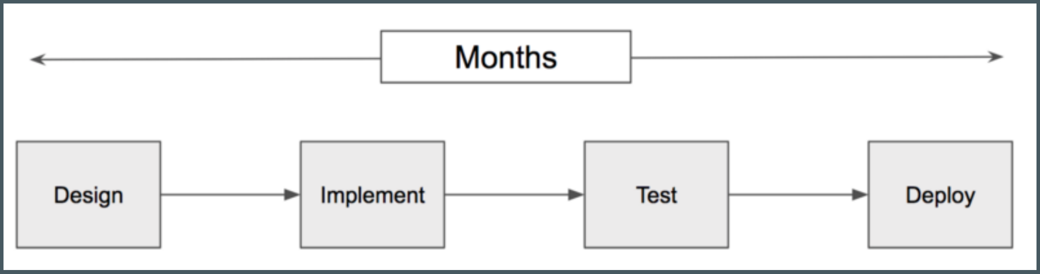
Tests must be fast

- Developers can’t wait
- They want to know *now* that the code runs
- They have to commit *now*

@giltayar

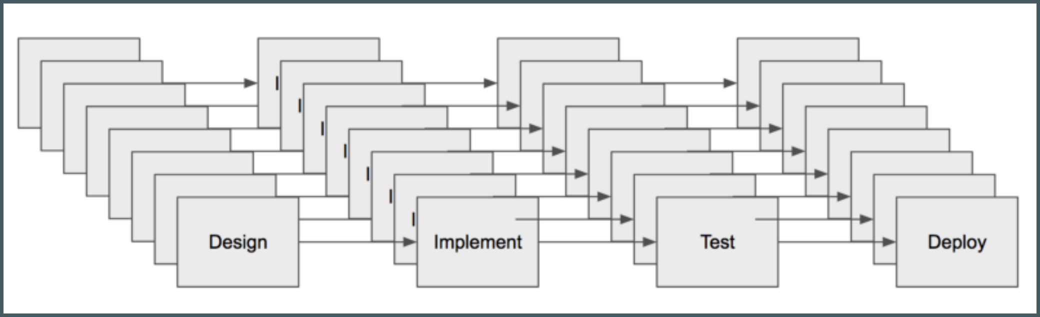
applitools

This is the "Waterfall" Method

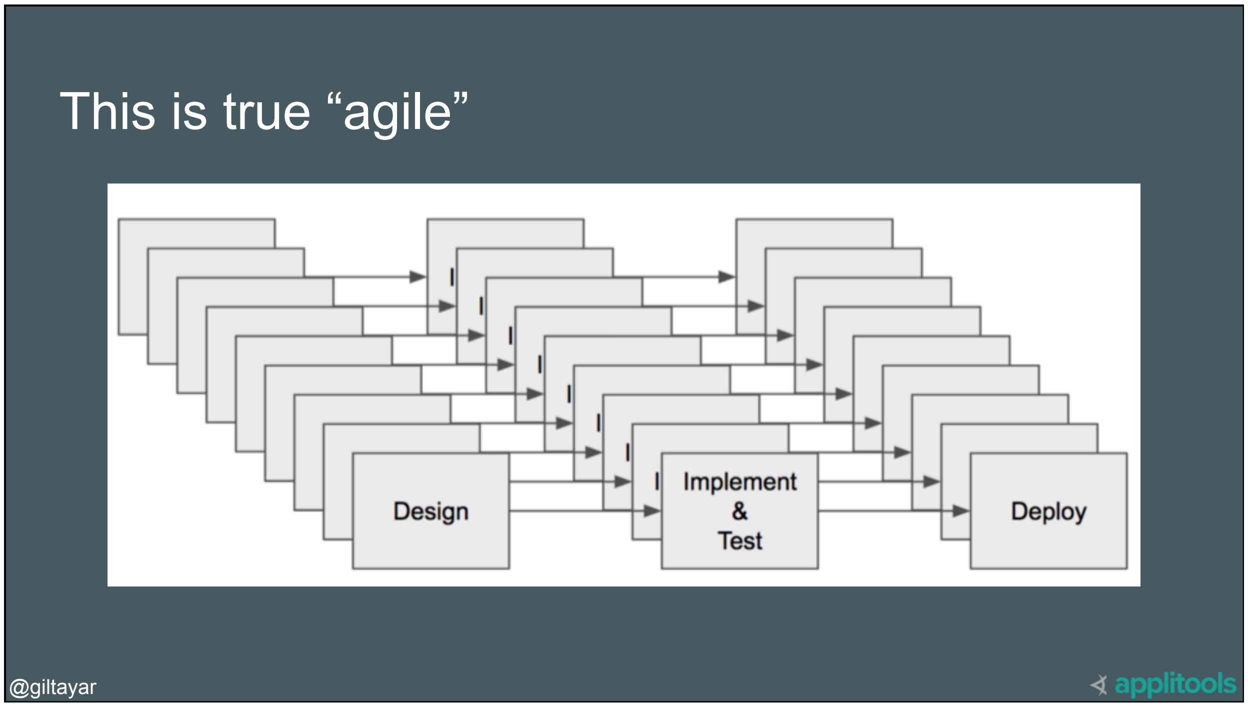


@giltayar applitools

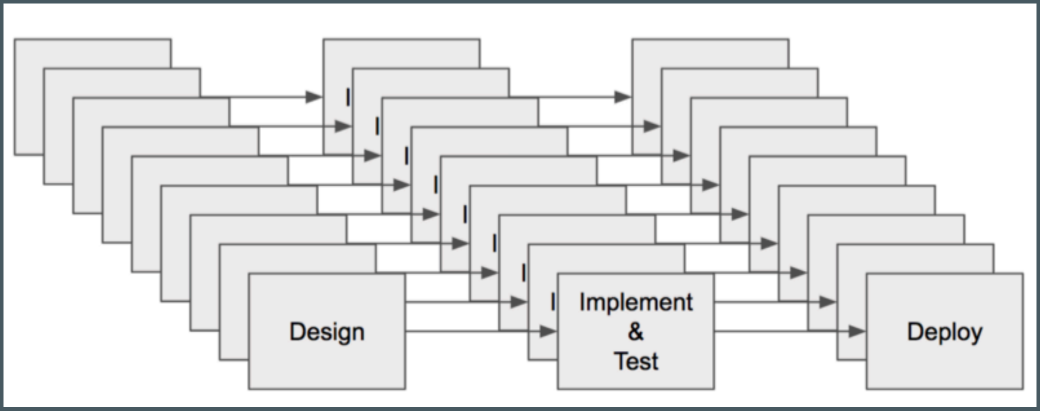
This is better, but not good enough



@giltayar applitools

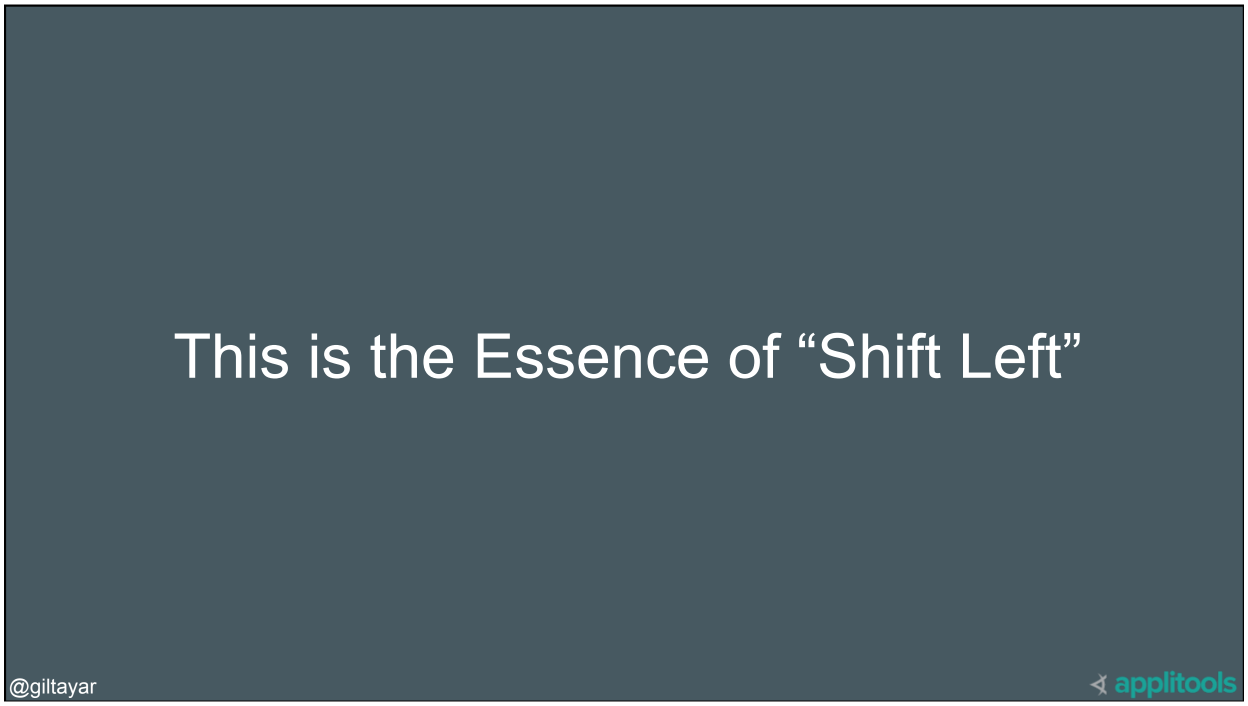


This is true "agile"



@giltayar

applitools



This is the Essence of "Shift Left"

@giltayar

applitools

Yay!
Shift Left!
Yay!

Except that...

@giltayar

applitools

Developers Don’t Test

Why Don’t Developers Test?

- They’re lazy bums
- They just “wing it”.
- “It’s gonna be alright”

Backend and Frontend Developers

@giltayar

applitools

Backend Developers Test More!

- More years building methodologies
- Easier

Frontend Developers Test Less

- It’s a young discipline
- More difficult

Frontend Testing is Young

- The whole modern Frontend Stack didn’t exist 5 years ago
  - The previous stack was impossible to test
- The current stack *is* testable
  - It took time to solidify
- But it *has* solidified now.
- There *is* a methodology that is used for frontend testing

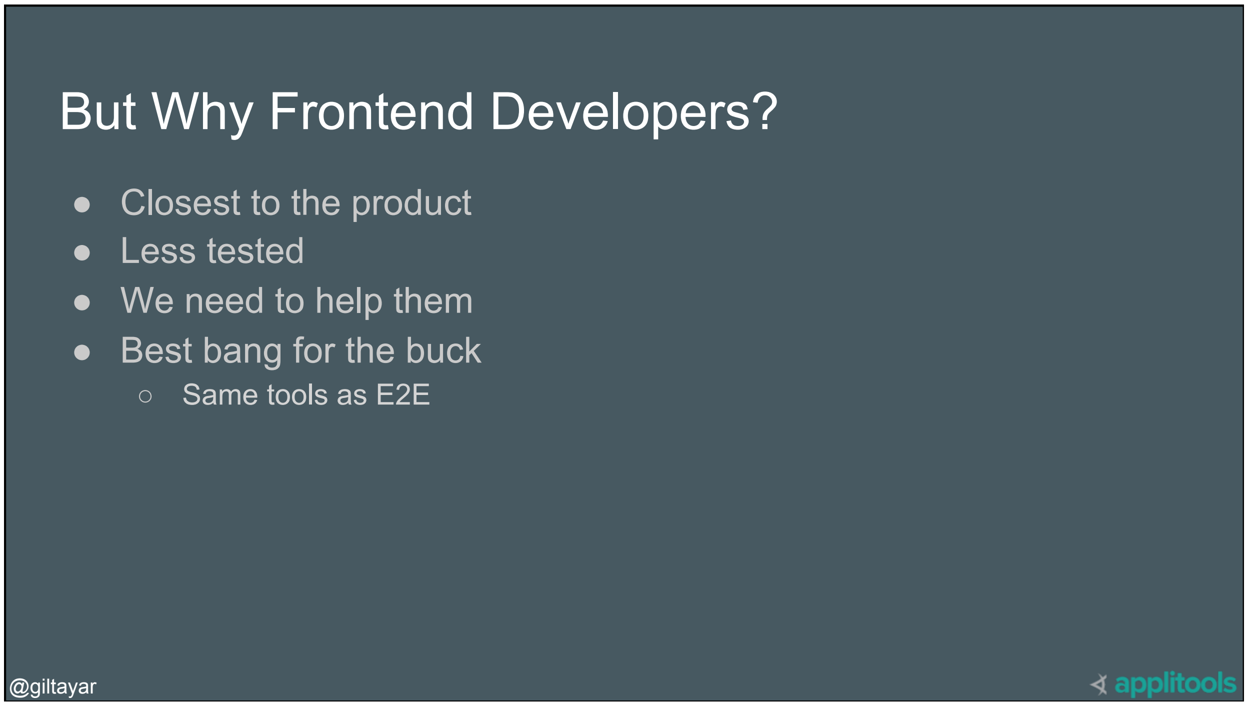
@giltayar

applitools

But Why Frontend Developers?

@giltayar

applitools



But Why Frontend Developers?

- Closest to the product
- Less tested
- We need to help them
- Best bang for the buck
  - Same tools as E2E

@giltayar

applitools

And… they’re cooler!

@giltayar

applitools

Which brings us to the second part...

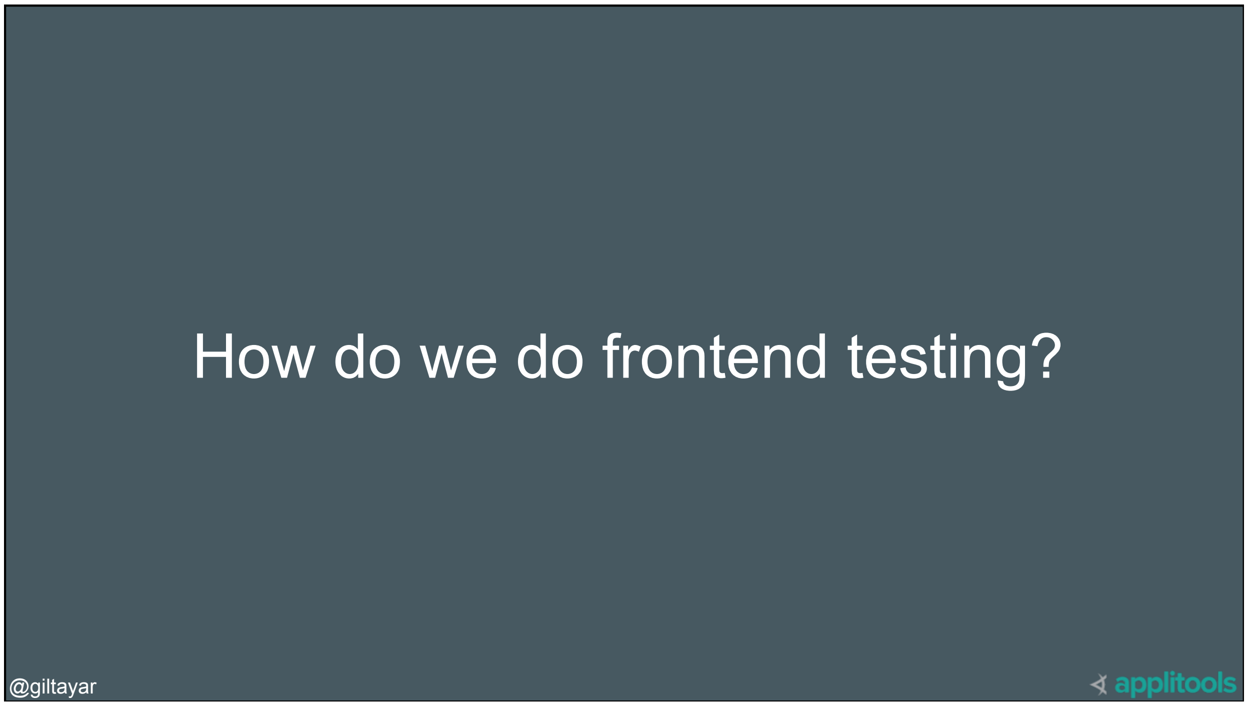
@giltayar

applitools

How?

@giltayar

applitools



How do we do frontend testing?

