

# *“Approaches and tools for Automated E2E testing of web applications”*

A multi-perspective analysis during software evolution

Andrea Stocco  
UBC  
[astocco@ece.ubc.ca](mailto:astocco@ece.ubc.ca)

# The World “Wild” Web



# The World “Wild” Web

What does make a web application unique?

Is testing web apps different than testing a desktop app?

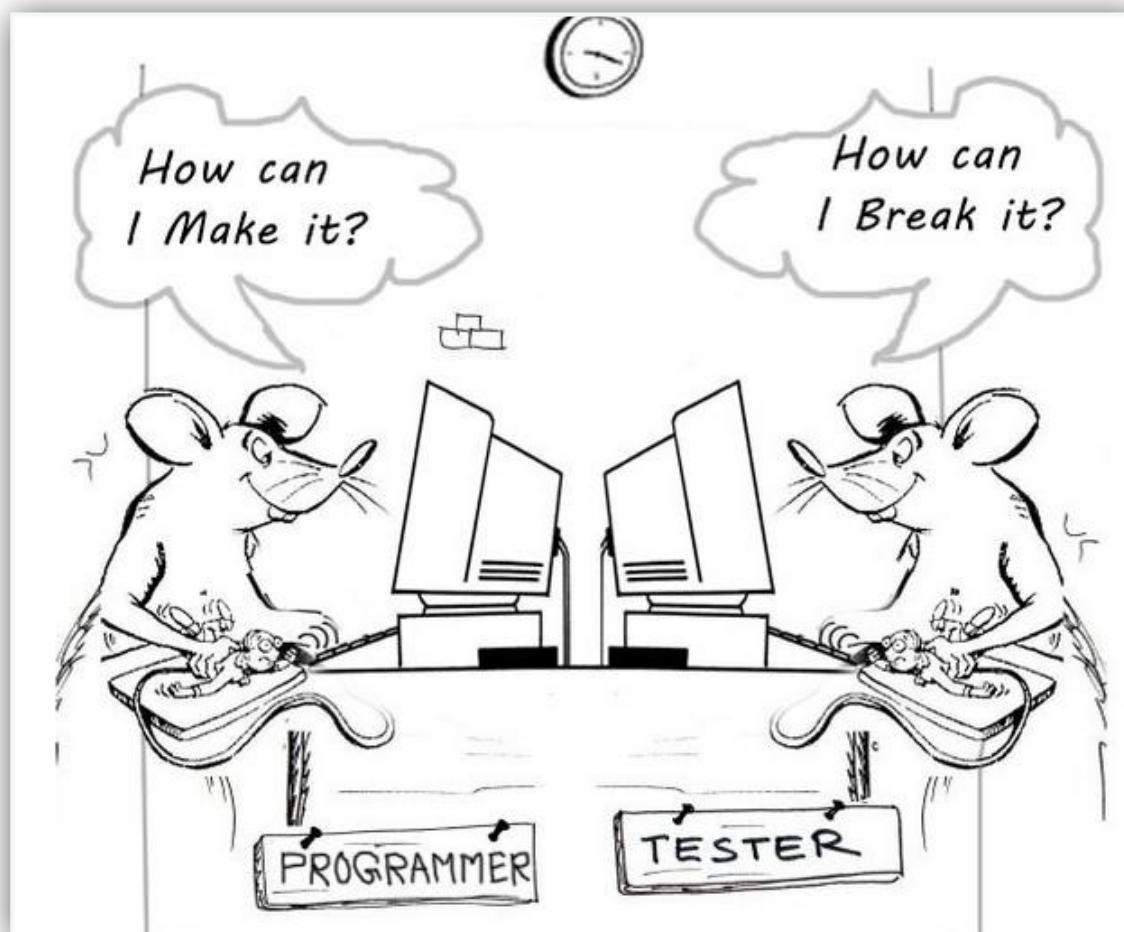
What makes testing web apps complex?

