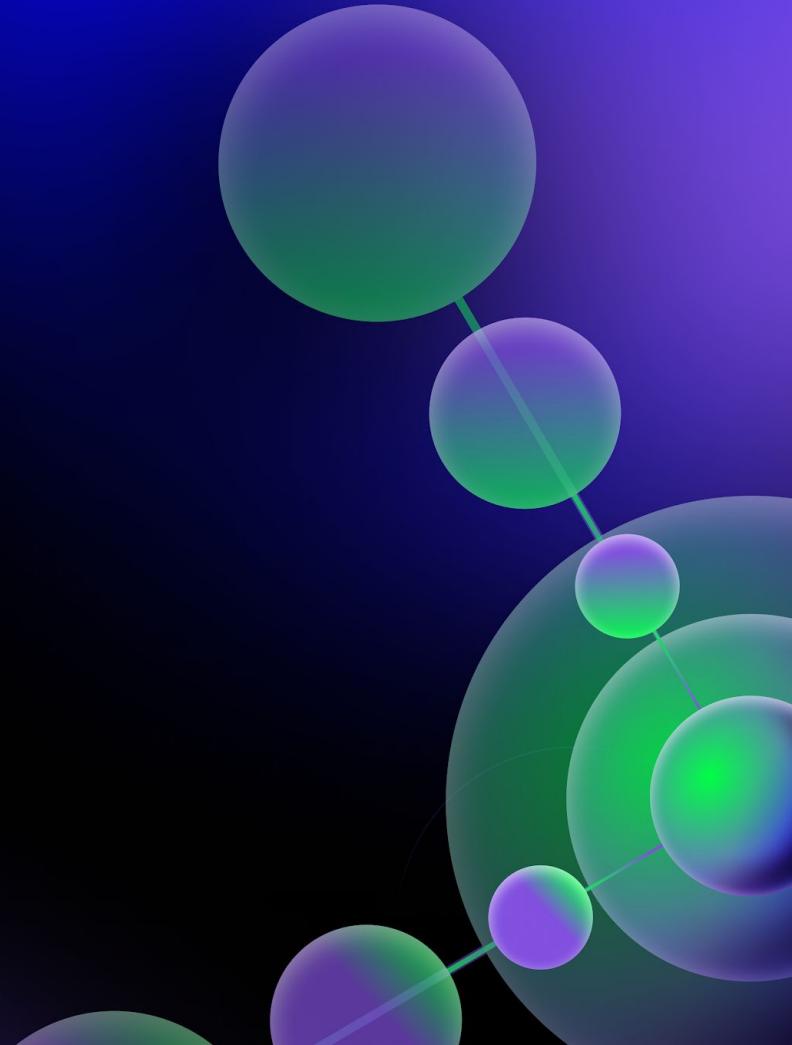


Test Driven Development



Tyler McGoffin - @jtmcg
Senior Software Engineer - GitHub CLI



Intro to @jtmcg



Professionally:

- Sr. Software Eng for GitHub CLI
- Formerly Manager and Engineer at CircleCI, Atlassian

Historically:

- B.S. Physics & B.S. Mathematics
- Research Scientist - Photovoltaics
- Teacher - Game Design & Coding
- Game Designer - Offworld Games

Personally:

- Triathlete
- D&D nerd
- Husband and dog dad



Introduction to @jtmeg



Let's start with a story...



Test Driven Development



AI & automation

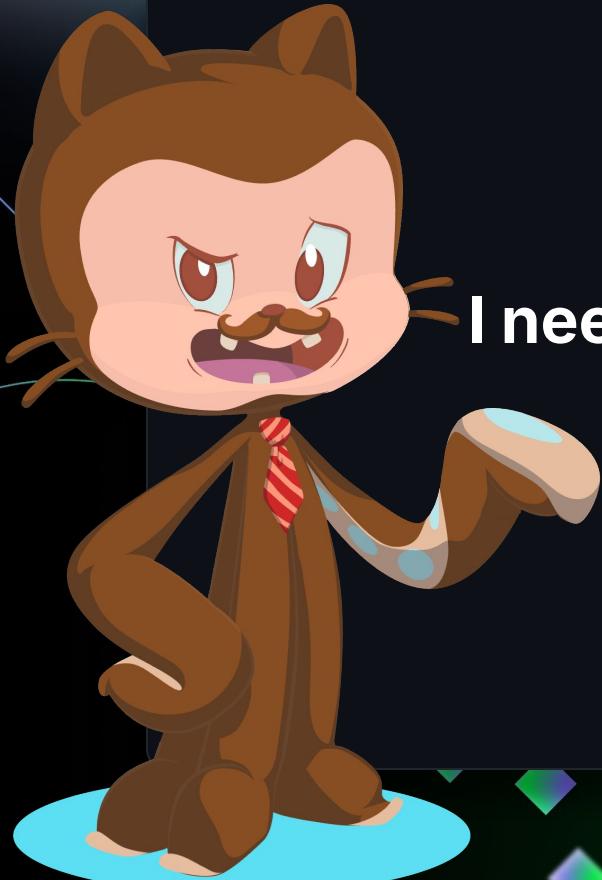


Q&A

Agenda



Story time



“

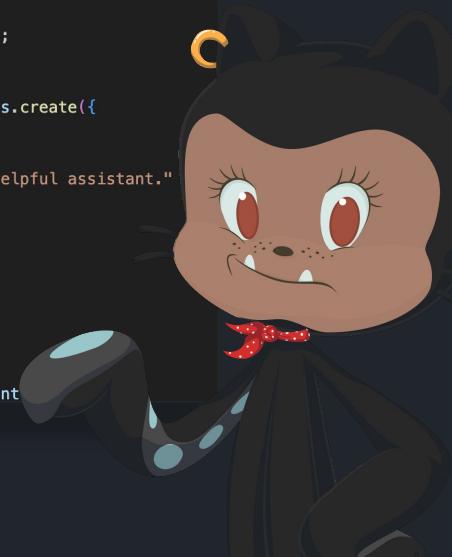
**I need a feature that will chat with
OpenAI via a file**

Your Product Manager, once upon a time

Easy-peasy!