@giltayar

applitools

Let's start with the language

JavaScript

@giltayar

applitools

JavaScript isn't serious

- "JavaScript is a toy language"
- "JavaScript shouldn't be taken seriously"
- "It's nice for small programs"
- "0.2 + 0.1 == 0.30000000000000004"

This was true 5 to 10 years ago. Not true now

(and the last one is true in *most* languages)

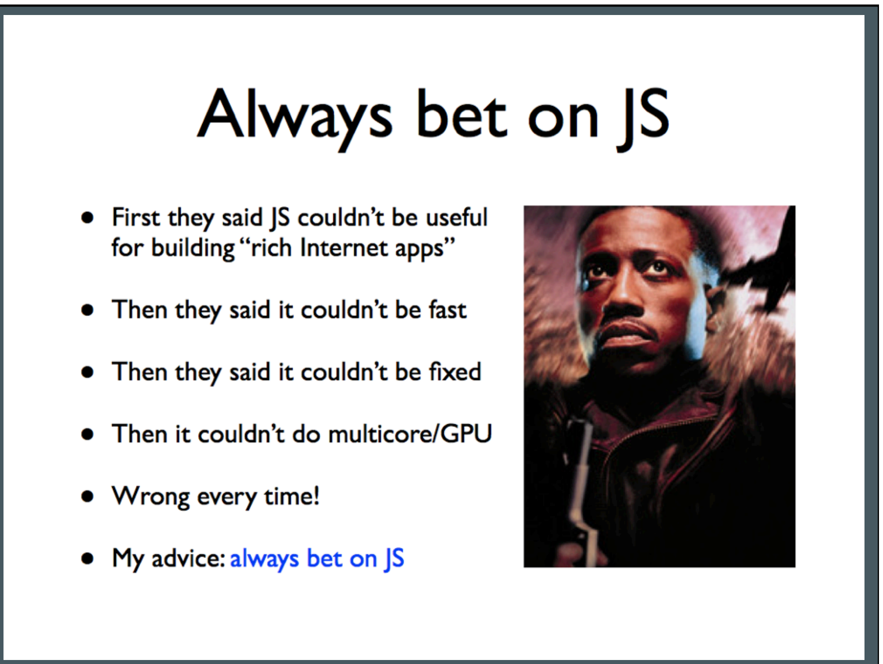
@giltayar

applitools

I have two quotes for you...

@giltayar

applitools



Always bet on JS

- First they said JS couldn't be useful for building "rich Internet apps"
- Then they said it couldn't be fast
- Then they said it couldn't be fixed
- Then it couldn't do multicore/GPU
- Wrong every time!
- My advice: always bet on JS

Brendan Eich

@giltayar

applitools

Atwood's Law

If it *can* be written in JavaScript, it *will* be written in JavaScript

@giltayar

applitools

Code Written in JavaScript

- Gmail
- Google Maps
- Twitter UI
- Facebook
- Large parts of server-side Netflix
- My favorite example:
  a CPU+hardware emulator that boots Linux

@giltayar

applitools

The JavaScript Renaissance

JavaScript today is...

- Modern
- Powerful
- Concise
- Functional
- Readable
- Ubiquitous (browser, server, CLI, IoT)
- Has the richest and largest 3rd party library *in the world*
- ...and is continually evolving

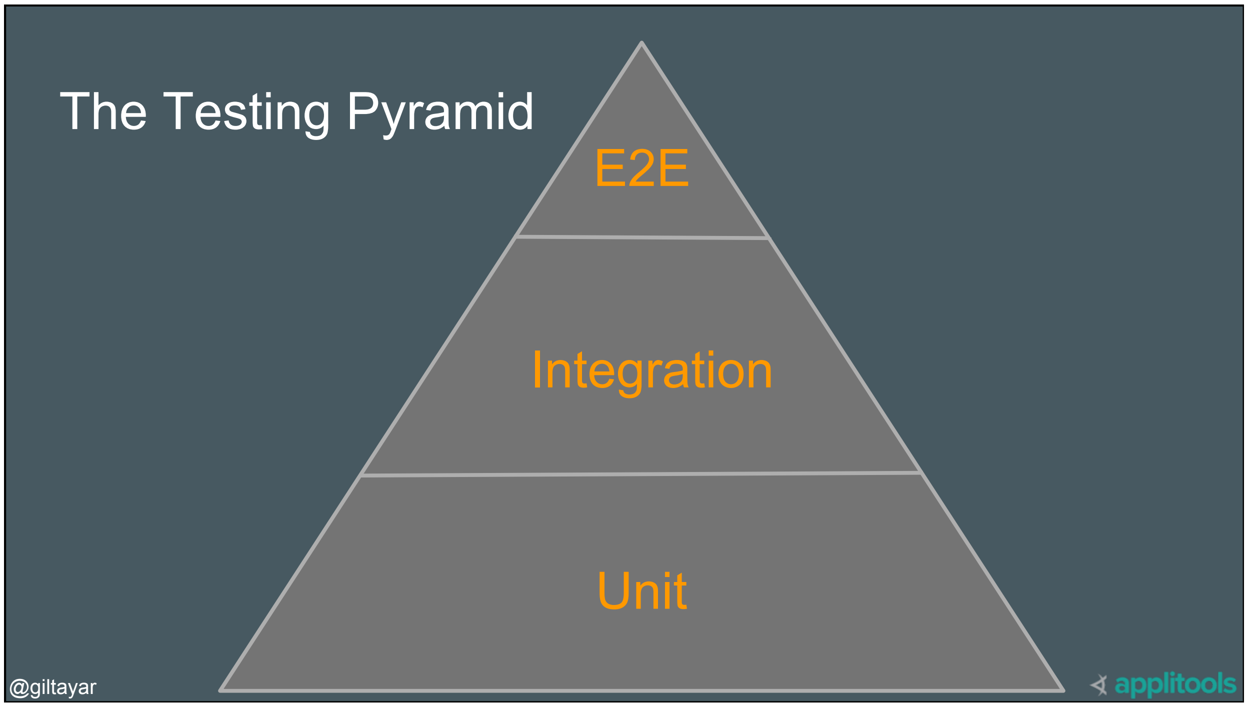
@giltayar

applitools

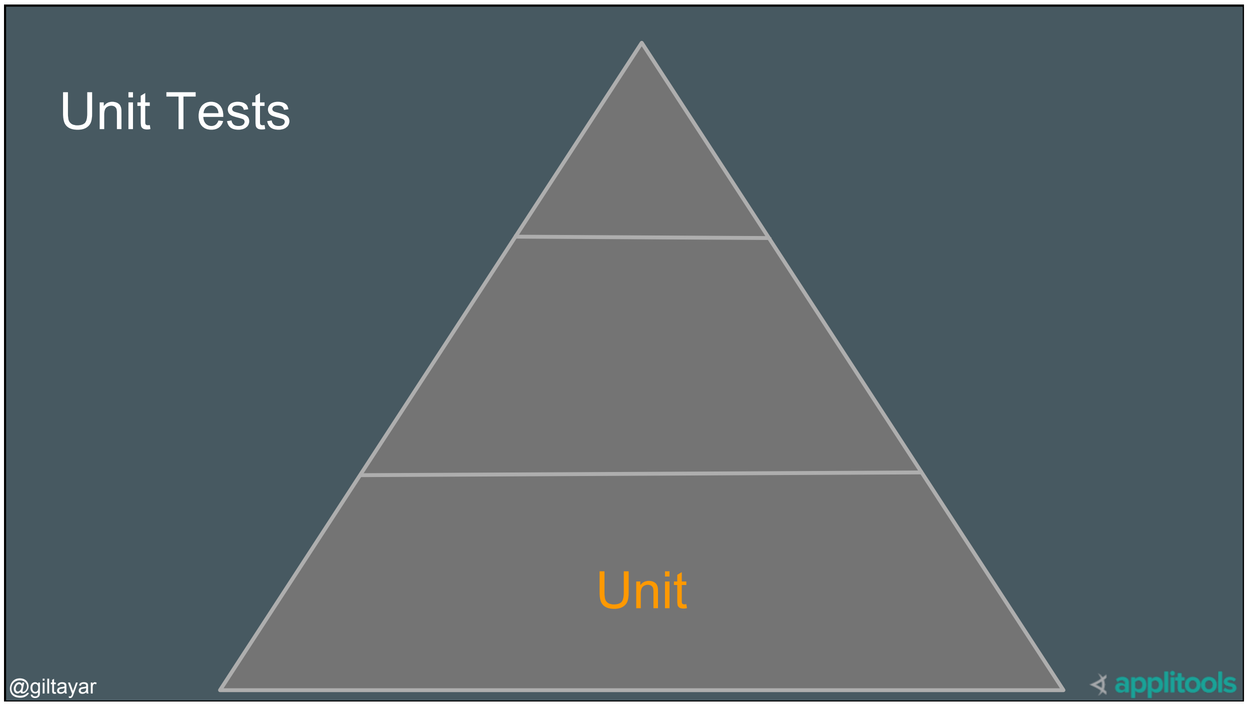
Next Thing: Testing Methodology

@giltayar

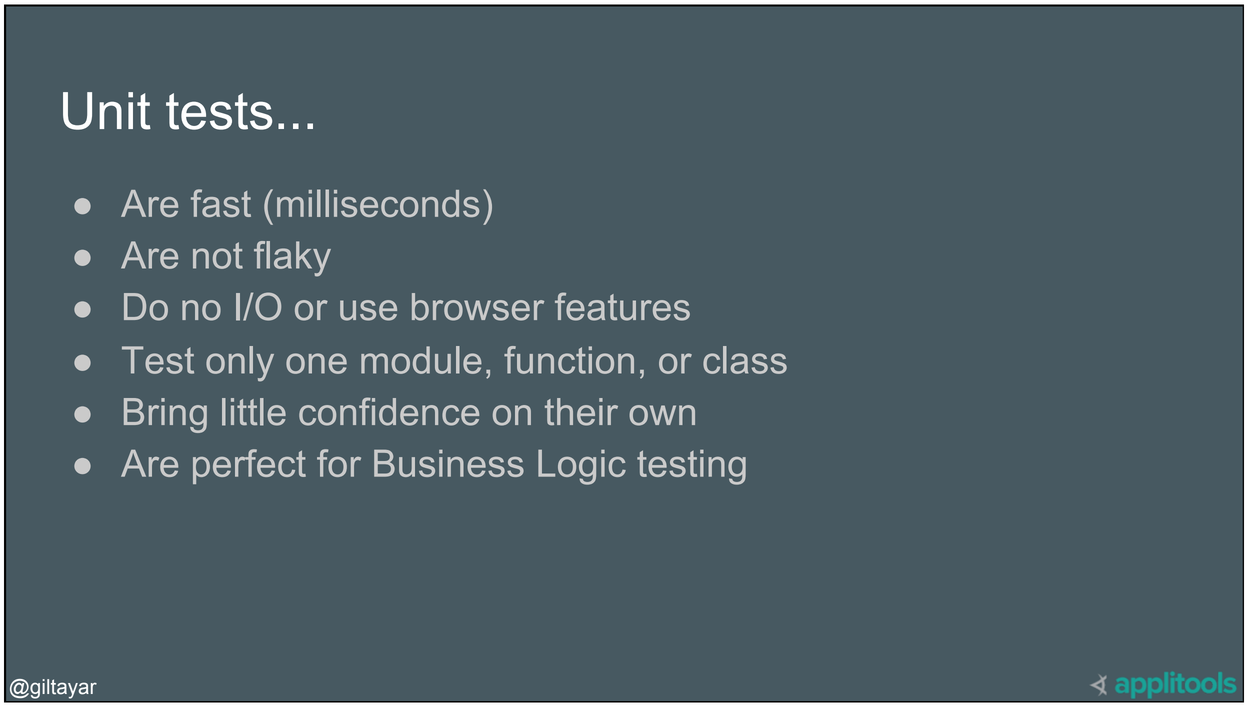
applitools



The Testing Pyramid



Unit Tests



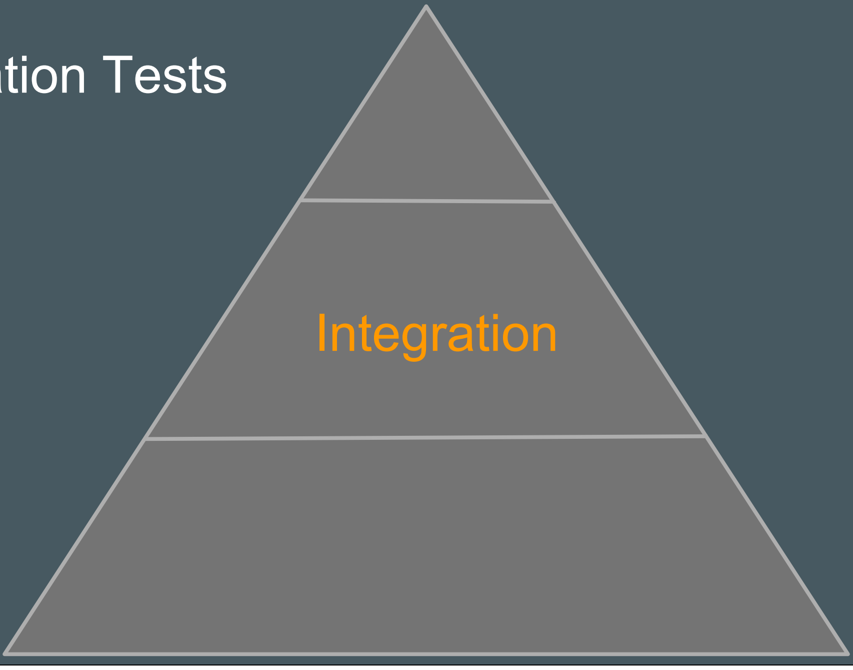
Unit tests...

- Are fast (milliseconds)
- Are not flaky
- Do no I/O or use browser features
- Test only one module, function, or class
- Bring little confidence on their own
- Are perfect for Business Logic testing

@giltayar

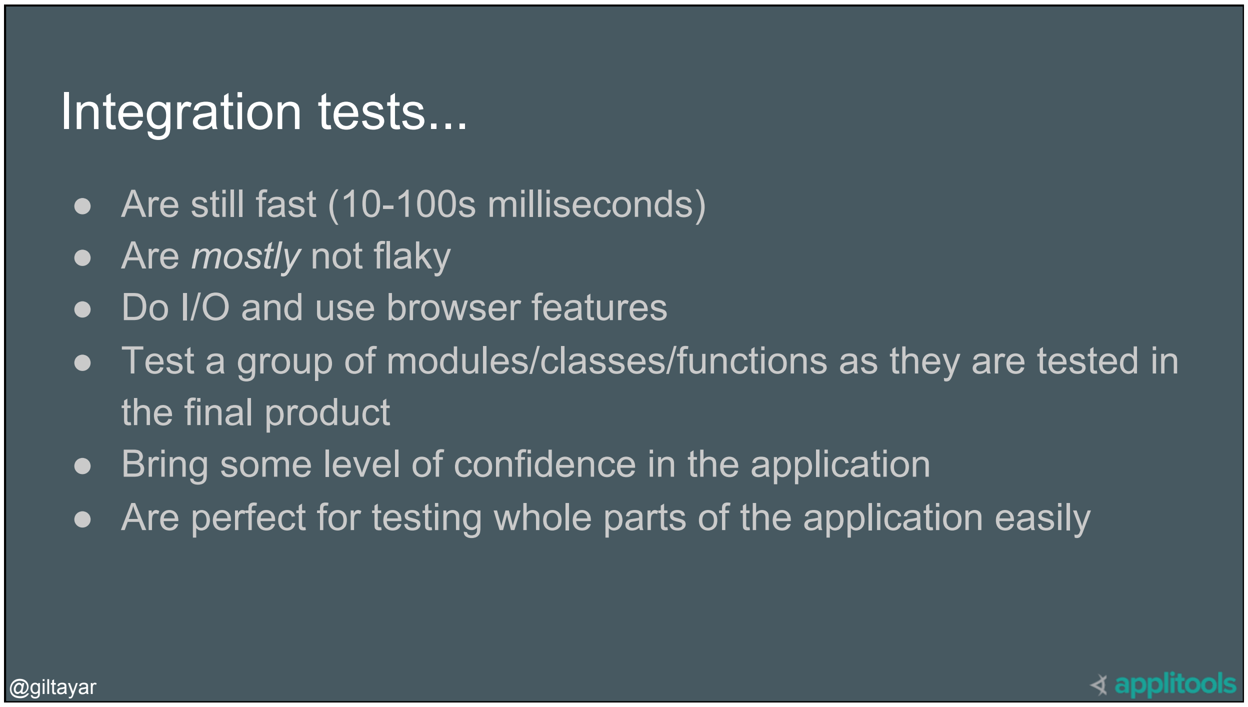
applitools

Integration Tests



@giltayar

applitools



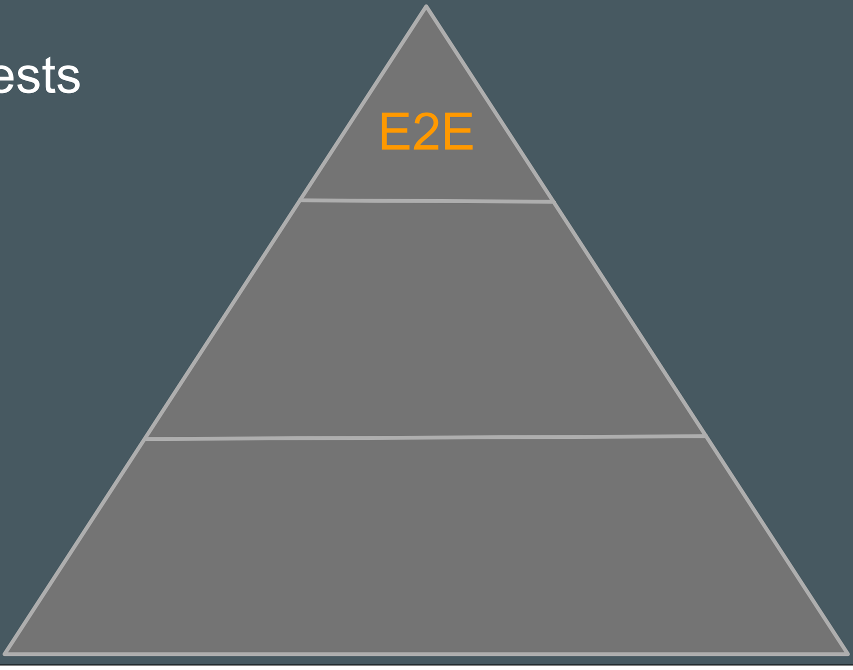
Integration tests...