# Modern Web Applications

- complex **infrastructure**
- integration of **different** technologies
- **asynchronous** interactions
- **different** platforms (web, smartphones)
- used by billions of people
- **ultra-rapid** evolution

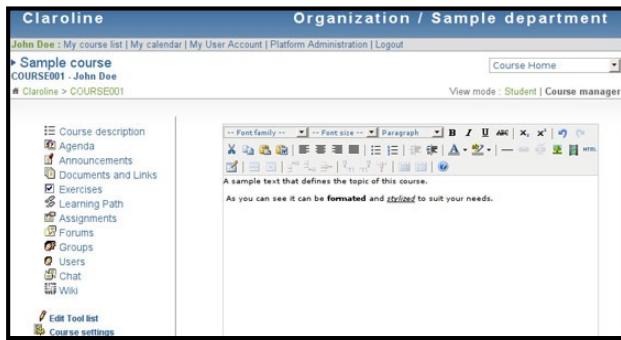


# Web Development vs Testing



# Functional Test Automation

## Web App



## Functional Requirements Analysis

```
// Double click on DataGrid line to open User Details form
3 FlexDataGrid("dg")>itemDoubleClick();
4 // Double Click on DataGrid line to open User Details form
5 FlexDataGrid("dg")>itemDoubleClick();
6 // Double Click on DataGrid line to open User Details form
7 F 5 3 FlexDataGrid("dg")>itemDoubleClick();
8 F 6 4 "X" Click to select/unselect | "Admin" | Administrator | nomail@riatest.com";
9 F 7 5
10 F 8 6 // Change User Details
11 F 9 7 FlexTextArea("Name::nameInput")>textSelectionChange(0,13);
12 F 10 8 FlexTextArea("Name::nameInput")>textInput("John Smith");
13 F 11 9 FlexTextArea("Name::nameInput")>keyFocusChange();
14 F 12 10 FlexTextArea("Email::emailInput")>textInput("somemail@nodomain.com");
15 F 13 11 FlexTextArea("Email::emailInput")>keyFocusChange();
16 F 14 12 FlexTextArea("New Password::passwordInput")>textInput("mypass");
17 F 15 13 FlexTextArea("New Password::passwordInput")>keyFocusChange();
18 F 16 14 FlexTextArea("Verify Password::passwordInput2")>textInput("mypass");
19 F 17 15
20 w 18 16 // Click OK to close form and save details
21 F 19 17 FlexButton("OK")>click();
22 F 20 18
23 F 21 19 // Wait while user details are saved and appear in the DataGrid
24 waiFor(FlexDataGrid("dg")>selectedItem.indexOf("John Smith")>0);
```

## Test Scripts Creation

## Test Scripts Automation

## Test Results

## Bug Reports



# Functional Test Automation

Web App



Why have such tools focused at the E2E (GUI) level?

Functional  
Requirements  
Analysis

```
1 // Open user details page to open user details form
2 F.openDetails("dp") == itemDetails.click()
3 // Click on "Edit" link to open update details form
4 F.click("edit") == itemDetails.click()
5 // Enter new email address
6 F.sendKeys("new_email@example.com", "email", "new_email@test.com") == itemDetails.change(0, 13)
7 F.click("update") == itemDetails.click()
8 F.assertText("Changed email address successfully", "success") == itemDetails.successChange(0, 13)
9 F.assertText("Email address has been changed", "message") == itemDetails.messageChange(0, 13)
10 F.assertText("Email address has been changed", "label") == itemDetails.labelChange(0, 13)
11 F.assertText("Email address has been changed", "text") == itemDetails.textChange(0, 13)
12 F.assertText("Email address has been changed", "button") == itemDetails.buttonChange(0, 13)
13 F.assertText("Email address has been changed", "checkbox") == itemDetails.checkboxChange(0, 13)
14 F.assertText("Email address has been changed", "radio") == itemDetails.radioChange(0, 13)
15 F.assertText("Email address has been changed", "select") == itemDetails.selectChange(0, 13)
16 F.assertText("Email address has been changed", "textarea") == itemDetails.textareaChange(0, 13)
```

