Design Patterns for QA Automation

Anton Semenchenko

Agenda, part 1 (general)

1. Main challenges
2. Solution
3. Design Patterns – the simplest definition
4. Design Patterns language – the simplest definition
5. Encapsulation – the most important OOP principle

Agenda, part 2 (main patterns)

1. Page Element
2. Page Object
3. Action

Agenda, part 3 (less popular patterns)

1. Flow (Fluent Interface)
   - Ubiquitous language
   - Key word driven
   - Behavior Driven Development (BDD)
2. Domain Specific Language (DSL)
   - Flow
3. Navigator (for Web)

Agenda, part 4 (take away points)

1. “Rules” and principles
2. A huge set of useful links
3. A huge set of examples

2 main challenges in our every day work (interview experience)

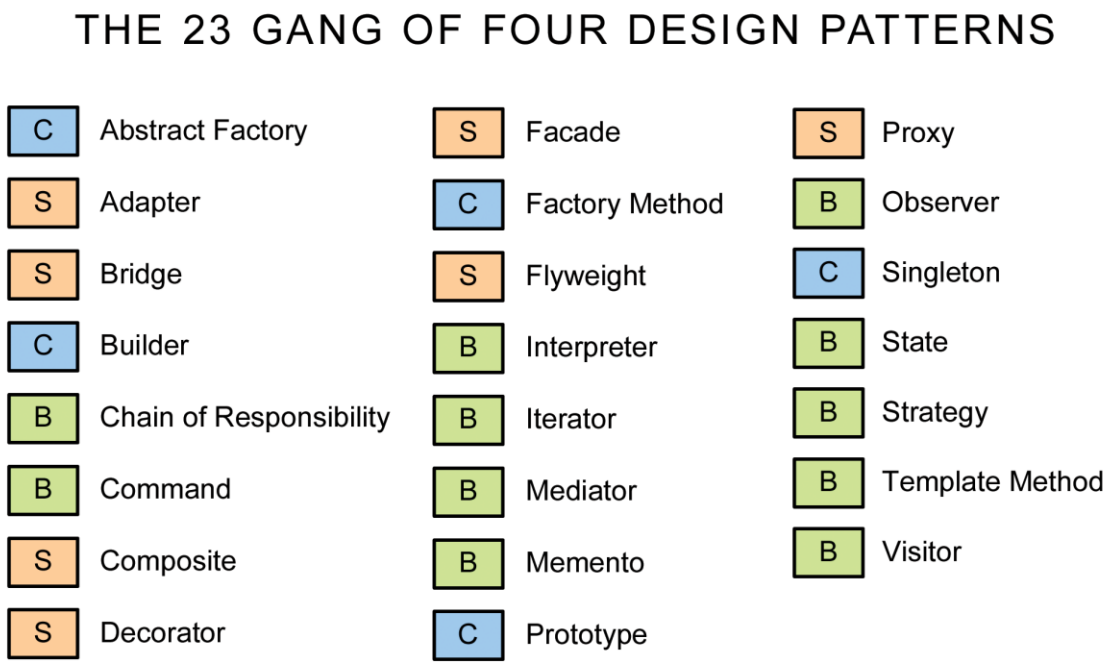
1. Pure design
2. Over design

Solution

1. Find a balance

Design Patterns – the simplest definition

1. Elements (blocks) of reusable object-oriented software;
2. The re-usable form of a solution to a design problem;



Design Patterns – as a language

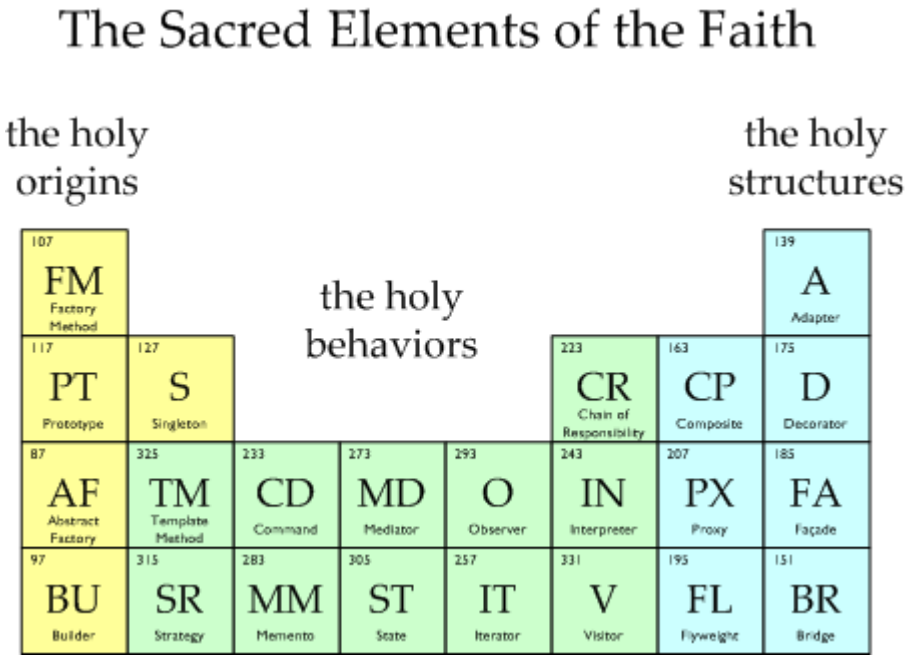
1. Design patterns that relate to a particular field (for example QA Automation) is called a pattern language

2. Design Patterns language gives a common terminology for discussing the situations specialists are faced with:

   – “The elements of this language are entities called patterns”;
   – “Each pattern describes a problem that occurs over and over again in our (QA Automation) environment”;
   – “Each pattern describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice!”

Design Patterns for QA Automation

1. Design patterns that relate to a QA Automation field:
   - Page Element
   - Page Object
   - Action
   - Flow
   - DSL
   - Navigator (for Web)

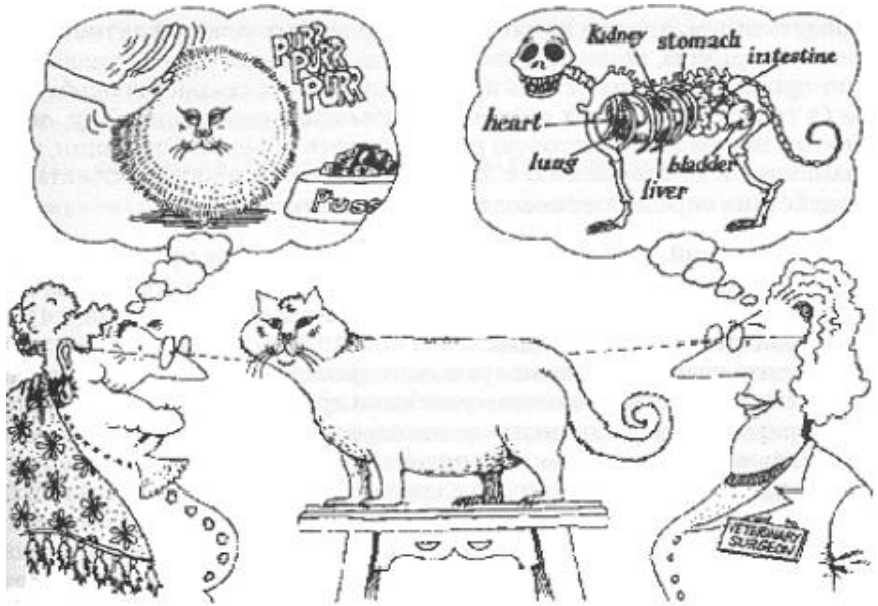


Encapsulation – the most important OOP principle

1. Ask yourself "how can I hide some details from the rest of the software?“
2. As with any encapsulation this yields two benefits (QA Automation, Page Object Design Pattern context):
   - store logic that manipulates the UI to a single place you can modify it there without affecting other components in the system;
   - it makes the client (test) code easier to understand because the logic there is about the intention of the test and not cluttered by UI details.

Page Element

1. Page Element – encapsulates “complexity” of UI element, canonical example – table as a part of UI.
2. Page Element – the simplest Page Object (Page Object with one and only one UI element)
3. Let’s focus on Page Objects and then return back to Page Element Design Pattern.