You, the baddest developer in
the room

```
4
5  const queryOpenAI = async () => {
6    // Set up the OpenAI Client
7    const API_KEY = process.env.OPEN_API_KEY_TDD_WORKSHOP
8    const ORGANIZATION_ID = process.env.OPENAI_ORGANIZATION_ID
9    const PROJECT_ID = process.env.OPENAI_TDD_WORKSHOP_PROJECT_ID
10
11   const openai = new OpenAI({
12     organization: ORGANIZATION_ID,
13     project: PROJECT_ID,
14     apiKey: API_KEY,
15   });
16
17   // Read the query from our text file
18   const filePath = "./query.txt";
19   const query = fs.readFileSync(filePath, "utf8");
20
21   // Query OpenAI
22   const completion = await openai.chat.completions.create({
23     model: "gpt-4o-mini",
24     messages: [
25       { role: "system", content: "You are a helpful assistant." },
26       {
27         role: "user",
28         content: query,
29       },
30     ],
31   });
32
33   // Log the response
34   console.log(completion.choices[0].message.content)
```





“

**Great! Now make it work for Claude,
Gemini, and LLaMa too. Oh, and let
users pick which one to use**

Your Product Manager, twice upon a time?

“Cool, I’ll just change this here...”

Your teammate



```
4
5  const queryOpenAI = async () => {
6    // Set up the OpenAI Client
7    const API_KEY = process.env.OPEN_API_KEY_TDD_WORKSHOP
8    const ORGANIZATION_ID = process.env.OPENAI_ORGANIZATION_ID
9    const PROJECT_ID = process.env.OPENAI_TDD_WORKSHOP_PROJECT_ID
10
11   const openai = new OpenAI({
12     organization: ORGANIZATION_ID,
13     project: PROJECT_ID,
14     apiKey: API_KEY,
15   });
16
17   // Read the query from our text file
18   const filePath = "./query.txt";
19   const query = fs.readFileSync(filePath, "utf8");
20
21   // Query OpenAI
22   const completion = await openai.chat.completions.create({
23     model: "gpt-4o-mini",
24     messages: [
25       { role: "system", content: "You are a helpful assistant." },
26       {
27         role: "user",
28         content: query,
29       },
30     ],
31   });
32
33   // Log the response
34   console.log(completion.choices[0].message.content);
```

How do you know it all still works?



Why write tests?

Reliability:

- Early bug detection
- Automation

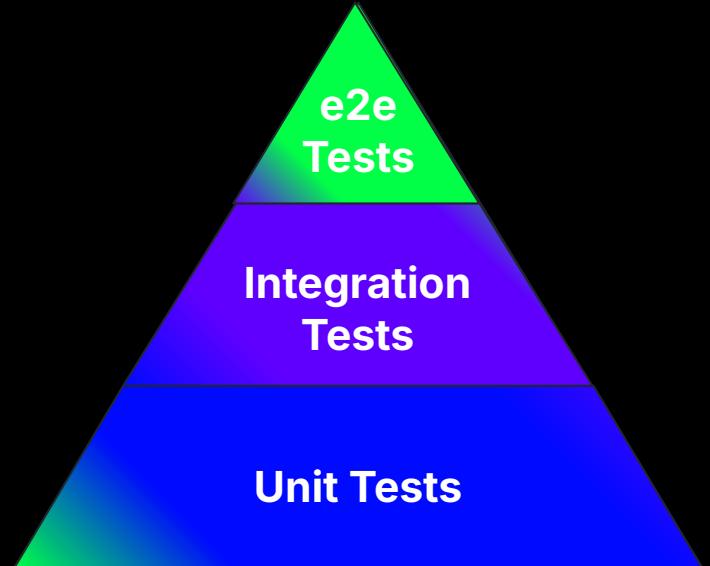
Maintainability:

- Safe refactoring
- Edge case confidence/discovery
- Reduction of tech debt

Productivity:

- Facilitates collaboration
- Developer confidence
- Faster delivery
- Documentation of behavior

