flowup

How TDD helps you build modular Angular apps

Matěj Chalk

Full-stack engineer

@ flowup

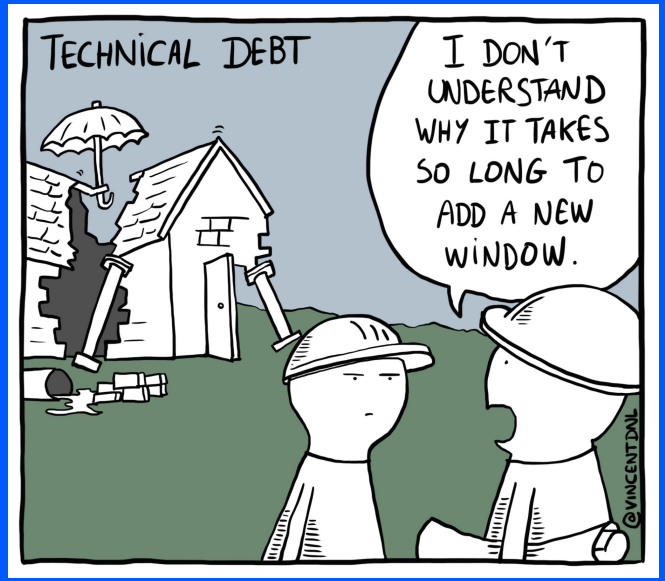
matejchalk

matejchalk

Drowning in tech debt?

flow^up

- large portions of code are considered legacy
- adding features is harder than before, mostly fixing bugs
- fixing a bug causes other unexpected bugs
- refactoring is too scary, developers are scared to touch existing code
- senior developers are burned out
- onboarding process is slow



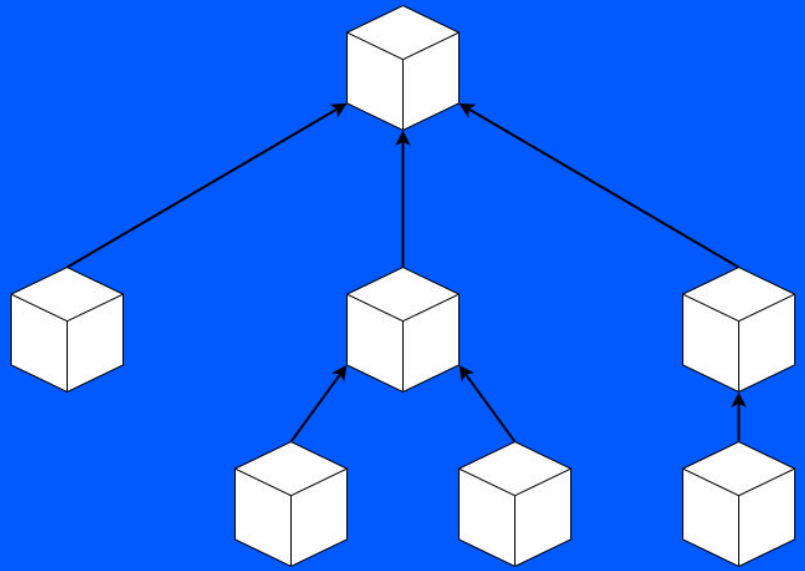
How can this be prevented?

Modular architecture design

flow^up

- as engineers, our job is to solve problems
- to solve complex problems, break them down into simple problems

analysis



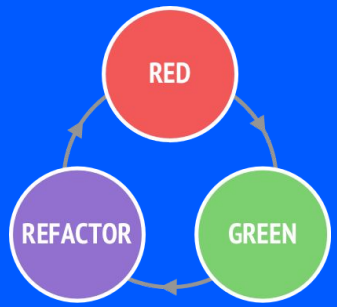
implementation

Unit tests and TDD

flowup

- unit tests
  - bottom-up approach to testing
  - fast to write, fast to run

- test-driven development (TDD)
  - design code with testability in mind
  - ability to write tests before implementation (TDD cycle)
  - increases effort in initial phases, reduces effort in later phases



Example of TDD flow

flowup

1. design function signature without implementing

```
export function dateDiffInDays(fromDate: string, toDate: string): number {
  // TODO: implement
  return Number.POSITIVE_INFINITY;
}
```