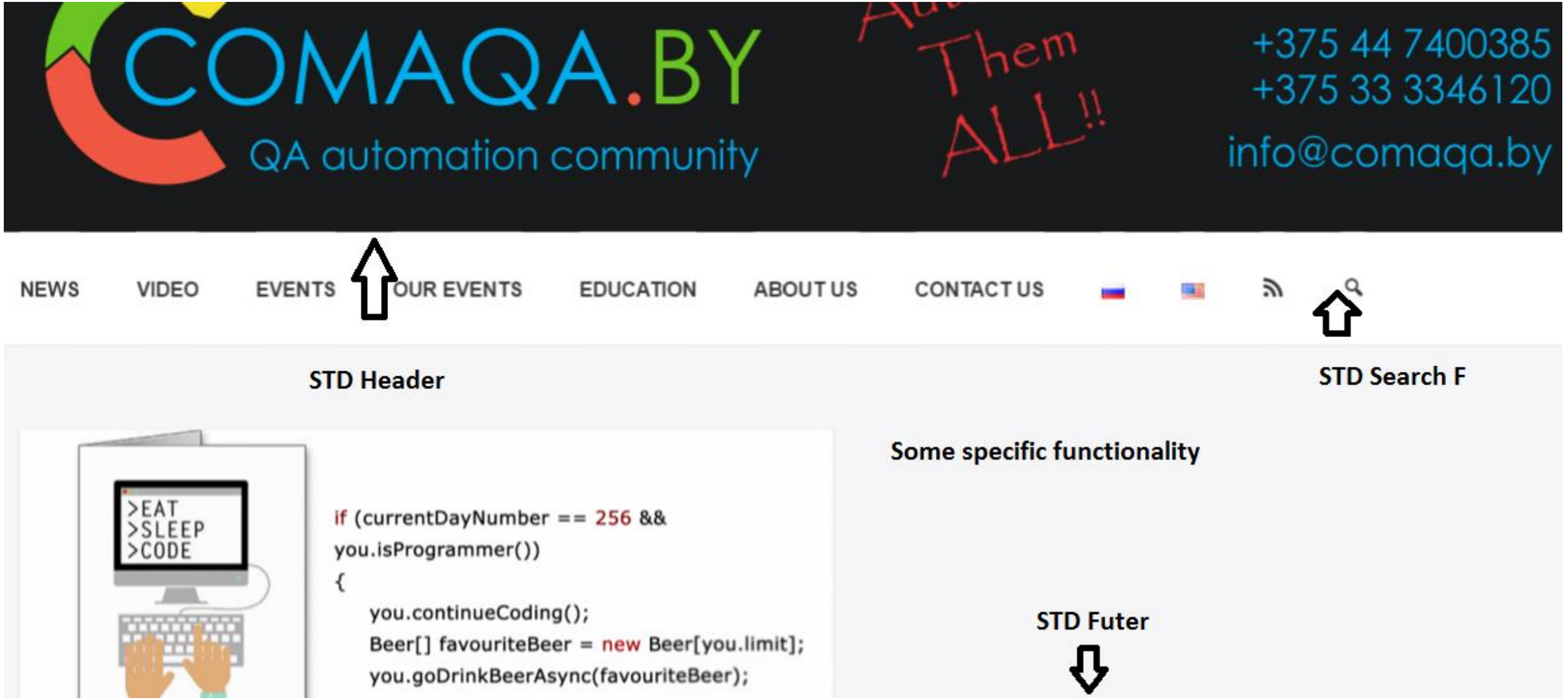
Page Object

1. Page Objects – encapsulates the way of identification and logical grouping of widgets.

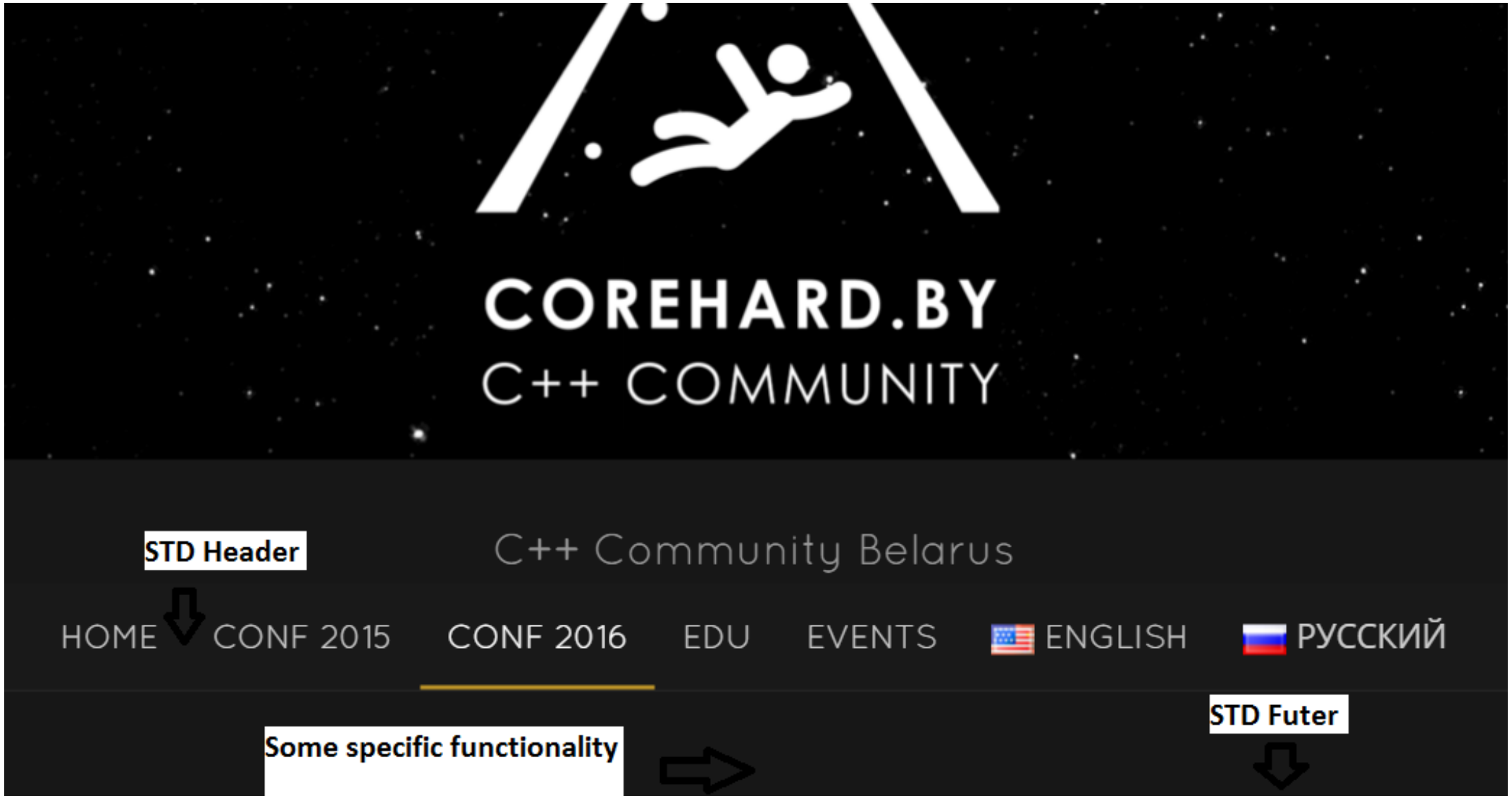
2. Page Object == Logical Page

3. Page Object != Physical Page

Page Object – classical example (state-less approach)



Page Object – classical example (state-less approach) ☺



2 ways (main, in 99% of cases) of re-usage any entity in OOP

1. Aggregation
2. Inheritance
   - Much more complicated then aggregation.
3. Summary:
   - prefer aggregation to inheritance in mooooost cases;
   - before start implementation based on inheritance, please, take a break for a minute, and re-thing everything again, possibly you can find a proper solution based on aggregation.

State-less or state-full solution?

1. Let’s compare:
   - Photo
     - Share – looks like parallelism (easy parallelism).
   - Video
     - Share – looks like parallelism (not trivial parallelism).

State-less or state-full solution?

1. How easy transform solution from “single” to “multi” threading (to decrease “QA Automation Windows”)

   – State-less – like share a photo
     • Just 5 minutes of work.
   – State-full – like share a video
     • Not trivial task, could be a night mare.

2. Summary

   – prefer state-less solutions to state-full solutions in mooooost cases;
   – before start implementation a state-full solution, please, take a break for a minute, and re-thing everything again, possibly you can find a proper state-less solution.

Object or static class \ State-less or state-full solution?

1. Static class
   - could be implemented as a state-less solution easily
2. Object
   - State-full solution in 99,99% cases
3. Summary
   - prefer static class based solutions (state-less) to object based (state-full) in mooooost cases;
   - before start implementation based on objects, please, take a break for a minute, and re-thing everything again, possibly you can find a proper solution based on static classes.

Page Object – the simplest implementation

1. Static classes based

2. State-less

3. Summary:

   – Such an implementation (the simplest one, state-less) – is a proper one in most cases;
   – You can find dozens of examples in this presentation.

UI Map

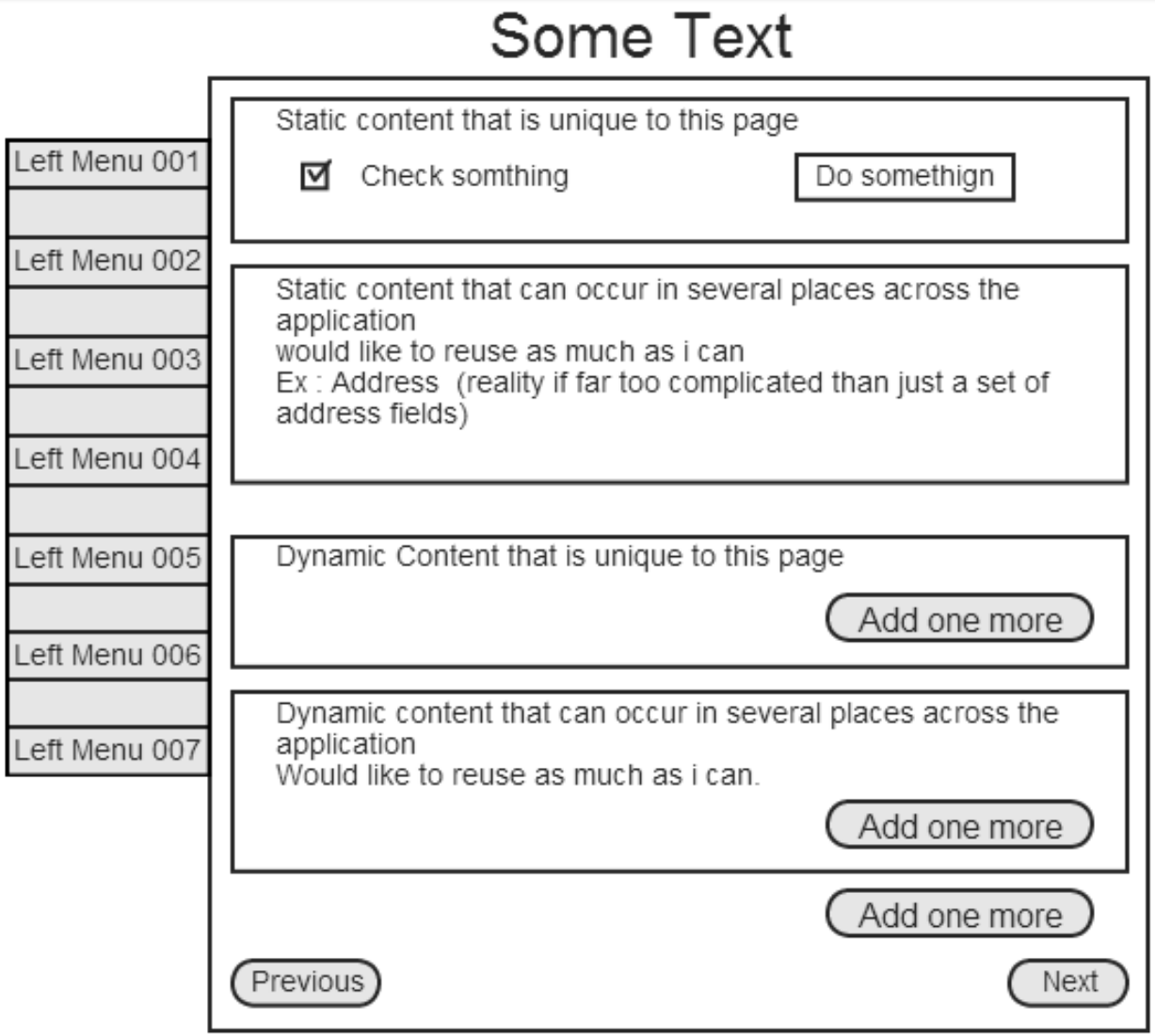
1. UI Map – one entry point

   – One entry point, tree of Page Objects;
   – One entry point, tree of locators.