The testing pyramid



Unit Tests

Verify units of code are working as expected

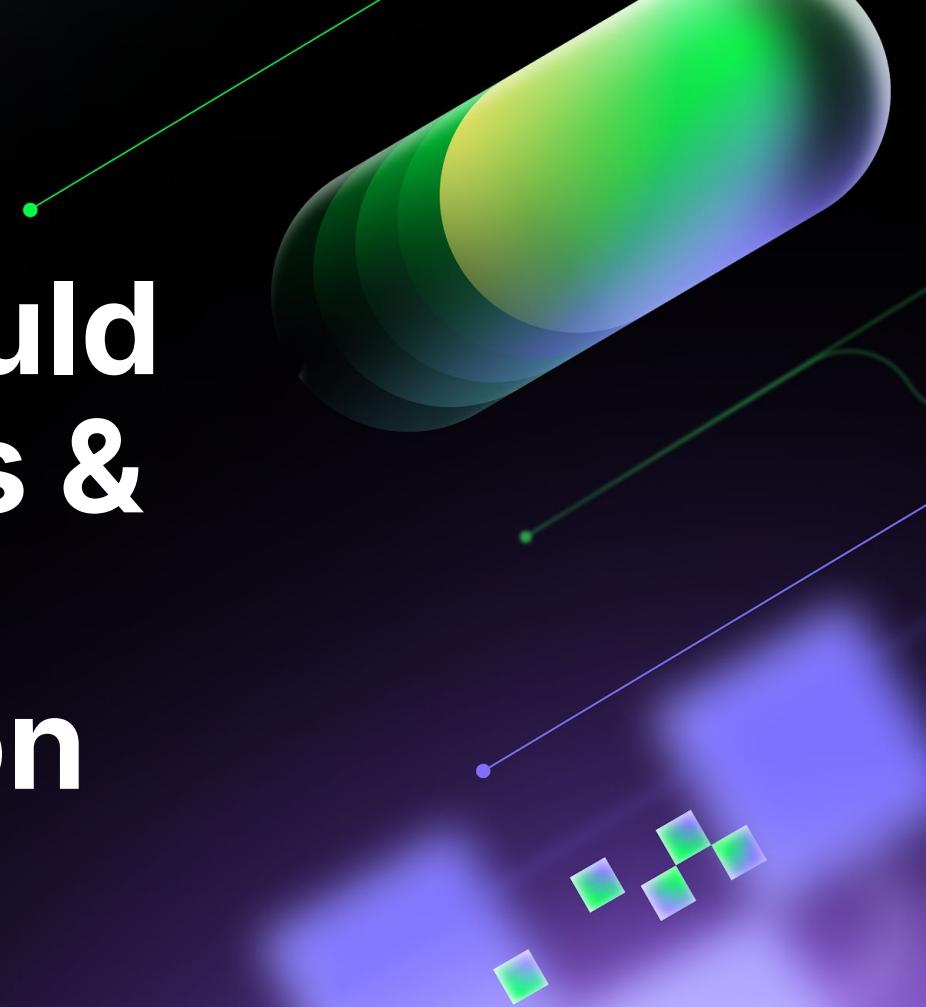
Integration Tests

Verify features are working as expected

End-to-end (e2e) Tests

Verify features are working in production

Unit tests should validate inputs & outputs, not implementation



Code that is easy to test is good code

Idiomatic

Facilitates language-specific patterns in coding

Extensible

Small, orthogonal, and composable pieces of code

Understandable

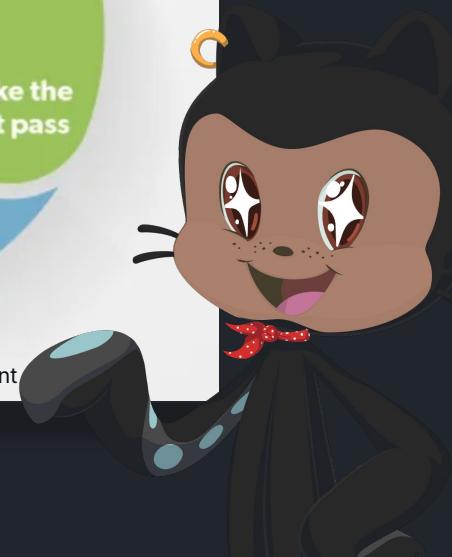
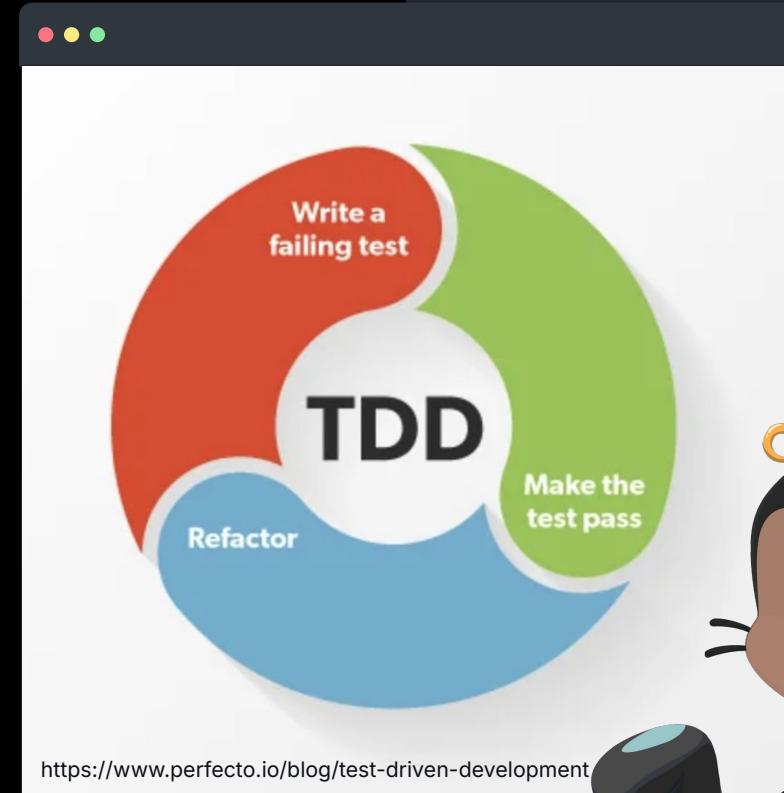
Easy to read and maintain



Test Driven Development (TDD)

"

I need a feature that
will chat with OpenAI
via a file



Principles of TDD

Write tests first

You must have a failing test in place before writing code to make it pass

Keep it simple & stupid

Results in easy to understand code, reliable tests, and novel solutions

Refactor often

Be willing to change everything to accommodate the next test

Don't overtest

If you can't write a new failing test, you're done

Red: Write a failing test

```
goodExample > JS goodExample.test.js > ...
```

```
1 import queryOpenAI from './goodExample'  
2  
3 describe('goodExample', () => {  
4     it('should return a response string', () => {  
5         expect(typeof queryOpenAI()).toBe("string");  
6     })  
7 })
```