- Are still fast (10-100s milliseconds)
- Are *mostly* not flaky
- Do I/O and use browser features
- Test a group of modules/classes/functions as they are tested in the final product
- Bring some level of confidence in the application
- Are perfect for testing whole parts of the application easily

@giltayar

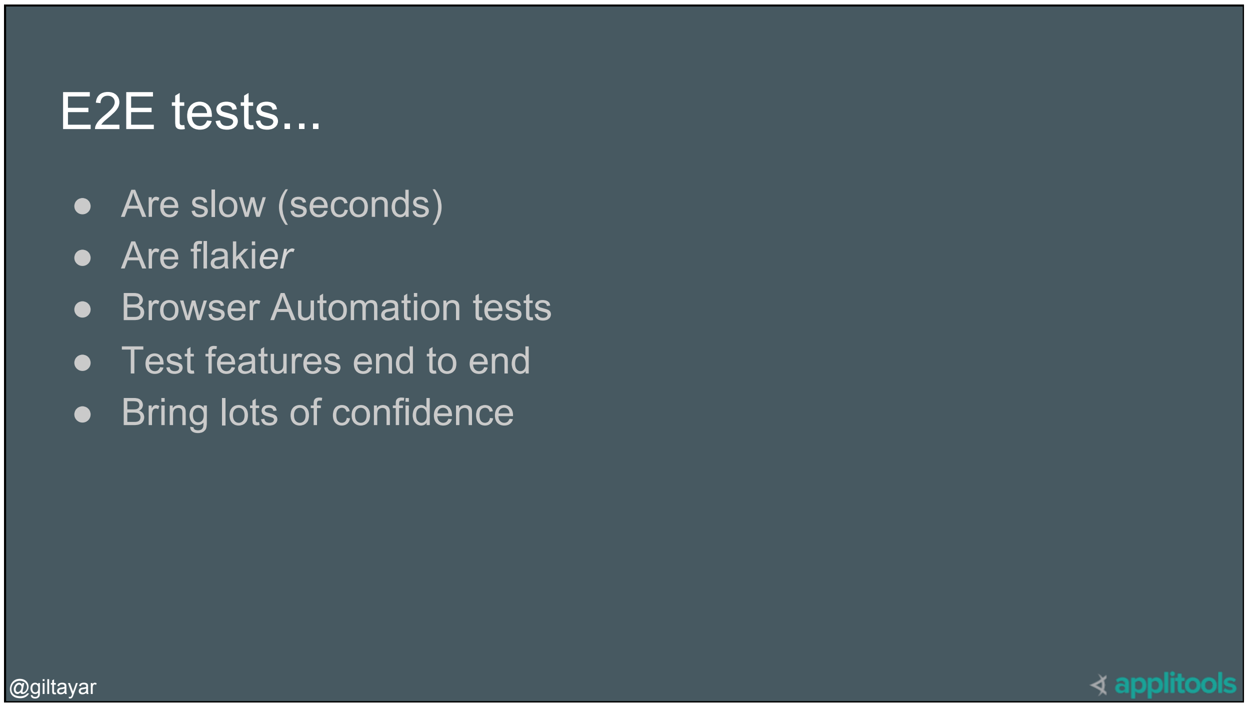
applitools

E2E Tests



@giltayar

applitools

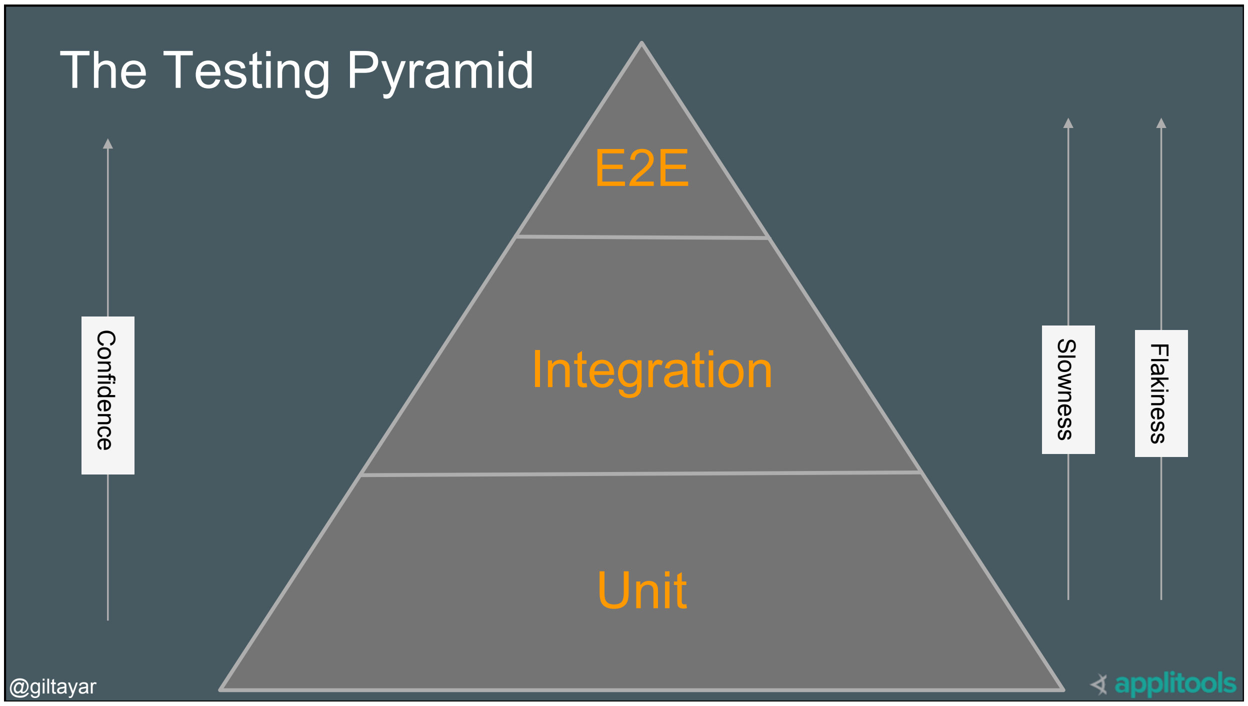


E2E tests...

- Are slow (seconds)
- Are flak*ier*
- Browser Automation tests
- Test features end to end
- Bring lots of confidence

@giltayar

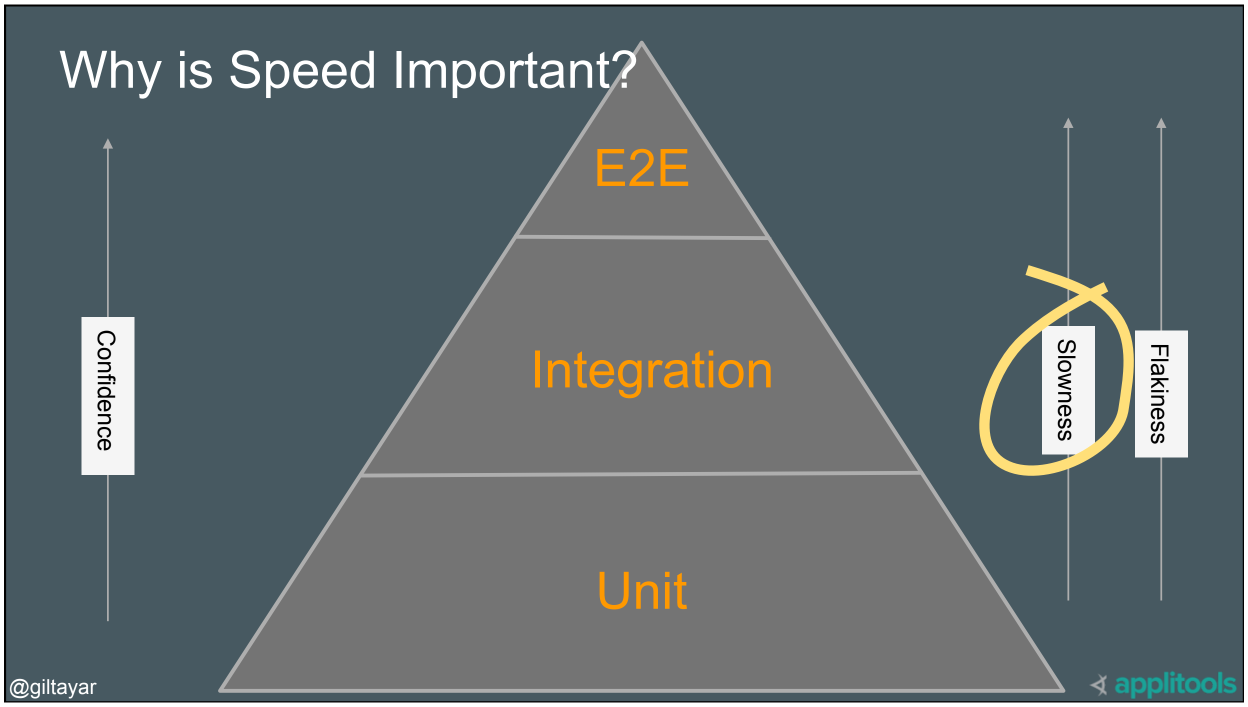
applitools



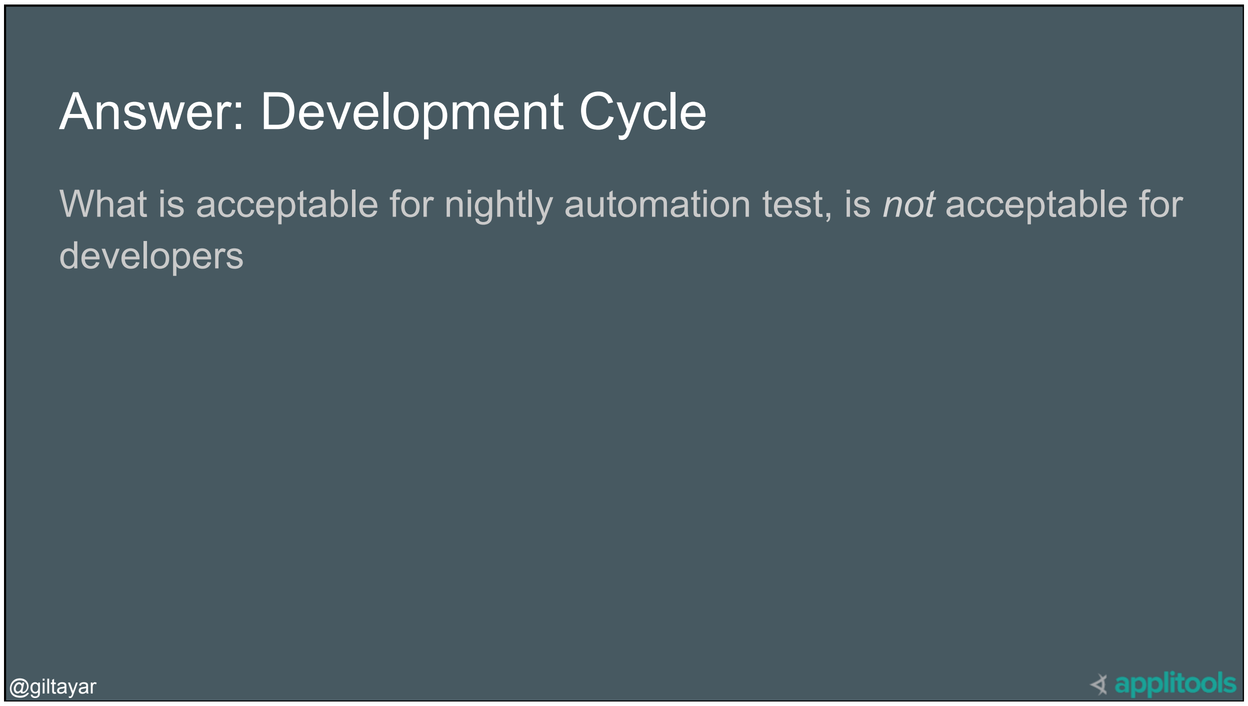
The Testing Pyramid

@giltayar

applitools



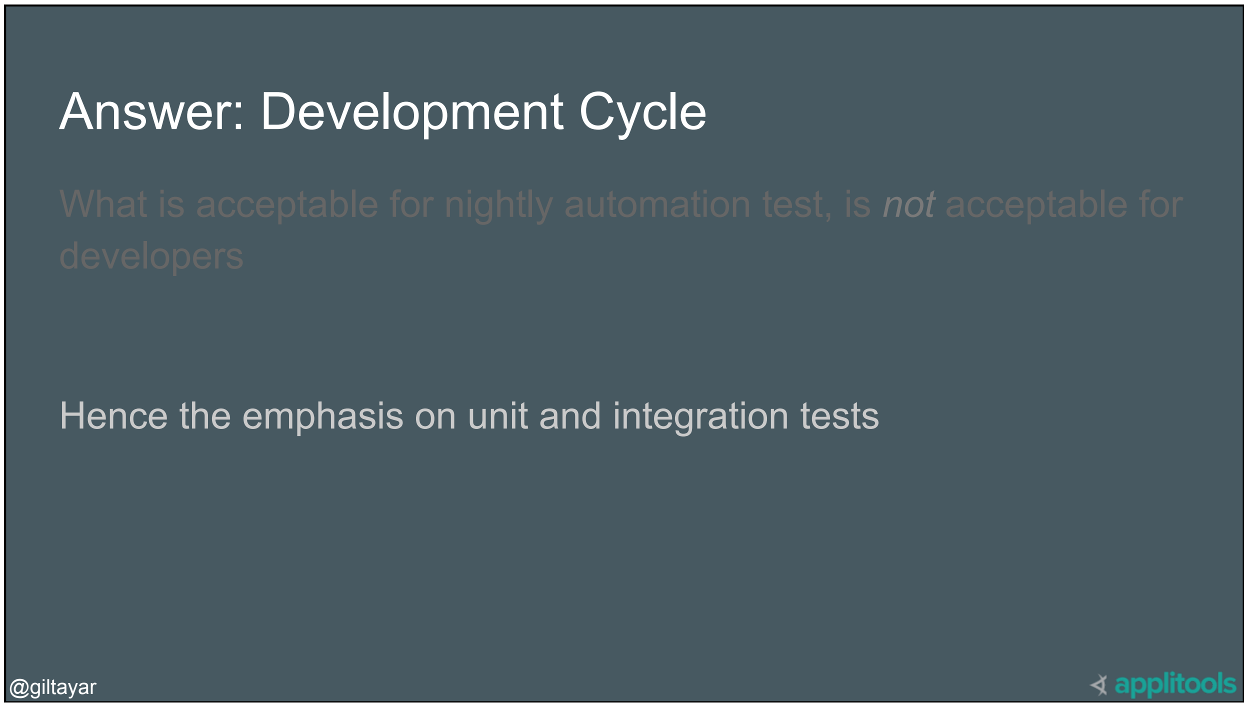
Why is Speed Important?



