

Prompt 19: Large classes

Smell: Large classes

End to End test code without smell

```
```js
// createTask.cy.js

describe('Mantis test', () => {

 const expectedText = {
 taskExpect: "Operation performed successfully."
 };

 it('Create Task', { tags: '@smoke' }, () => {

 const username = Cypress.config('username');
 const password = Cypress.config('password');

 cy.visit(Cypress.config('baseUrl') + Cypress.env('loginUrl'), {
 failOnStatusCode: false });

 cy.get('[id="username"]').type(username);
 cy.get('input[type="submit"]').click();
 cy.get('[id="password"]').type(password);
 cy.get('input[type="submit"]').click();

 cy.get(':nth-child(3) > a > .menu-text').click();

 cy.get('input[value="Selecionar Projeto"]').click();

 const uuid = () => Cypress._.random(0, 1e6)
 const id = uuid()
 const testname = `testname${id}`
 const testSummary = `summary${id}`

 const summary = testname
 const description = testSummary

 cy.get('[id="summary"]').type(summary);

 cy.get('[id="description"]').type(description);

 cy.get('input[value="Submit Issue"]').click();

 cy.get('.bold').invoke('text').should('include',
 expectedText.taskExpect);

 })

})
```
```

```

```js

// project.cy.js

describe('Mantis test', () => {

 it('Create Project', { tags: '@smoke' }, () => {

 const username = Cypress.config('username');
 const password = Cypress.config('password');

 cy.visit(Cypress.config('baseUrl') + Cypress.env('loginUrl'), {
 failOnStatusCode: false });

 cy.get('[id="username"]').type(username);
 cy.get('input[type="submit"]').click();
 cy.get('[id="password"]').type(password);
 cy.get('input[type="submit"]').click();

 cy.get(':nth-child(7) > a > .menu-text').click();

 cy.get('a[href="/manage_proj_page.php"]').click();

 const name = "Project Name";
 const description = "Project Description";
 const nameP = name;

 cy.xpath('//button[@type="submit" and text()="Create New Project
"]').click();

 cy.get('[id="project-name"]').type(name);

 cy.get('[id="project-description"]').type(description);

 cy.xpath('//input[@type="submit"]').click();

 cy.get('a').invoke('text').should('include', nameP);

 })
})
```

```

```

```js

// loginTest.cy.js

describe('Mantis test', () => {

 const expectedText = {
 textMenu: 'administrator'
 };

 it('Login Test', { tags: '@smoke' }, () => {

```

```

 const username = Cypress.config('username');
 const password = Cypress.config('password');

 cy.visit(Cypress.config('baseUrl') + Cypress.env('loginUrl'), {
failOnStatusCode: false });

 cy.get('[id="username"]').type(username);
 cy.get('input[type="submit"]').click();
 cy.get('[id="password"]').type(password);
 cy.get('input[type="submit"]').click();

 cy.get('.user-info').invoke('text').should('include',
expectedText.textMenu);

 });
})
```

```

```

```js
// newUser.cy.js

describe('Mantis test', () => {

 const expectedText = {
 textMenu: 'Edit User'
 };

 it('New User', { tags: '@smoke' }, () => {

 const username = Cypress.config('username');
 const password = Cypress.config('password');

 const uuid = () => Cypress._.random(0, 1e6)
 const id = uuid()

 const nameUser = `UserTestLarissaProjectAutomationProfile${id}` ;
 const nameReal = `Test Automation Larissa Project Profile${id}`;
 const emailUser = nameUser+"@gmail.com";

 cy.visit(Cypress.config('baseUrl') + Cypress.env('loginUrl'), {
failOnStatusCode: false });

 cy.get('[id="username"]').type(username);
 cy.get('input[type="submit"]').click();
 cy.get('[id="password"]').type(password);
 cy.get('input[type="submit"]').click();

 cy.get(':nth-child(7) > a > .menu-text').click();

 cy.get('a[href="/manage_user_page.php"]').click();

 cy.get('a').contains('Create New Account').click();

 cy.get('[id="user-username"]').type(nameUser);

```

```

 cy.get('[id="user-realname"]').type(nameReal);

 cy.get('[id="email-field"]').type(emailUser);

 cy.get('input[value="Create User"]').click();

 cy.get('#edit-user-form > .widget-box > .widget-header >
 .widget-title').invoke('text').should('include', expectedText.textMenu);

