Lessons Learned
Web Application Testing in .NET

GÁSPÁR NAGY
gaspar.nagy@techtalk.at
http://gasparnagy.com

NDC Oslo, 6/6/2014
Is UI automation bad?

costly
(implementation, maintenance)

senseless
(not testing app behavior)
Google Trends for web automation technologies

Source: http://bit.ly/1jacnQ4
Moving target

• Web technology is changing
  • HTML5/CSS3, REST, JSON, jQuery, WebSocket, etc.

• Web applications are changing
  • responsive, SPA, js-heavy, offline, desktop-like UX, touch, etc.

• Web automation is changing
  • WebDriver W3C specification, browser as IDE, headless browsers, cloud testing
Web Automation in .NET

Web application testing principles should not depend on the technology you use...

But it does.

The current version of ASP.NET is tightly coupled to IIS.

ASP.NET vNext – we wait for you desperately!
Goals for today

Scope: ASP.NET MVC “classic” business apps with automated functional tests (BDD/ATDD)

1. Discover possibilities of doing test-first web development

2. See options how to ease the pain caused by the out-proc testing nature of ASP.NET MVC web testing

3. Enumerate useful tools & resources
Test-first web development
Test-first development

Failing end-to-end Tests

Passing end-to-end Tests

Deploy

Refactor
Test-first web development

• Outside-in approach
  • You model & design your application based on the actual required outcome

• Ensures clean and consistent domain model
  • In modern web applications, HTML is more part of the model than the presentation (which is in CSS/js)

• Ensures fast and stable browser automation from the beginning
  • Better than fixing it later with a huge automation code base
Demo: Test-first web development

Controller-level
integration testing

Coypu

Page Objects

SpecsFor.MVC

Pure Selenium WebDriver
with FireFox
Demo: Test-first web development

```csharp
driver.Navigate().GoToUrl("http://localhost/Home/Index");

↓

app.NavigateTo<HomeController>(c => c.Index());
```
Expressing tests – Display case

<table>
<thead>
<tr>
<th>Arrange</th>
<th>Act</th>
<th>Assert</th>
</tr>
</thead>
<tbody>
<tr>
<td>var controller = new HomeController();</td>
<td>var app = new MvcWebApp();</td>
<td>var driver = new FirefoxDriver();</td>
</tr>
<tr>
<td>var result = controller.Index();</td>
<td>app.NavigateTo &lt;HomeController&gt;(c =&gt; c.Index());</td>
<td>driver.Navigate().GoToUrl(&quot;http:...&quot;);</td>
</tr>
<tr>
<td>var list = app.FindElement(By.Id(&quot;questions&quot;));</td>
<td>var questions = list.FindElements(By.TagName(&quot;li&quot;));</td>
<td>var questions = list.FindElements(By.TagName(&quot;li&quot;));</td>
</tr>
<tr>
<td>Assert.AreEqual(3, result.Model.Count);</td>
<td>Assert.AreEqual(3, questions.Count);</td>
<td>Assert.AreEqual(3, questions.Count);</td>
</tr>
</tbody>
</table>
## Expressing tests – Edit case

<table>
<thead>
<tr>
<th>Arrange</th>
<th>Controller-level</th>
<th>SpecsFor.MVC</th>
<th>Pure WebDriver</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

```csharp
var result = controller.Ask(new QuestionModel { Title = "foo" });
```

```csharp
app.NavigateTo <HomeController>(c => c.Ask());
driver.Navigate().GoToUrl("http:...");
```

```csharp
app.FindFormFor <QuestionModel>()
    .Field(qm => qm.Title)
    .SetValueTo("foo")
driver.FindElement(By.Id("Title"))
    .SendKeys("foo")
```

```csharp
.getResult().Submit();
driver.FindElement(By.Id("askform"))
    .Submit();
```

```csharp
// "follow redirect"
result = controller.Index();
```

// check model

```csharp
// grab question list from page and check
```
## Expressing tests – Validation case