Test  
Scripts

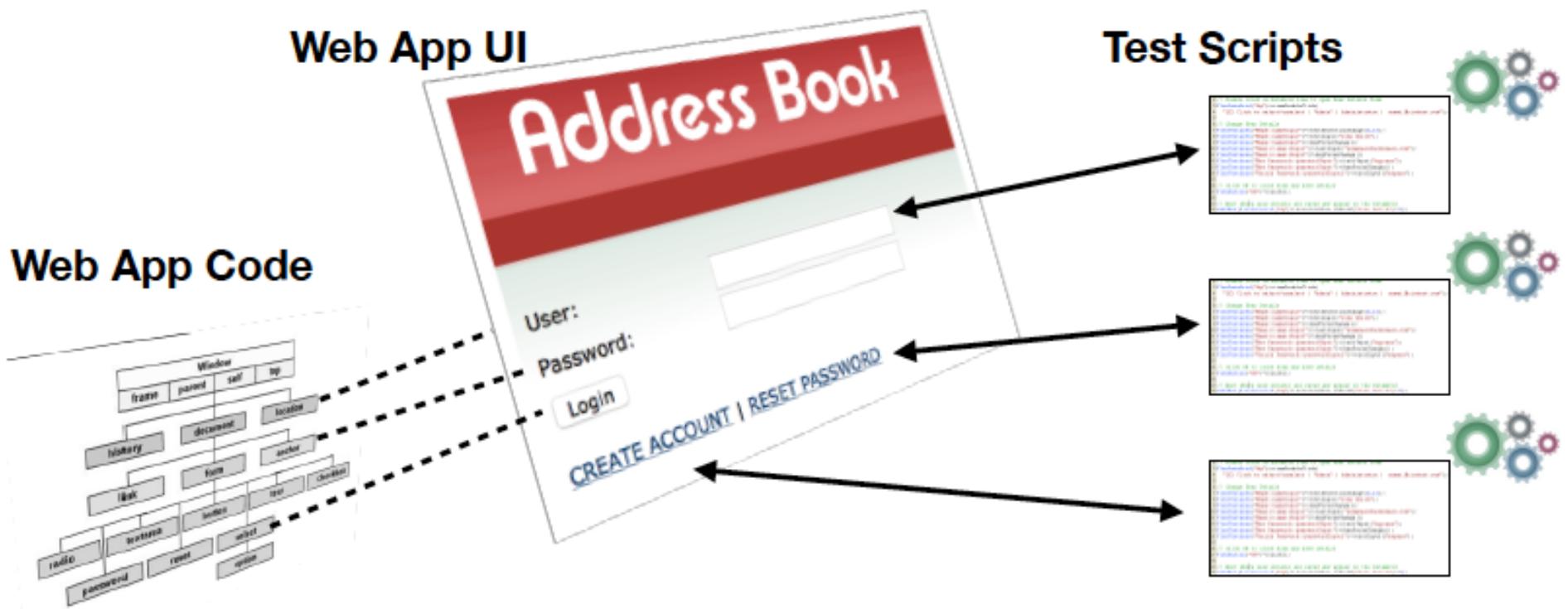
Test Scripts  
Automation

Test Results

Bug Reports



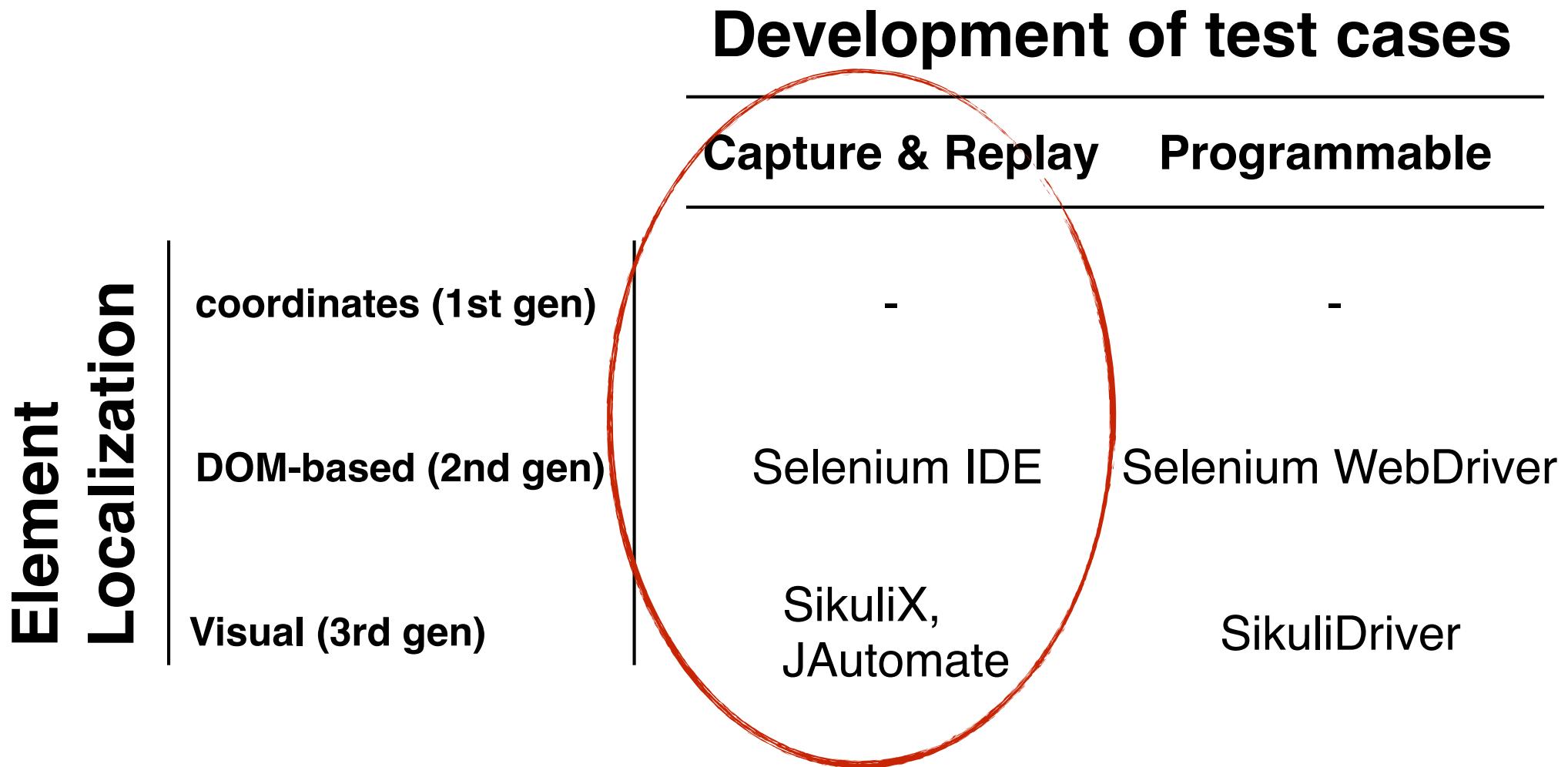