Answer: Development Cycle

What is acceptable for nightly automation test, is *not* acceptable for developers

@giltayar
applitools



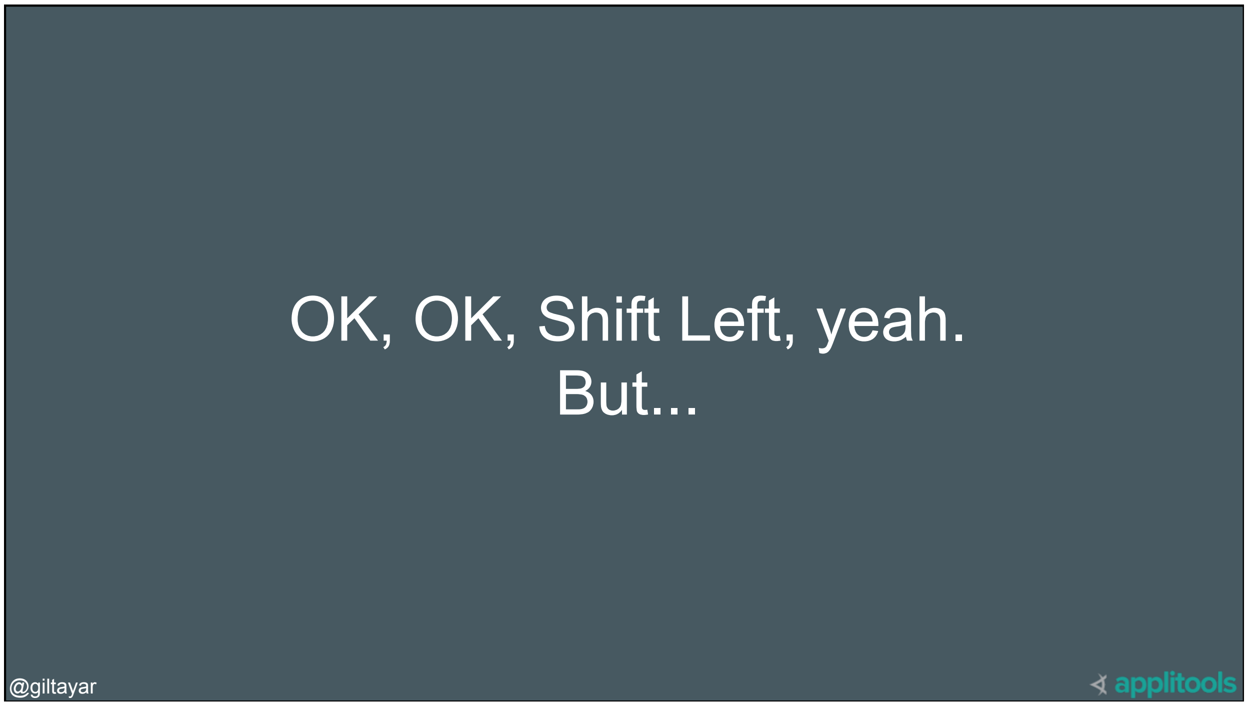
Answer: Development Cycle

What is acceptable for nightly automation test, is *not* acceptable for developers

Hence the emphasis on unit and integration tests

@giltayar

applitools



OK, OK, Shift Left, yeah.
But...

@giltayar

applitools

What's the Tester's Role?

- Educate and monitor
  - They *are* lazy bums, after all. 😉
- Work on the tests *with* the frontend developers
- Write the real E2E tests
- And… Shift Right. E2E tests in production!
  - Which you can (and should) still do with JS

@giltayar

applitools

OK, OK. But how?

How do I write tests?

Show me some code!

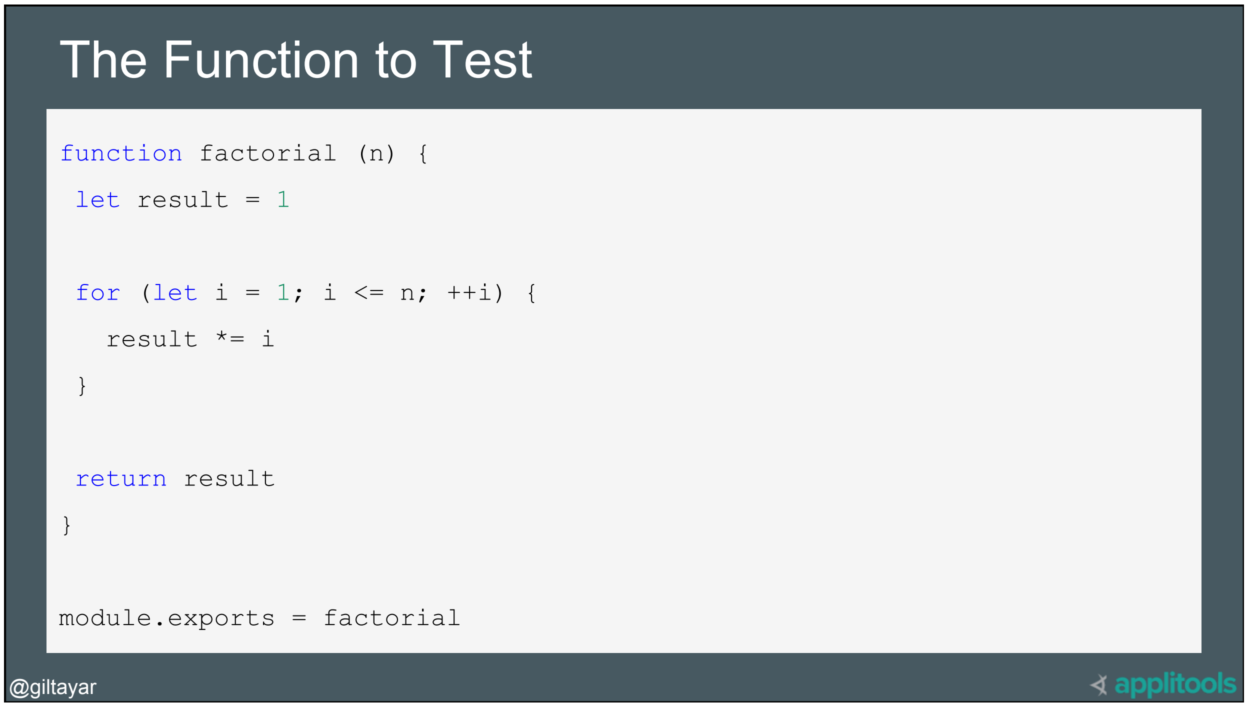
Writing Unit Tests

@giltayar

Remember….

- Unit tests test only one module, function, or class
- Bring little confidence on their own
- Are perfect for Business Logic testing
- Are very fast (milliseconds)

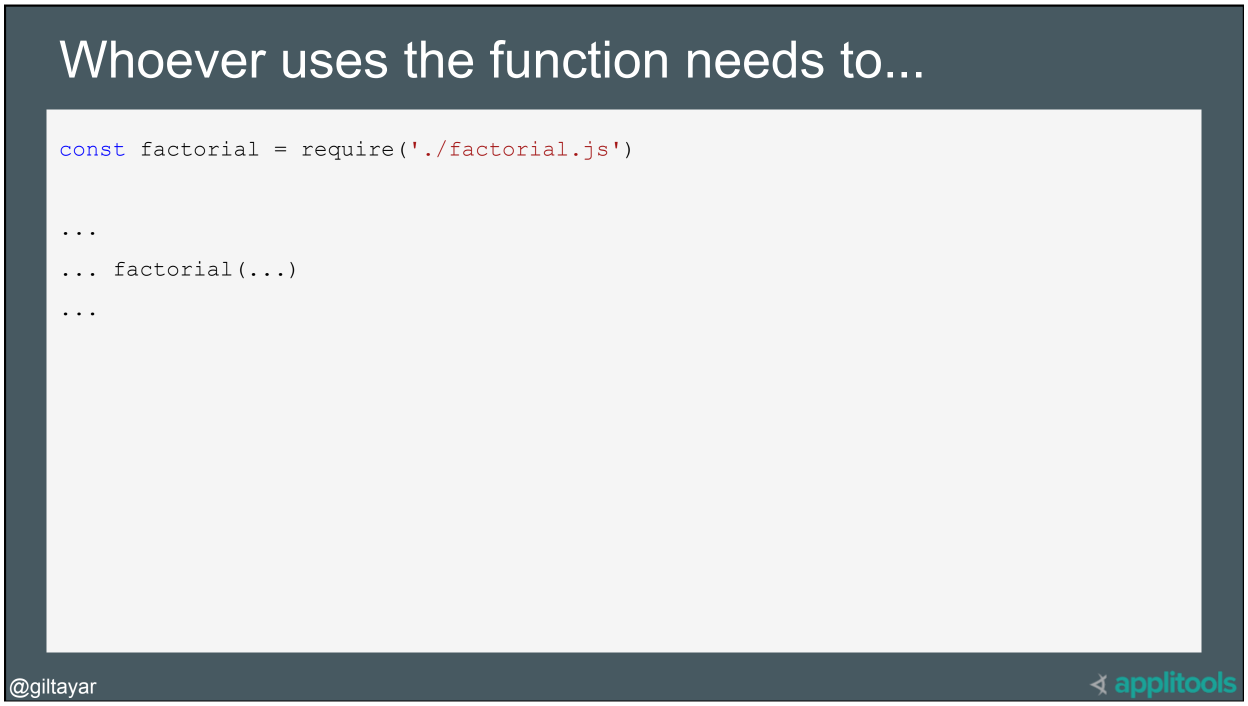
@giltayar



The Function to Test

```
function factorial (n) {
 let result = 1

 for (let i = 1; i <= n; ++i) {
   result *= i
 }

 return result
}

module.exports = factorial
```

@giltayar



Whoever uses the function needs to...

```
const factorial = require('./factorial.js')

...
... factorial(...)
...
```

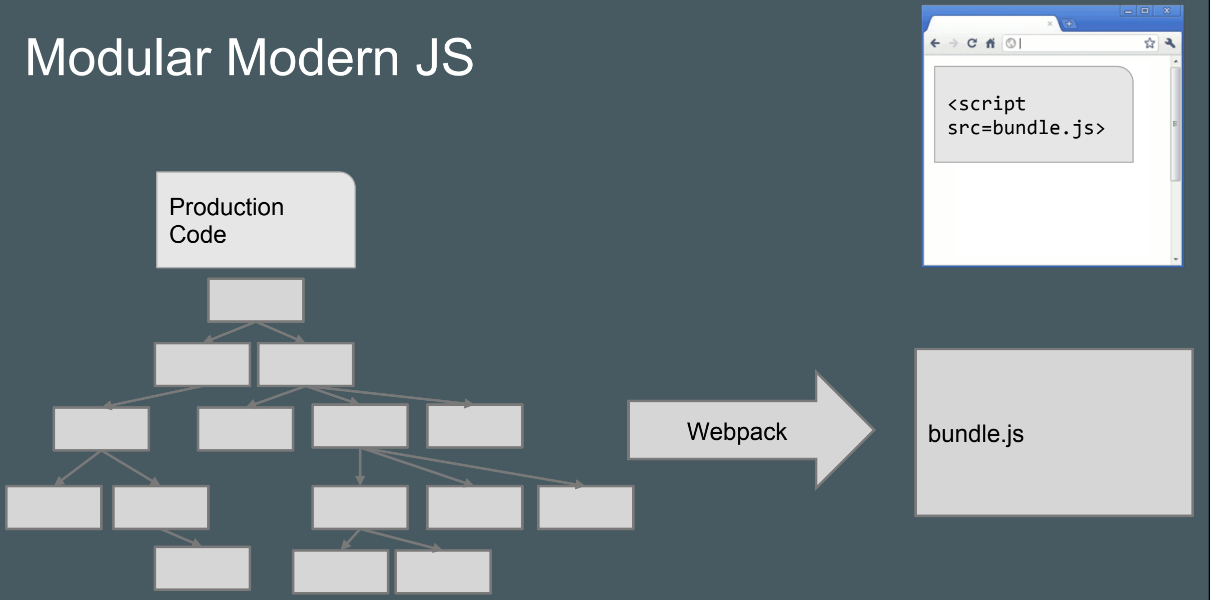
@giltayar

Does the browser support `module.exports`? No!

```
function factorial (n) {
 let result = 1

 for (let i = 1; i <= n; ++i) {
   result *= i
 }

 return result
}

module.exports = factorial
```

@giltayar applitools

Modular Modern JS



```
<script
src=bundle.js>
```

@giltayar applitools

What do we want to test?

- factorial(0) == 1
- factorial(1) == 1
- factorial(2) == 2
- factorial(5) == 120

@giltayar

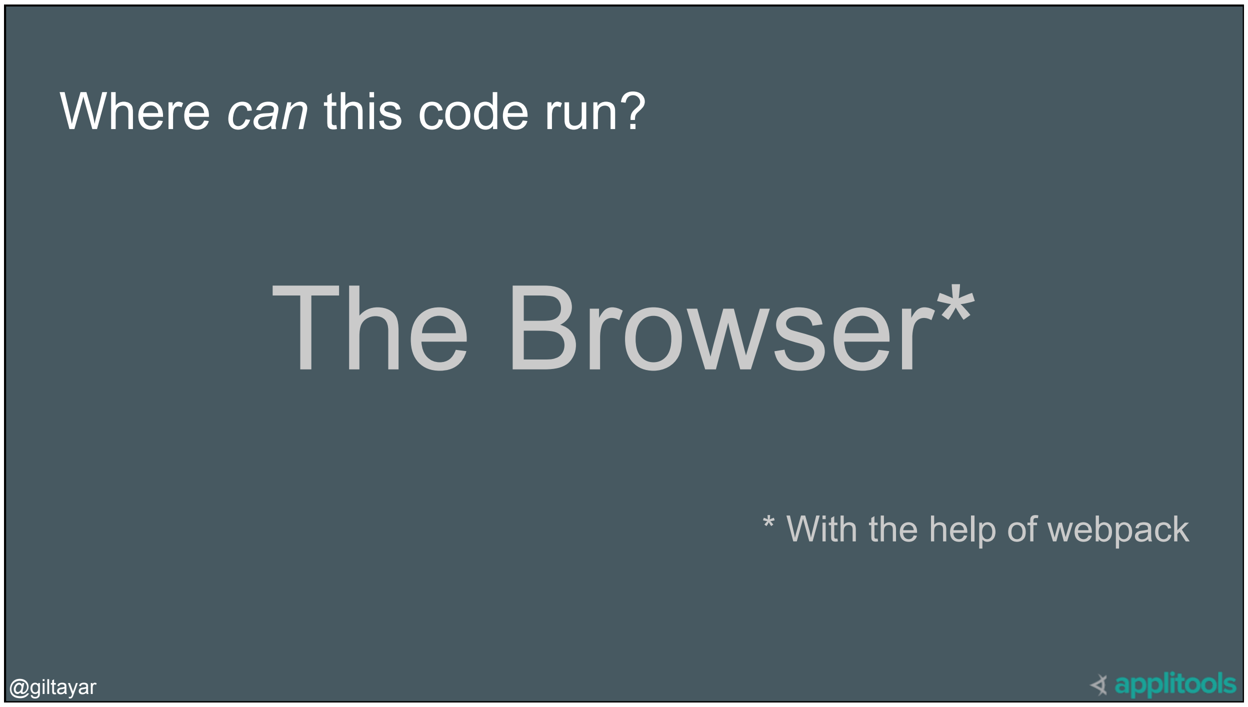
applitools

Test Factorial

```
const assert = require('assert')
const factorial = require('../../lib/factorial')

assert.strict.equal(factorial(0), 1)
assert.strict.equal(factorial(1), 1)
assert.strict.equal(factorial(2), 2)
assert.strict.equal(factorial(5), 120)
```

@giltayar

applitools

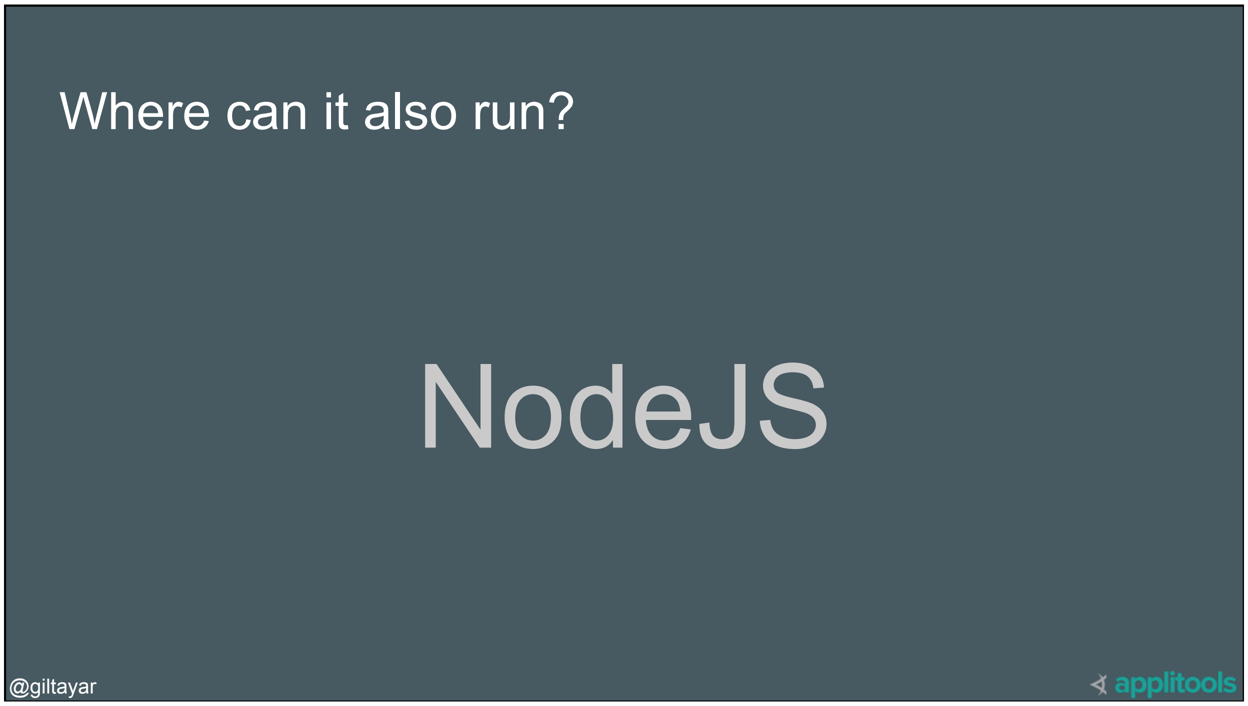


Where *can* this code run?