Green: Make the test pass

```
goodExample > JS goodExample.test.js > ...
```

```
1 import queryOpenAI from './goodExample'  
2  
3 describe('goodExample', () => {  
4     it('should return a response string', () => {  
5         expect(typeof queryOpenAI()).toBe("string");  
6     })  
7 })
```

```
goodExample > JS goodExample.js > [?] default
```

```
1 const queryOpenAI = () => {  
2     return "Asshole Driven Development 😞"  
3 }  
4  
5 export default queryOpenAI;
```

Repeat: Red → Green

```
import queryOpenAI from './goodExample'

const mockCreate = jest.fn();

describe('goodExample', () => {
  var openAIClientMock;

  beforeEach(() => {
    mockCreate.mockReset();
    openAIClientMock = {
      chat: {
        completions: {
          create: mockCreate
        }
      }
    }
  })

  it('should return a response string', () => {
    expect(typeof queryOpenAI()).toBe("string");
  })

  it('should call OpenAI for a completions', () => {
    queryOpenAI(openAIClientMock);
    expect(mockCreate).toBeCalled();
  })
})
```

```
const queryOpenAI = (openAIClient) => {
  const completion = openAIClient.chat.completions.create()
  return "Asshole Driven Development 😞"
}

export default queryOpenAI;
```

```
$ npm test ./doItLive/goodExample.test.js
> tdd-workshop@0.0.0 test
> jest ./doItLive/goodExample.test.js
FAIL doItLive/goodExample.test.js
secondTest
  ✓ should return a response string (2 ms)
  ✘ should call OpenAI for a completion (1 ms)

● secondTest > should call OpenAI for a completion

  expect(jest.fn()).toBeCalled()

    Expected number of calls: >= 1
    Received number of calls:   0
      23 |     it('should call OpenAI for a completion', () => {
      24 |       queryOpenAI(openAIClientMock);
      25 |       expect(mockCreate).toBeCalled();
      26 |     })
      27 |   }

    at Object.toBeCalled (doItLive/goodExample.test.js:25:28)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 1 passed, 2 total
Snapshots:  0 total
Time:        2.176 s
Ran all test suites matching ./doItLive/goodExample.test.js/i.
```

Refactor

```
1 import queryOpenAI from './goodExample'
2
3 const mockCreate = jest.fn();
4
5 describe('goodExample', () => {
6   var openAIClientMock;
7
8   beforeEach(() => {
9     mockCreate.mockReset();
10    openAIClientMock = {
11      chat: {
12        completions: {
13          create: mockCreate
14        }
15      }
16    }
17  })
18
19 it('should return a response string', () => {
20   expect(typeof queryOpenAI(openAIClientMock))
21 })
22
23 it('should call OpenAI for a completions', () => {
24   queryOpenAI(openAIClientMock);
25   expect(mockCreate).toBeCalled();
26 })
27 })
```

```
const queryOpenAI = (openAIClient) => {
  const completion = openAIClient.chat.completions.create()
  return "Asshole Driven Development 😝"
}

export default queryOpenAI;
```

```
$ npm test ./doItLive/goodExample.test.js
> tdd-workshop@0.0.0 test
> jest ./doItLive/goodExample.test.js

PASS doItLive/goodExample.test.js
secondTest
  ✓ should return a response string (1 ms)
  ✓ should call OpenAI for a completion

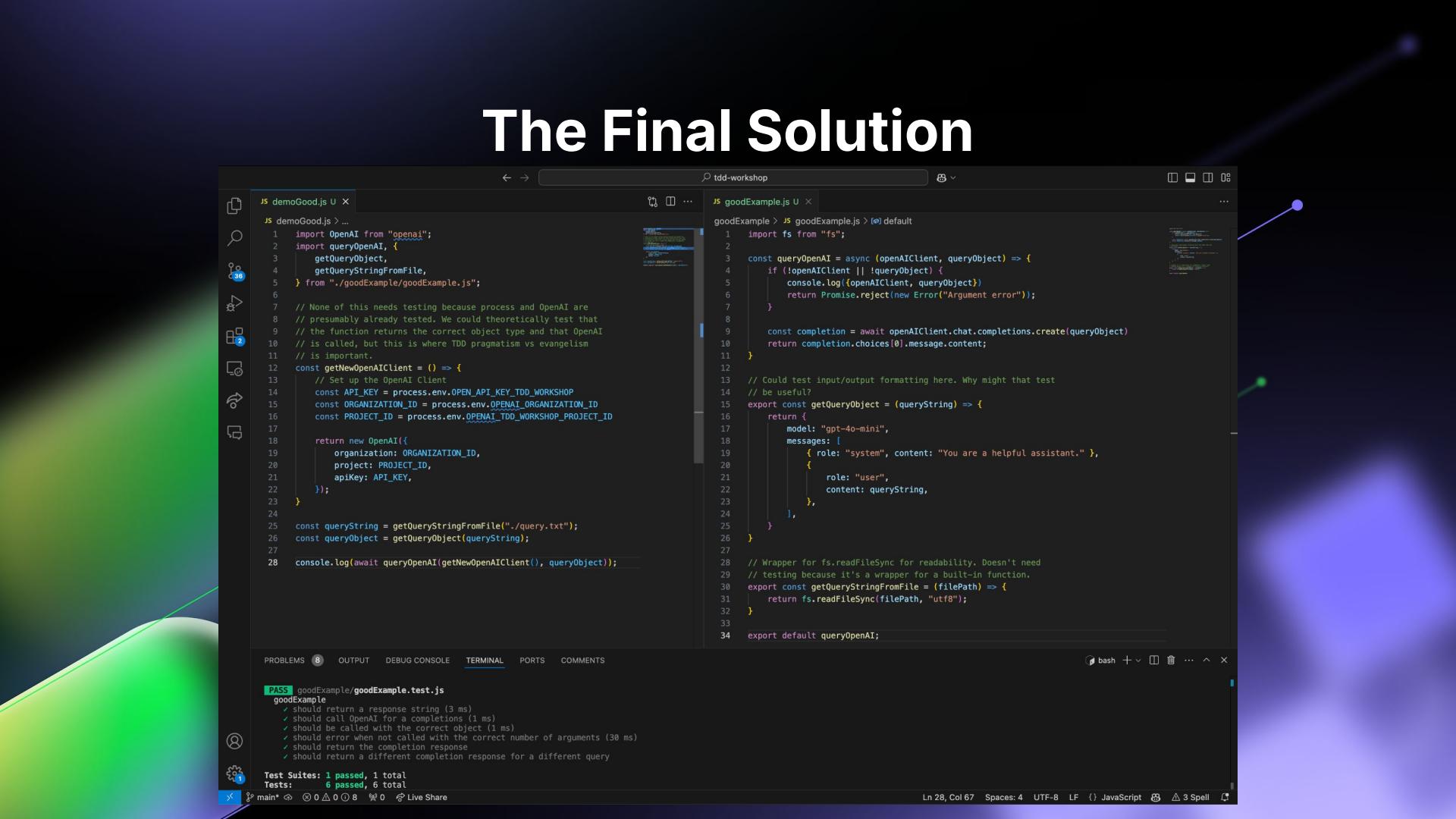
Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:  0 total
Time:        0.278 s
Ran all test suites matching ./doItLive/goodExample.test.js/i.
```