# E2E Test Automation



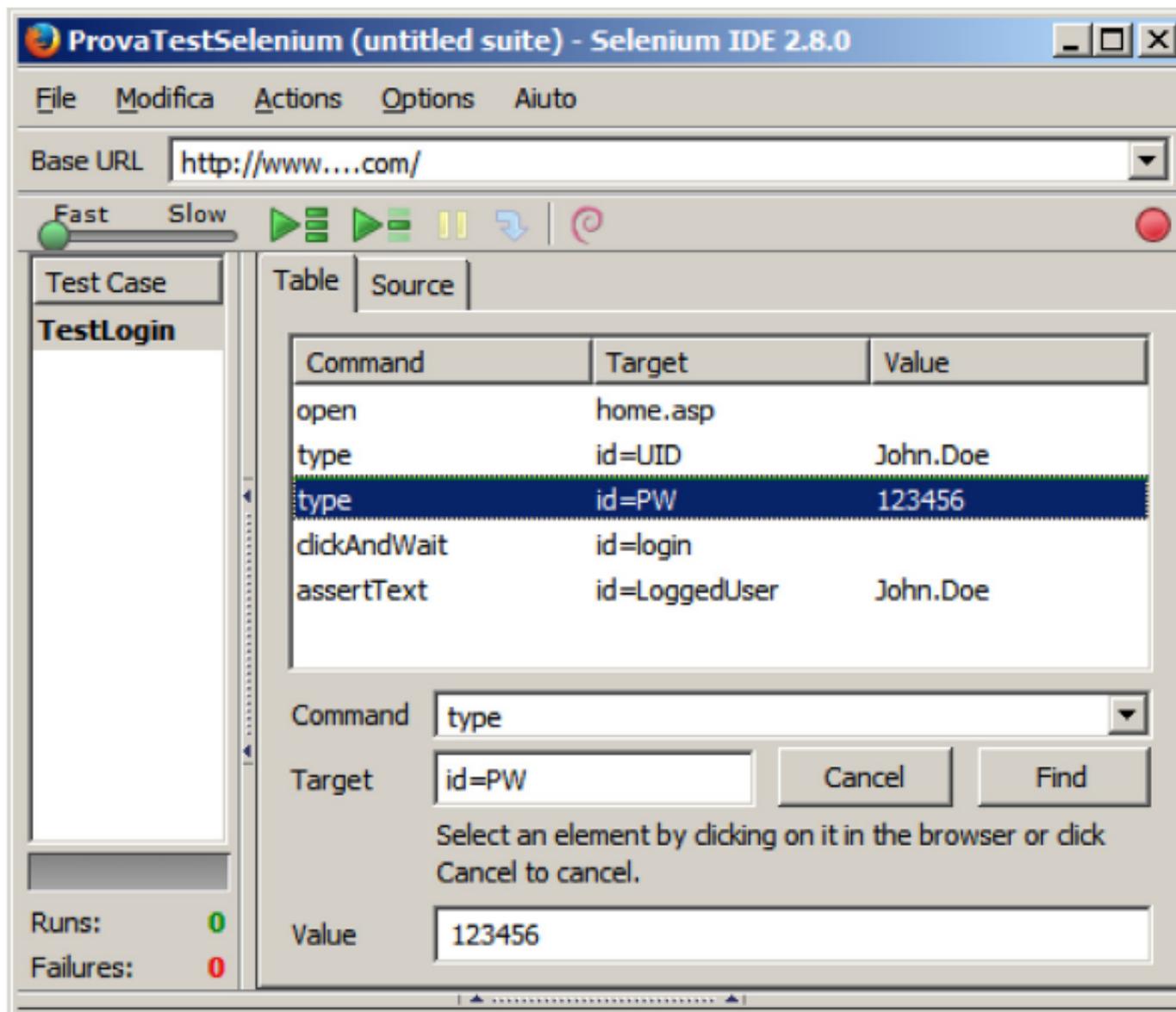
# Approaches to E2E Test Automation

		<b>Development of test cases</b>	
		<b>Capture &amp; Replay</b>	<b>Programmable</b>
<b>Element Localization</b>	<b>coordinates (1st gen)</b>	-	-
	<b>DOM-based (2nd gen)</b>	Selenium IDE	Selenium WebDriver
	<b>Visual (3rd gen)</b>	SikuliX, JAutomate	SikuliDriver