Page Objects – state-full, special cases

1. Web UI that behaves like a Wizard
2. Web UI in combination with Mobile in one use case
3. Internet of Things (in most cases)
4. More then 1 page during 1 test (for example several portals or several instances of one portal to implement one “business use case”):
   - Really seldom;
   - Looks like integration tests (in most cases):
     - Std solution- some type of White Box Testing.
5. Many others “special cases”

Page Object – special case example (state-full approach)



Page Object by Martin Fowler

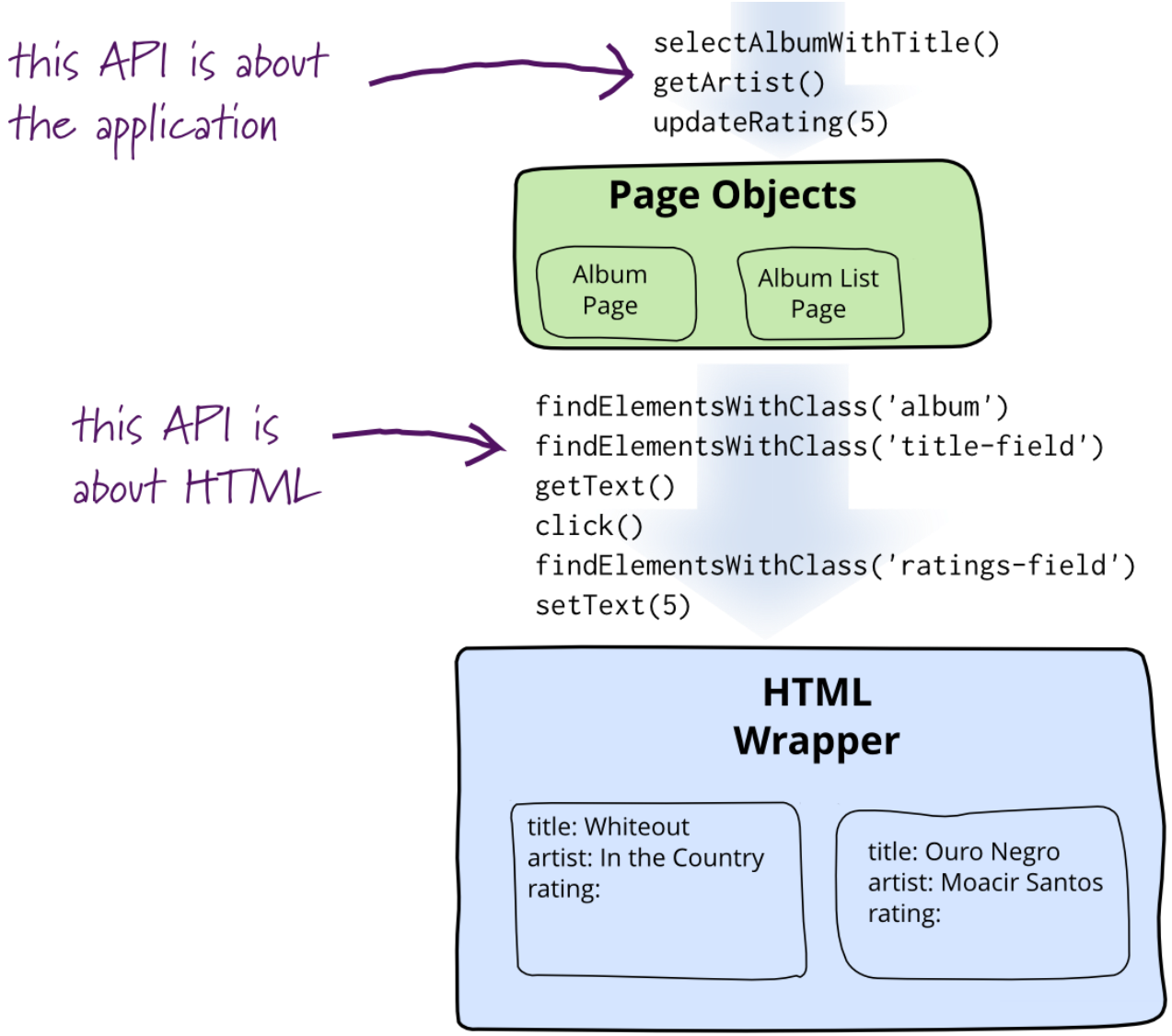
1. “Page objects are a classic example of encapsulation - they hide the details of the UI structure and widgetry from other components (the tests).”

   – “store logic that manipulates the UI to a single place you can modify it there without affecting other layers in the QA Automation solution (architecture)”;
   – “it makes the test code easier to understand because the logic there is about the intention of the test (focus on business logic) and not cluttered by UI details”.

Page Object by Martin Fowler, “general” rules

1. “When you write tests against a web page, you need to refer to elements within that web page in order to click links and determine what's displayed.”
2. “However, if you write tests that manipulate the HTML elements directly your tests will be brittle to changes in the UI.”
3. “A page object wraps an HTML page, **or** fragment, with an application-specific API, allowing you to manipulate page elements without digging around in the HTML.”
4. ”Page Object should allow a software client to do anything and see anything that a human can.”
5. ”Page Object should also provide an interface that's easy to program to and hides the underlying widgetry in the window.”

Page Object by Martin Fowler



Page Object by Martin Fowler, “encapsulation” rule

1. “The page object should **encapsulate** the mechanics required to find and manipulate the data in the UI control itself”
2. “**Changing the concrete control - the page object interface shouldn't change**.”
3. “A page object **wraps** an HTML page, **or** fragment, with an application-specific API, **encapsulate** a way of page elements manipulation (without digging around in the HTML).”
4. ”A page object should also provide an interface that's easy to program to and **hides** the underlying widgetry in the window.”

Page Object by Martin Fowler, “logical page” rule

1. “A page object wraps an HTML page, **or fragment**, with an **application-specific API**, allowing you to manipulate page elements without digging around in the HTML.”
2. “Despite the term "page" object, these objects shouldn't usually be built for each page, but rather **for the significant elements on a page**”
3. “A **header page object** and a **footer page object** – canonical examples.”

Page Object by Martin Fowler, “hierarchy of a complex UI” rule

1. “Model the structure in the page that **makes sense to the user of the application**.”
2. ”Page Object should allow a software client to do anything and see anything that a human can.”
3. “Some of the hierarchy of a complex UI is only there in order to structure the UI - such **composite structures shouldn't be “showed” by the page objects**.”

Page Object by Martin Fowler, “should return” rule

1. “To access a text field you should have **accessor methods that take and return** a string, check boxes should use booleans, and buttons should be represented by action oriented method names.”

2. “Page object operations should return **fundamental types** (strings, dates) or **other page objects**.”

3. “If you navigate to another page, the initial page object should return **another page object** for the new page.”

Page Object by Martin Fowler, “assertion” rule

1. “There are differences of opinion on whether page objects should include assertions themselves, or just provide data for test scripts to do the assertions.”
2. “Advocates of including assertions in page objects say that this helps avoid duplication of assertions in test scripts, makes it easier to provide better error messages, and supports a more TellDontAsk style API.”
3. Asserts in Page Objects increase QA Automation window dramatically.

Page Object by Martin Fowler, “assertion” rule

1. “Advocates of assertion-free page objects say that **including assertions mixes the responsibilities of providing access to page data with assertion logic, and leads to a bloated page object**.”
2. **“I favor having no assertions in page objects.”**
3. “I think you can avoid duplication by **using assertion libraries** (there is a huge set such a frameworks) for common assertions - which can also make it easier to provide good diagnostics.”

Page Object - notes

1. We've described this pattern in terms of HTML, but the same pattern applies equally well to any UI technology. I've seen this pattern used effectively to hide the details of a Java swing UI and I've no doubt it's been widely used with just about every other UI framework out there too.

2. Patterns that aim to move logic out of UI elements (such as Presentation Model, Supervising Controller, and Passive View) make it less useful to test through the UI and thus reduce the need for page objects.

Page Object – alternative areas of usage

1. Page objects are most commonly used in testing, but can also be used to provide a scripting interface on top of an application.

2. It's best to put a scripting interface underneath the UI, that's usually less complicated and faster.

3. However with an application that's put too much behavior into the UI then using page objects may make the best of a bad job. (But look to move that logic if you can, it will be better both for scripting and the long term health of the UI.)

Page Element

1. Page Element – encapsulates “complexity” of UI element, canonical example – table as a part of UI.
2. Page Element – the simplest Page Object (Page Object with one and only one UI element)

Column 1	Column 2	Column 3
Some value		
	Some value	

Action

1. Action – a set (tiny or huge) of lines of code based on “primitive” \ “low level” API (for example Selenium or some wrapper) calls.

2. Action is usually used in a combination with Page Element and Page Object Design Patterns.

3. Action layer could be separated from, combined with Page Objects layer … or even both approached in one solution.

Action

1. 2 “types” of Actions:
   - QA Automation specialist oriented;
   - Business (~Product Owner) oriented;

2. Action – isn’t a right place for asserts in mooooooooost cases:
   - There is no sense to check the same functionality in the same build dozens of times;
   - Such an approach seriously increase QA Automation windows;

Action

1. QA Automation specialist oriented Action can contain just several lines of code, to simplify manipulations with Widgets.

2. Business oriented Action can be a “copy” of some test (without all asserts).

3. In general Action layer could be implemented ether as a classical API or as a DSL\Flow based API.

Flow – Fluent Interface, “logical chain”

1. Ubiquitous language
   - Domain model
   - Domain driven design (DDD)
   - In fact – really-really useful, general purpose practice, part of fully implemented Agile process
2. Key word driven QA Automation
3. Behavior Driven Development (BDD) approach – as a special case of DSL based QA Automation solutions

Flow – Fluent Interface, “logical chain”

1. Domain specific language (DSL) based QA Automation
2. Flow – as a way of implementation DSL\BDD
3. State-full solution

Flow by wiki

1. Flow- **fluent interface** is an implementation of an object oriented API that aims to provide more readable code.

2. A fluent interface is normally implemented by using method cascading (concretely method chaining) to relay the instruction context of a subsequent call (but a fluent interface entails more than just method chaining).