The Browser*

* With the help of webpack

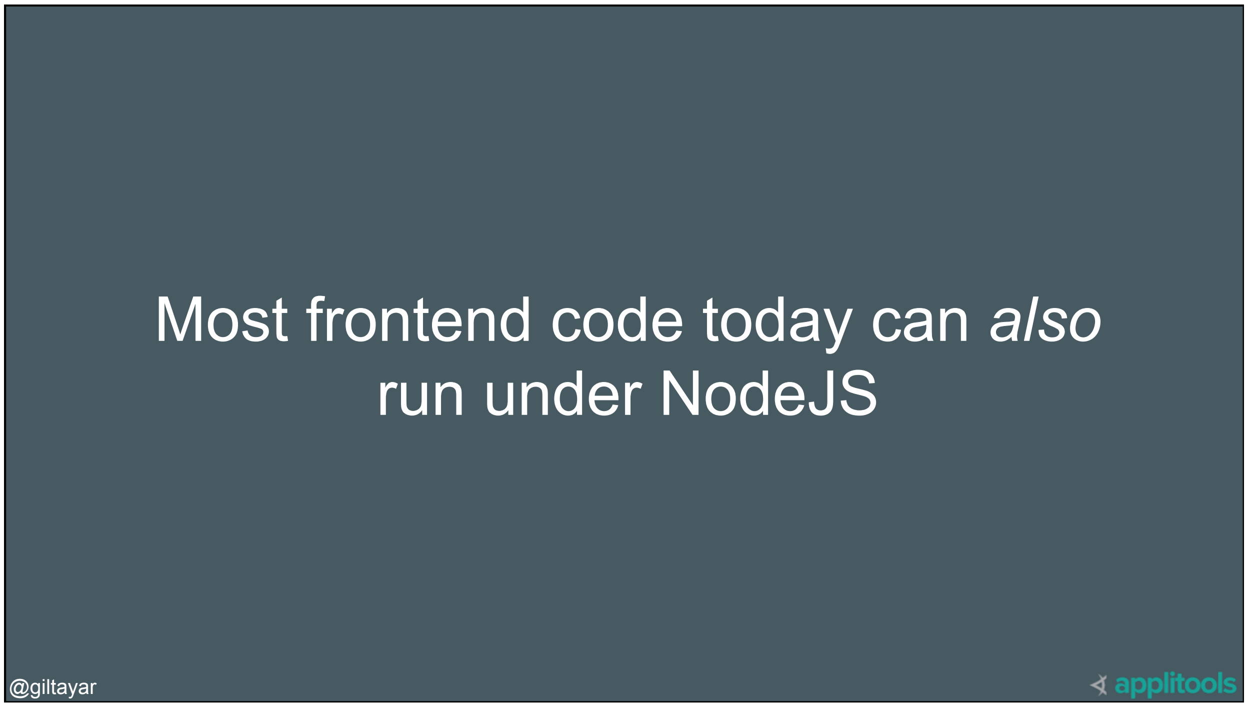
@giltayar



Where can it also run?

NodeJS

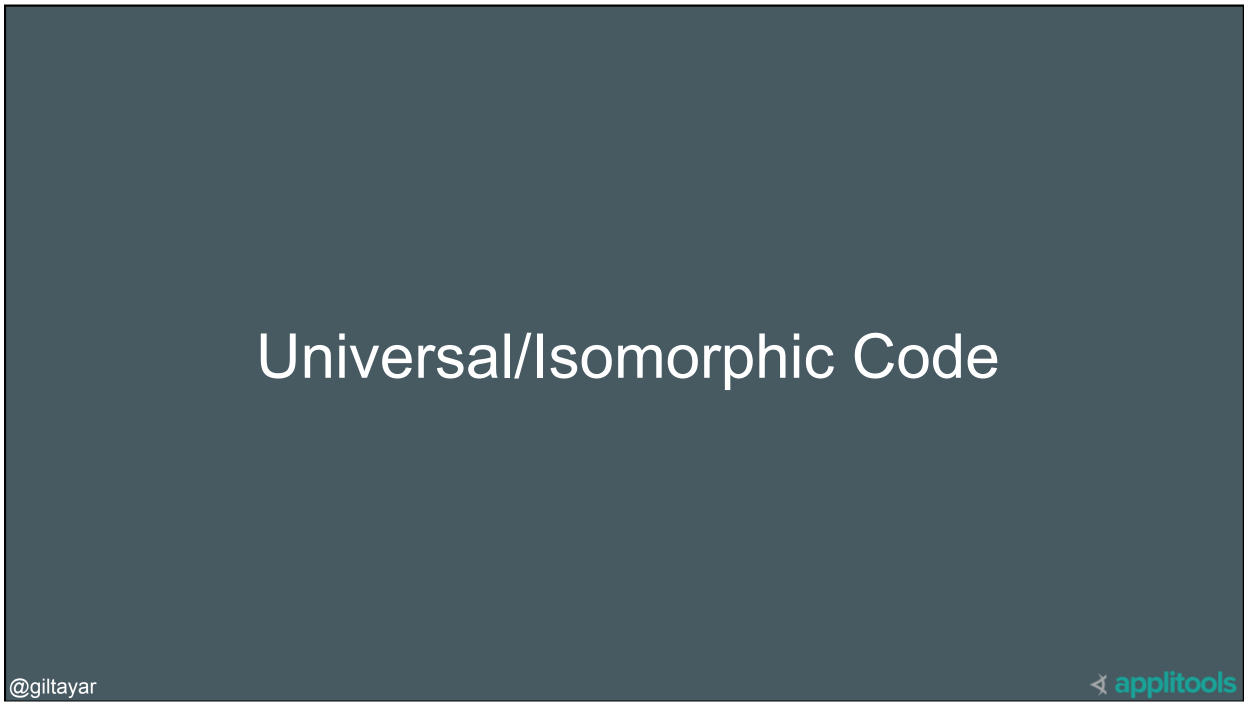
@giltayar



Most frontend code today can *also* run under NodeJS

@giltayar

applitools

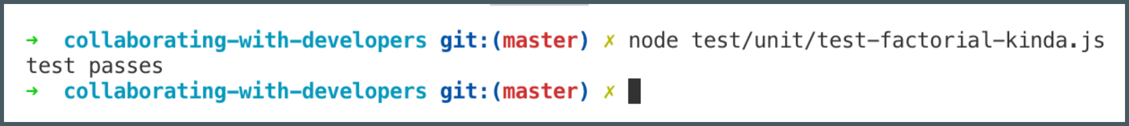


Universal/Isomorphic Code

@giltayar

applitools

Let’s run it under NodeJS



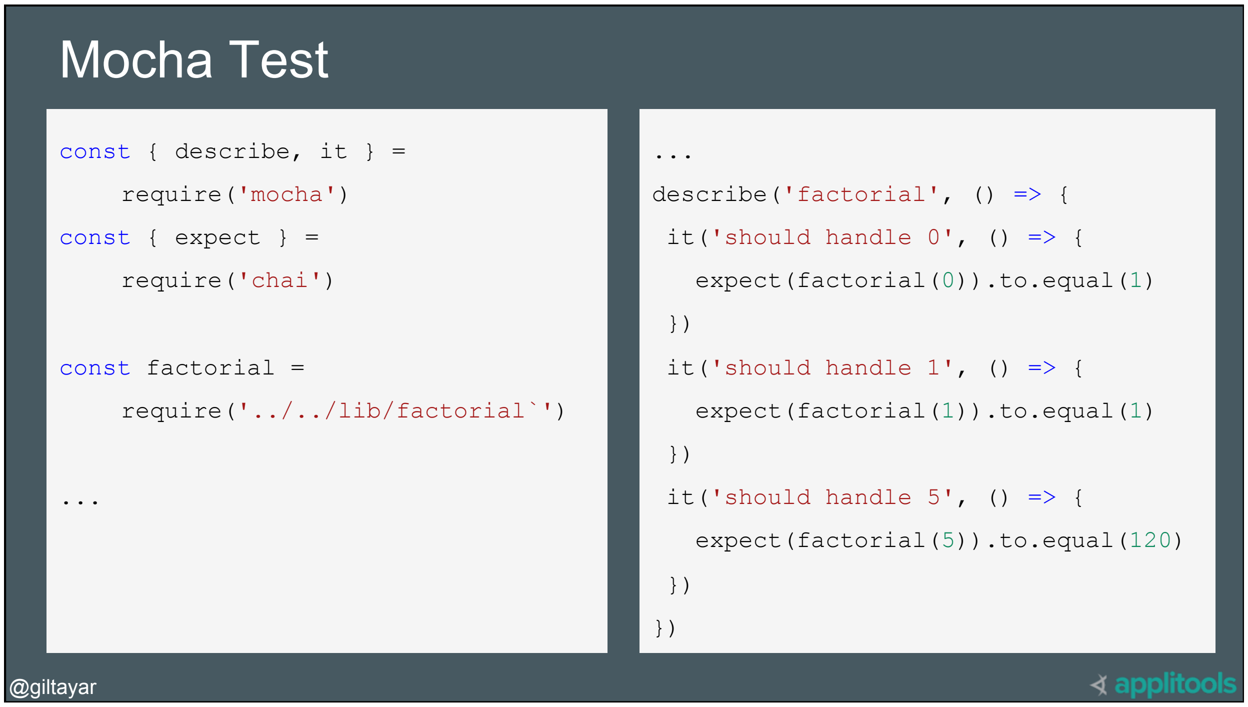
```
➜  collaborating-with-developers git:(master) ✗ node test/unit/test-factorial-kinda.js
test passes
➜  collaborating-with-developers git:(master) ✗ █
```

@giltayar

Awkward to Test Like This

- We need a **Test Runner**
- Just like jUnit, NUnit, pytest, test-unit, … in other languages
- NodeJS has lots of them:
  - Mocha, Jest, Ava, Tape, Jasmine.
  - And the list goes on…
- The most popular are Mocha and Jest
- We’ll be demoing using Mocha

@giltayar



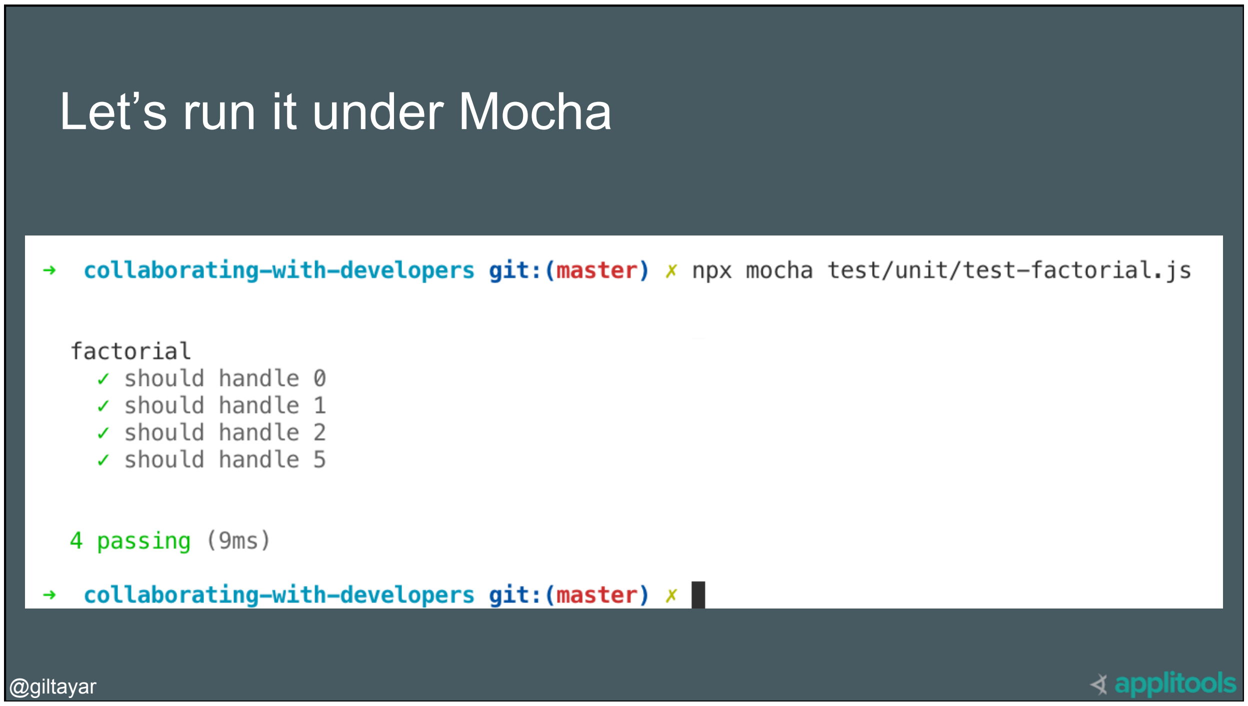
Mocha Test

```
const { describe, it } =
    require('mocha')
const { expect } =
    require('chai')

const factorial =
    require('../../lib/factorial`')

...
```

```
...
describe('factorial', () => {
 it('should handle 0', () => {
   expect(factorial(0)).to.equal(1)
 })
 it('should handle 1', () => {
   expect(factorial(1)).to.equal(1)
 })
 it('should handle 5', () => {
   expect(factorial(5)).to.equal(120)
 })
})
```

@giltayar

applitools

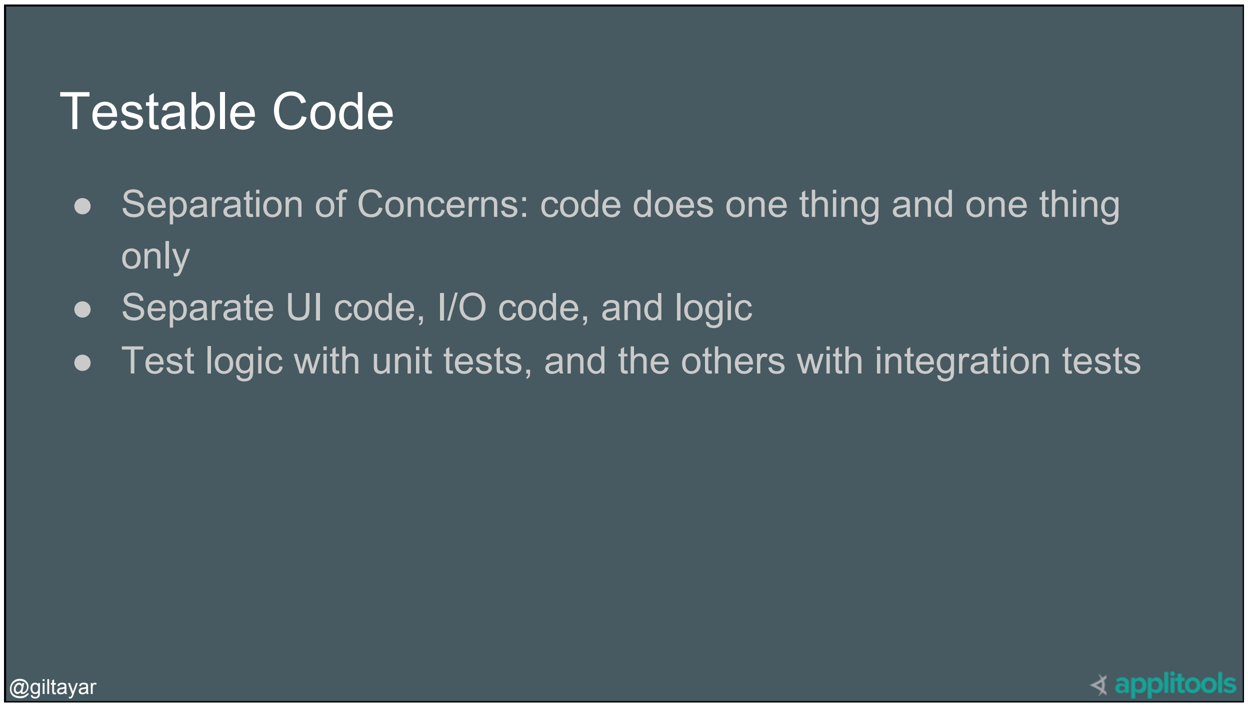


Let’s run it under Mocha

```
➜  collaborating-with-developers git:(master) ✗ npx mocha test/unit/test-factorial.js


  factorial
    ✓ should handle 0
    ✓ should handle 1
    ✓ should handle 2
    ✓ should handle 5


  4 passing (9ms)

➜  collaborating-with-developers git:(master) ✗ █
```

