# Where we are

You should have several tests by now. They have a high amount of duplication – the same locators and steps repeated over and over. Let’s fix that.

Before you begin refactoring, watch TA Approaches and Test Case Design videos for more background on what we will be doing.

Also watch Frameworks video, which contains a high-level explanation of framework structure, which we will implement next.

# Part 1 – Page Objects

The first thing we should do is implement the Page Object (PO) pattern. We do this by creating a class for each page that contains all information and logic related to that page:

1. Each web page (e.g. BbcNewsPage) should have a separate class with a descriptive name;
2. Move all IWebElements that you use in your tests to private (!) fields of these new classes;
3. Create public (or private if useful) methods that represent actions that you take on these pages. For example, to do a search on the news page, create a method Search(string toSearch) in the NewsPage class. All code that uses only one page should be separated and put in its PO;
4. Refactor all your tests to use these new POs;
5. Use PageFactory (PageFactory.InitElements(driver, page)) to initialize your POs. This can be called from its constructor. For this to work, all your private IWebElement fields need an attribute [FindsBy(How = How.Xpath, Using = “xpath”)];
6. Page Factory is a simple initialization method, which may not do all that you want it to. Implement your own initialization where this happens.

# Part 2 – Improve your Xpaths

Your Xpaths (especially if you simply copied them from Chrome) are likely quite long and you don’t understand what they do by looking at them. Let’s try a few things to fix that:

1. Xpaths for buttons can be shortened by using button text. You can even create a method that returns Xpath of element number X [since there are usually several elements with the same text] with a certain text, like GetTextXpath(String elementLabel, int elementNumber);
2. For all other Xpaths – try to find a unique attribute value somewhere close to the target element (or in the target element itself) that you can bind to. For example, top stories have a parent element with attribute data-entityid=’container-top-stories#X’, where X is the number of the story.

# Part 3 – add a class representing a complex element

1. Your objective is to create a class that represents a complex page element. This element will then be added to the relevant Page Object. The specific element is different for each task, detailed below;
2. For BBC1 and BBC2 tasks, there is a form involved: a set of controls that you use to input information and then press Submit (or similar button) to send the information to the server.

Xpaths for the form can usually be optimized and reused for other similar forms.

Your class will contain a method FillForm(Dictionary<string, string> values), which will fill this form (and in fact, any similarly implemented form) with values based only on fieldname-value pairs stored in the dictionary.

Of course, the simplest approach is to hardcode all XPaths of form elements. However, field name is always present on the page as an attribute or text value – if you base your XPath on that, you can dramatically reduce their number (for a text box form – to 1 Xpath).

1. For Lorem Ipsum tasks, the special element is the radio button group. Your class will only contain SetValue method, which will take desired value (bytes, words, etc) and click the appropriate radio button.

Simplest solution is to hard-code everything. Ideally though, your class will contain one XPath to find all radio buttons (based on root XPath that selects the entire group), and will find the text values of these buttons automatically. Such a class will not need to be modified if the radio button group changes.

1. For BBC3, you need to create a Scoreboard class. It should have a method GetScore that takes 2 team names as parameters, and returns score. This can be accomplished by using the team name in the Xpath (don’t forget that you can do “find me an element, which has a child that has a name, and then find its child that is a yellow block” quite simply with Xpath).

You will also need to make sure the two found scores are adjacent on the page. This can also be done with Xpath. It may be useful to remember that you can search for elements within a found IWebElement by using IWebElement.FindElement and starting your Xpath with a period (.) .

# Part 4 – add Business Logic layer

1. You will probably need just one Business Logic Layer (BLL) class;
2. Create methods that implements logical actions that a user can do with your part of functionality, for example: “submit a question to BBC”;
3. Move logic into these methods until your tests call \*only\* BLL methods, while BLL class calls creates PO instances and \*only\* calls PO methods;
4. Make sure POs never call or use other POs (with some rare exceptions, such as returning another PO’s instance in a method that switches to that page).

# Voila

We have a framework. 😊 As you can see, each test became much shorter than it was, and you can add new tests using methods from BL layer extremely quickly.