2. prepare unit tests

```
test('dateDiffInDays', () => {
  expect(dateDiffInDays('2021-03-12', '2021-03-12')).toBe(0);
  expect(dateDiffInDays('2021-03-12', '2021-03-15')).toBe(3);
  expect(dateDiffInDays('2021-01-01', '2021-02-28')).toBe(58);
  expect(dateDiffInDays('2020-03-12', '2021-03-12')).toBe(365);
  expect(dateDiffInDays('2021-03-12', '2021-03-11')).toBe(-1);
});
```

3. implement function (tests go from failing to passing)

```
export function dateDiffInDays(fromDate: string, toDate: string): number {
  const timeDiff = new Date(toDate).getTime() - new Date(fromDate).getTime();
  return timeDiff / (1000 * 60 * 60 * 24);
}
```

Selective unit testing

flow^up

- *Selective Unit Testing - Costs and Benefits* by Steve Sanderson
- unit testing is most beneficial for complex logic
  - algorithms - parsing, formatting, transformations, filtering, ...
  - pure functions are ideal
- unit testing code with many dependencies requires a lot of mocking
  - side-effects (e.g. API requests, DOM interaction), integration layers (e.g. adapters, wrappers), ...
  - dependencies increase maintenance effort

😊 String formatting

flowup

```
describe('generateCombinationString', () => {
  test('should handle 0 items', () => {
    expect(generateCombinationString([])).toBe('');
  });

  test('should handle 1 item', () => {
    expect(generateCombinationString(['foo'])).toBe('foo');
  });

  test('should handle 2 items', () => {
    expect(generateCombinationString(['foo', 'bar'])).toBe('foo and bar');
  });

  test('should handle 2+ items', () => {
    expect(generateCombinationString(['1', '2', '3', '4'])).toBe('1, 2, 3 and 4');
  });

  test('should allow custom separator', () => {
    expect(generateCombinationString(['1', '2', '3', '4'], ' or ')).toBe('1, 2, 3 or 4');
  });
});
```

😊 Regular expressions

flowup

```
describe('domain regex', () => {
  test('valid domains', () => {
    expect(DOMAIN_VALIDATOR_REGEX.test('flowup.cz')).toBe(true);
    expect(DOMAIN_VALIDATOR_REGEX.test('fake-domain')).toBe(true);
    expect(DOMAIN_VALIDATOR_REGEX.test('www.google.com')).toBe(true);
  });

  test('invalid domains', () => {
    expect(DOMAIN_VALIDATOR_REGEX.test('https://www.flowup.cz')).toBe(false);
    expect(DOMAIN_VALIDATOR_REGEX.test('www.flowup.cz/')).toBe(false);
    expect(DOMAIN_VALIDATOR_REGEX.test('double..dot')).toBe(false);
  });
});
```

😊 Object transformations

flowup

```
describe('entitiesArrayToMap', () => {
  test('should identify entities by key string', () => {
    expect(
      entitiesArrayToMap(
        [
          { id: 1, title: 'First' },
          { id: 2, title: 'Second' },
          { id: 3, title: 'Third' },
          { id: 4, title: 'Fourth' },
        ],
        'id',
      ),
    ).toEqual({
      [1]: { id: 1, title: 'First' },
      [2]: { id: 2, title: 'Second' },
      [3]: { id: 3, title: 'Third' },
      [4]: { id: 4, title: 'Fourth' },
    });
  });

  test('should throw error if ID field is not a string, number or symbol', () => {
    expect(() => entitiesArrayToMap([{ id: [1, 2, 3] }], 'id')).toThrowError();
  });
});
```

😕 Wrapper service for HttpClient (1)