<table>
<thead>
<tr>
<th>Arrange</th>
<th>Controller-level</th>
<th>SpecsFor.MVC</th>
<th>Pure WebDriver</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>app.NavigateTo</td>
<td>driver.Navigate()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;HomeController&gt;(...)</td>
<td>.GoToUrl(&quot;http:...&quot;);</td>
</tr>
</tbody>
</table>

```csharp
define q =
    new QuestionModel
    { Title = "" };
//TODO: model validation
var result =
    controller.Ask(q);
```

```csharp
app.FindFormFor<QuestionModel>()
    .Field(qm => qm.Title)
    .SetValueTo(""")
    .Submit();
driver.FindElement(By.Id("Title"))
    .SendKeys("");
driver.FindElement(By.Id("askform"))
    .Submit();
```

```csharp
var errors =
    result.ViewData.ModelState
    ["Title"].Errors;
Assert.Greater(0, errors.Count);
```

```csharp
app.FindFormFor<QuestionModel>()
    .Field(qm => qm.Title)
    .ShouldBeInvalid();
```

```csharp
Assert.IsNotNull(
    driver.FindElement(By.CssSelector("span.field-validation-error[data-valmsg-for=Title]")));`
# Expressing tests – A subjective comparison

<table>
<thead>
<tr>
<th></th>
<th>Controller-level</th>
<th>SpecsFor.MVC</th>
<th>Pure WebDriver</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Dev. order</strong></td>
<td>Test – Controller/BL – View</td>
<td>Test – (Controller/BL</td>
<td>View)</td>
</tr>
<tr>
<td><strong>Good design</strong></td>
<td>♥ (not driven by outcomes)</td>
<td>♥ ♥ ♥ (driven by domain)</td>
<td>♥ ♥ (mixed domain)</td>
</tr>
<tr>
<td><strong>Clean test</strong></td>
<td>♥ ♥ ♥ (except infr. hack)</td>
<td>♥ ♥ (needs switching to WD)</td>
<td>♥ (selectors as string)</td>
</tr>
<tr>
<td><strong>Clean HTML</strong></td>
<td>♥ (no design help)</td>
<td>♥ ♥ ♥ (~stays with domain)</td>
<td>♥ ♥ (mixed domain)</td>
</tr>
<tr>
<td><strong>Maturity</strong></td>
<td>♥ ♥ ♥ (~your code)</td>
<td>♥ (3rd party framework)</td>
<td>♥ ♥ (W3C-driven framework)</td>
</tr>
<tr>
<td><strong>Score</strong></td>
<td>8 ♥</td>
<td>9 ♥</td>
<td>7 ♥</td>
</tr>
</tbody>
</table>
Out-proc vs. in-proc testing
Is mocking bad?

• The automation trade-off also appears at the component level
  • Some components cannot be tested without mocks/stubs
    • E.g: e-mail sending, time, authentication, etc.
  • Mock/stub components can improve the efficiency

• Demo: Let’s try to use in-memory database access to our tests!
Testing through controller in-proc
Testing through browser out-proc

Legend
- blue: test code
- green: application code
- black solid line: data flow
- orange square: mock/stub
- black dotted line: app state manipulation
Testing through browser in-proc (wish)

Possible for:
- WebAPI
- NancyFX
- vNext

Still a wish:
- ASP.NET MVC
- WebAPI used in a web project
- ASP.NET WebForms
NUnit inside IIS
Testing through browser in-proc with “MvcIntegrationTesting” (Steven Sanderson)

Test Process

Test AppDomain

IIS-Like AppDomain

Test Controller

Tests

Application

remoting

Browser

DB

Legend

- test code
- application code
- data flow
- mock/stub
- app state manipulation
Testing through browser in-proc with SpecFlow+ Web (alpha)
Out-proc vs. in-proc testing

• Subbing and mocking are important tools to fight for better efficiency
• The out-proc nature of ASP.NET web testing makes this hard
  • Frameworks can help to host your tests in the same AppDomain
  • If this is not possible, you can implement backdoor interfaces to your application (e.g. WebAPI/REST)
• OWIN improves this, but for MVC only in vNext
Our journey...

Selenium WebDriver

Page Objects

Coypu

SpecsFor.MVC

MvcIntegration Test

NUnit inside IIS

SpecFlow+ Web
Conclusions

• Reevaluate web testing strategy, as the circumstances rapidly changing

• Define a testing strategy based on test-first

• Be the master of the tools – you need to carefully pick and combine them to get the best result

- WebDriver W3C Draft
- Coypu
- Selenium Page Object Framework
- SpecsFor.MVC (good intro post)
- Phantomjs (NuGet package)
- SimpleBrowser, SimpleBrowser.WebDriver
- MvcIntegrationTestFramework (Steven Sanderson)
  - FakeHost – recent fork
- ASP.NET vNext
- SpecFlow+ Web
Thank you!

GÁSPÁR NAGY

gn@techtalk.at

http://gasparnagy.com

twitter: @gasparnagy

NDC Oslo, 6/6/2014