Prompt 18: Long methods

Smell: **Long methods**

End to End test code <u>without smell</u>

~~~
```js
// manageUser.cy.js

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

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

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

      const username = Cypress.config('username');
      const password = Cypress.config('password');
      const nameUser = "UserTestLarissaProjectAutomation" ;
      const nameReal = "Test Automation Larissa Project";
      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

// project.cy.js

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

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

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

      const name = "Project Name";

      const description = "Project Description";

      const nameP = name;

      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();


      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

// manageMark.cy.js


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

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

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

      const descriptionMark = "Test Description Automation Mark Larissa
Project";
      const tagName = "name Tag Larissa Project";

      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_tags_page.php"]').click();

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

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

      cy.get('input[name="config_set"]').click();


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

   })

})
```
~~~

~~~
```js

// manageCustomFields.cy.js

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

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

      const username = Cypress.config('username');
      const password = Cypress.config('password');
      const customField = "test Automation Larissa Project";

      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_custom_field_page.php"]').click();

      cy.get('input[class="input-sm"]').type(customField);

      cy.get('input[value="New Custom Field"]').click();

      cy.get('h4').invoke('text').should('include', 'Edit custom field
');

   })

})

```
~~~

~~~
```js

// manageGlobalProfiles.cy.js

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

   it('Manage Test', () => {

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

      const description = "Test Description Automation LARISSA Project";
      const platform = "Custom Platform";
      const so = "Windows";
      const versionOs = "14";


      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_prof_menu_page.php"]').click();

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

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

      cy.get('[id="os-version"]').type(versionOs);

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

      cy.get('input[value="Add Profile"]').click();
~~~

```
      cy.get('select').select(platform+ " "+ so + " "+versionOs);

   })

})
```
```

End to End test code with smell

// longMethods.js

````
```js
//File : manageTest.cy.js


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

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

   it('Manage Test', () => {

      const username = Cypress.config('username');
      const password = Cypress.config('password');
      const nameUser = "UserTestLarissaProjectAutomation" ;
      const nameReal = "Test Automation Larissa Project";
      const emailUser = nameUser+"@gmail.com";
      const name = "Project Name";
      const description = "Project Description";

      const descriptionMark = "Test Description Automation Mark Larissa 
Project";
      const tagName = "name Tag Larissa Project";
      const customField = "test Automation Larissa Project";
      const platform = "Custom Platform";
      const so = "Windows";
      const versionOs = "14";
      const nameP = name;

      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);


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


      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);

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

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

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

      cy.get('input[name="config_set"]').click();


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

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

      cy.get('input[class="input-sm"]').type(customField);

      cy.get('input[value="New Custom Field"]').click();

      cy.get('h4').invoke('text').should('include', 'Change Custom
Field');


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

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

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

      cy.get('[id="os-version"]').type(versionOs);

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

~~~
      cy.get('input[value="Add Profile"]').click();


      cy.get('select').select(platform+ " "+ so + " "+versionOs);
   })

})
```
~~~

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: manageTest.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
//File : manageTest.cy.js
```

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

   const expectedText = {
      textMenu: 'Alterar Usuário'
   };

   it('Manage Test', () => {

      const username = Cypress.config('username');
      const password = Cypress.config('password');
      const nameUser = "UserTestLarissaProjectAutomation" ;
      const nameReal = "Test Automation Larissa Project";
      const emailUser = nameUser+"@gmail.com";
      const name = "Project Name";
      const description = "Project Description";
      const descriptionMark = "Test Description Automation Mark Larissa
Project";
      const tagName = "name Tag Larissa Project";
      const customField = "test Automation Larissa Project";
      const platform = "Custom Platform";
      const so = "Windows";
      const versionOs = "14";
      const nameP = name;

      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('Criar nova conta').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="Criar Usuário"]').click();


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


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


      cy.xpath('//button[@type="submit" and text()="Criar Novo
Projeto"]').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);

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

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

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

      cy.get('input[name="config_set"]').click();


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

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

      cy.get('input[class="input-sm"]').type(customField);

      cy.get('input[value="New Custom Field"]').click();

      cy.get('h4').invoke('text').should('include', 'Alterar campo
personalizado');