Red, Green, Refactor, Repeat...

See the [doltLive](#) folder in the repo



The Final Solution



JS demoGood.js U x

```
1 import OpenAI from "openai";
2 import queryOpenAI, {
3   getQueryObject,
4   getQueryStringFromFile,
5 } from "./goodExample/goodExample.js";
6
7 // None of this needs testing because process and OpenAI are
8 // presumably already tested. We could theoretically test that
9 // the function returns the correct object type and that OpenAI
10 // is called, but this is where TDD pragmatism vs evangelism
11 // is important.
12 const getNewOpenAIClient = () => {
13   // Set up the OpenAI Client
14   const API_KEY = process.env.OPEN_API_KEY_TDD_WORKSHOP
15   const ORGANIZATION_ID = process.env.OPENAI_ORGANIZATION_ID
16   const PROJECT_ID = process.env.OPENAI_TDD_WORKSHOP_PROJECT_ID
17
18   return new OpenAI({
19     organization: ORGANIZATION_ID,
20     project: PROJECT_ID,
21     apiKey: API_KEY,
22   });
23 }
24
25 const queryString = getQueryStringFromFile("./query.txt");
26 const queryObject = getQueryObject(queryString);
27
28 console.log(await queryOpenAI(getNewOpenAIClient(), queryObject));
```

JS goodExample.js U x

```
goodExample > JS goodExample.js > (e) default
1 import fs from "fs";
2
3 const queryOpenAI = async (openAIClient, queryObject) => {
4   if (!openAIClient || !queryObject) {
5     console.log(openAIClient, queryObject);
6     return Promise.reject(new Error("Argument error"));
7   }
8
9   const completion = await openAIClient.chat.completions.create(queryObject)
10  return completion.choices[0].message.content;
11 }
12
13 // Could test input/output formatting here. Why might that test
14 // be useful?
15 export const getQueryObject = (queryString) => {
16   return (
17     model: "gpt-4o-minim",
18     messages: [
19       {
20         role: "system", content: "You are a helpful assistant."
21       },
22       {
23         role: "user",
24         content: queryString,
25       },
26     ],
27   );
28
29 // Wrapper for fs.readFileSync for readability. Doesn't need
30 // testing because it's a wrapper for a built-in function.
31 export const getQueryStringFromFile = (filePath) => {
32   return fs.readFileSync(filePath, "utf8");
33 }
34
35 export default queryOpenAI;
```

