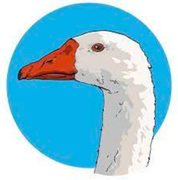
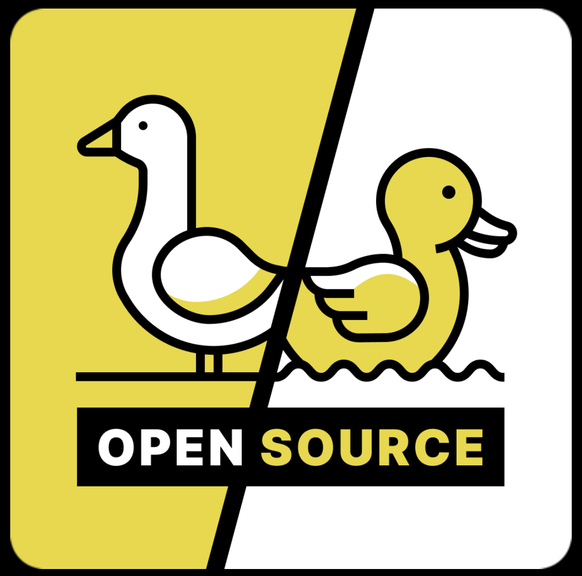
TDD не работает?

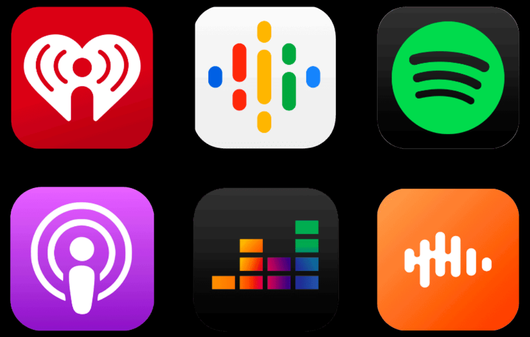
у меня для вас **плохие** новости…

От Гуся  для HolyJS

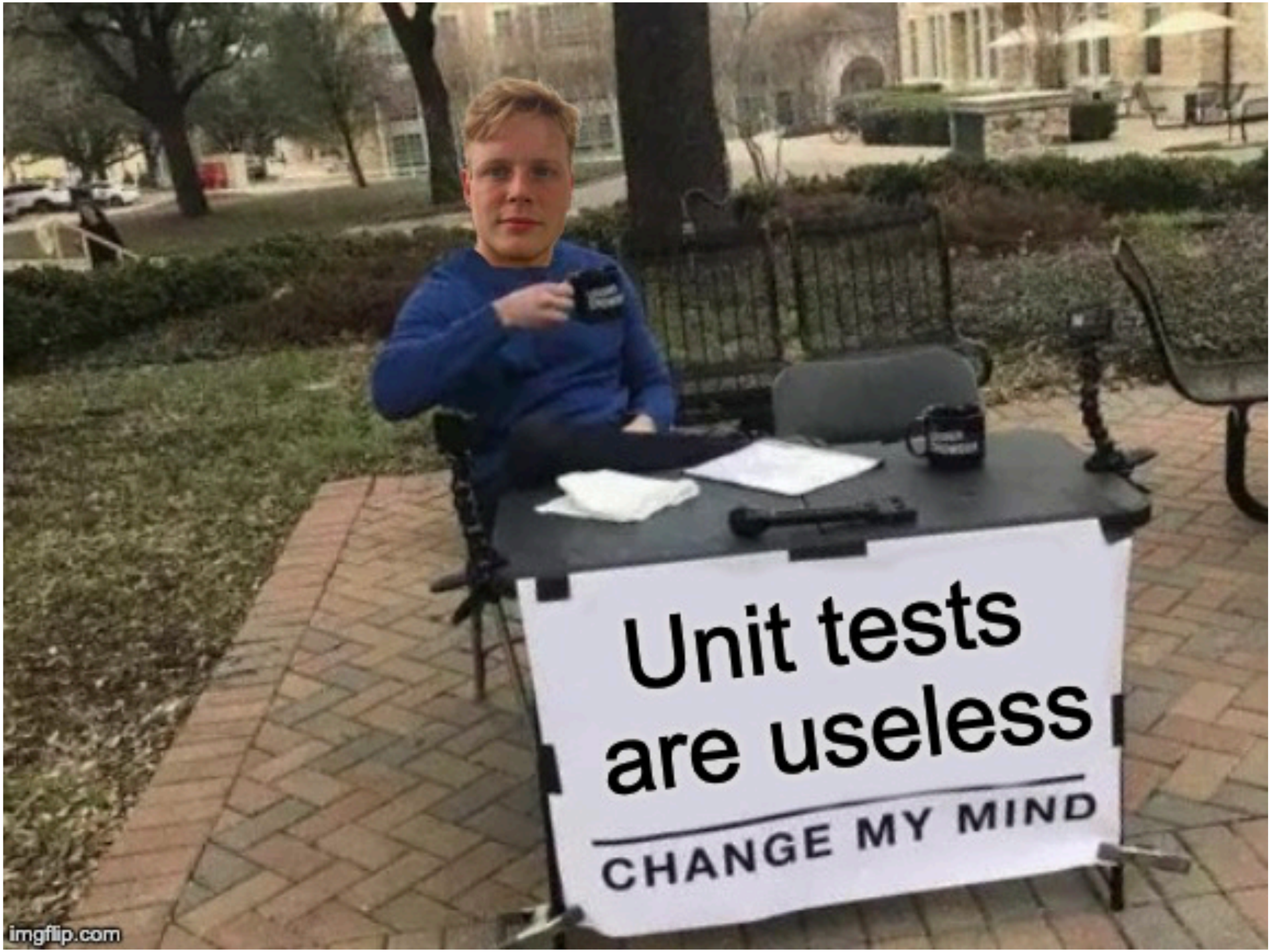
КТО ТАКОВ?