flowup

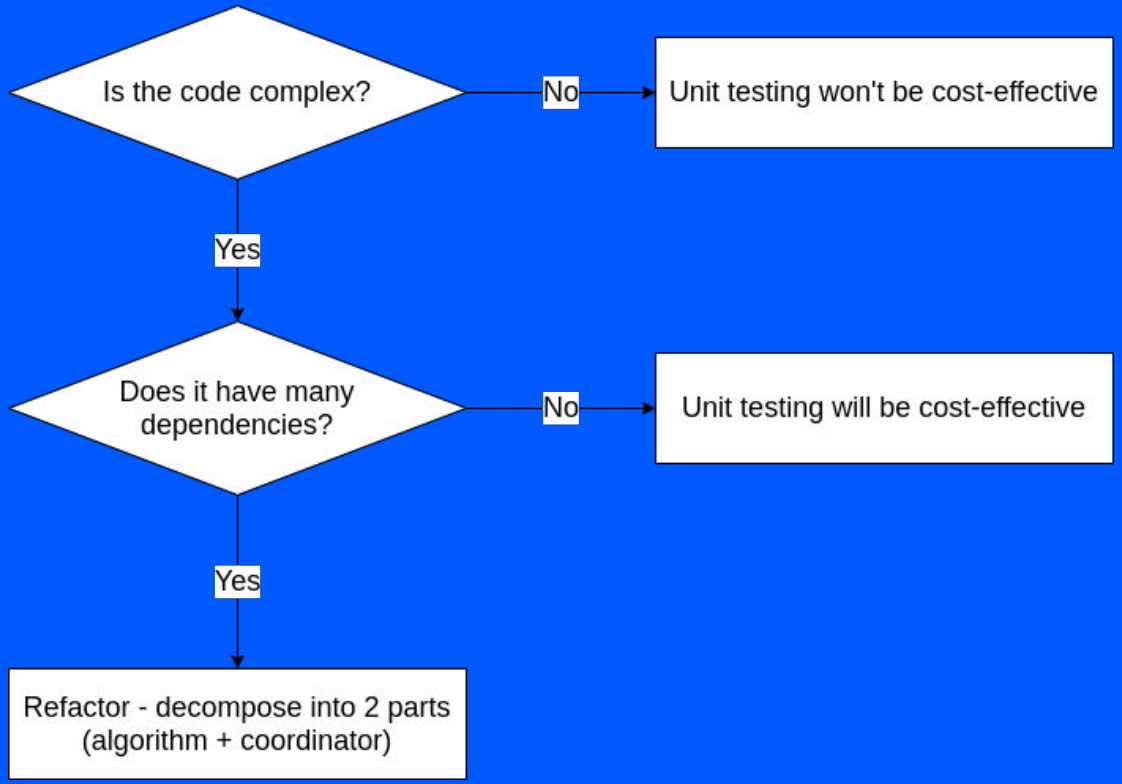
```ts
@Injectable()
export class ApiClient {
  constructor(private readonly http: HttpClient) {}

  getComments(): Observable<CommentModel[]> {
    return this.http.get<CommentModel[]>('/api/comments');
  }

  postComment(body: { text: string; author: string }): Observable<CommentModel> {
    return this.http.post<CommentModel>('/api/comments', body);
  }
}
```

😒 Wrapper service for HttpClient (2) flowup

```
describe('API client', () => {
  test('should get comments', () => {
    const mockHttpClient = {
      get: jest.fn().mockReturnValue(of([{ text: 'Hello' }, { text: 'Bye' }])),
    };
    const apiClient = new ApiClient(mockHttpClient as any);

    apiClient.getComments().subscribe(response => {
      expect(response).toEqual([{ text: 'hello' }, { text: 'Bye' }]);
    });
    expect(mockHttpClient.get).toHaveBeenCalledWith('/api/comments');
  });

  test('should post comment', () => {
    const mockHttpClient = {
      post: jest.fn().mockReturnValue(EMPTY),
    };
    const apiClient = new ApiClient(mockHttpClient as any);

    apiClient.postComment({ text: 'Hi', author: 'john.doe' }).subscribe();

    expect(mockHttpClient.post).toHaveBeenCalledWith('/api/comments', {
      text: 'Hi',
      author: 'john.doe',
    });
  });
});
```

Decision tree for unit testing

flow^up

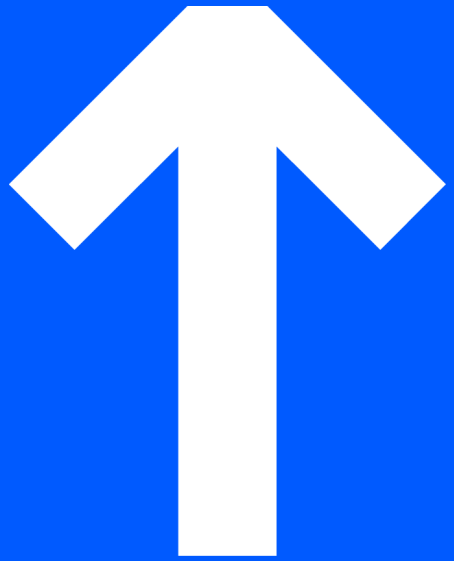


Tips for writing good unit tests

flowup

- a good test works as documentation
  - describes behaviour
  - automatic proof (unlike other doc formats)
- each unit test should be independent of others
- forget DRY principle (Don't Repeat Yourself)
- stick to KISS principle (Keep It Simple, Stupid)

flow^up



Demo time!