PROBLEMS 8 OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

PASS goodExample/goodExample.test.js

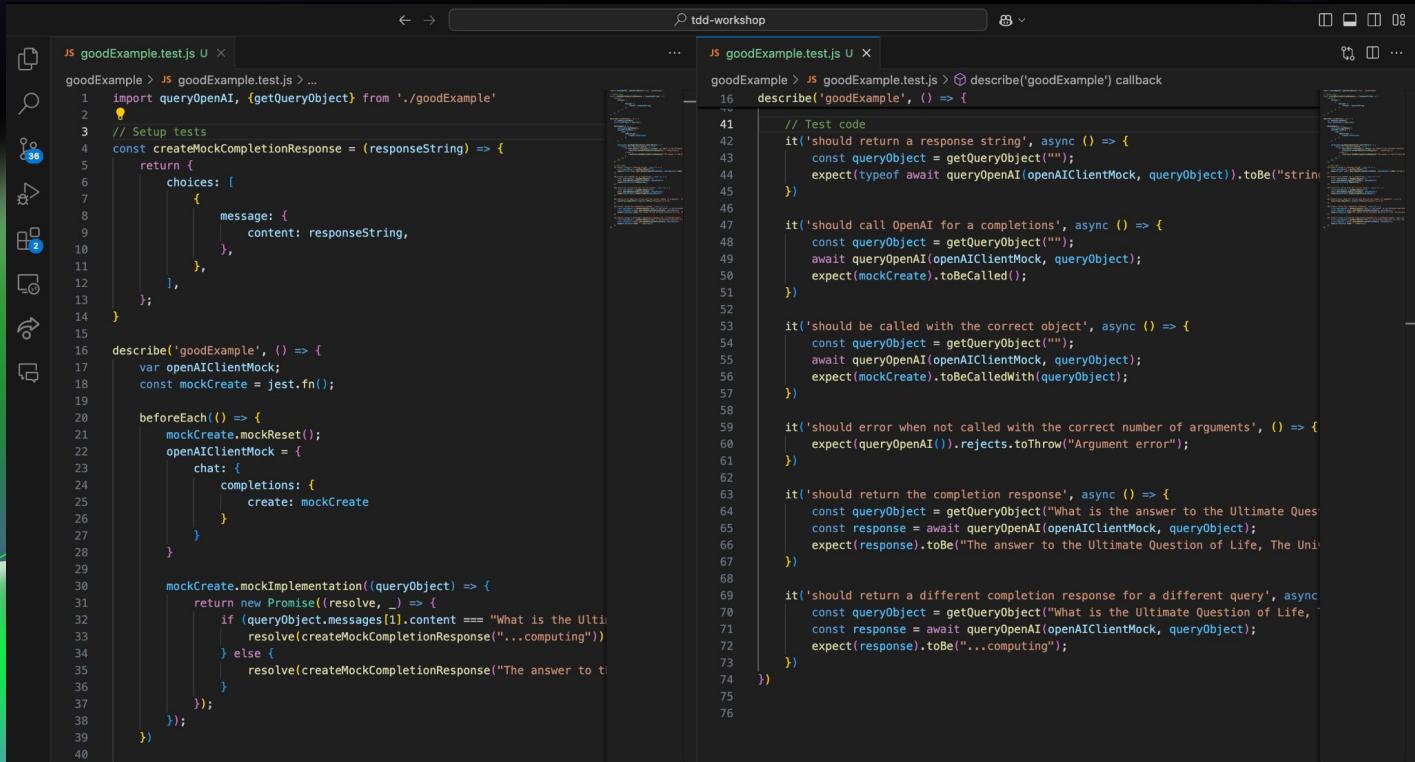
```
goodExample
✓ should return a response string (3 ms)
✓ should call OpenAI for a completions (1 ms)
✓ should be called with the correct object (1 ms)
✓ should error when not called with the correct number of arguments (30 ms)
✓ should return the completion response
✓ should return a different completion response for a different query
```

Test Suites: 1 passed, 1 total
Tests: 6 passed, 6 total

main* ⇨ 0 ⌂ 0 ⌂ 8 ⌂ 0 ⌂ Live Share

In 28, Col 67 Spaces: 4 UTF-8 LF () JavaScript ⚡ 3 Spell

The Final Tests



```
JS goodExample.test.js U x
goodExample > JS goodExample.test.js > ...
1 import queryOpenAI, {getQueryObject} from './goodExample'
2
3 // Setup tests
4 const createMockCompletionResponse = (responseString) => {
5   return {
6     choices: [
7       {
8         message: {
9           content: responseString,
10          },
11        },
12      ],
13    };
14  }
15
16 describe('goodExample', () => {
17   var openAIClientMock;
18   const mockCreate = jest.fn();
19
20   beforeEach(() => {
21     mockCreate.mockReset();
22     openAIClientMock = {
23       chat: {
24         completions: {
25           create: mockCreate
26         }
27       }
28     }
29
30     mockCreate.mockImplementation((queryObject) => {
31       return new Promise((resolve, _) => {
32         if (queryObject.messages[1].content === "What is the Ulti")
33         resolve(createMockCompletionResponse("...computing"))
34       } else {
35         resolve(createMockCompletionResponse("The answer to t")
36       }
37     });
38   });
39 }
40

JS goodExample.test.js U x
goodExample > JS goodExample.test.js > ⚡ describe('goodExample') callback
16 describe('goodExample', () => {
17
18   // Test code
19   it('should return a response string', async () => {
20     const queryObject = getQueryObject("");
21     expect(typeof await queryOpenAI(openAIClientMock, queryObject)).toBe("string")
22   })
23
24   it('should call OpenAI for a completions', async () => {
25     const queryObject = getQueryObject("");
26     await queryOpenAI(openAIClientMock, queryObject);
27     expect(mockCreate).toHaveBeenCalled();
28   })
29
30   it('should be called with the correct object', async () => {
31     const queryObject = getQueryObject("");
32     await queryOpenAI(openAIClientMock, queryObject);
33     expect(mockCreate).toHaveBeenCalledWith(queryObject);
34   })
35
36   it('should error when not called with the correct number of arguments', () => {
37     expect(queryOpenAI()).rejects.toThrow("Argument error");
38   })
39
40   it('should return the completion response', async () => {
41     const queryObject = getQueryObject("What is the answer to the Ultimate Ques")
42     const response = await queryOpenAI(openAIClientMock, queryObject);
43     expect(response).toBe("The answer to the Ultimate Question of Life, The Uni")
44   })
45
46   it('should return a different completion response for a different query', async () => {
47     const queryObject = getQueryObject("What is the Ultimate Question of Life, T")
48     const response = await queryOpenAI(openAIClientMock, queryObject);
49     expect(response).toBe("...computing");
50   })
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76 })
```

Holy crap, Tyler,
that's a lot of
work...