ВАШ ПОКОРНЫЙ СЛУГА



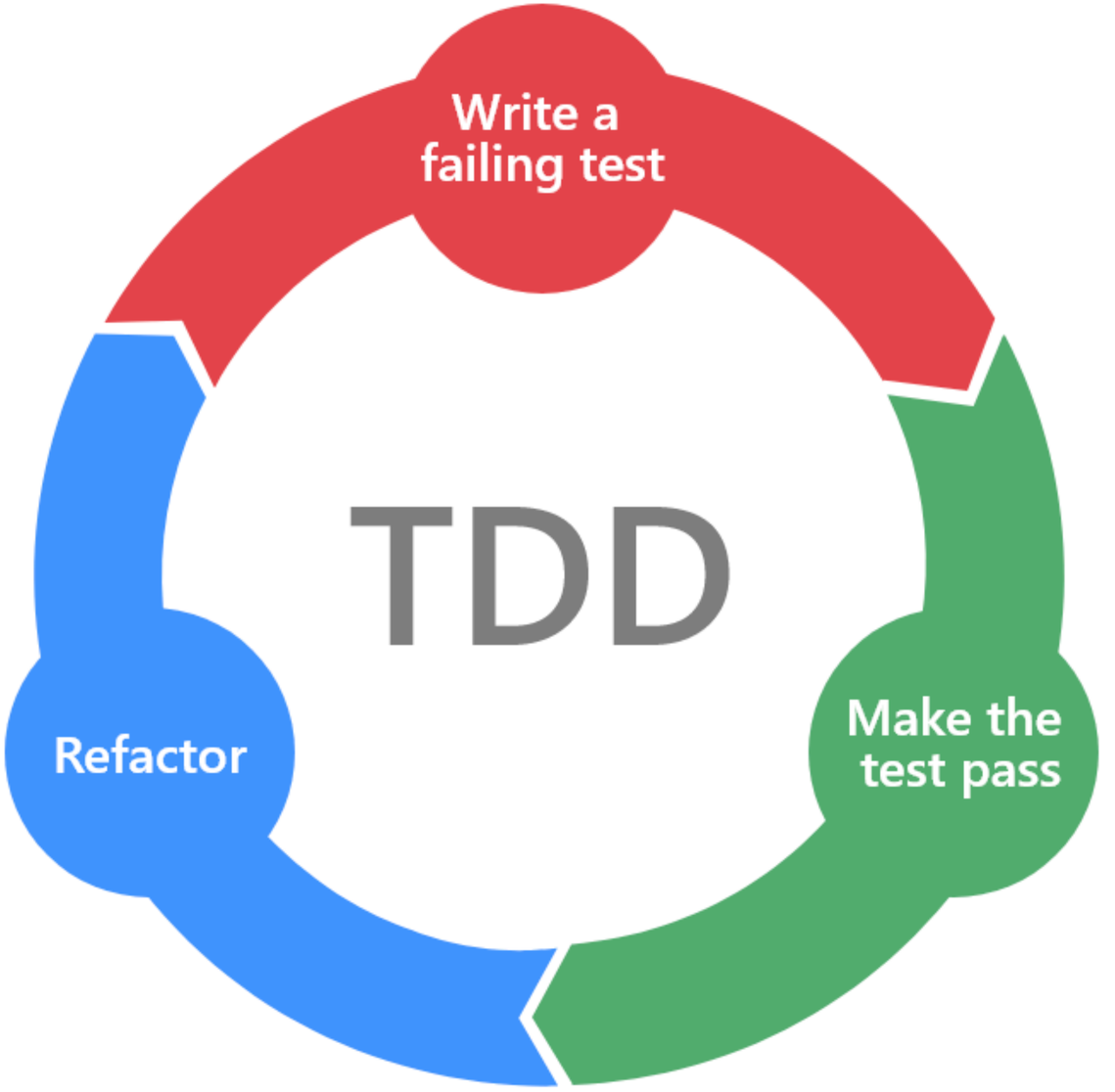


Holy war


Unit tests
are useless
CHANGE MY MIND
imgflip.com


Секретный секрет
успешного успеха

Про TDD
слышали
наверное все

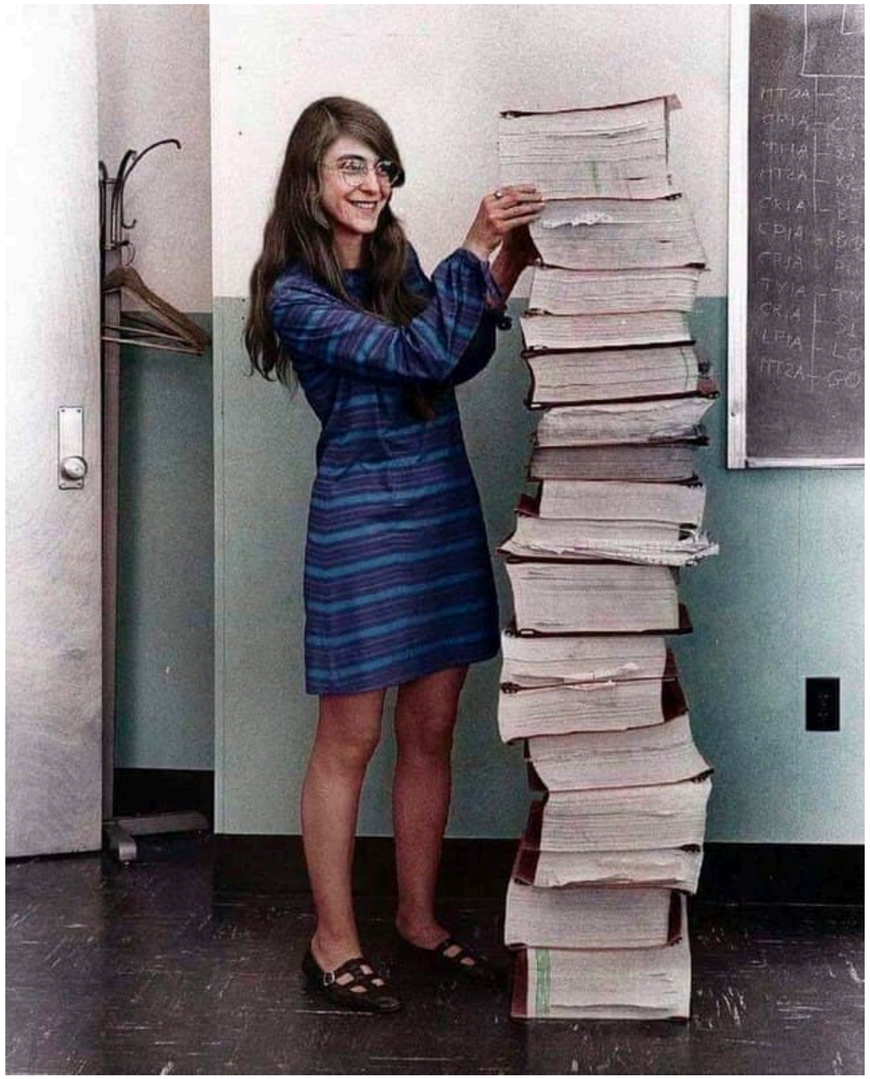

Write a
failing test
TDD
Make the
test pass
Refactor


IF ONLY I USED TDD

TDD
краткий экскурс

1964

Маргарет Гамильтон рядом с распечатками текста программы, которую она написала для миссии «Аполлон-11»

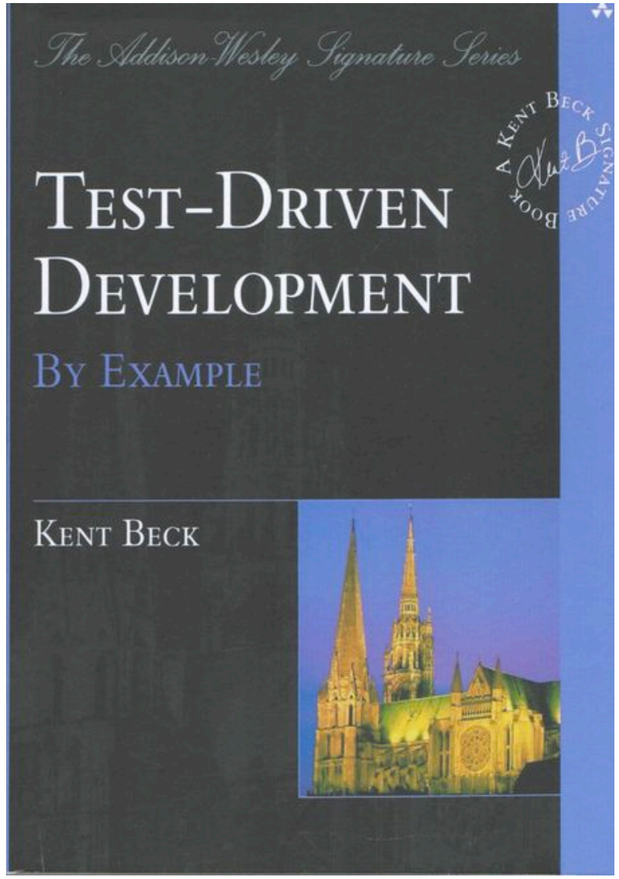


Я выкатываю на прод код, который писал последний год

1994

SUNIT

2002


The Addison-Wesley Signature Series
A Kent Beck Signature Book
Test-Driven Development
By Example
Kent Beck

TDD

Effects of Test-Driven Development: A Comparative Analysis of Empirical Studies

Simo Mäkinen and Jürgen Münch

University of Helsinki,
Department of Computer Science,
P.O. Box 68 (Gustaf Hällströmin katu 2b),
FI-00014 University of Helsinki, Finland
{simo.makinen,juergen.muench}@cs.helsinki.fi

Abstract. Test-driven development is a software development practice where small sections of test code are used to direct the development of program units. Writing test code prior to the production code promises several positive effects on the development process itself and on associated products and processes as well. However, there are few comparative studies on the effects of test-driven development. Thus, it is difficult to assess the potential process and product effects when applying test-driven development. In order to get an overview of the observed effects of test-driven development, an in-depth review of existing empirical studies was carried out. The results for ten different internal and external quality attributes indicate that test-driven development can reduce the amount of introduced defects and lead to more maintainable code. Parts of the implemented code may also be somewhat smaller in size and complexity. While maintenance of test-driven code can take less time, initial development may last longer. Besides the comparative analysis, this article sketches related work and gives an outlook on future research.