@giltayar

applitools

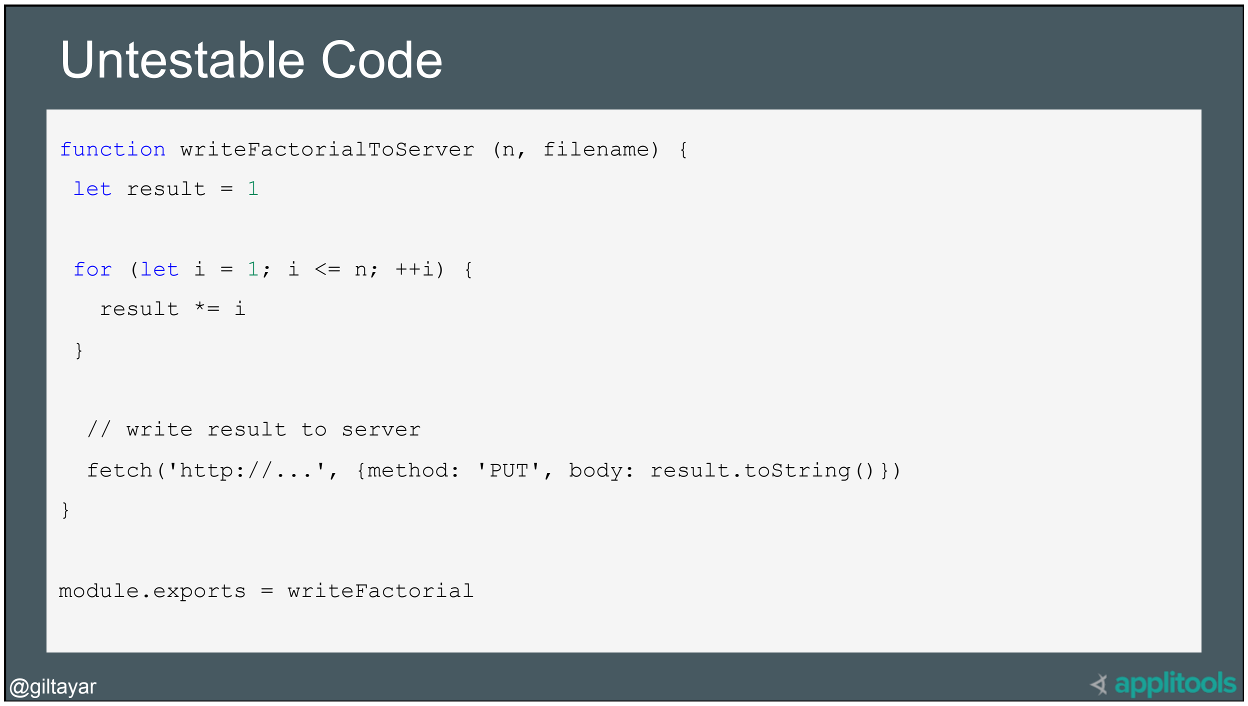


Testable Code

- Separation of Concerns: code does one thing and one thing only
- Separate UI code, I/O code, and logic
- Test logic with unit tests, and the others with integration tests

@giltayar

applitools



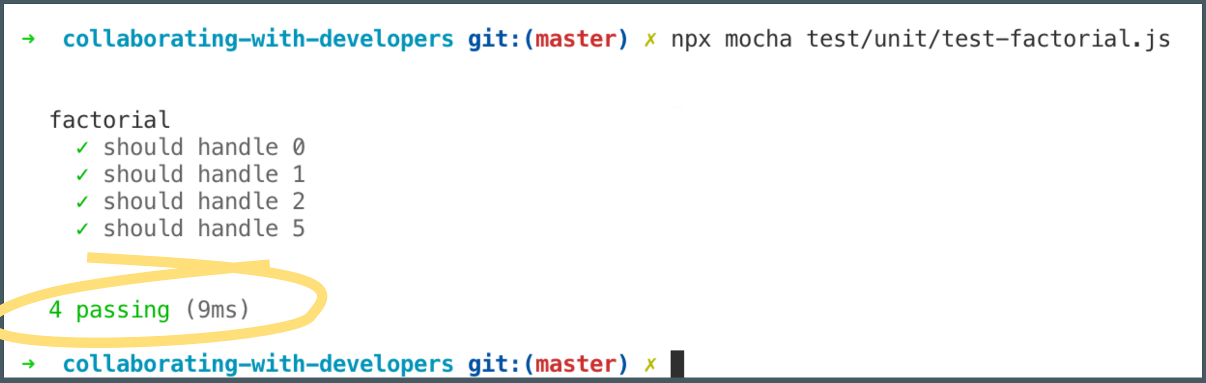
Untestable Code

```
function writeFactorialToServer (n, filename) {
 let result = 1

 for (let i = 1; i <= n; ++i) {
   result *= i
 }

  // write result to server
  fetch('http://...', {method: 'PUT', body: result.toString()})
}

module.exports = writeFactorial
```

@giltayar

applitools

Notice how important speed is...



```
➜  collaborating-with-developers git:(master) ✗ npx mocha test/unit/test-factorial.js


  factorial
    ✓ should handle 0
    ✓ should handle 1
    ✓ should handle 2
    ✓ should handle 5


  4 passing (9ms)

➜  collaborating-with-developers git:(master) ✗ █
```

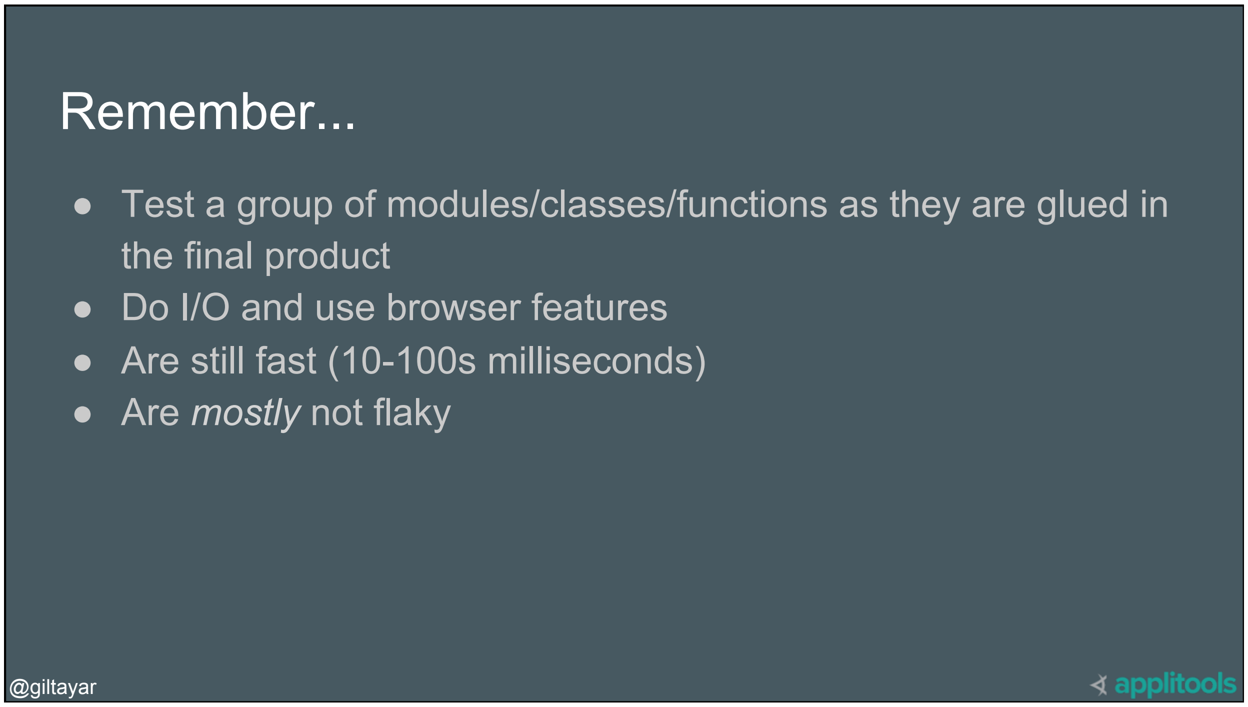
@giltayar

applitools

Writing Integration Tests

@giltayar

applitools

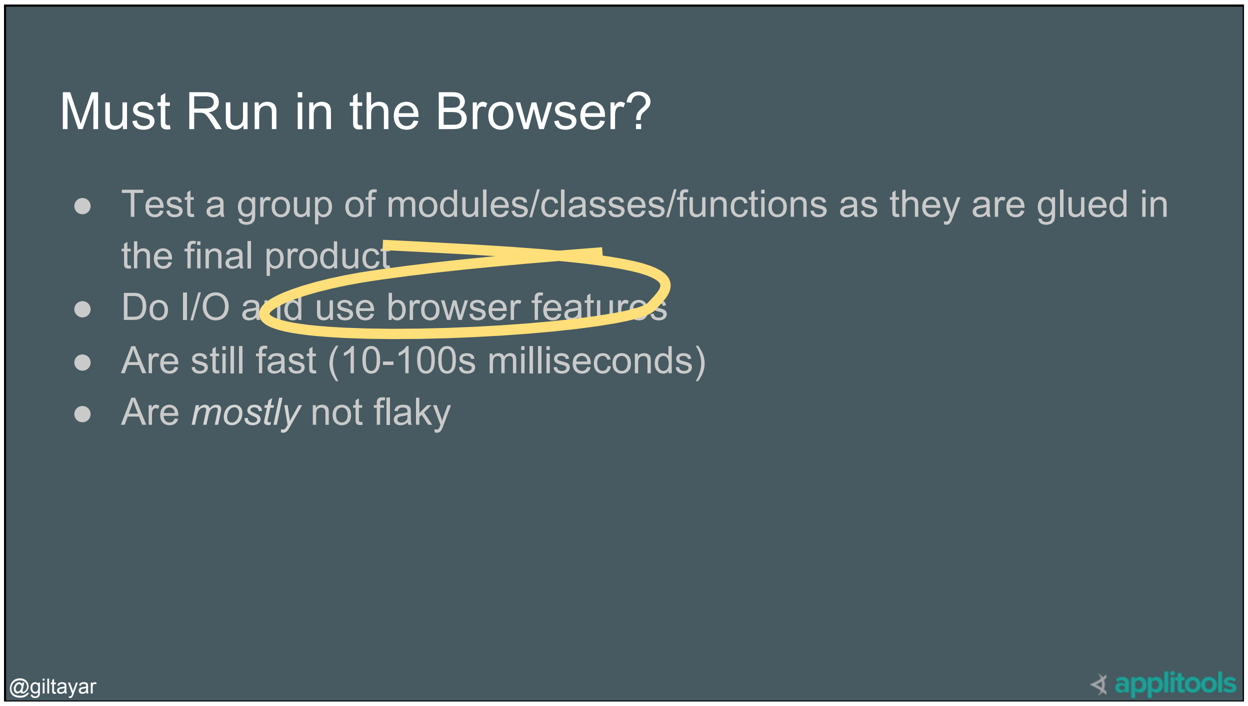


Remember...

- Test a group of modules/classes/functions as they are glued in the final product
- Do I/O and use browser features
- Are still fast (10-100s milliseconds)
- Are *mostly* not flaky

@giltayar

applitools

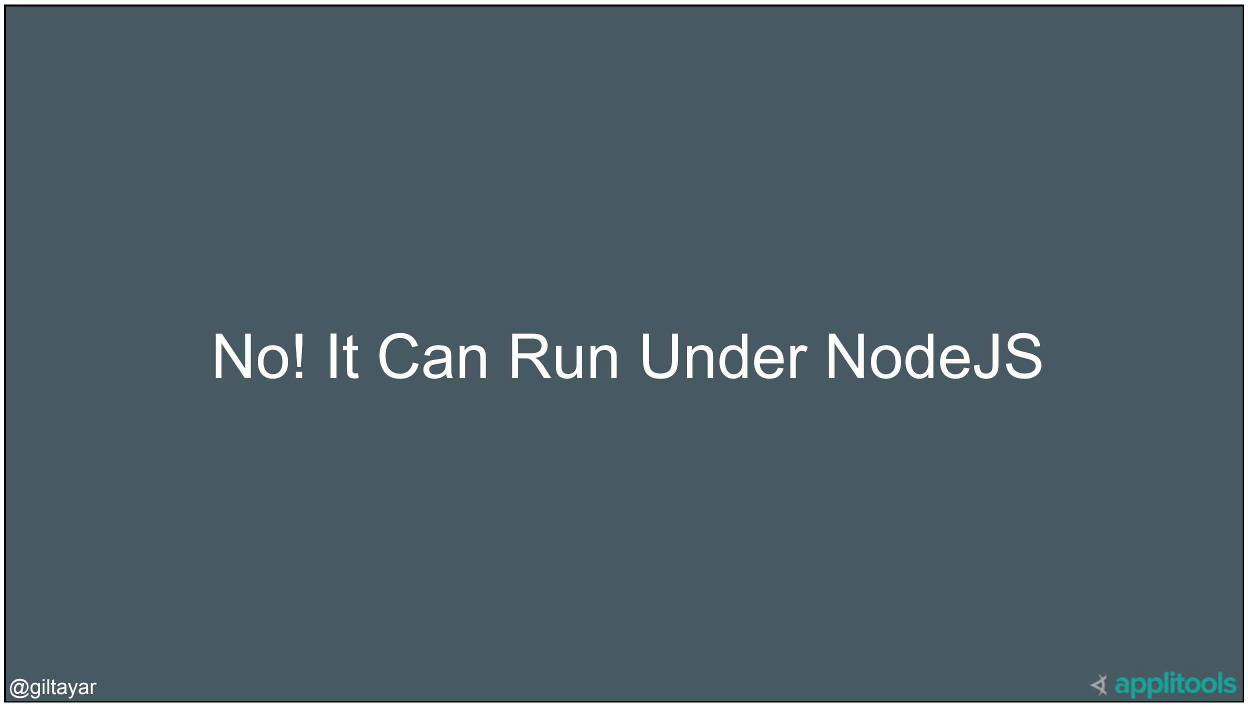


Must Run in the Browser?

- Test a group of modules/classes/functions as they are glued in the final product
- Do I/O and use browser features
- Are still fast (10-100s milliseconds)
- Are *mostly* not flaky