Compare Solutions

```
3 const queryOpenAI = async (openAIClient, queryObject) => {
4   if (!openAIClient || !queryObject) {
5     console.log({openAIClient, queryObject})
6     return Promise.reject(new Error("Argument error"));
7   }
8
9   const completion = await openAIClient.chat.completions.create(queryObject)
10  return completion.choices[0].message.content;
11 }
```

```
4
5  const queryOpenAI = async () => {
6    // Set up the OpenAI Client
7    const API_KEY = process.env.OPEN_API_KEY_TDD_WORKSHOP
8    const ORGANIZATION_ID = process.env.OPENAI_ORGANIZATION_ID
9    const PROJECT_ID = process.env.OPENAI_TDD_WORKSHOP_PROJECT_ID
10
11   const openai = new OpenAI({
12     organization: ORGANIZATION_ID,
13     project: PROJECT_ID,
14     apiKey: API_KEY,
15   });
16
17   // Read the query from our text file
18   const filePath = "./query.txt";
19   const query = fs.readFileSync(filePath, "utf8");
20
21   // Query OpenAI
22   const completion = await openai.chat.completions.create({
23     model: "gpt-4o-mini",
24     messages: [
25       { role: "system", content: "You are a helpful assistant." },
26       {
27         role: "user",
28         content: query,
29       },
30     ],
31   });
32
33   // Log the response
34   console.log(completion.choices[0].message.content);
```

Compare Solutions

```
3 const queryOpenAI = async (openAIClient, queryObject) => {
4   if (!openAIClient || !queryObject) {
5     console.log({openAIClient, queryObject})
6     return Promise.reject(new Error("Argument error"));
7   }
8
9   const completion = await openAIClient.chat.completions.create(queryObject)
10  return completion.choices[0].message.content;
11 }
```

Great! Now make it work for Claude, Gemini, and LLaMa too. Oh, and let users pick which one to use



```
4
5  const queryOpenAI = async () => {
6    // Set up the OpenAI Client
7    const API_KEY = process.env.OPEN_API_KEY_TDD_WORKSHOP
8    const ORGANIZATION_ID = process.env.OPENAI_ORGANIZATION_ID
9    const PROJECT_ID = process.env.OPENAI_TDD_WORKSHOP_PROJECT_ID
10
11   const openai = new OpenAI({
12     organization: ORGANIZATION_ID,
13     project: PROJECT_ID,
14     apiKey: API_KEY,
15   });
16
17   // Read the query from our text file
18   const filePath = "./query.txt";
19   const query = fs.readFileSync(filePath, "utf8");
20
21   // Query OpenAI
22   const completion = await openai.chat.completions.create({
23     model: "gpt-4o-mini",
24     messages: [
25       { role: "system", content: "You are a helpful assistant." },
26       {
27         role: "user",
28         content: query,
29       },
30     ],
31   });
32
33   // Log the response
34   console.log(completion.choices[0].message.content);
```

Compare Solutions

```
3 const queryOpenAI = async (openAIClient, queryObject) => {
4   if (!openAIClient || !queryObject) {
5     console.log({openAIClient, queryObject})
6     return Promise.reject(new Error("Argument error"));
7   }
8
9   const completion = await openAIClient.chat.completions.create(queryObject)
10  return completion.choices[0].message.content;
11 }
```



Guess I'll just pass
this a different
client



```
4
5  const queryOpenAI = async () => {
6    // Set up the OpenAI Client
7    const API_KEY = process.env.OPEN_API_KEY_TDD_WORKSHOP
8    const ORGANIZATION_ID = process.env.OPENAI_ORGANIZATION_ID
9    const PROJECT_ID = process.env.OPENAI_TDD_WORKSHOP_PROJECT_ID
10
11  const openai = new OpenAI({
12    organization: ORGANIZATION_ID,
13    project: PROJECT_ID,
14    apiKey: API_KEY,
15  });
16
17  // Read the query from our text file
18  const filePath = "./query.txt";
19  const query = fs.readFileSync(filePath, "utf8");
20
21
22  // Query OpenAI
23  const completion = await openai.chat.completions.create({
24    model: "gpt-4o-mini",
25    messages: [
26      { role: "system", content: "You are a helpful assistant." },
27      {
28        role: "user",
29        content: query,
30      },
31    ],
32  });
33
34
35  // Log the response
36  console.log(completion.choices[0].message.content);
```

Key Takeaways

Write tests **WITH** code

Easy to test = good code and testing as you code begets good code the first time

Test interfaces, not implementation

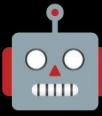
Your tests shouldn't care about how the code is written