Key words: test-driven development; test-first programming; software testing; software verification; software engineering; empirical study

1 Introduction

Red. Green. Refactor. The mantra of test-driven development [1] is contained in these words: *red* refers to the fact that first and foremost implementation of any feature should start with a failing test, *green* signifies the need to make that test pass as fast as possible and *refactor* is the keyword to symbolize that the code should be cleaned up and perfected to keep the internal structure of the code intact. But the question is, what lies behind these three words and what do we know about the effects of following such guidelines? Test-driven development reshapes the design and implementation of software [1] but does the change propagate to the associated software products and in which way are the processes altered with the introduction of this alternative way of development? The objective here was to explore these questions and to get an overview of the observed effects of test-driven development.

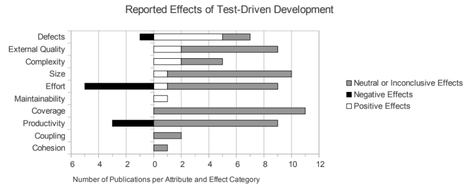


Fig. 1. The occurrence of positive, neutral and negative effects for each quality attribute as reported by the test-driven development publications included in the review

article need a careful analysis of the respective contexts before applying in other environments. The completeness of the integrative literature review was based on the ranking algorithm of the search engines and might have been enforced more strictly. Other threats to validity concern the use of qualitative inclusion and exclusion criteria as well as the selection of databases, search terms, and the chosen timeframe. Due to these factors, there could be a selection bias related to the selection of the publications. This needs to be taken into care when interpreting and using the results of this integrative literature review.

5 Conclusion

This integrative literature review analyzed the effects of test-driven development from existing empirical studies. The detailed review collected empirical findings for different quality attributes and found out varying effects to these attributes. Based on the results, prominent effects include the reduction of defects and the increased maintainability of code. The internal quality of code in terms of coupling and cohesion seem not to be affected so much but code complexity might be reduced a little with test-driven development. With all the tests written, the whole code base becomes larger but more source code lines are being covered by tests. Test code is faster to write than the code implementing the test but many of the studies report increased effort in development.

The quality map constructed as part of the review shows some possible directions for future research. One of the promising effects was the increased maintainability and reduced effort it took to maintain code later but at the time of the review there was only a single study from Dogša and Batič [14] which had specifically focused on maintainability. This could be one of the areas for further research on test-driven development.

https://www.researchgate.net/publication/256848134_Effects_of_Test-Driven_Development_A_Comparative_Analysis_of_Empirical_Studies

Quality of Testing in Test Driven Development

Adnan Čaušević, Sasikumar Punnekkat and Daniel Sundmark
Mälardalen University, Sweden
firstname.lastname@mdh.se

***Abstract*—Test-driven development is an essential part of eXtreme Programming approach with the preference of being followed in other Agile methods as well. For several years, researchers are performing empirical investigations to evaluate quality improvements in the resulting code when test-driven development is being used. However, very little had been reported into investigating the quality of the testing performed in conjunction with test-driven development.**

In this paper we present results from an experiment specifically designed to evaluate the quality of test cases created by developers who used the test-first and the traditional test-last approaches. On an average, the quality of testing in test-driven development was almost the same as the quality of testing using test-last approach. However, detailed analysis of test cases, created by test-driven development group, revealed that 29% of test cases were "negative" test cases (based on non-specified requirements) but contributing as much as 65% to the overall tests quality score of test-first developers.

We are currently investigating the possibility of extending test-driven development to facilitate non-specified requirements to a higher extent and thus minimise the impact of a potentially inherent effect of positive test bias.

***Index Terms*—software testing; test case quality; test driven development; experiment;**

I. Introduction

Quality of agile methods is often a focus of empirical studies by researchers due to the inability to formally prove the benefits arising from the usage of such methods. Several factors may contribute to the overall software product quality when using agile methods, such as, usage of short cycles, close customer relationship, pair programming, test-driven development, continuous integration, and many more. When performing empirical investigations, researchers usually try to isolate one factor in particular and evaluate its effects on the code quality which is often the main metric of evaluation. However, when isolating test-driven development factor, researchers tend to omit another important metric, quality of test cases. We believe that measuring the quality and characteristics of test cases generated during test-driven development is an important step towards making it more industrially acceptable.

Test-driven development (TDD) was introduced as a practice within eXtreme Programming (XP) methododology [1]. Developers using TDD write automated unit tests before they write the actual code, and hence it is also referred as a test-first approach in literature [2]. Tests are written in the form of assertions and in TDD their purpose is to define code requirements. By using TDD, developers build the systems in cycles of test, development and refactoring.

Test-driven development was identified, in our industrial survey [3], as a most preferred but lesser used practice in industry. Interpretation of a main finding of this study could be: "Respondents would like to use TDD to a significantly higher extent than they actually do currently". This preference towards using TDD could be based on academic research results often pointing improvements of the code quality when TDD is used ([4]–[8]), but also due to the success of early adopters. As a follow up, we performed a systematic literature review [9] for the purpose of identifying any obstacles in the path of full scale adoption of TDD in the industry. Seven factors, which are potentially limiting full adoption of TDD, were identified and listed. Inability of developers to write automated test cases (in an efficient and effective way) is considered to be one of these limiting factors. In the current paper we are presenting analysis results of an experiment formulated and defined in a way to investigate the significance of such a limiting factor.

An experiment was conducted during the autumn semester in 2011 with master students enrolled in the Software Verification and Validation course at the Mälardalen University, with the intention of comparing testing efficiency and effectiveness of agile (test-first) and traditional (test-last) developers. This experiment allowed us to investigate the quality of testing in test-driven development by using the created test cases as a main metric of evaluation.

The remaining of this paper is organised as follows. Section II presents the related research work followed by the experimental design and its execution in section III. The analysis of quality attributes are presented in section IV followed by a detailed investigation on test cases in section V. In section VI, we discuss threats to validity of our study followed by conclusions and future research plans in section VII.

II. Related Work

During the identification of potential limiting factors of TDD adoption, our systematic literature review [9] listed 48 empirical studies that had effects of TDD as the focus of the investigation. Most of the studies had TDD as a primary focus of investigation, but in some cases effects of TDD were investigated in conjunction with some other practice, e.g. pair-programming. Goal of the studies investigating effects of TDD was related in most cases with respect to: (i) the internal or the external code quality improvements, (ii) performance improvements or (iii) a general perception of using TDD. However, we identified only one study [10] where the focus of the investigation was quality attributes of *test cases* when test-first approach was used.

2002 !== 2022

Программисты раньше

Я написал полноценную 3D игру с разрешением 640x480 на чистом ассемблере

Программисты сейчас

О боже, помогите мне. Я не могу выйти из VIM


Менеджер по логистике
Маркетинг менеджер
Менеджер по кадрам
Начальник охраны
IT менеджер
Менеджер по связям
Менеджер проекта
Начальник отдела
PR менеджер
Менеджер по разработкам
Вася

```
import { Repository } from '@hoorns/dbs'
import { MyService } from '@hoorns/services'

mock(Repository, () => ({
  users: [{ id: 1, name: 'test', email: 'test@testcom'}]
}));

test("My API works", async () => {
  const user = await MyService.getUser();

  expect(user.id).toBe(1);
  expect(user.name).toBe('test');
})
```

```
import { Repository } from "@hoorns/db/UserRepository";
import { MyService } from "@hoorns/services/MyService";

mock(Repository, () => ({
  users: [{ id: 1, name: "test", email: "test@testcom" }],
}));

Add only
test("My API works", async () => {
  const user = await MyService.getUser();

  expect(user.id).toBe(1);
  expect(user.name).toBe("test");
});
```

Неудобно!