3. Generally, the context is defined through the return value of a called method self-referential, where the new context is “equivalent” to the last context terminated through the return of a void context.

Flow – an abstract example

```
LoginPage.Instance().Navigate()
                    .Login()
                    .Search("some entity")
                    .ClickImages()
                    .SetSize(Sizes.Large)
                    .SetColor(Colors.BlackWhite)
                    .SetTypes(Types.Clipart)
                    .SetPeople(People.All)
                    .SetDate(Dates.PastYear)
                    .SetLicense(Licenses.All);
```

Flow by Martin Fowler

1. “Building a fluent API like this leads to some unusual API habits.”
2. “One of the most obvious ones are **setters that return a value**.”
3. “The common convention in the curly brace world is that modifier methods are void, which I like because it follows the principle of CommandQuerySeparation. This convention does get in the way of a fluent interface, so I'm inclined to suspend the convention for this case.”
4. “You should choose your return type based on what you need to continue fluent action.”
5. “The key test of fluency, for us, is the Domain Specific Language quality. The more the use of the API has that language like flow, the more fluent it is.”

DSL – Domain Specific Language

1. DSL = Domain (ether technical or business … or both – Gherkin for specific domain) + Language
2. Language = Dictionary + Structure
3. Dictionary = Ubiquitous language
4. Structure = some rules how to combine words (business terms) from dictionary in a proper ways (based on business logic)
5. Way of implementation (one of the ways) – Flow Design Pattern

DSL by Martin Fowler

1. The basic idea of a domain specific language (DSL) is a computer language that's targeted to a particular kind of problem (QA Automation or even QA Automation in exact domain), rather than a general purpose language that's aimed at any kind of software problem. Domain specific languages have been talked about, and used for almost as long as computing has been done.

2. DSLs are very common in computing: examples include CSS, regular expressions, make, rake, ant, SQL, HQL, many bits of Rails, expectations in JMock …

3. It's common to write tests using some form of DomainSpecificLanguage, such as Cucumber or an internal DSL. If you do this it's best to layer the testing DSL over the page objects so that you have a parser that translates DSL statements into calls on the page object.

DSL types by Martin Fowler

1. **Internal DSLs** are particular ways of using a host language to give the host language the feel of a particular language. This approach has recently been popularized by the Ruby community although it's had a long heritage in other languages - in particular Lisp. Although it's usually easier in low-ceremony languages like that, **you can do effective internal DSLs in more mainstream languages like Java and C#**. Internal DSLs are also referred to as embedded DSLs or FluentInterfaces

2. **External DSLs** have their own custom syntax and you write a full parser to process them. There is a very strong tradition of doing this in the **Unix community**. **Many XML configurations have ended up as external DSLs**, although XML's syntax is badly suited to this purpose.

3. Mixed (internal with external)

4. Graphical DSLs requires a tool along the lines of a Language Workbench.

Navigator (for Web)

1. Navigator (for Web) – follows “DRY” and “Single source of truth” principles, encapsulates “complexity” of links \ transitions between web pages and store this information in one and only one place.

2. Usually:

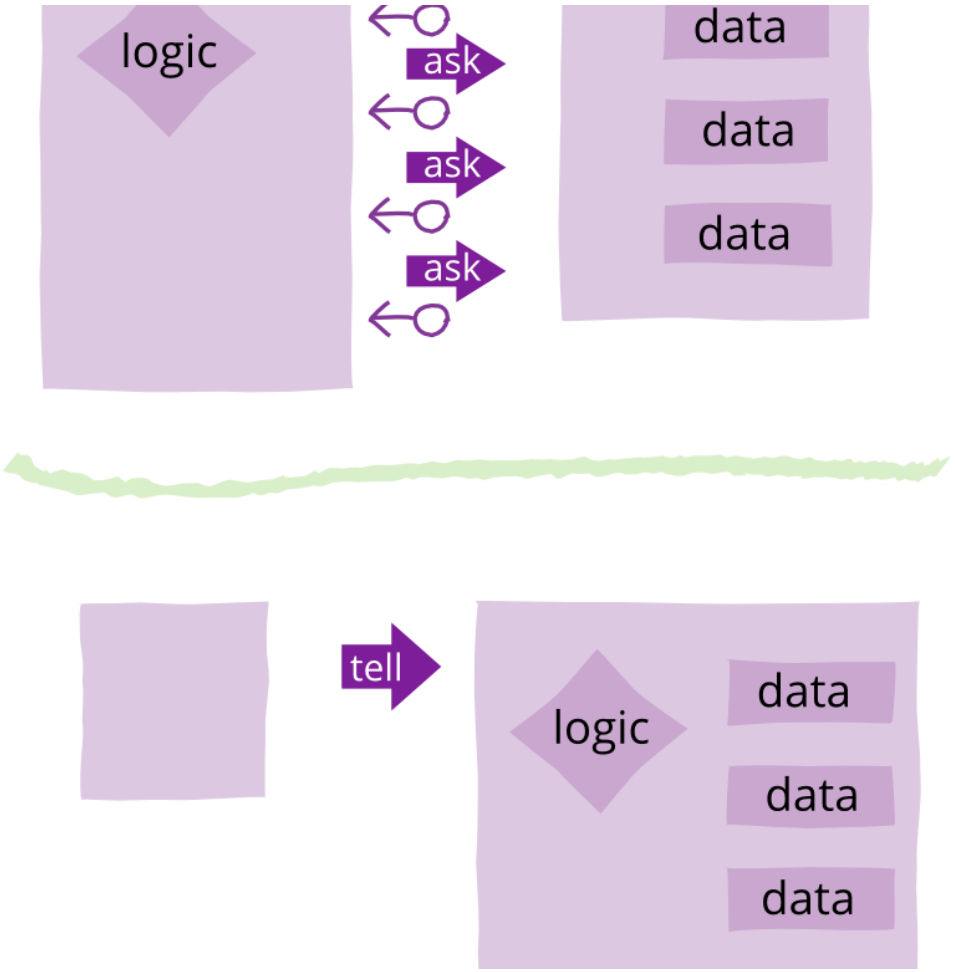
   - works in a combination with a “Page Object” and “Action” Design Patterns for QA Automation;
   - “Action” layer implements via Flow or DSL approaches;
   - state-full approach;
   - applies for really big projects.

3 “main” principles \ “rules” ☺

1. *“If you have WebDriver APIs in your test methods, You're Doing It Wrong.” - Simon Stewart*
2. Don't repeat yourself (DRY): “*Every piece of knowledge must have a single, unambiguous, authoritative representation within a system*” - Andy Hunt and Dave Thomas in their book *The Pragmatic Programmer*
3. “Broken windows theory”

“Tell-Don't-Ask” principle

This is a principle that helps people remember that object-orientation is about bundling data with the functions that operate on that data. It reminds us that rather than asking an object for data and acting on that data, we should instead tell an object what to do. This encourages to move behavior into an object to go with the data.



Useful “principles”

1. “Test-driven development”
2. “Single responsibility principle”
3. “Single source of truth”
4. “Interface segregation principle”
5. “Occam's razor”
6. “Poka-yoke”

Project A

Protected network monitoring system

- *Category:* Web application
- *Technologies:* HTML, CSS, Javascript, JQuery
- *Stage*: Updating the functionality of the application
- *Automation scope*: Performance testing, functional testing

Challenge

- Existing solution unable to fully test functionality and performance of the highly secure app
- Provide suggestions for performance improvement of highly loaded application
- Work with image recognition tests