github.com/matejchalk/tdd-demo

Nx workspace

flow^up

- “smart, extensible build framework” by Nrwl
- minimal learning curve for Angular developers
- modern tech stack
  - Jest > Karma/Jasmine
  - ESLint > TSLint
  - Cypress > Protractor

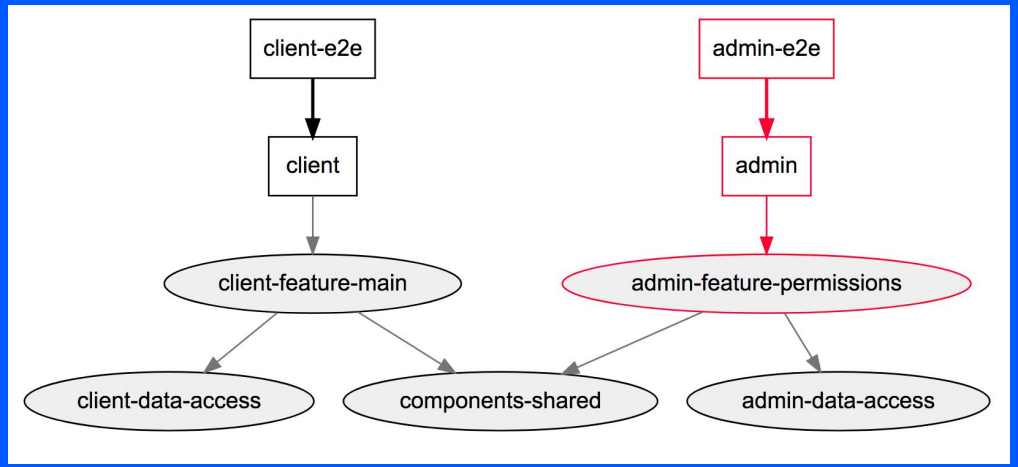
- first-class support for multiple FE/BE frameworks within same monorepo
  - e.g. Angular + Nest.js, React + Express, Next.js, Gatsby, ...

```
$ npx create-nx-workspace
```

Nx workspace (continued)

flowup

- organizes reusable code into libraries (modular architecture)
- manages dependency graph
  - automatic constraints (workspace lint rules)
  - optimized CI (only affected apps/libs)



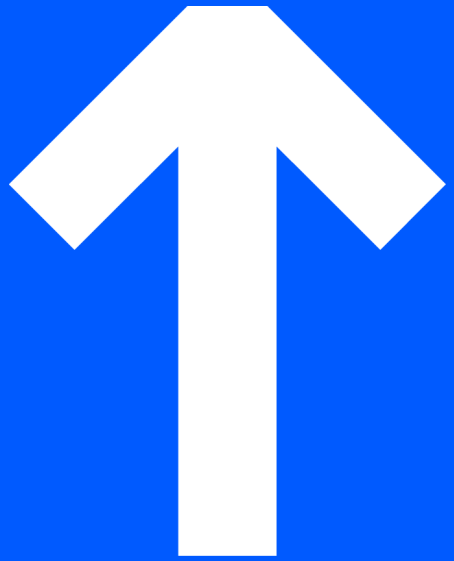
Storybook

flowup

- tool for building UI components in isolation
- document use cases as stories
- highly extensible with addons

```
$ npm i -D @nrwl/storybook
$ nx g
@nrwl/angular:storybook-configuration
```

flow^up



Demo time!

github.com/matejchalk/tdd-demo

Other tips to increase code quality

flow^up

- don't start implementing straight away
  - first analyze the problem and design your solution
- think hard about naming
  - *"There are only two hard things in Computer Science: cache invalidation and naming things."*
  - names tend to stick around, so pick a good one (self-documenting)
  - be consistent to avoid confusion (no synonyms, consider a glossary)
- make a habit of breaking up large source modules into smaller ones - before it's too late!
  - agree on a max value for LOC in your project (lint rule)

Summary

flowup

- tech debt can be effectively managed by modularizing app architecture
  - well-worn idea, but still hard to achieve without guidance
- selective unit testing imposes useful constraints re: maintainability
  - deconstructing into small isolated modules
  - separation of concerns (e.g. business logic vs UI rendering)
  - single responsibility principle
- tools like Nx help maintain a clean dependency graph
  - avoid circular dependencies
  - loose coupling, clear relationships
- tools like Storybook help isolate UI concerns from app logic
  - “smart” vs “dumb” (aka presentational) components

flowup

Q&A

Matěj Chalk

Full-stack engineer

@ flowup

matejchalk

matejchalk