~~EASY~~ GRAPHQL

```
const EasyGraphQLTester = require('easygraphql-tester')

const schema = `
  type FamilyInfo {
    id: ID!
    isLocal: Boolean!
  }

  type Query {
    getFamilyInfoByIsLocal(isLocal: Boolean!): FamilyInfo
  }
`

const query = `
  query TEST($isLocal: Boolean!) {
    getFamilyInfoByIsLocal(isLocal: $isLocal) {
      id
      isLocal
    }
  }
`

function getFamilyInfoByIsLocal(__, args, ctx) {
  return {
    id: 1,
    isLocal: args.isLocal
  }
}

const resolvers = {
  Query: {
    getFamilyInfoByIsLocal
  }
}

const tester = new EasyGraphQLTester(schema, resolvers)

tester.graphql(query, undefined, undefined, { isLocal: false })
  .then(result => console.log(result))
  .catch(err => console.log(err))
```

Unit test vs. Integration test

ВАСИЛИЙ

airbnb-app

- gateway
  - prod 2
  - dev 1
- database
  - prod 2
  - dev 4

+ NEW WORKSPACE

Q topExperiences | M book

PRETTIFY | HISTORY | https://airbnb.now.sh | COPY CURL | SHARE PLAYGROUND

```
1 {
2   topExperiences {
3     id
4     category {
5       name
6     }
7     title
8   }
9 }
```

```
{
  "data": {
    "topExperiences": [
      {
        "id": "cja8e0vvg1b060156al3tucfd",
        "category": null,
        "title": "Raise a glass to Prohibition"
      },
      {
        "id": "cja8e32k81n5201515yyc35p9",
        "category": null,
        "title": "Raise a glass to Prohibition"
      }
    ]
  }
}
```

SCHEMA

TRACING

Request	210 ms
topExperiences	90 ms
0.id	21 µs
0.category	4 µs
0.title	6 µs
1.id	4 µs
1.category	2 µs
1.title	3 µs
Response	12 ms

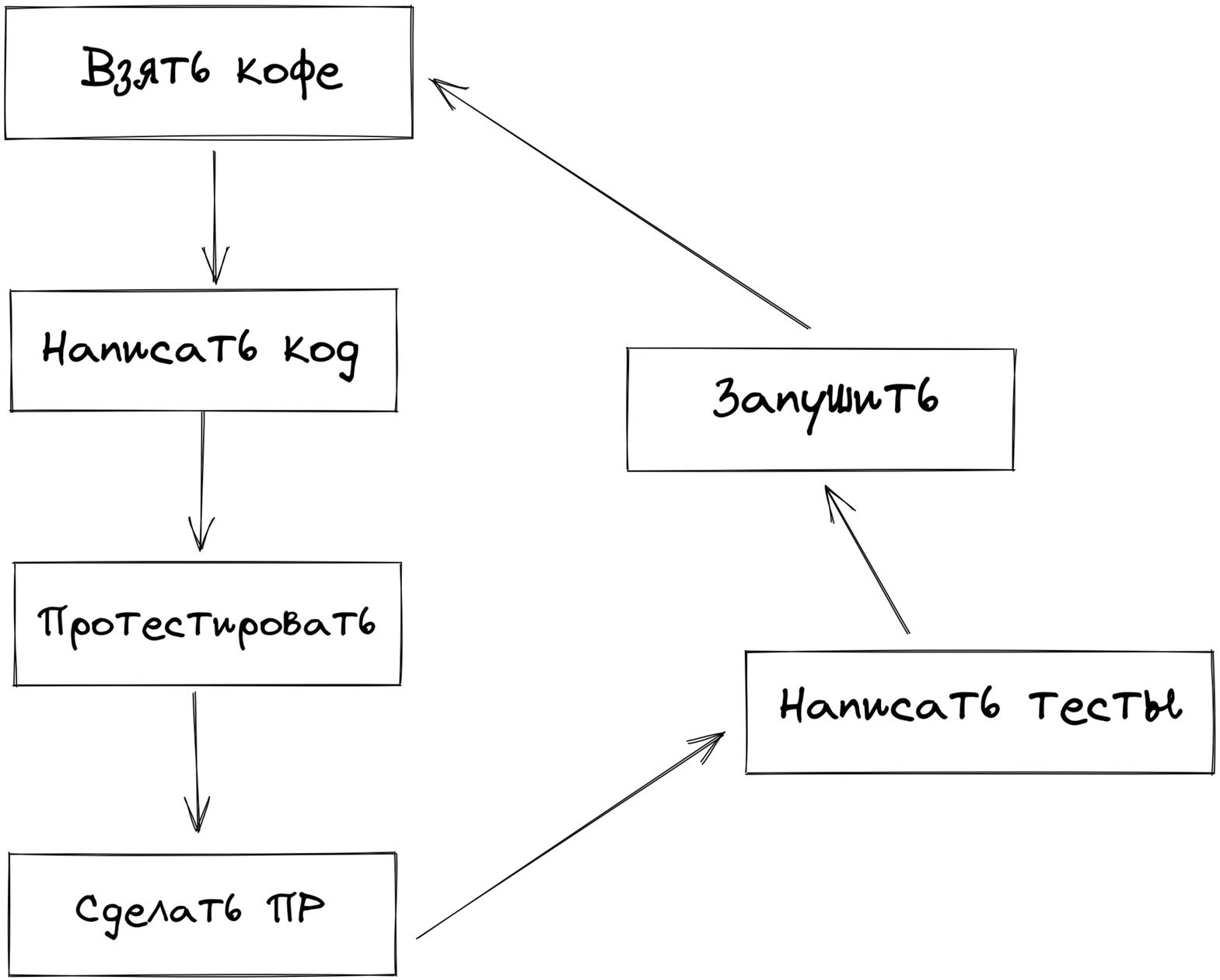
QUERY VARIABLES HTTP HEADERS (1)

```
import { db } from '@hoorns/db';

test("My API works", async () => {
  await db.createUser({ id: 1, name: 'test' });

  const { currentUser } = await runRequest(graphql`
    query Test {
      user(id: $id) {
        id
        name
      }
    }
  `, { id: 1 });

  expect(currentUser.id).toBe(1);
  expect(currentUser.name).toBe('test');
})
```