 })

})
```

```

```

```js

// registerTest.cy.js

describe('Mantis test', () => {

 const expectedText = {
 register: "No Change Log information available. Issues are
 included once projects have versions and issues are resolved with \"fixed
 in version\" set."
 };

 it('Click in Registration', () => {

 const username = Cypress.config('username');
 const password = Cypress.config('password');

 cy.visit(Cypress.config('baseUrl') + Cypress.env('loginUrl'), {
 failOnStatusCode: false });

 cy.get('[id="username"]').type(username);
 cy.get('input[type="submit"]').click();
 cy.get('[id="password"]').type(password);
 cy.get('input[type="submit"]').click();

 cy.get(':nth-child(4) > a > .menu-text').click();

 cy.get('.lead').invoke('text').should('include',
 expectedText.register);

 })

})
```

```

Continue others...

End to End test code with smell

```
``js

// totalUser.cy.js

describe('Mantis test', () => {

  const expectedText = {
    taskExpect: "Operation performed successfully.",
    textMenu: 'administrator',
    testUser: "Edit User",
    register: "No Change Log information available. Issues are
included once projects have versions and issues are resolved with \"fixed
in version\" set."

  };

  beforeEach(() => {
    const username = Cypress.config('username');
    const password = Cypress.config('password');

    cy.visit(Cypress.config('baseUrl') + Cypress.env('loginUrl'), {
failOnStatusCode: false });

    cy.get('[id="username"]').type(username);
    cy.get('input[type="submit"]').click();
    cy.get('[id="password"]').type(password);
    cy.get('input[type="submit"]').click();

  })

  it('Create Task', { tags: '@smoke' }, () => {

    cy.get(':nth-child(3) > a > .menu-text').click();

    cy.get('input[value="Selecionar Projeto"]').click();

    const uuid = () => Cypress._.random(0, 1e6)
    const id = uuid()
    const testname = `testname${id}`
    const testSummary = `summary${id}`

    const summary = testname
    const description = testSummary

    cy.get('[id="summary"]').type(summary);

    cy.get('[id="description"]').type(description);

    cy.get('input[value="Submit Issue"]').click();

    cy.get('.bold').invoke('text').should('include',
expectedText.taskExpect);

  })

  it('Create Project', { tags: '@smoke' }, () => {
```

```

        cy.get(':nth-child(7) > a > .menu-text').click();

        cy.get('a[href="/manage_proj_page.php"]').click();

        const name = "Project Name";
        const description = "Project Description";
        const nameP = name;

        cy.xpath('//button[@type="submit" and text()="Create New Project"]').click();

        cy.get('[id="project-name"]').type(name);

        cy.get('[id="project-description"]').type(description);

        cy.xpath('//input[@type="submit"]').click();

        cy.get('a').invoke('text').should('include', nameP);
    })

    it('Login Test', { tags: '@smoke' }, () => {

        cy.get('.user-info').invoke('text').should('include',
expectedText.textMenu);

    });

    it('New User', { tags: '@smoke' }, () => {

        const username = Cypress.config('username');
        const password = Cypress.config('password');

        const uuid = () => Cypress._.random(0, 1e6)
        const id = uuid()

        const nameUser = `UserTestLarissaProjectAutomationProfile${id}`;
        const nameReal = `Test Automation Larissa Project Profile${id}`;
        const emailUser = nameUser+"@gmail.com";

        cy.get(':nth-child(7) > a > .menu-text').click();

        cy.get('a[href="/manage_user_page.php"]').click();

        cy.get('a').contains('Create New Account').click();

        cy.get('#edit-user-form > .widget-box > .widget-header >
.widget-title').invoke('text').should('include', expectedText.testUser);

    })

    it('Click in Registration', { tags: '@smoke' }, () => {

        cy.get(':nth-child(4) > a > .menu-text').click();

        cy.get('.lead').invoke('text').should('include',
expectedText.register);