Project A

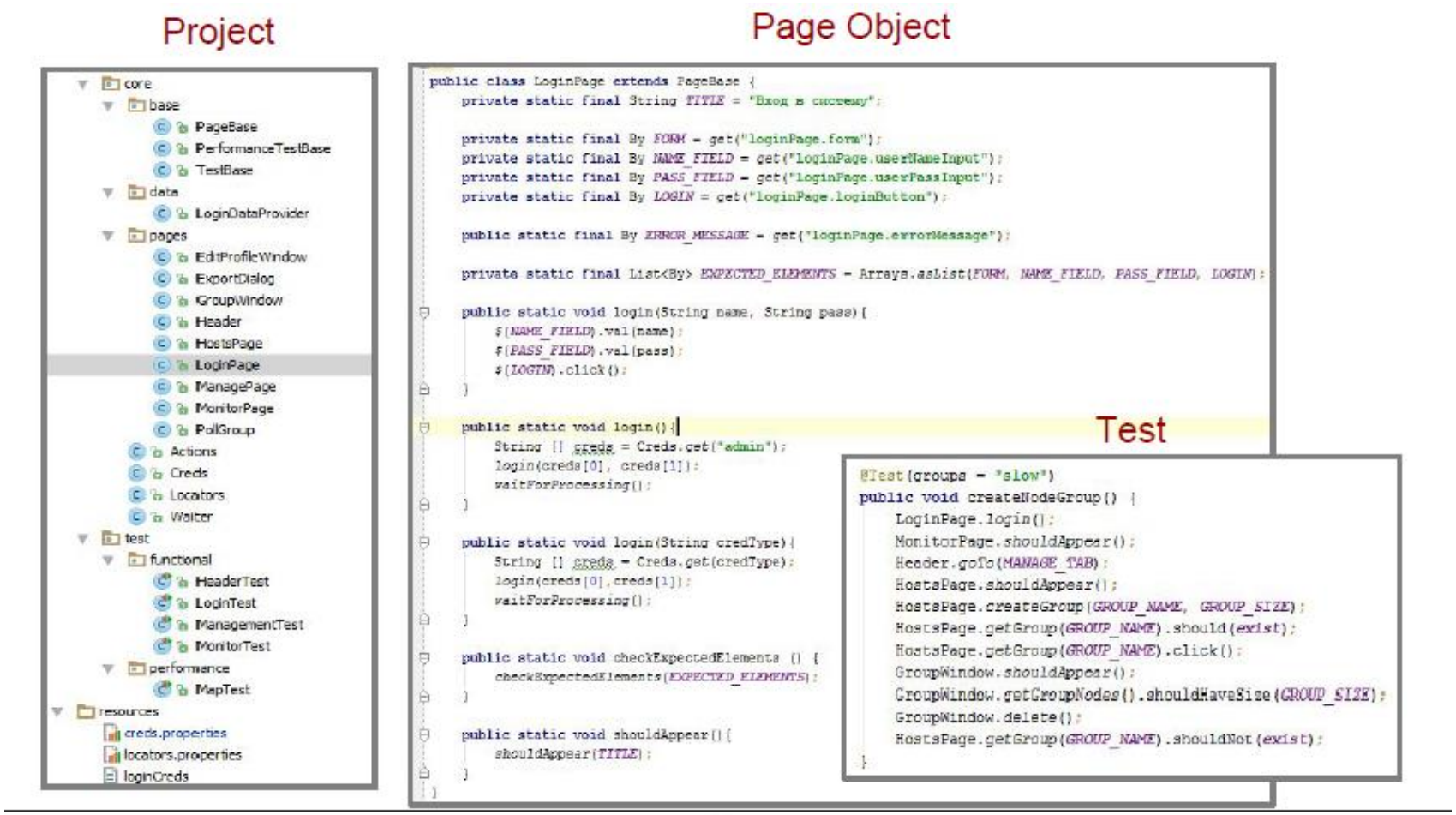
Solution

- *Performance*. BrowserMob Proxy allowed to use the same architecture and similar scenarios to test performance as well as functionality
- This stack also provided the opportunity to generate JMeter scenarios from client-side performance tests for free
- *Suggestions*. HAR-storage together with BrowserMob Proxy provided easy-to-interpret statistics and data for developing suggestions for performance improvement
- *Image recognition*. Sikuli-API perfectly solving the problem

Automation technology stack

- Selenium WebDriver (Java bindings)
- Selenide
- Sikuli-API
- BrowserMob Proxy
- HAR-Storage
- JMeter

Project A


Project
core
base
PageBase
PerformanceTestBase
TestBase
data
LoginDataProvider
pages
EditProfileWindow
ExportDialog
GroupWindow
Header
HostsPage
LoginPage
ManagePage
MonitorPage
PollGroup
Actions
Creds
Locators
Waiter
test
functional
HeaderTest
LoginTest
ManagementTest
MonitorTest
performance
MapTest
resources
creds.properties
locators.properties
loginCreds
Page Object
public class LoginPage extends PageBase {
private static final String TITLE = "Вход в систему";
private static final By FORM = get("loginPage.form");
private static final By NAME_FIELD = get("loginPage.userNameInput");
private static final By PASS_FIELD = get("loginPage.userPassInput");
private static final By LOGIN = get("loginPage.loginButton");
public static final By ERROR_MESSAGE = get("loginPage.errorMessage");
private static final List<By> EXPECTED_ELEMENTS = Arrays.asList(FORM, NAME_FIELD, PASS_FIELD, LOGIN);
public static void login(String name, String pass){
$(NAME_FIELD).val(name);
$(PASS_FIELD).val(pass);
$(LOGIN).click();
}
public static void login(){
String [] creds = Creds.get("admin");
login(creds[0], creds[1]);
waitForProcessing();
}
public static void login(String credType){
String [] creds = Creds.get(credType);
login(creds[0],creds[1]);
waitForProcessing();
}
public static void checkExpectedElements () {
checkExpectedElements(EXPECTED_ELEMENTS);
}
public static void shouldAppear(){
shouldAppear(TITLE);
}
}
Test
@Test(groups = "slow")
public void createNodeGroup() {
LoginPage.login();
MonitorPage.shouldAppear();
Header.goTo(MANAGE_TAB);
HostsPage.shouldAppear();
HostsPage.createGroup(GROUP_NAME, GROUP_SIZE);
HostsPage.getGroup(GROUP_NAME).should(exist);
HostsPage.getGroup(GROUP_NAME).click();
GroupWindow.shouldAppear();
GroupWindow.getGroupNodes().shouldHaveSize(GROUP_SIZE);
GroupWindow.delete();
HostsPage.getGroup(GROUP_NAME).shouldNot(exist);
}

Project B

Distributed networked desktop application

- *Category:* Medical data processing for healthcare centers
- *Technologies:* Windows forms, C#, MS SQL
- *Stage:* Fully functional application with new features in development
- *Automation scope:* Functional testing to reduce the volume of manual testing

Challenge

- Introduce automation under high security restrictions with no access to source code or environment settings
- Only remote access to the machine with application with no ability to install any automation software

Project B

Solution

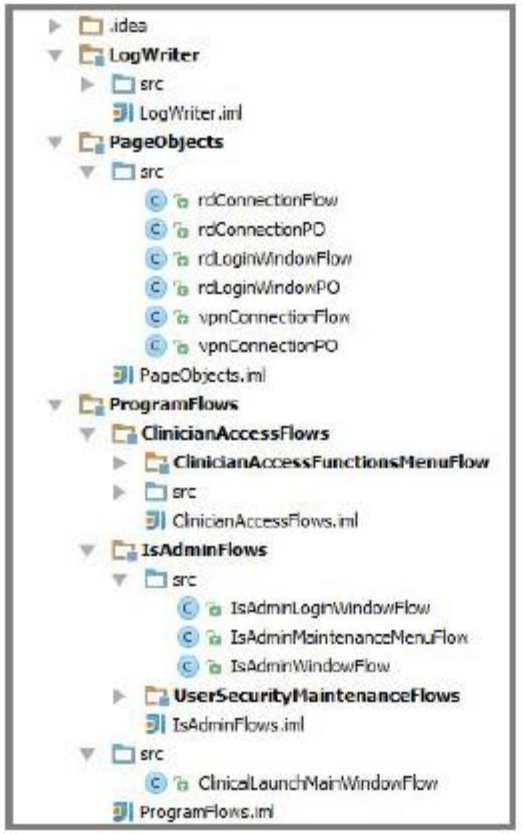
- *No source code:* using the source/technology-independent image-based automation tool – Sikuli Script to automate over 200 tests
- *Remote access:* Sikuli Script also allowed to work with the application remotely without any additional installations

Automation technology stack

- Sikuli Script (Java)

Project B

Project



Page Object

```
    public Region rdcLoginWindow() throws FindFailed
    {
        _rdcLoginWindow = _screen.wait(_rdcLogin, 15);

        return _rdcLoginWindow;
    }

    public Region rdcUserName() throws FindFailed
    {
        return _rdcLoginWindow.find(_rdcUsername);
    }

    public Region rdcPassword() throws FindFailed
    {
        return _rdcLoginWindow.find(_rdcPassword);
    }

    public Region rdcLogontoDropdown() throws FindFailed
    {
        return _rdcLoginWindow.find(_rdcLogontoDropdown);
    }

    public Region rdcLogontoRemote() throws FindFailed
    {
        return _rdcLoginWindow.find(_rdcLogontoSelect);
    }

    public Region rdcOkButton() throws FindFailed
    {
        return _rdcLoginWindow.find(_rdcOkButton);
    }
}
```

Test

```
clinicalLaunch
        .StartIsAdmin()
        .login("tech","nutshell")
        .gotoMaintenanceMenu()
        .gotoUserSecurityMaintenance()
        .gotoUserSpecificTab()
        .enterUsername("tech")
        .gotoClinicianAccessTab()
        .findCode0307000()
        .setCodeToNo();
```

Project C

Web application

- *Category:* Staff management application
- *Technologies*: ASP.NET MVC 4, C#, JQuery.js, Angular.js, Bootstrap.js
- *Stage:* Upgrading web application to use latest technologies
- *Automation scope:* Functional testing

Challenge

- Create effective and scalable E2E tests
- Lack of QA resources

Project C

Solution

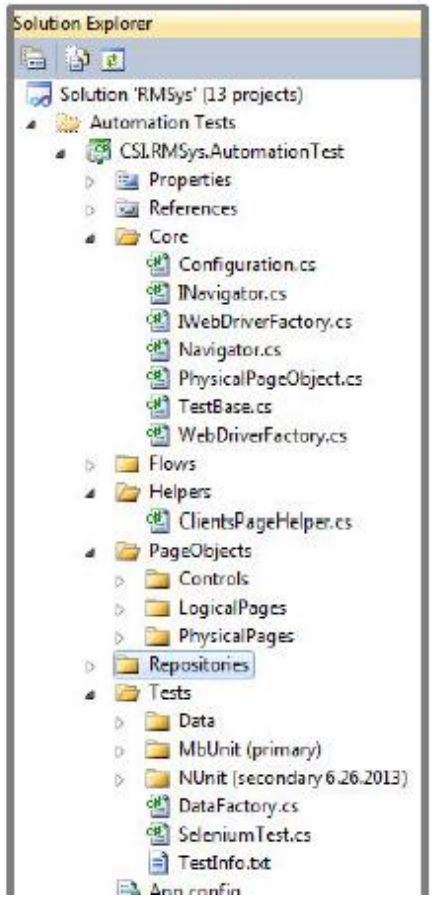
- *E2E tests:* more than 2 hundred tests run in 3 most popular browsers
- *Scalability:* initially local execution then moved to Team City and Selenium Grid for distributed execution

Automation technology stack

- Selenium Webdriver (C# bindings)
- Selenium Grid
- Nunit as test framework
- TFS as source control
- JQuery API (dealing with AJAX)
- Team City

Project C

Project



Page Object

```
using CSI.RMSys.AutomationTest.PageObjects.Controls;
using CSI.RMSys.AutomationTest.PageObjects.LogicalPages;
using OpenQA.Selenium;

namespace CSI.RMSys.AutomationTest.PageObjects.Pages
{
    public class HomePPO: LayoutPPO, IHomeLPO
    {
        public HomePPO() : base("Home/Index") { }

        private Button m_clearSearch;
        public Button ClearSearch
        {
            get { return m_clearSearch ?? (m_clearSearch = new Button(Driver, By.ClassName("global-search-deleteicon"))); }
        }

        private Button m_search;
        public Button Search
        {
            get { return m_search ?? (m_search = new Button(Driver, By.ClassName("search-button"))); }
        }

        private Button m_managementUsa;
        public Button ManagementUsa
```

Test

```
[Test, TestCaseSource(typeof(DataFactory), cDesiredCapabilitiesFactoryName)]
public void LoginAndCheckWelcomeText(DesiredCapabilities browser)
{
    using (var start = GetStart(browser))
    {
        start
            .LoginAndGoToHomePage()
            .AssertWelcomeText("Welcome, test_guest")
            .AssertTextIsPresent("Please, use navigate menu to  work");
    }
```