Взять кофе
Написать код
Протестировать
Сделать ПР
Написать тесты
Запушить

Используй свои тесты по назначению

Надежность

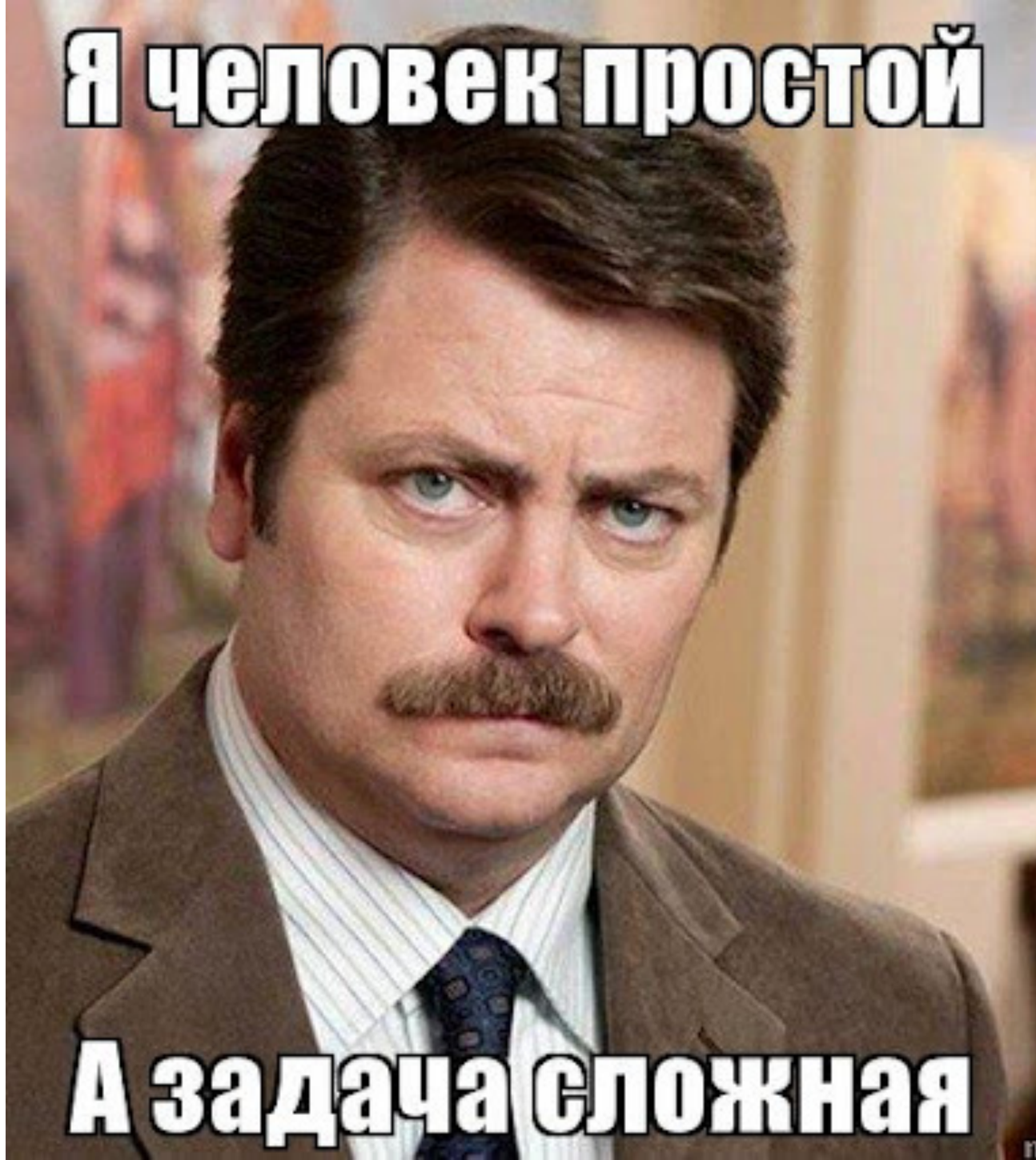
Антихрупкость

"Антихрупкость"

Нассим Талеб

“Антихрупкость – совсем не то, что эластичность, гибкость или неуязвимость. Гибкое либо эластичное противостоит встряске и остается прежним; антихрупкое, пройдя сквозь испытания, становится лучше прежнего.”

Задача Тестов



Неожиданные

Чёрные гуси

Влекут последствия

В ретроспективе очевидные

Технологии

GraphQL

cypress

material-ui-pickers
Dmitriy Kovalenko

material-ui-pickers
View all projects

- Latest runs
- Analytics
  - Run status
  - Run duration
  - Test suite size
  - Top failures
  - Slowest tests
  - Most common errors
  - Flaky tests
- Project settings