@giltayar

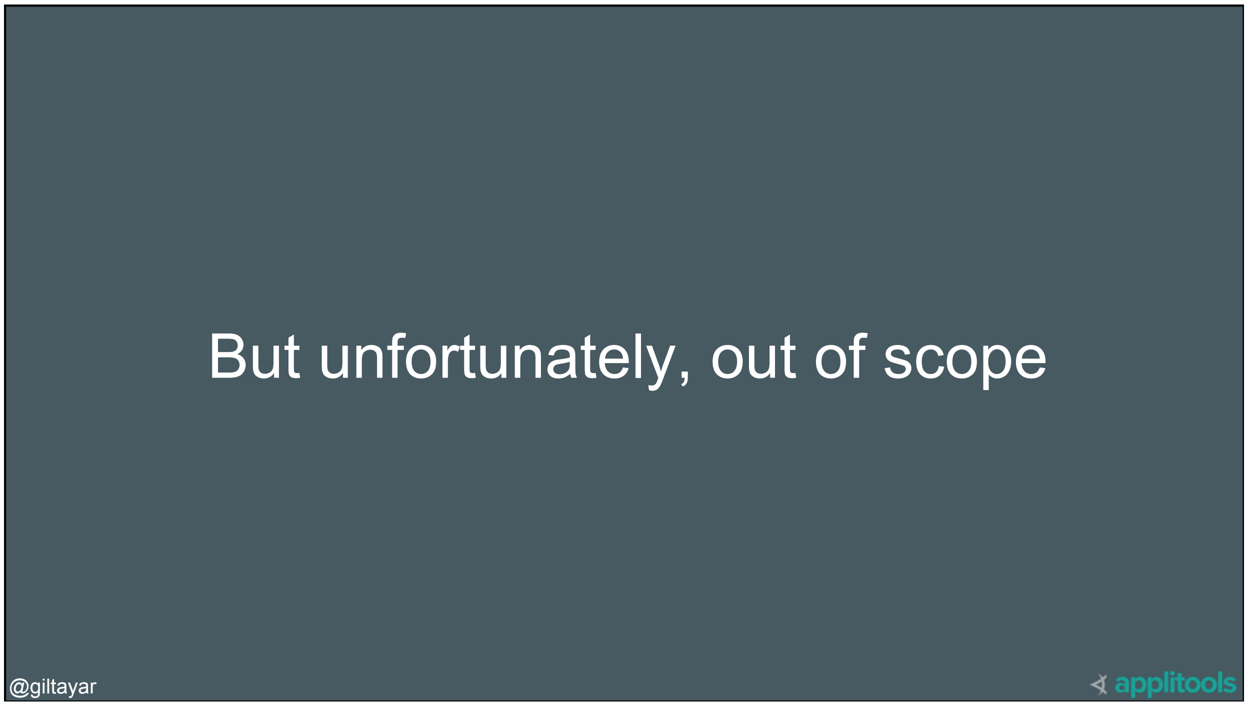
applitools



No! It Can Run Under NodeJS

@giltayar

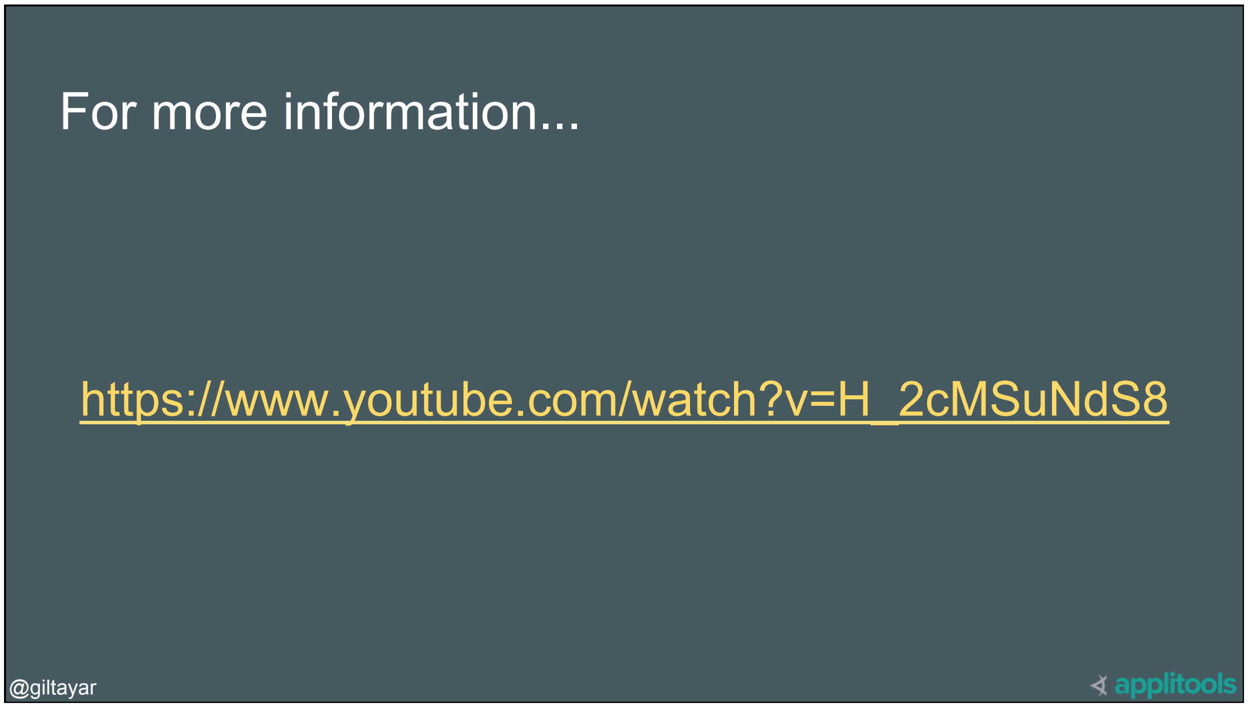
applitools



But unfortunately, out of scope

@giltayar

applitools

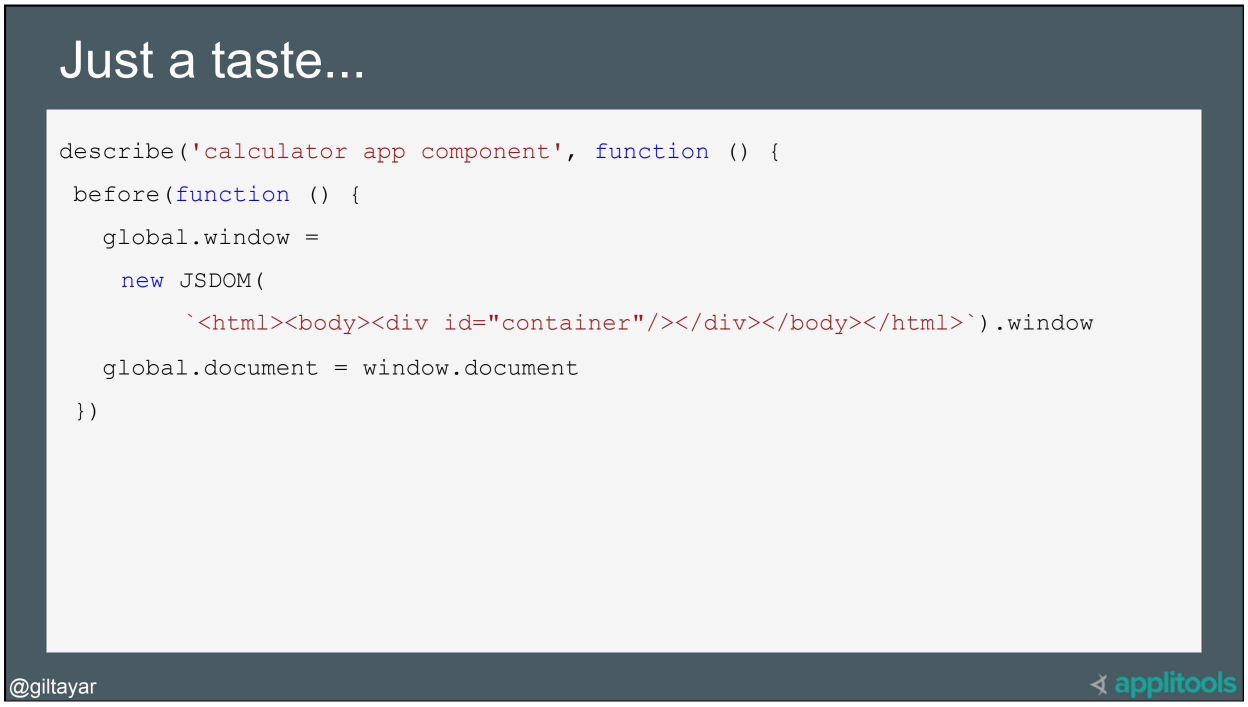


For more information...

https://www.youtube.com/watch?v=H_2cMSuNdS8

@giltayar

applitools

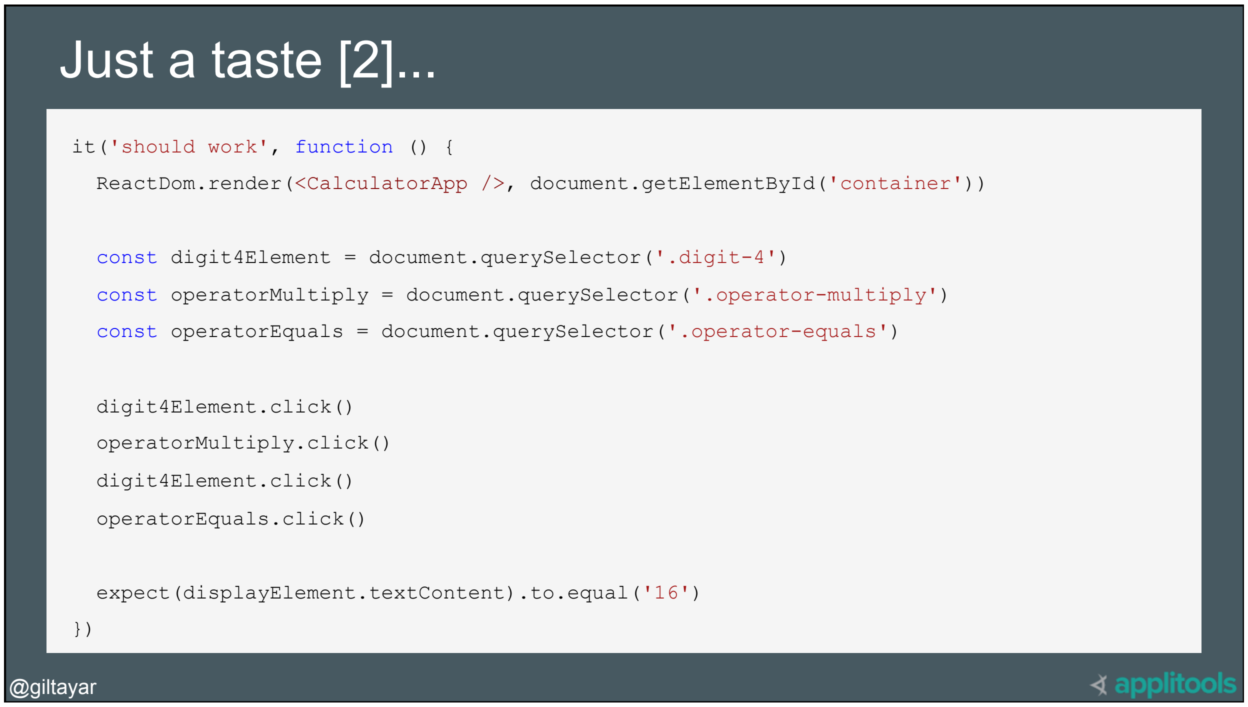


Just a taste...

```
describe('calculator app component', function () {
 before(function () {
   global.window =
     new JSDOM(
          `<html><body><div id="container"/></div></body></html>`).window
   global.document = window.document
 })
```

@giltayar

applitools

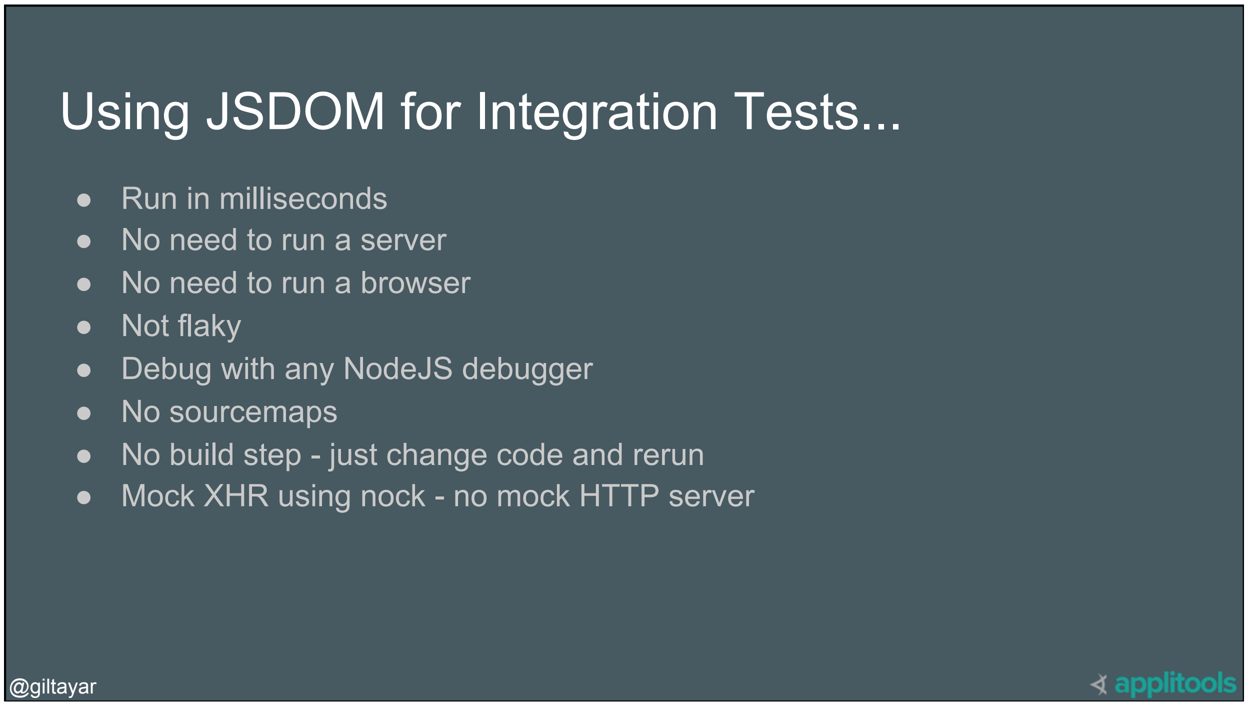


Just a taste [2]...

```
it('should work', function () {
  ReactDom.render(<CalculatorApp />, document.getElementById('container'))

  const digit4Element = document.querySelector('.digit-4')
  const operatorMultiply = document.querySelector('.operator-multiply')
  const operatorEquals = document.querySelector('.operator-equals')

  digit4Element.click()
  operatorMultiply.click()
  digit4Element.click()
  operatorEquals.click()

  expect(displayElement.textContent).to.equal('16')
})
```

@giltayar

applitools



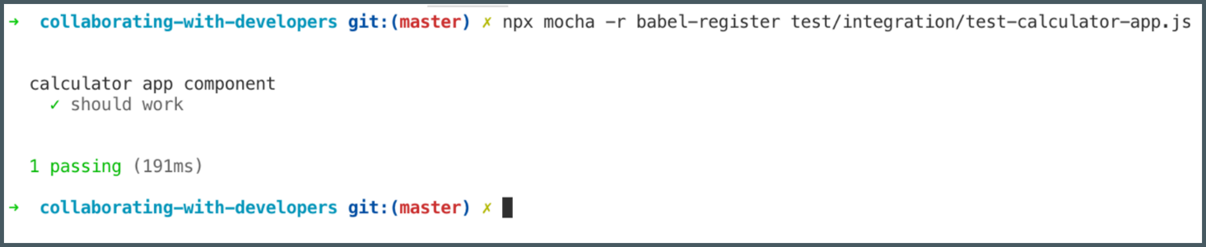
Using JSDOM for Integration Tests...

- Run in milliseconds
- No need to run a server
- No need to run a browser
- Not flaky
- Debug with any NodeJS debugger
- No sourcemaps
- No build step - just change code and rerun
- Mock XHR using nock - no mock HTTP server

@giltayar

applitools

Let's try it!



```
➜  collaborating-with-developers git:(master) ✗ npx mocha -r babel-register test/integration/test-calculator-app.js


  calculator app component
    ✓ should work


  1 passing (191ms)

➜  collaborating-with-developers git:(master) ✗ █
```

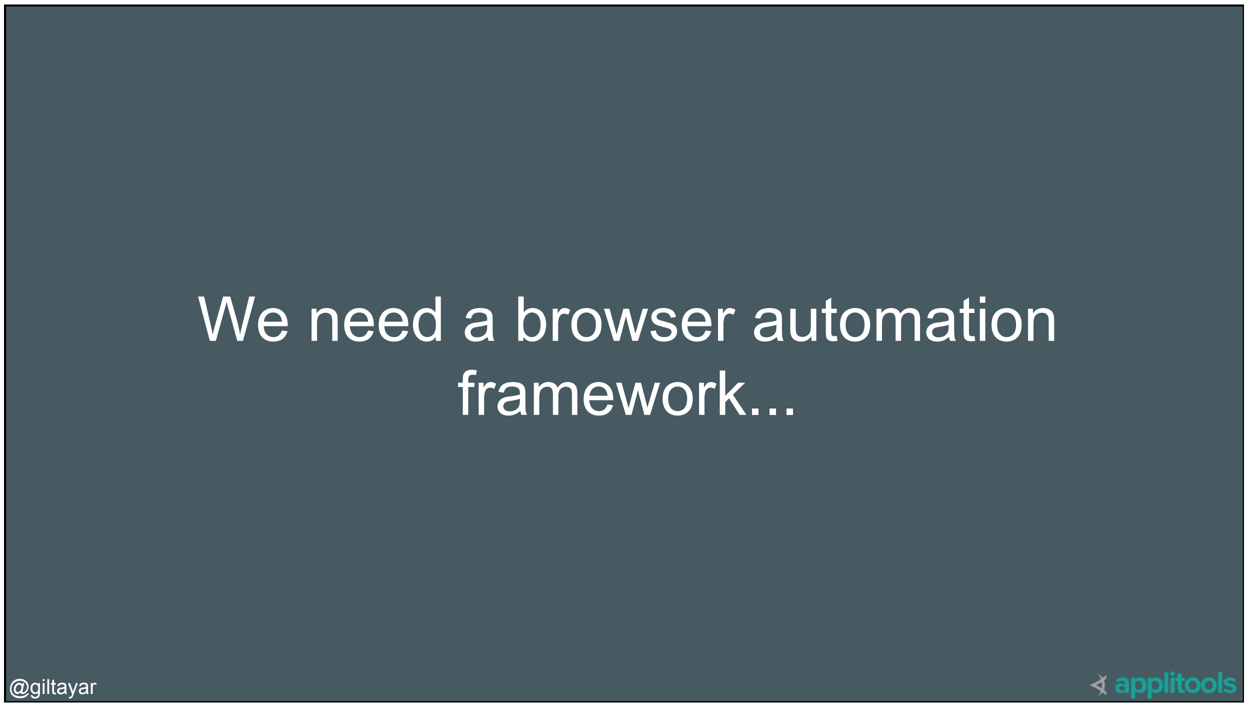
@giltayar

applitools

Writing E2E Tests (Browser Automation)