Project D

Cross-platform mobile application

- *Category:* Healthcare
- *Technologies:* PhoneGap, Sencha Touch
- *Stage:* Initial development
- *Automation scope:* Functional testing

Challenge

- Create cross-platform tests (IOS and Android)
- Tests that are easy to tweak and support
- Ability to execute on real devices

Project D

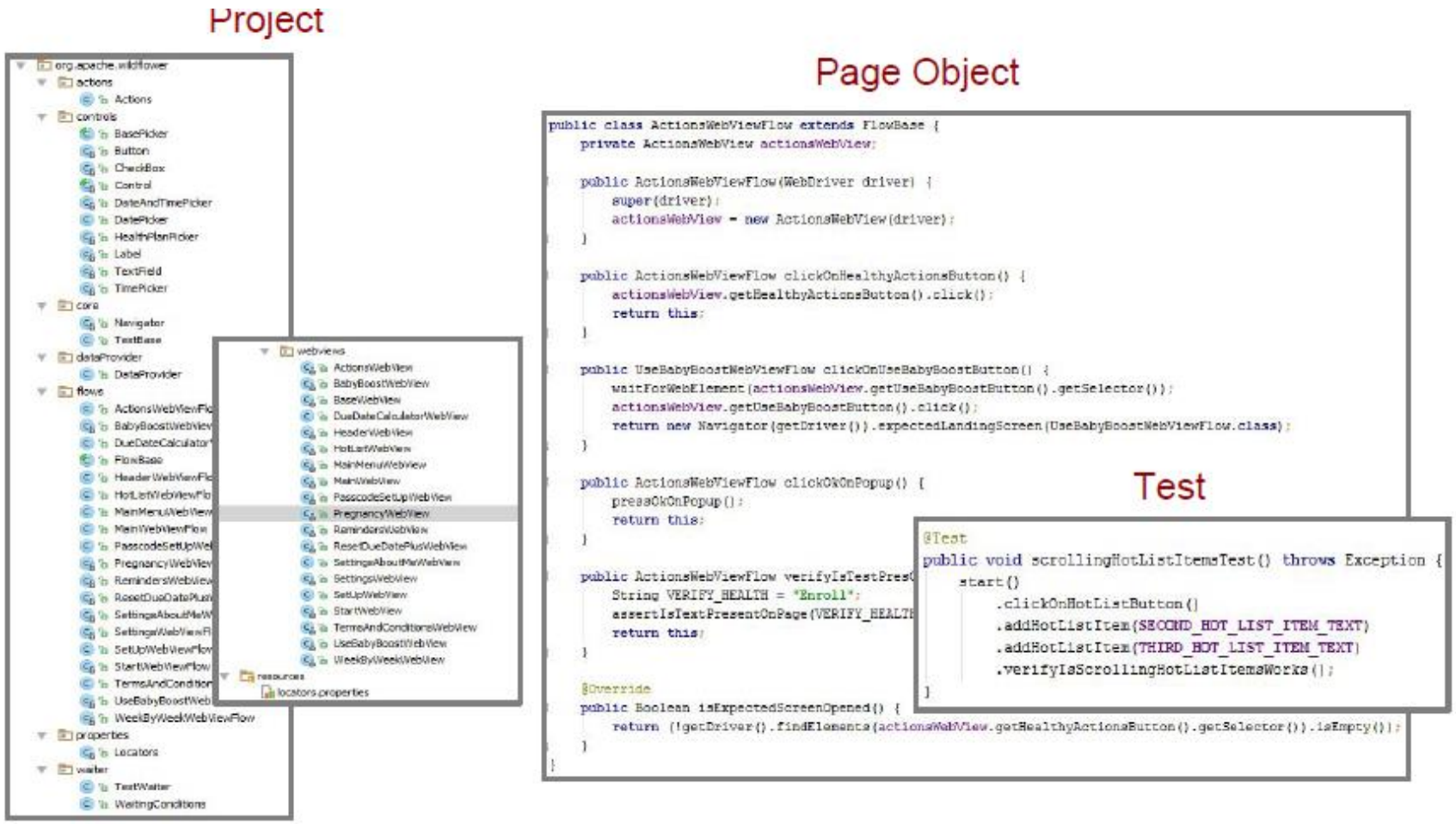
Solution

- *Cross-platform:* more than 80 tests that are run both on IOS and Android app
- *Maintainability:* using UI Map, static Page Object design patters for better readability and support
- *Devices:* Appium + Selenium Grid with real devices

Automation technology stack

- Appium (Java bindings) as automation tool
- TestNG as test framework
- Git as source control
- Maven as build tool
- Jenkins as CI tool
- Selenium Grid for distributed environment and real devices

Project D


Project
org.apache.wildflower
actions
Actions
controls
BasePicker
Button
CheckBox
Control
DateAndTimePicker
DatePicker
HealthPlanPicker
Label
TextField
TimePicker
core
Navigator
TestBase
dataProvider
DataProvider
flows
ActionsWebViewFlo
BabyBoostWebView
DueDateCalculator
FlowBase
HeaderWebViewFlo
HotListWebViewFlo
MainMenuWebView
MainWebViewFlow
PasscodeSetUpWeb
PregnancyWebView
RemindersWebView
ResetDueDatePlus
SettingsAboutMeW
SettingsWebViewFl
SetUpWebViewFlow
StartWebViewFlow
TermsAndCondition
UseBabyBoostWeb
WeekByWeekWebViewFlow
properties
Locators
waiter
TestWaiter
WaitingConditions
webviews
ActionsWebView
BabyBoostWebView
BaseWebView
DueDateCalculatorWebView
HeaderWebView
HotListWebView
MainMenuWebView
MainWebView
PasscodeSetUpWebView
PregnancyWebView
RemindersWebView
ResetDueDatePlusWebView
SettingsAboutMeWebView
SettingsWebView
SetUpWebView
StartWebView
TermsAndConditionsWebView
UseBabyBoostWebView
WeekByWeekWebView
resources
locators.properties
Page Object
public class ActionsWebViewFlow extends FlowBase {
private ActionsWebView actionsWebView;
public ActionsWebViewFlow(WebDriver driver) {
super(driver);
actionsWebView = new ActionsWebView(driver);
}
public ActionsWebViewFlow clickOnHealthyActionsButton() {
actionsWebView.getHealthyActionsButton().click();
return this;
}
public UseBabyBoostWebViewFlow clickOnUseBabyBoostButton() {
waitForWebElement(actionsWebView.getUseBabyBoostButton().getSelector());
actionsWebView.getUseBabyBoostButton().click();
return new Navigator(getDriver()).expectedLandingScreen(UseBabyBoostWebViewFlow.class);
}
public ActionsWebViewFlow clickOkOnPopup() {
pressOkOnPopup();
return this;
}
public ActionsWebViewFlow verifyIsTestPres
String VERIFY_HEALTH = "Enroll";
assertIsTextPresentOnPage(VERIFY_HEALTH
return this;
}
@Override
public Boolean isExpectedScreenOpened() {
return (!getDriver().findElements(actionsWebView.getHealthyActionsButton().getSelector()).isEmpty());
}
}
Test
@Test
public void scrollingHotListItemsTest() throws Exception {
start()
.clickOnHotListButton()
.addHotListItem(SECOND_HOT_LIST_ITEM_TEXT)
.addHotListItem(THIRD_HOT_LIST_ITEM_TEXT)
.verifyIsScrollingHotListItemsWorks();
}

Project E

Android mobile application

- *Category:* Healthcare
- *Technologies:* PhoneGap, Sencha Touch
- *Stage:* Initial development
- *Automation scope:* Functional testing

Challenge

- Tests that are easy to tweak and support
- Ability to execute on real devices and different platform versions

Project E

Solution

- *Cross-platform:* more than 60 tests that are run both on IOS and Android app
- *Maintainability:* using UI Map, static Page Object design patters and selenide for better readability and support
- *Different platforms:* Selendroid supports wide range of Android versions

Automation technology stack

- Selendroid (Java bindings) as automation tool
- TestNG as test framework
- Selenide as Selenium wrapper
- Git as source control
- Maven as build tool
- Jenkins as CI tool

Project E

Project

Page Object

```
public class PasscodeView extends BaseView {

    private static final String TITLE ="Passcode Set Up";
    private static final  String DEFAULT_PASSCODE = "1234";

    private static By choosePasscodeButton = get("PasscodeView.chooseMyPasscodeButton");
    private static By passcodeTextField = get("PasscodeView.passcodeTextField");
    private static By okPasscodeButton = get("PasscodeView.okPasscodeButton");
    private static By changePasscodeButton = get("PasscodeView.changePasscodeButton");
    private static By cancelPasscodeButton = get("PasscodeView.cancelMyPasscodeButton");
    private static By removePasscodeButton = get("PasscodeView.removeMyPasscodeButton");


    public static void chooseMyPasscode () {
        $(choosePasscodeButton).click();
        $(passcodeTextField).sendKeys(DEFAULT_PASSCODE);
        $(okPasscodeButton).click();

    }

    public static void chooseMyPasscode (String message) {
        $(choosePasscodeButton).click();
        $(passcodeTextField).sendKeys(message);
        $(okPasscodeButton).click();

    }

    public static void changeMyPasscode (String message) {
        $(changePasscodeButton).click();
        $(passcodeTextField).sendKeys(message);
        $(okPasscodeButton).click();
    }

    public static void cancelPasscode () {
        $(cancelPasscodeButton).click();
    }

    public static  void removePasscode () {
        $(removePasscodeButton).click();
    }

    public static void shouldAppear() {
        shouldAppear(TITLE);
    }
```

Test

```
@Test
public void defaultRegistrationWithMedIdTest() {
    SetupView.setupDefault();
    MedicalAidView.setupDefault();
    TermsView.setupDefault();
    MainView.maskShouldAppear();
    Actions.tap();
    MainView.shouldAppear();
}
```

Project F

Android mobile application

- *Category:* Healthcare
- *Technologies:* PhoneGap, Sencha Touch
- *Stage:* Mid-development
- *Automation scope:* Functional testing