cypress

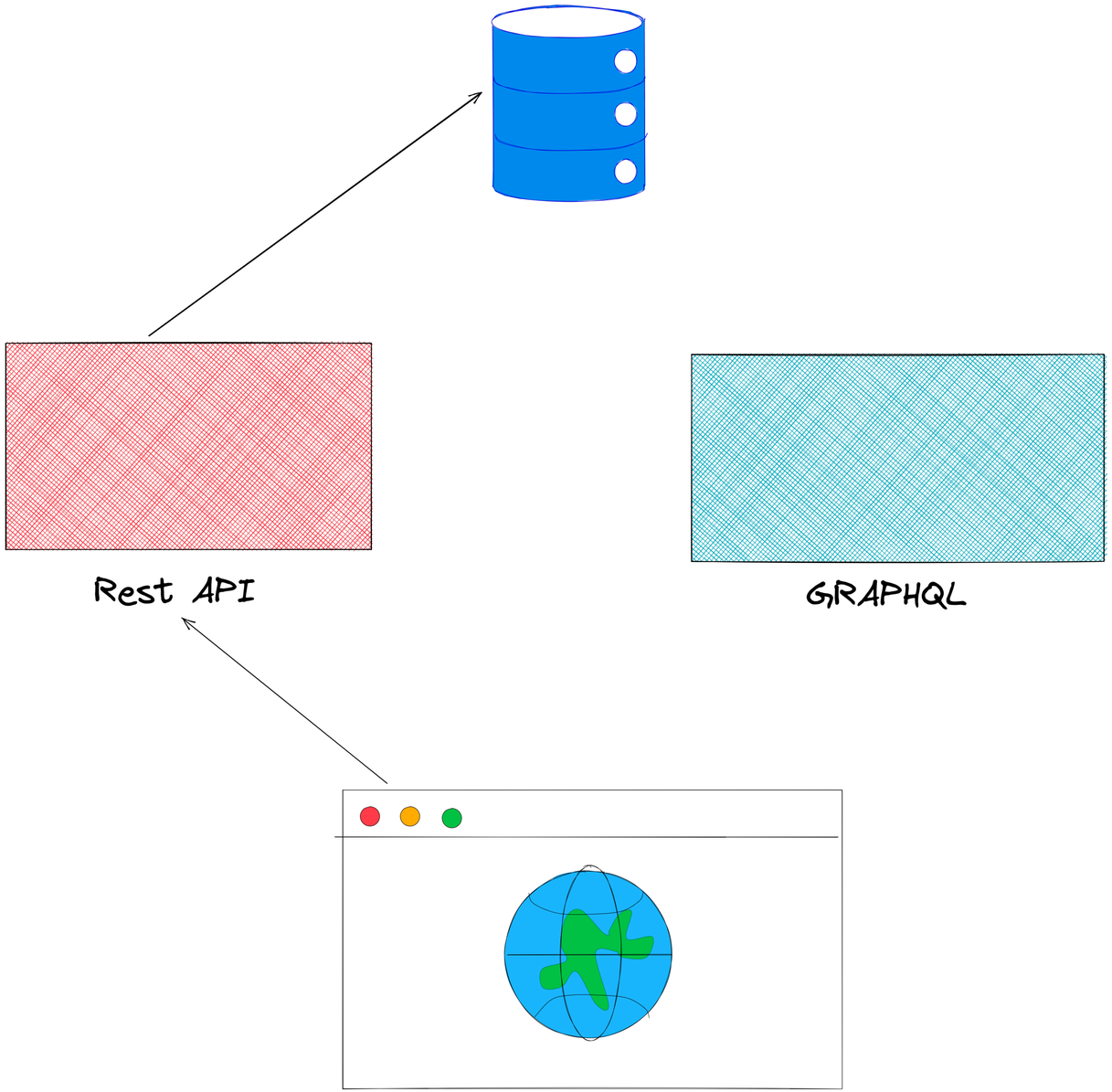
Support | Documentation

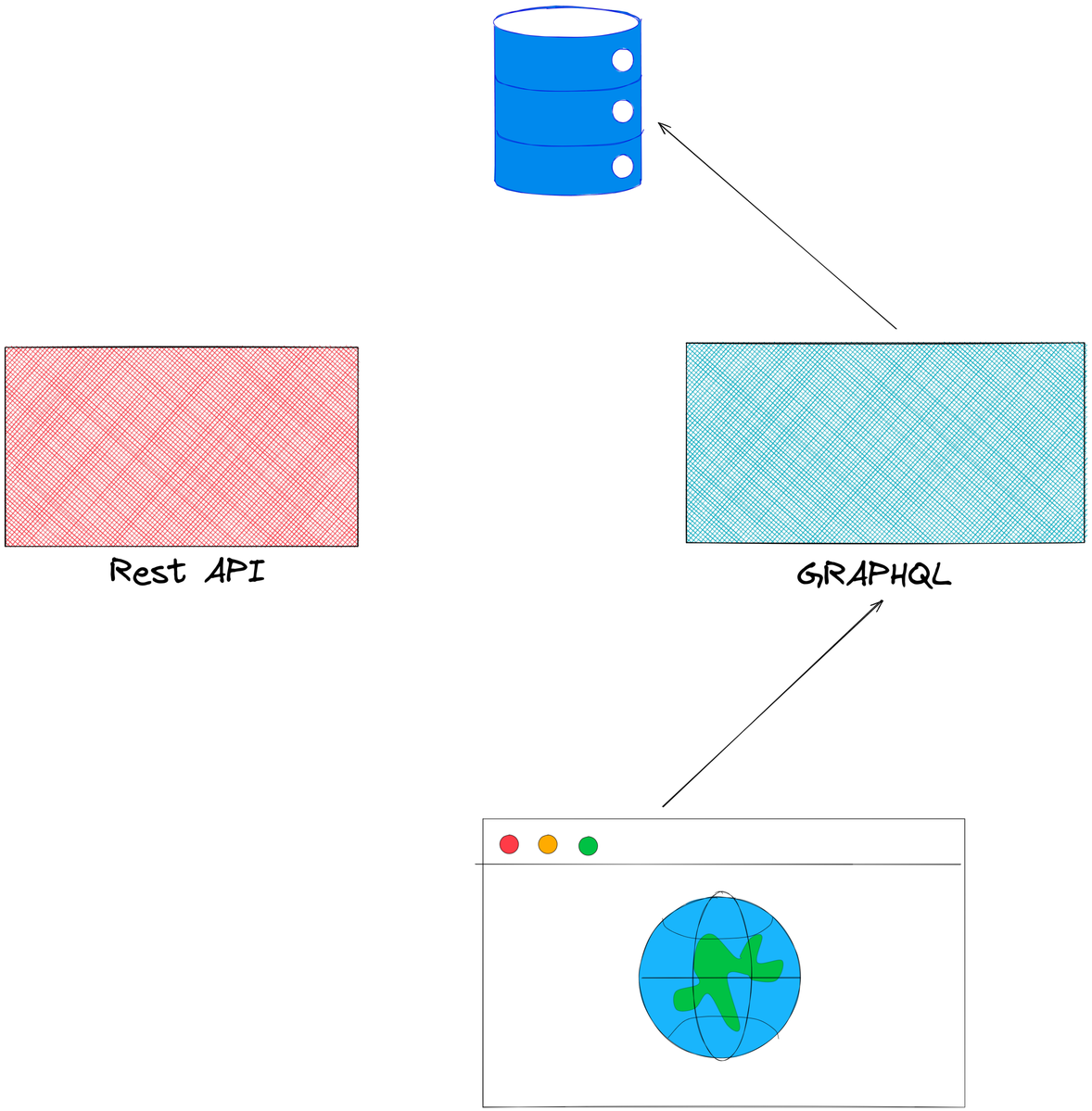
Latest runs

FILTER BY

All Time | Status | Branch | Committer | Tag | Flaky Tests

- Add renovate.json
  Renovate Bot • Ran 4 months ago • 01:44 • renovate/configure • CircleCI 422135 • # 1564
- [docs] Fix date-fns documentation link (#2181)
  Jonathan Chen • Ran 7 months ago • 01:40 • next • CircleCI 0277f3 • # 1563
- Update parsing.mdx
  Jonathan Chen • Ran 7 months ago • 02:11 • pull/2181 • CircleCI 14a100 • # 1562
- update lib/.size-snapshot.json
  Olivier Tassinari • Ran 8 months ago • 01:04 • v3-x • CircleCI 20a2b1 • # 1561
- Allow React 17 in the peer dependencies (#2174)
  Jolse Maginnis • Ran 8 months ago • 01:07 • v3-x • CircleCI bb7716 • # 1560
- update to match Material-UI v4
  Olivier Tassinari • Ran 8 months ago • 01:03 • pull/2174 • CircleCI 3563dc • # 1559
- Bump prismjs from 1.21.0 to 1.23.0
  dependabot[bot] • Ran 8 months ago • 01:41 • dependabot/npm_and_yarn/prismjs-1.23.0 • CircleCI 689d07 • # 1558
- Support for react 17
  Jolse Maginnis • Ran 8 months ago • 01:01 • pull/2174 • CircleCI 744fb1 • # 1557
- Bump ini from 1.3.5 to 1.3.7
  dependabot[bot] • Ran 11 months ago • 01:51 • dependabot/npm_and_yarn/ini-1.3.7 • CircleCI 37076d • # 1556
- Fix typo
  Dmitriy Kovalenko • Ran 11 months ago • 01:39 • next • CircleCI fca267 • # 1555
- Update usage.mdx
  getsetbro • Ran a year ago • 01:42 • pull/2158 • CircleCI c103b5 • # 1554
- Update README.md
  Dmitriy Kovalenko • Ran a year ago • 01:37 • next • CircleCI 06dfac • # 1553
- Fix allowSameDateSelection prop in DateRangePicker
  Karimov Damir • Ran a year ago • 01:42 • pull/2145 • CircleCI a677eb • # 1552
- Fix allowSameDateSelection prop in DateRangePicker
  Karimov Damir • Ran a year ago • 01:50 • pull/2145 • CircleCI 3bb15c • # 1551
- Fix allowSameDateSelection prop in DateRangePicker
  Karimov Damir • Ran a year ago • 01:38 • pull/2145 • CircleCI 573a10 • # 1550
- Fix allowSameDateSelection prop in DateRangePicker
  Karimov Damir • Ran a year ago • 01:35 • pull/2145 • CircleCI 15dbd8 • # 1549
- Fix allowSameDateSelection prop in DateRangePicker


Rest API
GRAPHQL


Rest API
GRAPHQL

```
it("test", () => {
    cy.route('GET', '/users', 'fixtures:/user.json').as('getUser')
    cy.route('GET', '/permissions', { ... }).as('getUser')

    const user = { id: 1, name: 'test' }
    cy.route('POST', '/users', user).as('postUser')

    const me = { id: 1, name: user.name }
    cy.route('POST', '/me').as('me')
})
```

Моск-и

ЭТО ВРАГИ!

Elon Musk @elonmusk · Oct 23, 2021

Replying to @elonmusk

Regression in some left turns at traffic lights found by internal QA in 10.3. Fix in work, probably releasing tomorrow.

Elon Musk
@elonmusk

Seeing some issues with 10.3, so rolling back to 10.2 temporarily.