```

```
    })

    it('xpto1', { tags: '@smoke' }, () => {

        ....

    })

    it('xpto2', { tags: '@smoke' }, () => {

        ....

    })

    it('xpto3', { tags: '@smoke' }, () => {

        ....

    })

    it('xpto4', { tags: '@smoke' }, () => {

        ....

    })

    it('xpto5', { tags: '@smoke' }, () => {

        ....

    })

    it('xpto6', { tags: '@smoke' }, () => {

        ....

    })

    it('xpto7',{ tags: '@smoke' }, () => {

        ....

    })

    it('xpto8', { tags: '@smoke' }, () => {

        ....

    })

    it('xpto9', { tags: '@smoke' }, () => {

        ....

    })

    it('xpto10', { tags: '@smoke' }, () => {
```

```
    ....
  })

  it('xpto11', { tags: '@smoke' }, () => {

    ....

  })

  it('xpto12', { tags: '@smoke' }, () => {

    ....

  })

  it('xpto13', { tags: '@smoke' }, () => {

    ....

  })

  it('xpto14', { tags: '@smoke' }, () => {

    ....

  })

  it('xpto15', { tags: '@smoke' }, () => {

    ....

  })

  it('xpto16', { tags: '@smoke' }, () => {

    ....

  })

  it('xpto17', { tags: '@smoke' }, () => {

    ....

  })

  it('xpto18', { tags: '@smoke' }, () => {

    ....

  })

  it('xpto19', { tags: '@smoke' }, () => {

    ....

  })

  it('xpto20', { tags: '@smoke' }, () => {
```



```
    ....  
  
  })  
  
  })  
  },
```

Full prompt submitted to ChatGPT

Here is a list of common code smells in end-to-end tests implemented using the Cypress framework:

- * Incorrect use of asserts: using asserts (e.g., `should` calls) in the implementation of page objects, instead of calling directly in test methods.
- * Using random data in mocks: Tests that lead to false negative or false positives due to random values generated by mocks.

* Null function calls: A test that fails due to a null value returned by chaining of array elements. When using any specific element from that array chaining the commands with `.eq()`, `.first()`, `.last()`, etc. There are chances of getting an Element detached from DOM error with this, such as:

```
```js  
 cy.get('selector').eq(1).click()
```
```

* Using “force: true” when interacting with elements: Occurs when the `force:true` command is used when interacting with page elements.

* Using `cy.contains()` without selectors: A test that uses `cy.contains()` to select an element based on the text it contains, e.g., `cy.contains("please click here")`.

* Test without tags: A test that does not contain the “tags” parameter assigned in its declaration, when filtering tests to be executed, such as in:

```
```js  
 it('Description') , () => {

 }
```
```

* Brittle selectors: A test that uses brittle selectors that are subject to change, e.g., `cy.get(["customized id"])` instead of `cy.get(["random id"])`.

* Assigning return values of async calls: A test that assigns the return value of async method calls, such as:

```
```js  
 let x = cy.get('selector');
 x.click();
```

\*\*\*

- \* Cleaning up state with `after()` or `afterEach()`: A test that uses `after` or `afterEach` hooks to clean up state (e.g., to release resources acquired by the test).

- \* Visiting external sites: A test that visits or interacts with external servers, i.e., sites that we do not control.

- \* Starting Web servers: A test that starts a web server using `cy.exec()` or `cy.task()`.

- \* Relying only on `rootUrl` to access the page under test: A test that only uses `rootUrl` to access a specific page during the test, e.g., assuming that `xyz.com` redirects to `xyz.com/login`, which is the page that we need to test.

- \* Missing global `BaseUrl`: A test suite that calls `cy.visit()` without setting a `baseUrl`. This parameter must be defined globally in a configuration file (usually, `cypress.config.js`).

- \* Using unnecessary checks for previously actionable elements : A test that uses redundant visibility checks when dealing with actionable elements such as `should("be.visible")` or `should("exist")` before actions (click or type), such as in:

```
```js
  cy.get('selector').should('be.visible').click();
```
```

- \* Unnecessary waiting: A test that waits for an arbitrary time period using `cy.wait()`.

- \* Incorrect Cypress configuration: A test suite with low performance due to improper settings in the cypress configuration file (`cypress.json`), such as `port`, `timeout` or `video`.

- \* Duplicated code: When two or more tests share identical code snippets.

- \* Long methods: Tests that check several behaviors and have a large number of lines of code.

- \* Large classes: Classes with a very high number of end-to-end tests.

- \* Conditional overuse: Occurs when there is a condition in the test, i.e., due to a specific event the test may return different values.

Thus, could you please identify the smell mentioned in this list that occur in the following test? The test has 1 source code file: `totalTest.cy.js` . Each file is delimited using markdown code blocks. A comment in the first line of each file describes its name. Please, in your answer mention at most one smell.