Challenge

- Existing automated test support
- Tests that are easy to tweak and support

Project F

Solution

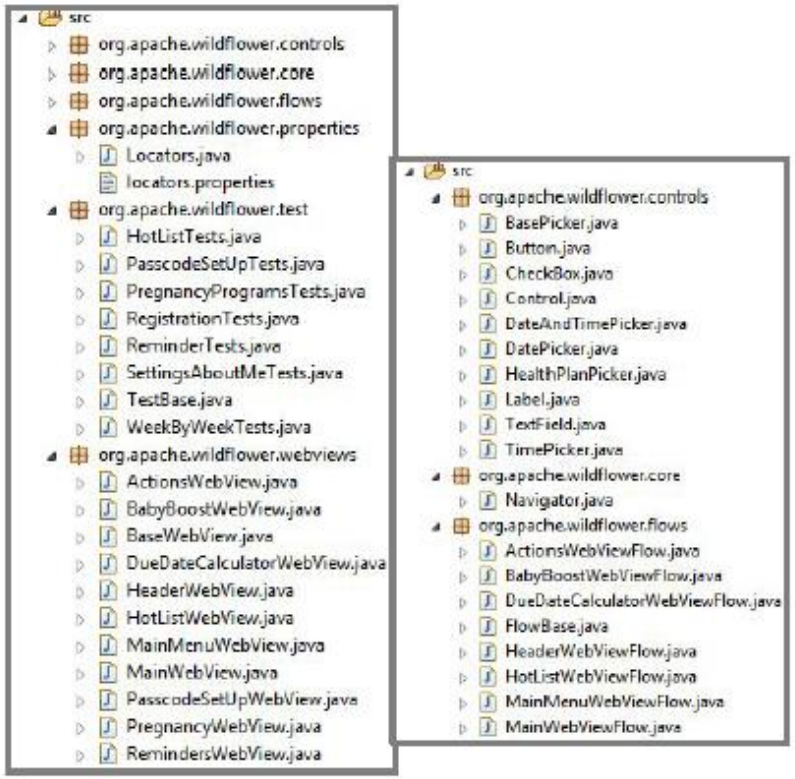
- *Consistency:* using Robotium (Java) for technology consistency
- *Maintainability:* using UI Map, static Page Object design patters and selenide for better readability and support

Automation technology stack

- Robotium(Java) as automation tool
- Git as source control
- Maven as build tool
- Jenkins as CI tool

Project F

Project



Page Object

```
public class HotListWebView extends BaseWebView {
    private Button addNewHotListItemButton;
    private Button doneHotListButton;
    private Button closeHotListButton;
    private TextField newHotListTextField;

    public HotListWebView(ExtSolo solo) {...

    public Button getAddNewHotListItemButton() {
        return addNewHotListItemButton;
    }

    public TextField getNewHotListTextField() {
        return newHotListTextField;
    }

    public Button getCloseHotListButton() {
        return closeHotListButton;
    }

    public Button getDoneHotListButton() {
        return doneHotListButton;
    }
}
```

Test

```
public void test_AlertPopupMessageAppearsIfDatesNotEntered() {
    startRegistration()
        .clickGoButton()
        .clickOnNextButton()
        .verifyIsPopupNotValidDatesAppeared()
        .clickOkOnPopup();
}
```

Project J

Web application

- *Category:* Online Shopping platform
- *Technologies:* Java EE, Ext.js
- *Stage:* Mid-development, existing manual test cases
- *Automation scope:* Functional testing

Challenge

- Deal with dynamic locators and frequent AJAX calls
- Test in many browsers
- Ability to scale testing effectively
- Automate file upload and testing emails

Project J

Solution

- *ExtJS:* using ExtJS API for locating elements and Selenide for dealing with AJAX timings
- *Cross-browser:* Selenium Webdriver as browser driver tool
- *Devices:* executing large number of tests in different browsers with help of Jenkins and Sauce Labs cloud
- *Emails:* Guerrilla Mail API

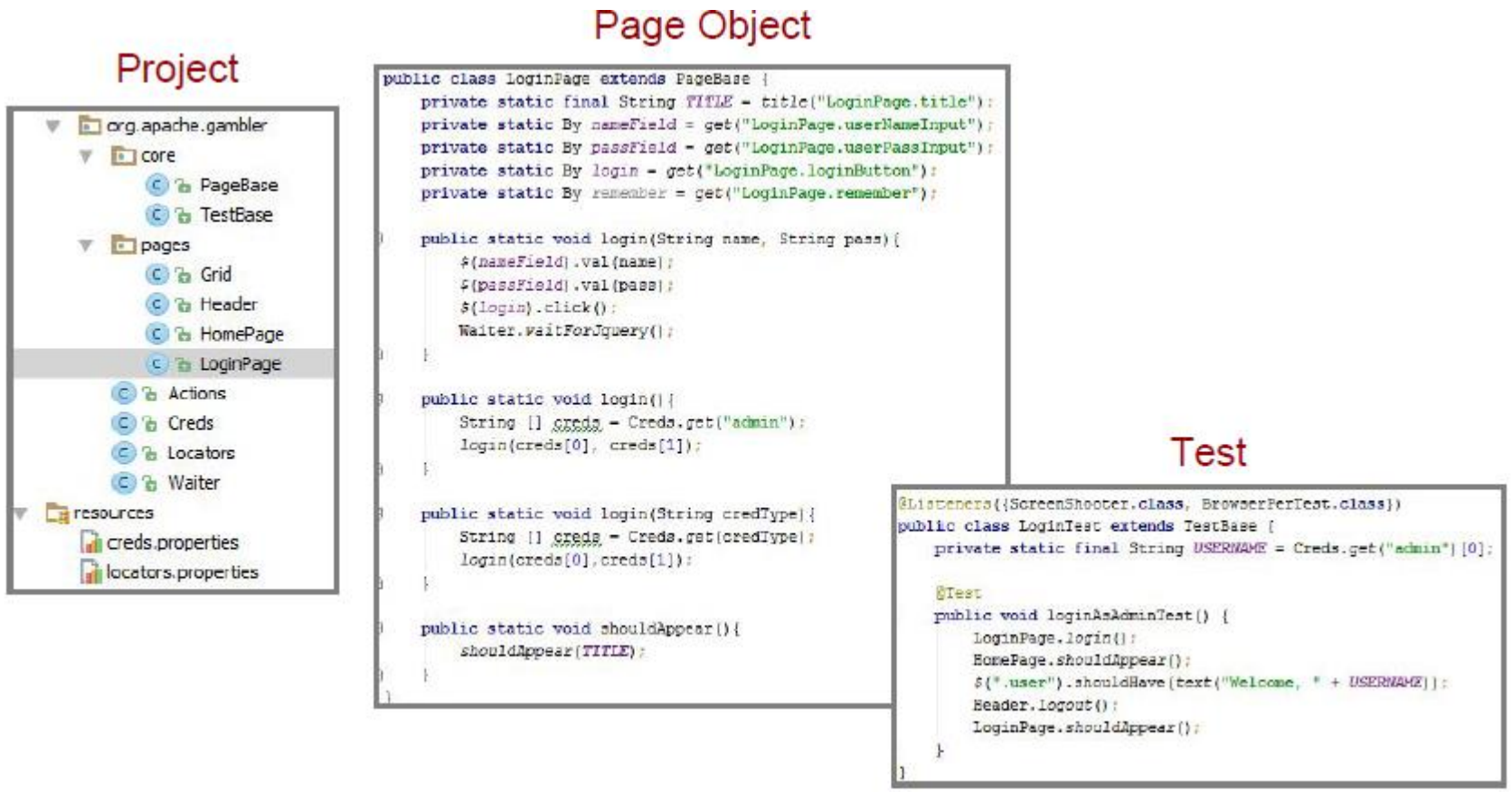
Automation technology stack

- Selenium Webdriver (Java bindings)
- Selenium Grid for distributed execution
- Selenide as AJAX helper
- JUnit as test framework
- Git as source control
- Jenkins as CI
- Sauce labs cloud for scaling test execution
- Allure as reporting framework

Jenkins

JUnit
Testing Framework

Project J


Project
org.apache.gambler
core
PageBase
TestBase
pages
Grid
Header
HomePage
LoginPage
Actions
Creds
Locators
Waiter
resources
creds.properties
locators.properties
Page Object
public class LoginPage extends PageBase {
private static final String TITLE = title("LoginPage.title");
private static By nameField = get("LoginPage.userNameInput");
private static By passField = get("LoginPage.userPassInput");
private static By login = get("LoginPage.loginButton");
private static By remember = get("LoginPage.remember");
public static void login(String name, String pass){
$(nameField).val(name);
$(passField).val(pass);
$(login).click();
Waiter.waitForJquery();
}
public static void login(){
String [] creds = Creds.get("admin");
login(creds[0], creds[1]);
}
public static void login(String credType){
String [] creds = Creds.get(credType);
login(creds[0],creds[1]);
}
public static void shouldAppear(){
shouldAppear(TITLE);
}
}
Test
@Listeners({ScreenShooter.class, BrowserPerTest.class})
public class LoginTest extends TestBase {
private static final String USERNAME = Creds.get("admin")[0];
@Test
public void loginAsAdminTest() {
LoginPage.login();
HomePage.shouldAppear();
$(".user").shouldHave(text("Welcome, " + USERNAME));
Header.logout();
LoginPage.shouldAppear();
}
}

Project H

Web application

- *Category:* Online Wine Shopping portal
- *Technologies:* HTML, CSS, Javascript, Angular JS
- *Stage:* Mid-development, existing manual test cases
- *Automation scope:* Functional testing

Challenge

- A lot of single-page-application-style components
- Multiple users and regions that affect expected results (using datasets)
- Automate testing both admin application and main application

Project H

Solution

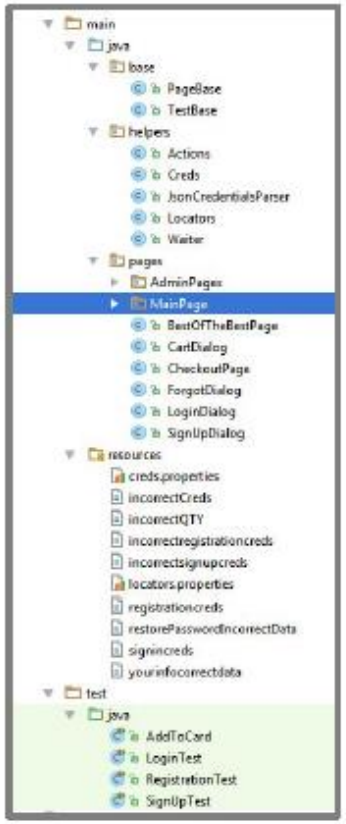
- *Angular JS:* Using Protractor to automate the AngularJS part of the application
- *Multiple users:* Using data-driven approach with TestNG Data Providers and JSON as data sources
- *Admin and main app testing:* Using separate packages inside a project for storing tests and architecture for both parts.