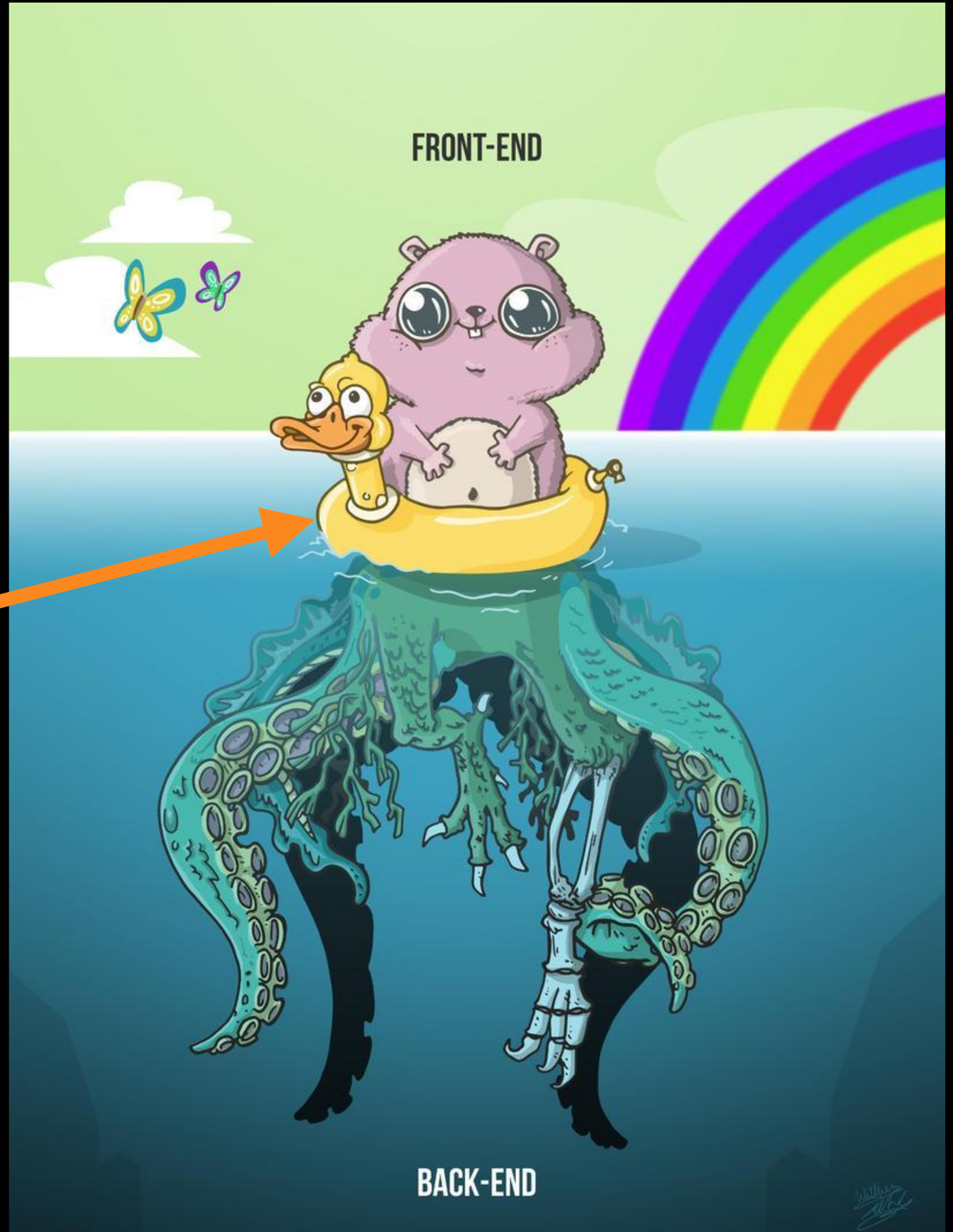
Please note, this is to be expected with beta software. It is impossible to test all hardware configs in all conditions with internal QA, hence public beta.

9:44 PM · Oct 24, 2021

22.8K 2.9K Copy link to Tweet

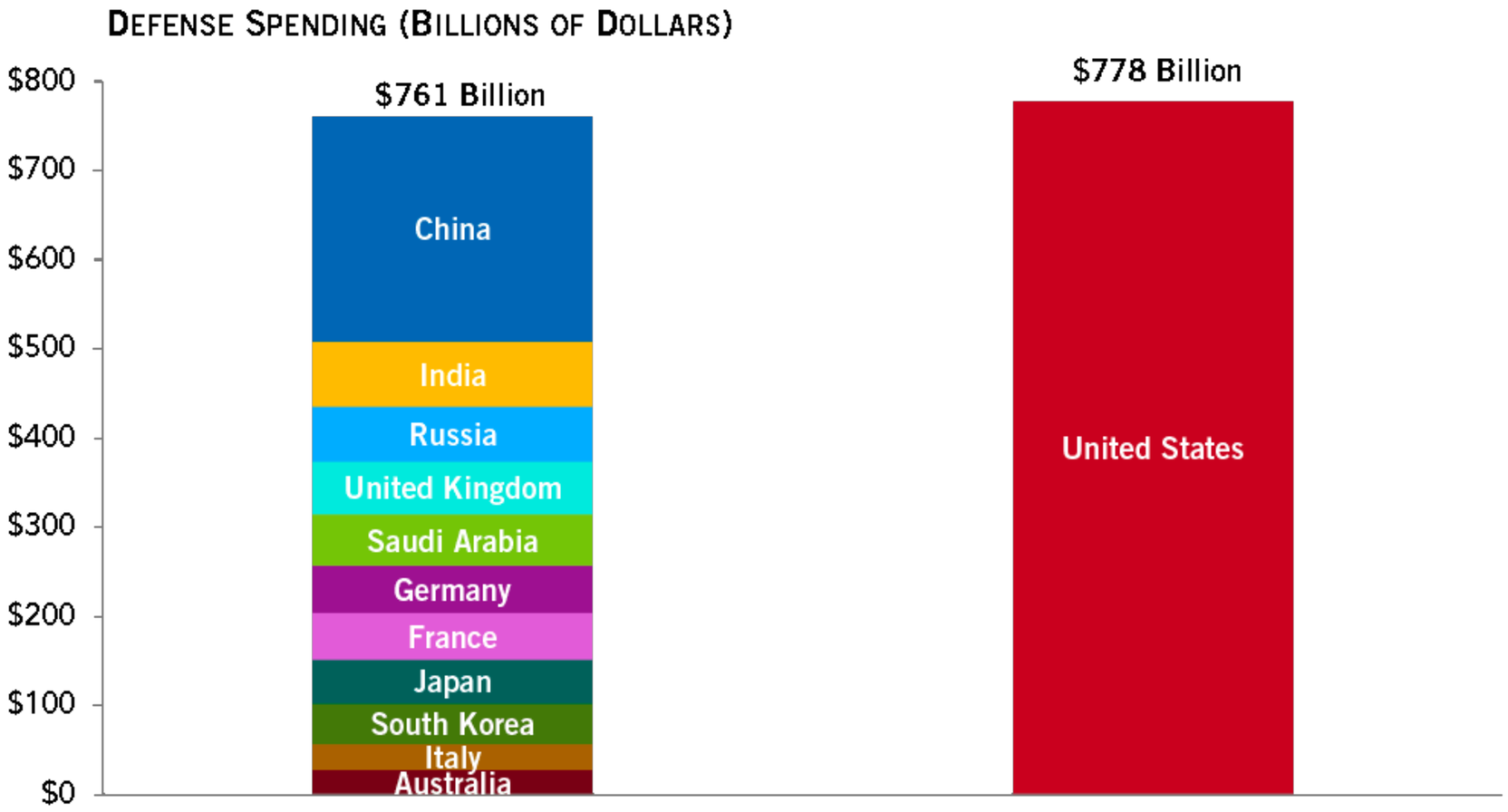
Tweet your reply

Do not mock what you own




STATE

The United States spends more on defense than the next 11 countries combined



SOURCE: Stockholm International Peace Research Institute, *SIPRI Military Expenditure Database*, April 2021.
NOTES: Figures are in U.S. dollars converted from local currencies using market exchange rates. Data for the United States are for fiscal year 2020, which ran from October 1, 2019 through September 30, 2020. Data for the other countries are for calendar year 2020. The source for this chart uses a definition of defense spending that is more broad than budget function 050 and defense discretionary spending.

LightSource

Welcome back, supply chain hero!

Don't have an account? Create one →

Email address

Password

Forgot your password?

Sign me in

73%

Не доверяют своим тестам

Dmitriy Kovalenko
@dmtrKovalenko

This week I am speaking about test confidence on a conference. So I have really important question!

Do you trust your tests? E.g. can you blindly push major dependency upgrade to production if your tests ✅

RT pls 🙏

Yes!	43%
Not really...	**57%**

100 votes · Final results

5:51 PM · Nov 1, 2021

9 2 Copy link to Tweet

Tweet your reply

А вы?

Тогда просто попробуйте использовать свои тесты во время разработки.