```
```js
describe('Mantis test', () => {

  const expectedText = {
    taskExpect: "Operation performed successfully.",
  }
})
```
```

```

 textMenu: 'administrator',
 testUser: "Edit User",
 register: "No Change Log information available. Issues are
included once projects have versions and issues are resolved with \"fixed
in version\" set."

 };

 beforeEach(() => {
 const username = Cypress.config('username');
 const password = Cypress.config('password');

 cy.visit(Cypress.config('baseUrl') + Cypress.env('loginUrl'), {
failOnStatusCode: false });

 cy.get('[id="username"]').type(username);
 cy.get('input[type="submit"]').click();
 cy.get('[id="password"]').type(password);
 cy.get('input[type="submit"]').click();

 })

 it('Create Task', { tags: '@smoke' }, () => {

 cy.get(':nth-child(3) > a > .menu-text').click();

 cy.get('input[value="Selecionar Projeto"]').click();

 const uuid = () => Cypress._.random(0, 1e6)
 const id = uuid()
 const testname = `testname${id}`
 const testSummary = `summary${id}`

 const summary = testname
 const description = testSummary

 cy.get('[id="summary"]').type(summary);

 cy.get('[id="description"]').type(description);

 cy.get('input[value="Submit Issue"]').click();

 cy.get('.bold').invoke('text').should('include',
expectedText.taskExpect);

 })

 it('Create Project', { tags: '@smoke' }, () => {

 cy.get(':nth-child(7) > a > .menu-text').click();

 cy.get('a[href="/manage_proj_page.php"]').click();

 const name = "Project Name";
 const description = "Project Description";
 const nameP = name;

 cy.xpath('//button[@type="submit" and text()="Create New

```

```

Project"]').click();

 cy.get('[id="project-name"]').type(name);

 cy.get('[id="project-description"]').type(description);

 cy.xpath('//input[@type="submit"]').click();

 cy.get('a').invoke('text').should('include', nameP);
 })

 it('Login Test', { tags: '@smoke' }, () => {

 cy.get('.user-info').invoke('text').should('include',
expectedText.textMenu);

 });

 it('New User', { tags: '@smoke' }, () => {

 const username = Cypress.config('username');
 const password = Cypress.config('password');

 const uuid = () => Cypress._.random(0, 1e6)
 const id = uuid()

 const nameUser = `UserTestLarissaProjectAutomationProfile${id}` ;
 const nameReal = `Test Automation Larissa Project Profile${id}`;
 const emailUser = nameUser+"@gmail.com";

 cy.get(':nth-child(7) > a > .menu-text').click();

 cy.get('a[href="/manage_user_page.php"]').click();

 cy.get('a').contains('Create New Account').click();

 cy.get('#edit-user-form > .widget-box > .widget-header >
.widget-title').invoke('text').should('include', expectedText.testUser);

 })

 it('Click in Registration', { tags: '@smoke' }, () => {

 cy.get(':nth-child(4) > a > .menu-text').click();

 cy.get('.lead').invoke('text').should('include',
expectedText.register);