Automation technology stack

- Selenium Webdriver (Java bindings)
- Selenide as a wrapper for more efficient Selenium usage
- Protractor for testing Angular JS part
- TestNG as test framework
- Git as source control
- Circle CI as CI
- Allure as reporting framework

Project H

Project



Page Object

```
public class LoginUserMainPage extends PageBase {
    private static String ProfileUrl = "https://preview.wineaccess.com/accounts/profile/";
    private static final By ACCOUNT_SETTINGS = get("LoginUserMainPage.AccountSettings");
    private static final By MENU = get("LoginUserMainPage.MenuButton");
    private static final By PROFILE_HEAD = get("ProfilePage.Head");
    private static final By LOGOUT = get("MainPage.LogOut");
    private static final By BEST_OF_THE_BEST_LINK = get("LoginUserMainPage.BestOfTheBest");

    public static void checkUserIsLoginnedIn(JsonCredentialsParser data){
        Waiter.waitForJquery();
        Waiter.waitForPageToLoad();
        Waiter.waitForJquery();
        Actions.checkUrl(MainUrl);
        $(MENU).click();
        Waiter.waitForPageToLoad();
        $(ACCOUNT_SETTINGS).click();
        Waiter.waitForPageToLoad();
        Assert.assertEquals($(PROFILE_HEAD).text(), data.FirstName + " "+ data.LastName);
        Assert.assertEquals($(MENU).text(),data.FirstName+" in "+data.Abbreviations);
    }
```

Test

```
//6.3.1 Add to cart for Loged In user (add to cart section)
@Test(groups = "fast",  dataProvider = "DataProviderQTY",dataProviderClass = JsonCredentialsParser.class)
public void addToCartFromSectionForLoginUserTest(String number) throws FileNotFoundException, UnsupportedEncodingException {
    JsonCredentialsParser logindata = (JsonCredentialsParser)JsonCredentialsParser.parameterProviderLoginValidCreds()[0][0];
    NonLoginUserMainPage.clickOnLoginButton();
    LoginDialog.loginCorrectData(logindata);
    AddToCartSection.setSectionQtyField(number);
    AddToCartSection.clickOnLineAddToCartButton();
    SignUpUserMainPage.compareQtyLabelForSignUpUser(number);
    CartDialog.deleteItemFromCard();
}
```

Project I

Web application

- *Category:* Online photograph sharing and storing social network
- *Technologies:* Groovy/Grails, Javascript
- *Stage:* Mid-development, existing manual test cases
- *Automation scope:* Functional testing

Challenge

- Complicated application structure
- Low optimization for different browsers
- Huge volumes of data which caused application lags
- Performance testing to reduce lags

Project I

Solution

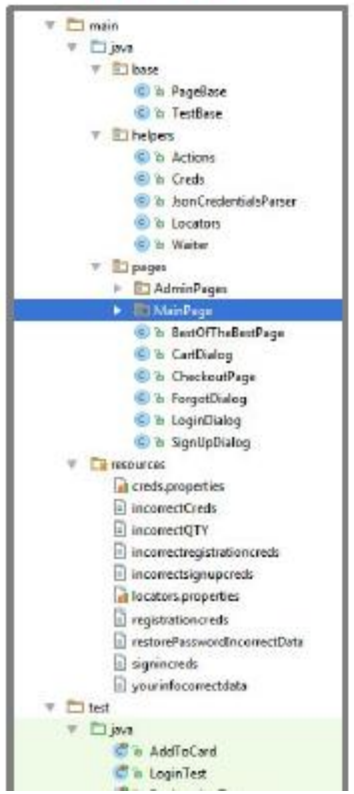
- *App structure:* Adding separate packages to store sets of page objects according to functionality
- *Low optimization:* Adding browser-helpers that use different actions depending on the browser
- *Application lags:* Advanced setting of Selenium waits with using custom Ajax and Jquery waiters
- Performance testing using JMeter.

Automation technology stack

- Selenium Webdriver (Java bindings)
- Selenide as a wrapper for more efficient Selenium usage
- Protractor for testing Angular JS part
- TestNG as test framework
- Git as source control
- Hudson as CI
- JMeter as performance testing tool
- Allure as reporting framework

Project I

Project



Page Object

```
public class LoginUserMainPage extends PageBase {
    private static String ProfileUrl = "https://preview.wineaccess.com/accounts/profile/";
    private static final By ACCOUNT_SETTINGS = get("LoginUserMainPage.AccountSettings");
    private static final By MENU = get("LoginUserMainPage.MenuButton");
    private static final By PROFILE_HEAD = get("ProfilePage.Head");
    private static final By LOGOUT = get("MainPage.LogOut");
    private static final By BEST_OF_THE_BEST_LINK = get("LoginUserMainPage.BestOfTheBest");

    public static void checkUserIsLoginnedIn(JsonCredentialsParser data){
        Waiter.waitForJquery();
        Waiter.waitForPageToLoad();
        Waiter.waitForJquery();
        Actions.checkUrl(MainUrl);
        $(MENU).click();
        Waiter.waitForPageToLoad();
        $(ACCOUNT_SETTINGS).click();
        Waiter.waitForPageToLoad();
        Assert.assertEquals($(PROFILE_HEAD).text(), data.FirstName + " "+ data.LastName);
        Assert.assertEquals($(MENU).text(),data.FirstName+" in "+data.Abbreviations);
    }
```

Test

```
//6.3.1 Add to cart for Loged In user (add to cart section)
@Test(groups = "fast",  dataProvider = "DataProviderQTY",dataProviderClass = JsonCredentialsParser.class)
public void addToCartFromSectionForLoginUserTest(String number) throws FileNotFoundException, UnsupportedEncodingException {
    JsonCredentialsParser logindata = (JsonCredentialsParser)JsonCredentialsParser.parameterProviderLoginValidCreds()[0][0];
    NonLoginUserMainPage.clickOnLoginButton();
    LoginDialog.loginCorrectData(logindata);
    AddToCartSection.setSectionQtyField(number);
    AddToCartSection.clickOnLineAddToCartButton();
    SignUpUserMainPage.compareQtyLabelForSignUpUser(number);
    CartDialog.deleteItemFromCard();
}
```

Project J

Web application

- *Category:* Travellers assistant portal
- *Technologies:* HTML5, CSS, Javascript
- *Stage:* Mid-development, existing manual test cases
- *Automation scope:* Functional testing

Challenge

- Long loading time because the app is filled with images and videos
- Complicated application structure (many sub-applications inside one)
- Unorthodox Admin panel login
- Login using third party application (facebook)
- File downloading

Project J

Solution

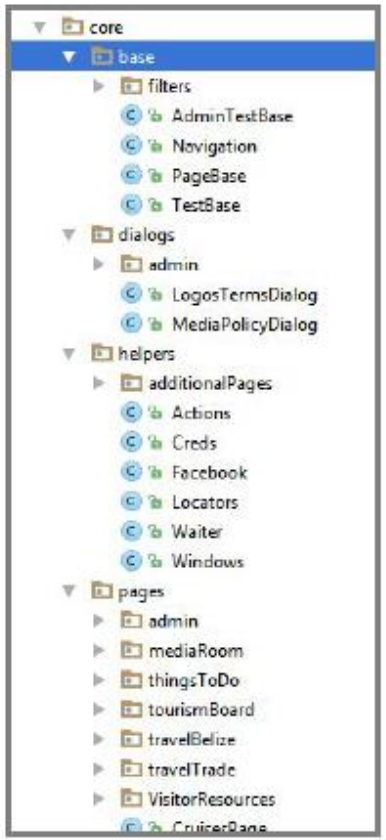
- *Loading time:* Advanced setting of Selenium waits with using custom Ajax and Jquery waiters
- *Application structure:* Adding separate packages to store sub-app data
- *Admin login:* Using direct HTTP request to login
- *Facebook login:* Using selenium window handlers
- *Downloads:* using Selenide download functionality.

Automation technology stack

- Selenium Webdriver (Java bindings)
- Selenide as a wrapper for more efficient Selenium usage
- TestNG as test framework
- Git as source control
- Allure as reporting framework


Project J

Project



Page Object

```
/* Asset Management tab */
public static final By ASSETS_MANAGEMENT_ASSETS=get("adminHeader.assetsManagement.assets");
public static final By ASSETS_MANAGEMENT_GALLERIES=get("adminHeader.assetsManagement.galleries");

/* Notifications tab */
public static final By NOTIFICATIONS_EMERGENCY_ALERTS=get("adminHeader.notifications.emergencyAlerts");

/* Notifications tab */
public static final By PARTNERS_IMPORTS=get("adminHeader.partners.imports");

/* Travel Agents tab */
public static final By TRAVEL_AGENTS_TRAVEL_AGENTS=get("adminHeader.travelAgents.travelAgents");

/* Travel Agents tab */
public static final By USERS_USERS=get("adminHeader.users.users");
public static final By USERS_GROUPS=get("adminHeader.users.groups");

public static void goToHomePagebyLogo() { $(LOGO).click(); }

public static void checkExpectedElements() {
    checkExpectedElements(Arrays.asList(LOGO, NAVIGATION_CONTAINER, LOGOUT_BUTTON, ADD_NEW_BUTTON));
}

public static void toPagesPages(){
    actions().clickAndHold($(PAGES_TAB)).release().clickAndH
}
```

Test

```
@Test
@Features("Travel Belize")
@TestCaseId("4.1.1")
public static void DestinationsWesternBelizeUITest(){
    HomePage.shouldAppear();
    Navigation.clickOnDestinationsWesternBelizeButton();
    for(String name:getNameOfPages(Navigation.WESTERN_BELIZE_LIST_OF_PAGES)) {
        switch(name){
            case "bengue viejo del carmen": break;
            case "san ignacio": break;
            case "western belize"+"\n"+"overview": break;
            default: Assert.assertFalse(true,"Unknown name of page: "+name);
        }
    }
}
```