TDD can be fun

Gamify your development experience with TDD, especially while pairing

Pragmatism > perfectionism

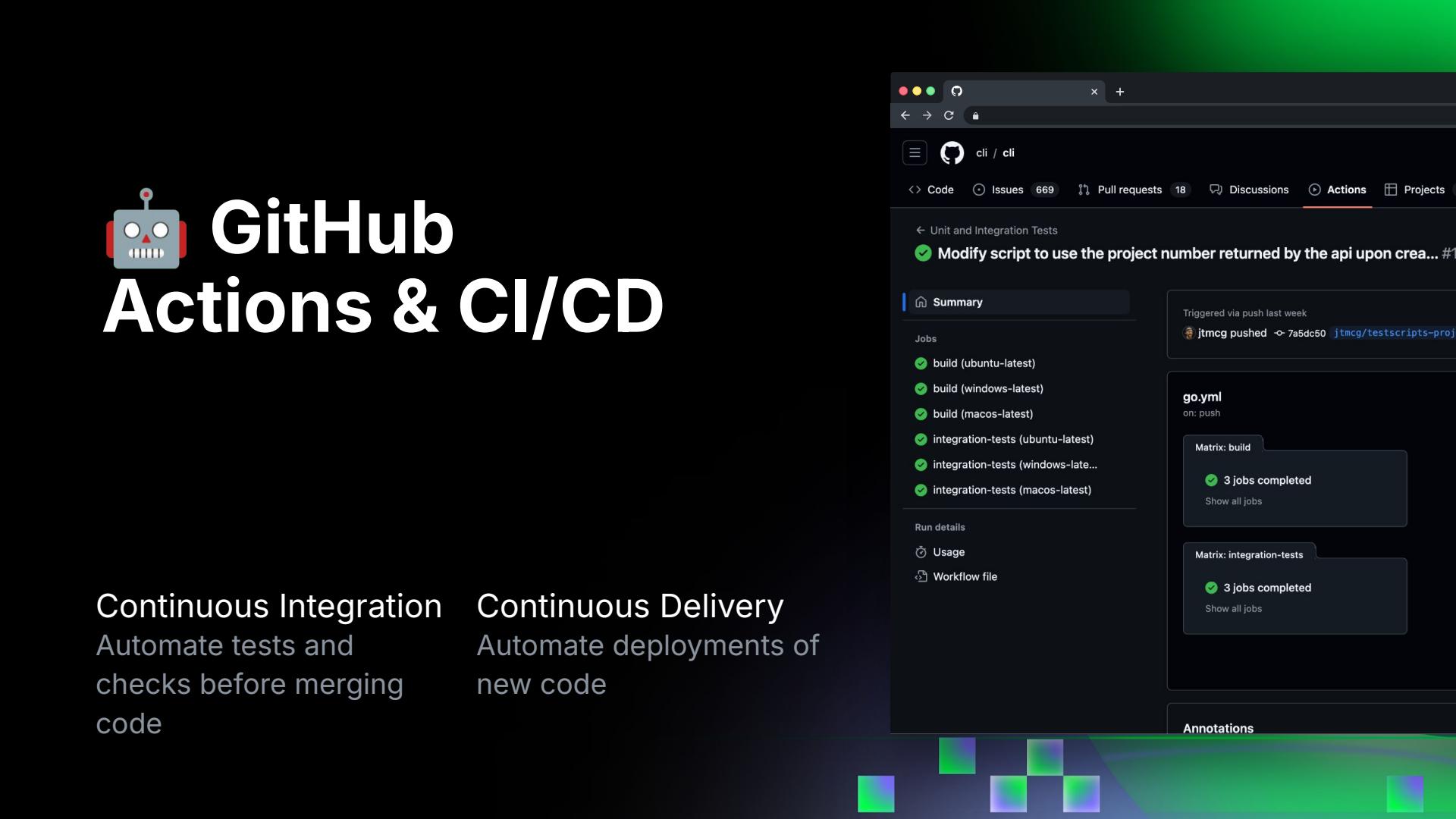
Beware TDD for the sake of TDD and overtesting



GitHub Copilot and Tests

Testing frameworks
can be hard to learn
Copilot can help
accelerate this learning
curve

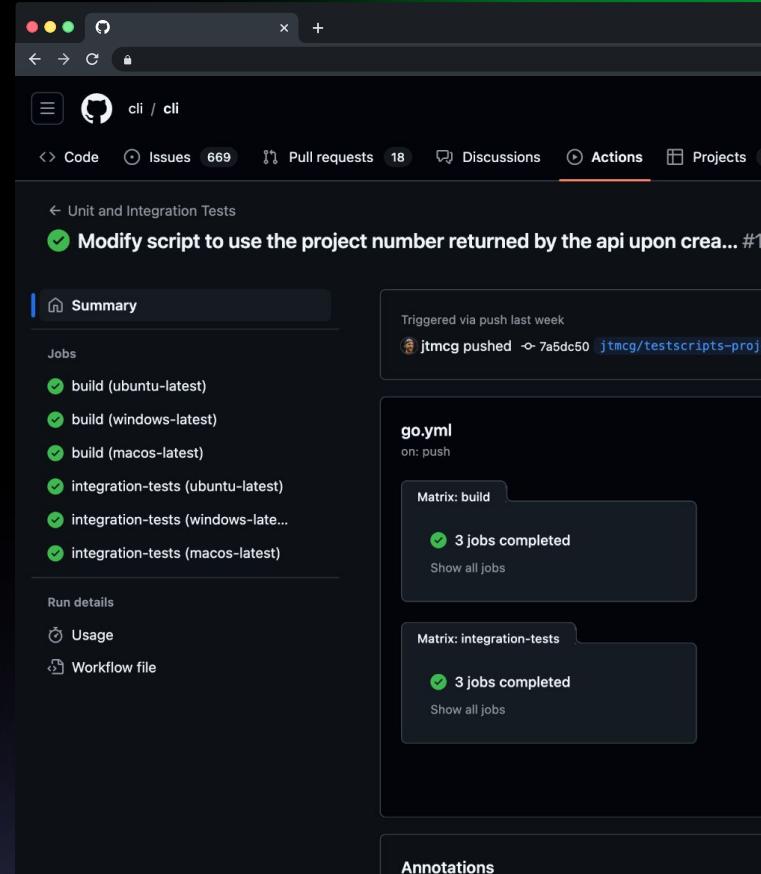
Tests are logically simpler than code
Copilot is really good at writing tests!



GitHub Actions & CI/CD

Continuous Integration
Automate tests and
checks before merging
code

Continuous Delivery
Automate deployments of
new code



A screenshot of a GitHub Actions workflow summary page. The page shows a list of jobs: build (ubuntu-latest), build (windows-latest), build (macos-latest), integration-tests (ubuntu-latest), integration-tests (windows-latest), and integration-tests (macos-latest). All jobs are marked as completed with green checkmarks. A message at the top says "Modify script to use the project number returned by the api upon crea... #". On the right, there are cards for "go.yml" and "Matrix: build" showing 3 completed jobs each. There are also cards for "Matrix: integration-tests" and "Annotations". The GitHub interface includes a sidebar with Code, Issues, Pull requests, Discussions, Actions (which is highlighted in red), and Projects.

Resources

Workshop Repo
github.com/jtmcg/tdd-workshop

Jest testing framework for JS
jestjs.io/docs/getting-started

OpenAI Docs
platform.openai.com/docs/overview

GitHub Actions
docs.github.com/en/actions

GitHub Copilot
github.com/features/copilot



Q&A



Thank you