 })

 it('xpto1', { tags: '@smoke' }, () => {

 })

 it('xpto2', { tags: '@smoke' }, () => {

```

```

 })

 it('xpto3', { tags: '@smoke' }, () => {

 })

 it('xpto4', { tags: '@smoke' }, () => {

 })

 it('xpto5', { tags: '@smoke' }, () => {

 })

 it('xpto6', { tags: '@smoke' }, () => {

 })

 it('xpto7',{ tags: '@smoke' }, () => {

 })

 it('xpto8', { tags: '@smoke' }, () => {

 })

 it('xpto9', { tags: '@smoke' }, () => {

 })

 it('xpto10', { tags: '@smoke' }, () => {

 })

 it('xpto11', { tags: '@smoke' }, () => {

 })
```

```

 it('xpto12', { tags: '@smoke' }, () => {

 })

 it('xpto13', { tags: '@smoke' }, () => {

 })

 it('xpto14', { tags: '@smoke' }, () => {

 })

 it('xpto15', { tags: '@smoke' }, () => {

 })

 it('xpto16', { tags: '@smoke' }, () => {

 })

 it('xpto17', { tags: '@smoke' }, () => {

 })

 it('xpto18', { tags: '@smoke' }, () => {

 })

 it('xpto19', { tags: '@smoke' }, () => {

 })

 it('xpto20', { tags: '@smoke' }, () => {

 })
 })
},

```

**ChatGPT full answer**

### (1 attempt)

One code smell that occurs in the provided test is "Duplicated code."

This smell can be observed in the `beforeEach` hook, where the code for logging in is duplicated:

```
```js
beforeEach(() => {
  const username = Cypress.config('username');
  const password = Cypress.config('password');

  cy.visit(Cypress.config('baseUrl') +
Cypress.env('loginUrl'), { failOnStatusCode: false });

  cy.get('[id="username"]').type(username);
  cy.get('input[type="submit"]').click();
  cy.get('[id="password"]').type(password);
  cy.get('input[type="submit"]').click();
})
```
```

The login code block is repeated in each test case, resulting in redundant code. Duplicated code makes the test suite harder to maintain, as any changes or updates need to be made in multiple places, increasing the chances of introducing errors and inconsistencies.

To improve this, the login code should be extracted to a reusable function or placed in a Cypress custom command, so that it can be invoked from different test cases without duplication. This would enhance the maintainability and readability of the test suite.

### (2 attempt)

One smell that occurs in the given test is "Duplicated code."

There is duplicated code related to logging in at the beginning of each test case. The login process is repeated in the `beforeEach()` hook for each individual test. This duplication can be avoided by extracting the login logic into a reusable function and calling that function in the `beforeEach()` hook. By doing so, the test code will be more maintainable and easier to update if there are any changes to the login process.

To address this smell, you can create a custom login function and call it inside the `beforeEach()` hook, like this:

```
```javascript
// Custom login function
function login() {
  const username = Cypress.config('username');
  const password = Cypress.config('password');

  cy.visit(Cypress.config('baseUrl') + Cypress.env('loginUrl'), {
failOnStatusCode: false });
}
```

```

    cy.get('[id="username"]').type(username);
    cy.get('input[type="submit"]').click();
    cy.get('[id="password"]').type(password);
    cy.get('input[type="submit"]').click();
  }

  // Test suite
  describe('Mantis test', () => {
    const expectedText = {
      taskExpect: "Operation performed successfully."
    };

    beforeEach(() => {
      login(); // Call the custom login function here
    });

    // ... rest of the test cases ...
  });
  ...

```

By refactoring the code in this way, you remove the duplicated login code and ensure that the login process is consistently executed before each test, without cluttering the individual test cases with the same login steps.

(3 attempt)

One code smell that occurs in the provided test is "Long methods."

The test case "Create Task" contains a significant amount of code, including generating UUIDs, setting up test data, and performing multiple actions on the page. The test case includes 10 lines of code within it, making it a relatively long method. Long methods can make tests harder to read, understand, and maintain.

To address this smell, it's generally a good practice to break down tests into smaller, more focused units that test specific functionality. This can improve readability and make it easier to identify issues when tests fail. Additionally, breaking down tests can help in reusing common test code in multiple test cases, reducing code duplication.

(4 attempt)

One code smell that occurs in the given test is "Large classes." The test suite contains a large number of test cases (20 in total) within a single `describe` block. Having such a large number of test cases in a single test suite can make it difficult to manage, read, and maintain the code. It can also lead to longer execution times and increased complexity.

To improve the maintainability and readability of the test suite, it is recommended to break down the test cases into smaller, focused test suites, each covering specific functionalities or scenarios. This way, the test suite can be more organized, and it will be easier to identify and fix issues when they arise.