Michael Kalygin @mkalygin · Oct 1, 2021

Replying to @dmtrKovalenko

Тесты были не мои, но однажды я фиксил либу для AWS S3. Версия S3 API обновилась, а тесты использовали моки для API старой версии. После обновления зависимости с AWS S3 все тесты были зелёными, а по факту код не работал. :–)

Lonli-Lokli

@sirlonlilokli

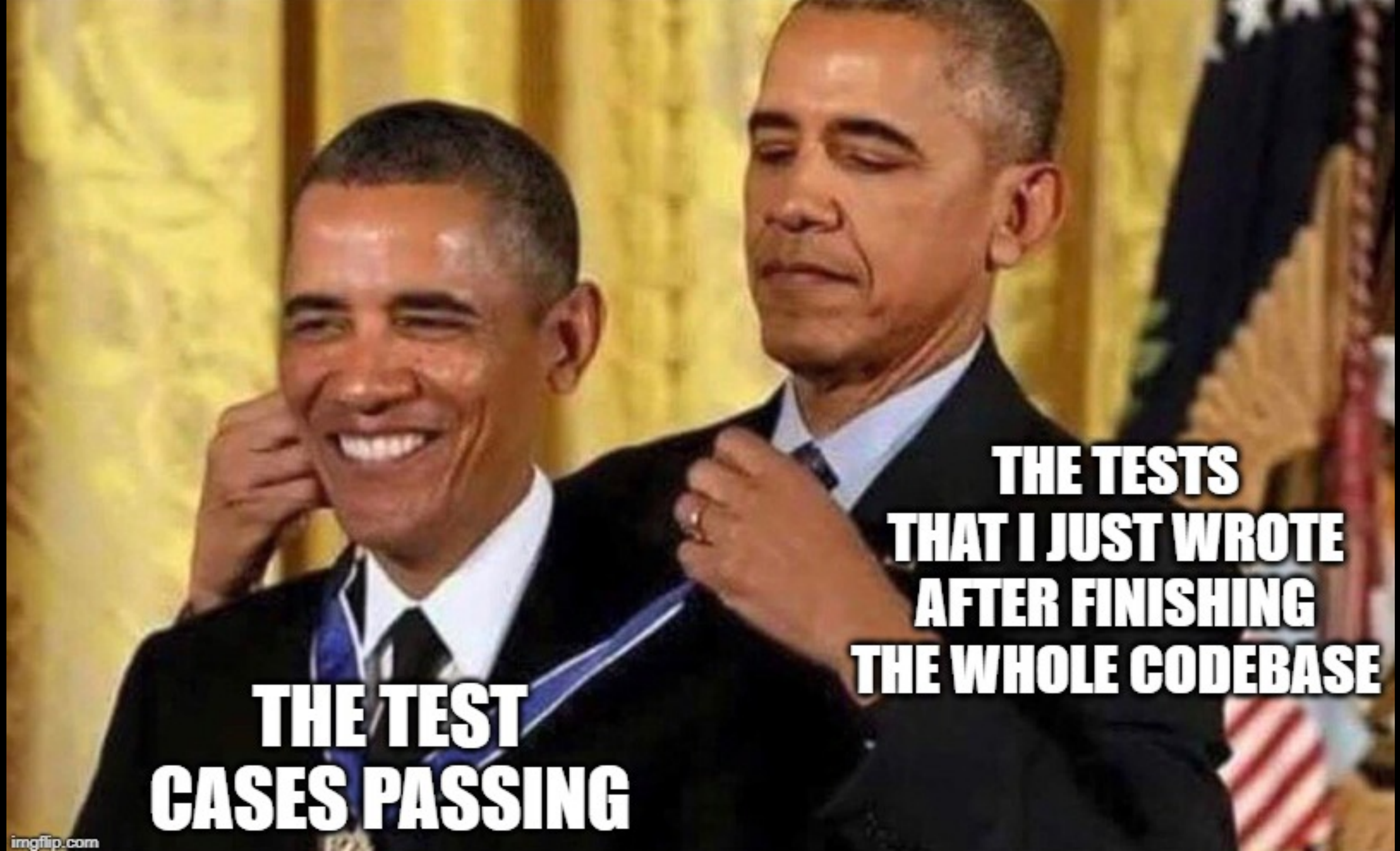
Поэтому нужны тесты что 'а вот сейчас должно упасть'. К сожалению, очень часто тесты пишут в расчете получить OK.

6:41 PM · Oct 2, 2021

6 1 Copy link to Tweet

Tweet your reply

Тестируем поведение


THE TESTS
THAT I JUST WROTE
AFTER FINISHING
THE WHOLE CODEBASE
THE TEST
CASES PASSING
imgflip.com

1. Напиши тест

Бесполезность

```
it("test", async () => {
  await request(app)
    .get('/goose')
    .expect(404)
})
```


Predictor
Zwangerschapstest
When you want to be sure.

Add only

```
it('does not call callback if no policies are found', () => {
  const run = chromePolicyCheck.getRunner({
    enumerateValues: _.constant([]),
  })

  const cb = sinon.stub()

  run(cb)

  expect(cb).to.not.be.called
})
```

Add only

```
it('fails silently if enumerateValues throws', () => {
  const run = chromePolicyCheck.getRunner({
    enumerateValues () {
      throw new Error('blah')
    },
  })

  const cb = sinon.stub()

  run(cb)

  expect(cb).to.not.be.called
})
})
```

1. Напиши тест

2. Пишем Код

НО

Это же медленно

Ну да

А вы чего хотели?

НО

А как же фронтенд?

Playwright Inspector

Record test.js

```
11  await page.goto('https://github.com/microsoft');
12
13  // Click input[aria-label="Find a repository…"]
14  await page.click('input[aria-label="Find a repository…"]');
15
16  // Fill input[aria-label="Find a repository…"]
17  await Promise.all([
18    page.waitForNavigation(/*{ url: 'https://github.com/microsoft?q=playwright&type=
19    page.fill('input[aria-label="Find a repository…"]', 'playwright')
20  ]);
21
22  // Click //a[normalize-space(.)='playwright']
23  await page.click('//a[normalize-space(.)=\'playwright\']');
24  // assert.equal(page.url(), 'https://github.com/microsoft/playwright');
25
26  // Click text="Issues"
27  await Promise.all([
28    page.waitForNavigation(/*{ url: 'https://github.com/microsoft/playwright/issues'
29    page.click('text="Issues"')
30  ]);
31  await page.pause();
32
33  // Click text="triaging"
34  await Promise.all([
35    page.waitForNavigation(/*{ url: 'https://github.com/microsoft/playwright/issues?
```

Explore //a[normalize-space(.)='playwright']

```
> page.goto(https://github.com/microsoft) ✓ — 1.3s
> page.click(input[aria-label="Find a repository...”]) ✓ — 135ms
> page.waitForNavigation ✓ — 4.6s
> page.fill(input[aria-label="Find a repository...”]) ✓ — 69ms
v page.click(//a[normalize-space(.)='playwright']) II
  waiting for selector "//a[normalize-space(.)='playwright']"
```

Playwright Trace Viewer

Playwright

0 200ms 400ms 600ms 800ms 1.0s 1.2s 1.4s 1.6s 1.8s 2.0s 2.2s 2.4s

page.click a:has-text("playwright")

Actions | Action | Before | After | Log | Source | Network

```
page.goto https://github.com/microsoft
page.click [placeholder="Find a repository...”]
page.fill [placeholder="Find a repository...”]
page.waitForNavigation
page.click a:has-text("playwright")
page.waitForNavigation
page.click text=Issues
page.click [placeholder="Search all issues"]
page.fill [placeholder="Search all issues"]
page.waitForNavigation
page.press [placeholder="Search all issues"]
```

```
waiting for selector "a:has-text("playwright")
  selector resolved to visible <a class="d-inli
attempting click action
  waiting for element to be visible, enabled ar
  element is visible, enabled and stable
  scrolling into view if needed
  done scrolling
  checking that element receives pointer ever
  element does receive pointer events
  performing click action
  click action done
  waiting for scheduled navigations to finish
  navigated to "https://github.com/microsoft/
  navigations have finished
```

TDD