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

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

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

      cy.get('[id="os-version"]').type(versionOs);

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

      cy.get('input[value="Add Profile"]').click();


      cy.get('select').select(platform+ " "+ so + " "+versionOs);
    })

})
```
~~~~

ChatGPT full answer

(1 attempt)

One code smell that occurs in the given test is:

Using `cy.wait()` for unnecessary waiting:

In the test, there is no clear reason why `cy.wait()` is used. It appears to be an arbitrary time-based wait, which is generally discouraged in automated tests. It can make the test flaky and unpredictable since it depends on the system's performance at runtime. This practice should be avoided as much as possible.

Here's the specific part of the code where `cy.wait()` is used:

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

cy.wait(3000); // <-- Unnecessary waiting

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

Instead of using `cy.wait()`, it is better to use Cypress commands that wait for specific conditions to be met or use explicit assertions to ensure the expected state before proceeding to the next steps.

For instance, you can use `.should('exist')` to wait for an element to be present before proceeding:

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

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

Please note that this is just one of the code smells that can be identified in the given test. There might be other code smells present as well, but this is the one I've identified from the provided code.

(2 attempt)

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

This smell is evident in the test at the following line:

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

In this line, the test is clicking on an element based on its position using the `:nth-child(7)` selector. However, relying on the position of an element in the DOM can lead to brittle tests. If the structure of the page changes, and the target element is no longer the 7th child, the test will fail.

To make this test more robust and maintainable, it is better to use a more reliable selector, such as an ID, class, or data attribute, that uniquely identifies the element being clicked.

As a general best practice, using meaningful selectors that are less likely to change with DOM modifications ensures that the test remains stable even if the page layout changes.

Additionally, some of the other code smells mentioned in the list might be present in the test, but only one smell was requested to be identified.

(3 attempt)

One code smell that occurs in the provided test is:

Unnecessary waiting: The test contains an unnecessary wait using `cy.wait()`. The test has a call to `cy.wait()` without any clear reason or context for the wait. Unnecessary waiting can slow down the test execution and make it less efficient. Here's the specific line with the smell:

~~~
```js
cy.wait(time in seconds);
```
~~~

This line should be reviewed to determine if the wait is necessary and if it serves a valid purpose in the test. If not, it should be removed to improve the test's performance.

(4 attempt)

One code smell that occurs in the given test is:

Using cy.contains() without selectors

In the test, there is a usage of `cy.contains()` without specifying any selectors to locate the element. Specifically, in the following line:

~~~
```js
cy.get('a').contains('Create new account').click();
```
~~~

`cy.contains('Create new account')` is used without providing a more specific selector to narrow down the search for the element. This can lead to potential issues if there are multiple elements containing the same text, and Cypress selects an unintended element.

To improve this, it's recommended to provide a more specific selector along with `cy.contains()` to ensure Cypress locates the desired element accurately. For example, you can add a class or an ID to the element and use it in combination with `cy.contains()`. Here's a hypothetical example:

~~~
```js
cy.get('a.some-class').contains('Create new account').click();
```
~~~

By providing a more specific selector, you reduce the risk of Cypress selecting unintended elements during the test execution.

(5 attempt)

One code smell that occurs in the provided test is **"Long methods"**. The test named "Manage Test" contains a significant number of lines of code, and it appears to be checking several behaviors related to managing users, projects, tags, custom fields, and profiles. Long methods can lead to decreased readability, maintainability, and understandability of the test code.

To address this code smell, the test could be refactored into smaller, more focused test cases, each targeting a specific functionality or scenario. Smaller, more focused tests are easier to read, understand, and maintain. Additionally, breaking the test into smaller parts would make it easier to identify the cause of any test failures, as the scope of each individual test case would be reduced.

For example, instead of having a single test case that covers user creation, project creation, tag creation, custom field creation, and profile addition, the test could be split into separate test cases for each of these functionalities, with descriptive names that clearly indicate what each test is responsible for. This approach follows the principle of "One test, one assertion" and helps to improve the test code quality.