@giltayar

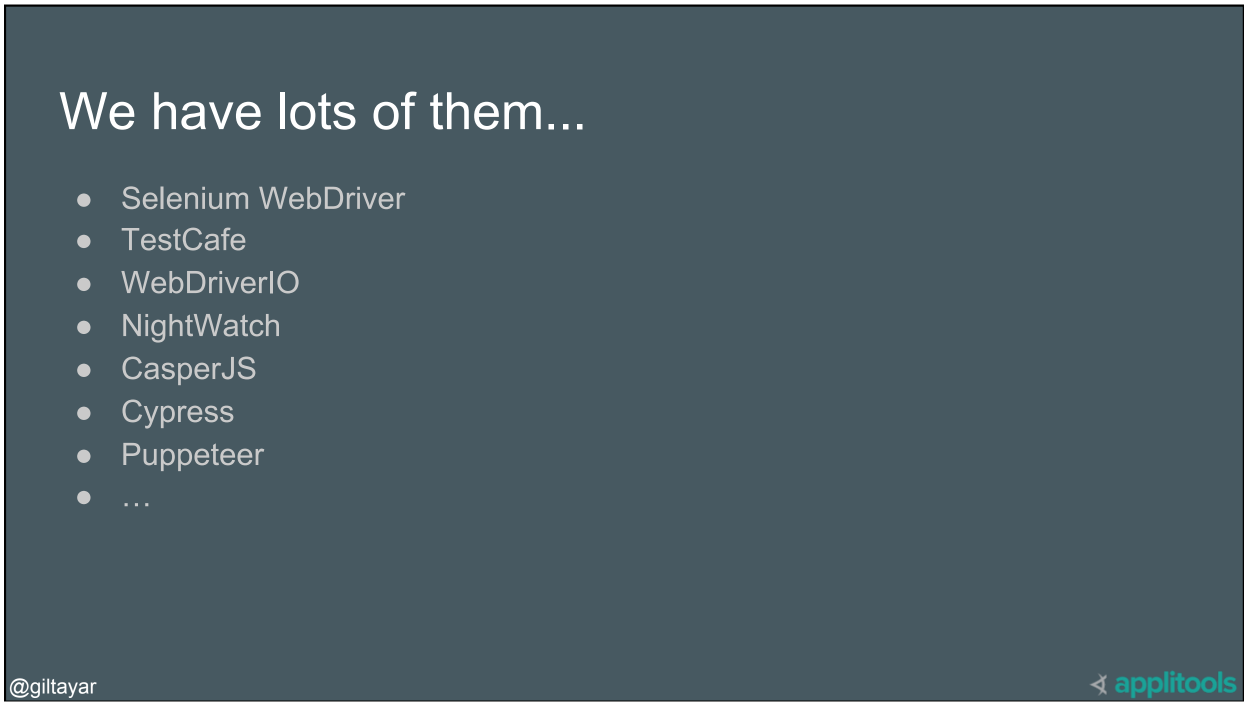
applitools



We need a browser automation framework...

@giltayar

applitools

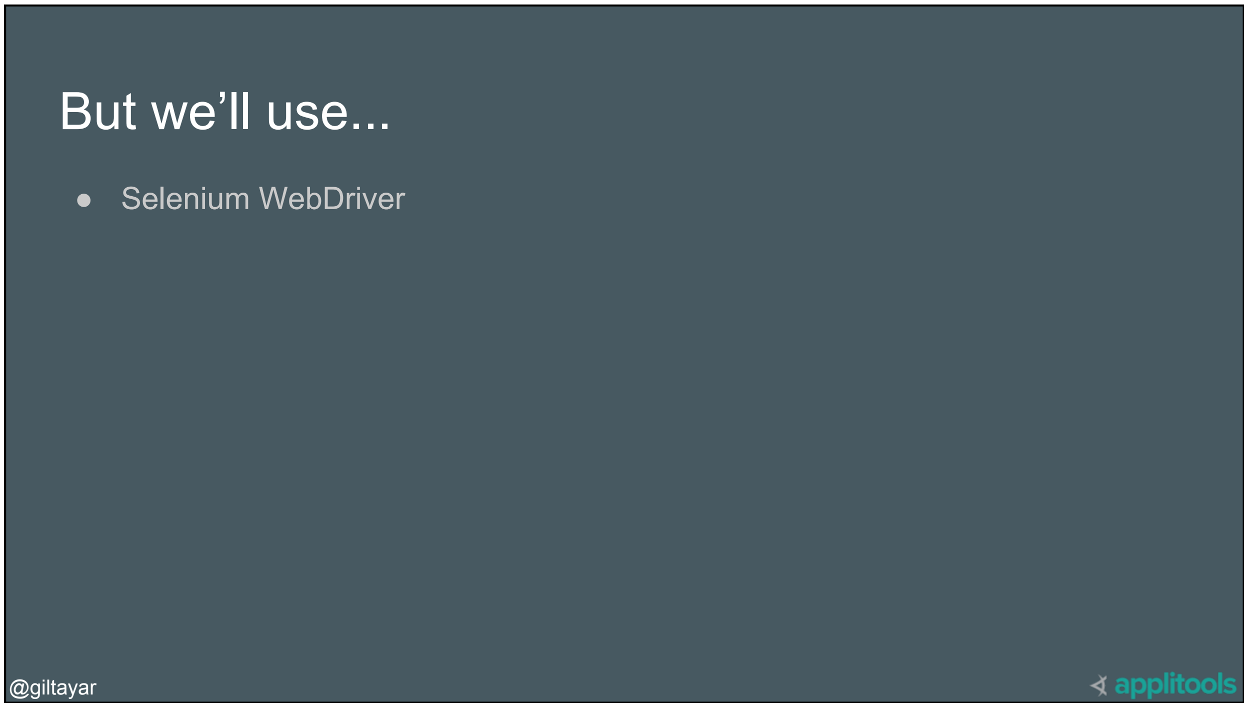


We have lots of them...

- Selenium WebDriver
- TestCafe
- WebDriverIO
- NightWatch
- CasperJS
- Cypress
- Puppeteer
- …

@giltayar

applitools

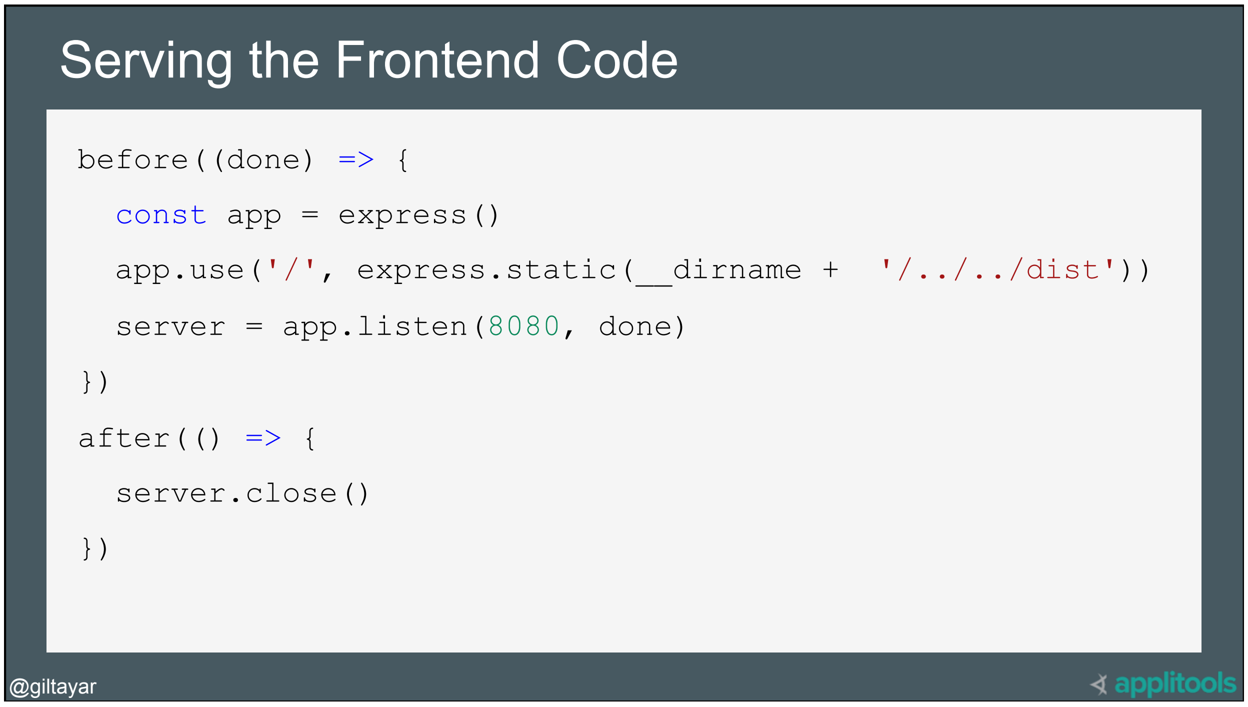


But we'll use...

- Selenium WebDriver

@giltayar

applitools



Serving the Frontend Code

```
before((done) => {
  const app = express()
  app.use('/', express.static(__dirname +  '/../../dist'))
  server = app.listen(8080, done)
})
after(() => {
  server.close()
})
```

@giltayar

applitools

Initializing WebDriver

```
before(async () => {
  driver = new webdriver.Builder()
    .forBrowser('chrome')
    .build()
})
after(async () => await driver.quit())
```

@giltayar

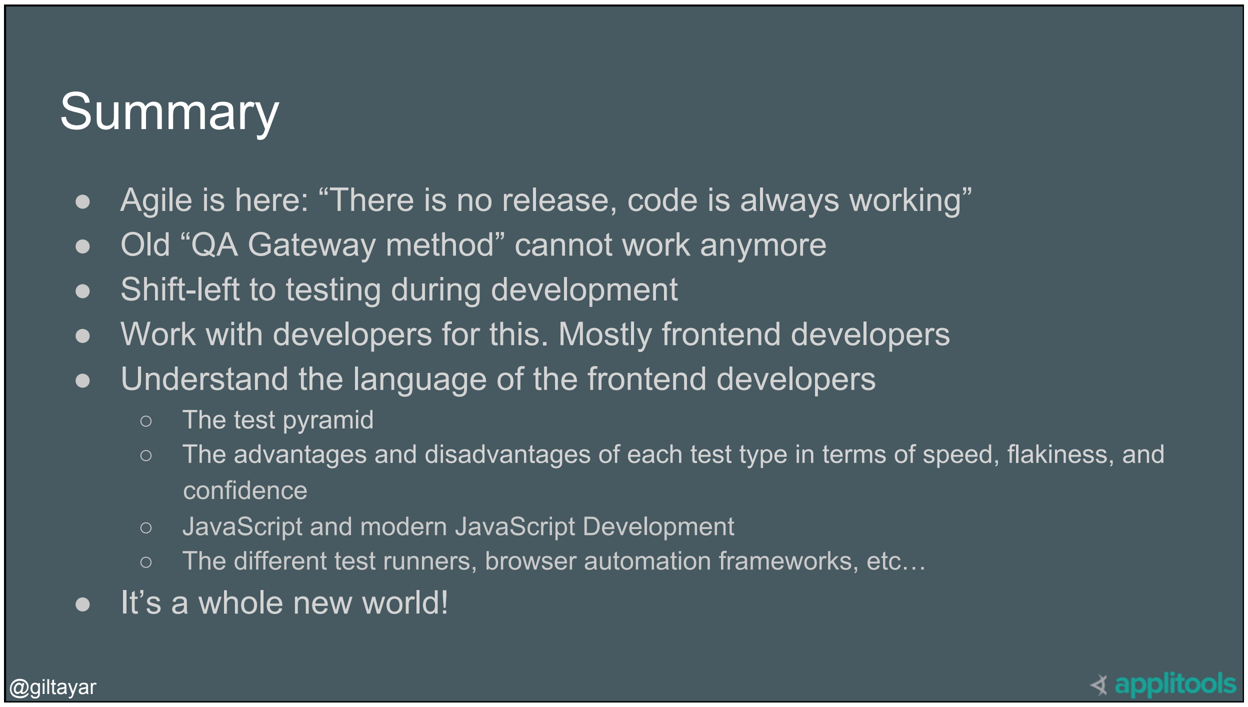
applitools

The Test

```
 it('should work', async function () {
   await driver.get('http://localhost:8080')

   const digit4Element = await driver.findElement(By.css('.digit-4'))
   const operatorMultiply = await driver.findElement(By.css('.operator-multiply'))
   const operatorEquals = await driver.findElement(By.css('.operator-equals'))

   await digit4Element.click()
   await operatorMultiply.click()
   await digit4Element.click()
   await operatorEquals.click()

   await driver.wait(until.elementTextIs(await driver.findElement(By.css('.display')),
'84'))
 })
```

@giltayar

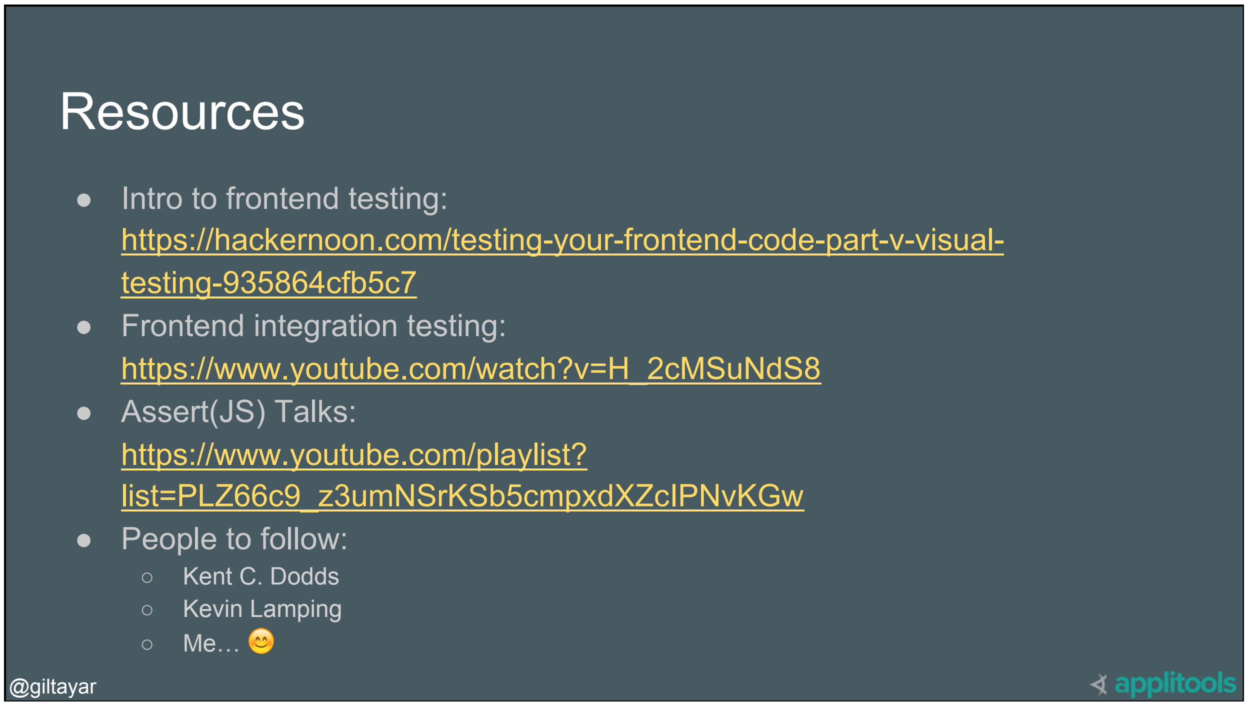
applitools



Summary

- Agile is here: “There is no release, code is always working”
- Old “QA Gateway method” cannot work anymore
- Shift-left to testing during development
- Work with developers for this. Mostly frontend developers
- Understand the language of the frontend developers
  - The test pyramid
  - The advantages and disadvantages of each test type in terms of speed, flakiness, and confidence
  - JavaScript and modern JavaScript Development
  - The different test runners, browser automation frameworks, etc…
- It’s a whole new world!

@giltayar applitools



Resources

- Intro to frontend testing: https://hackernoon.com/testing-your-frontend-code-part-v-visual-testing-935864cfb5c7
- Frontend integration testing: https://www.youtube.com/watch?v=H_2cMSuNdS8
- Assert(JS) Talks: https://www.youtube.com/playlist?list=PLZ66c9_z3umNSrKSb5cmpxdXZcIPNvKGw
- People to follow:
  - Kent C. Dodds
  - Kevin Lamping
  - Me… 😊

@giltayar applitools

Questions?

●●●

This presentation: http://bit.ly/collaborating-with-developers

Github repo: https://github.com/giltayar/collaborating-with-developers

@giltayar

applitools