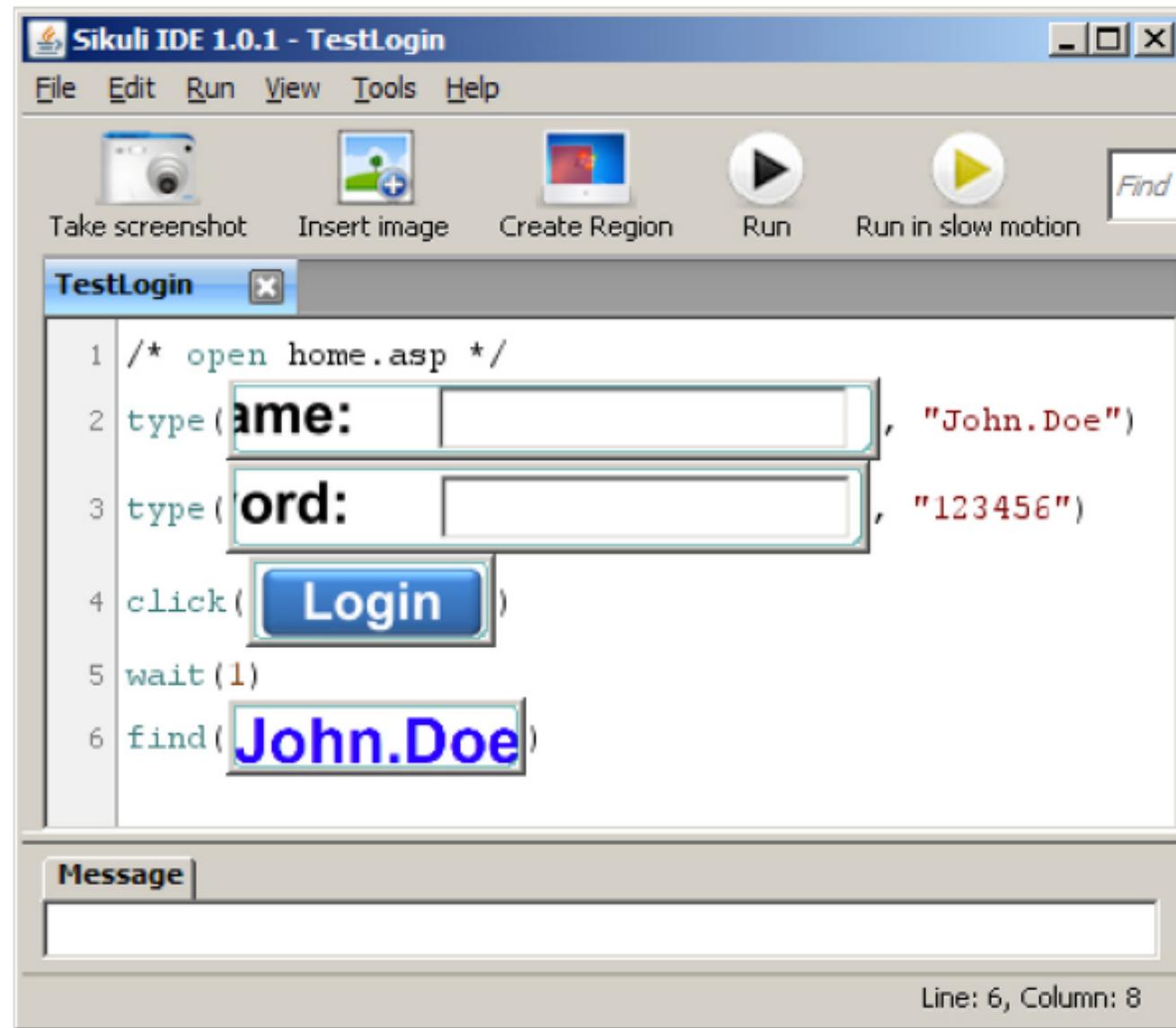
# Approaches to E2E Test Automation



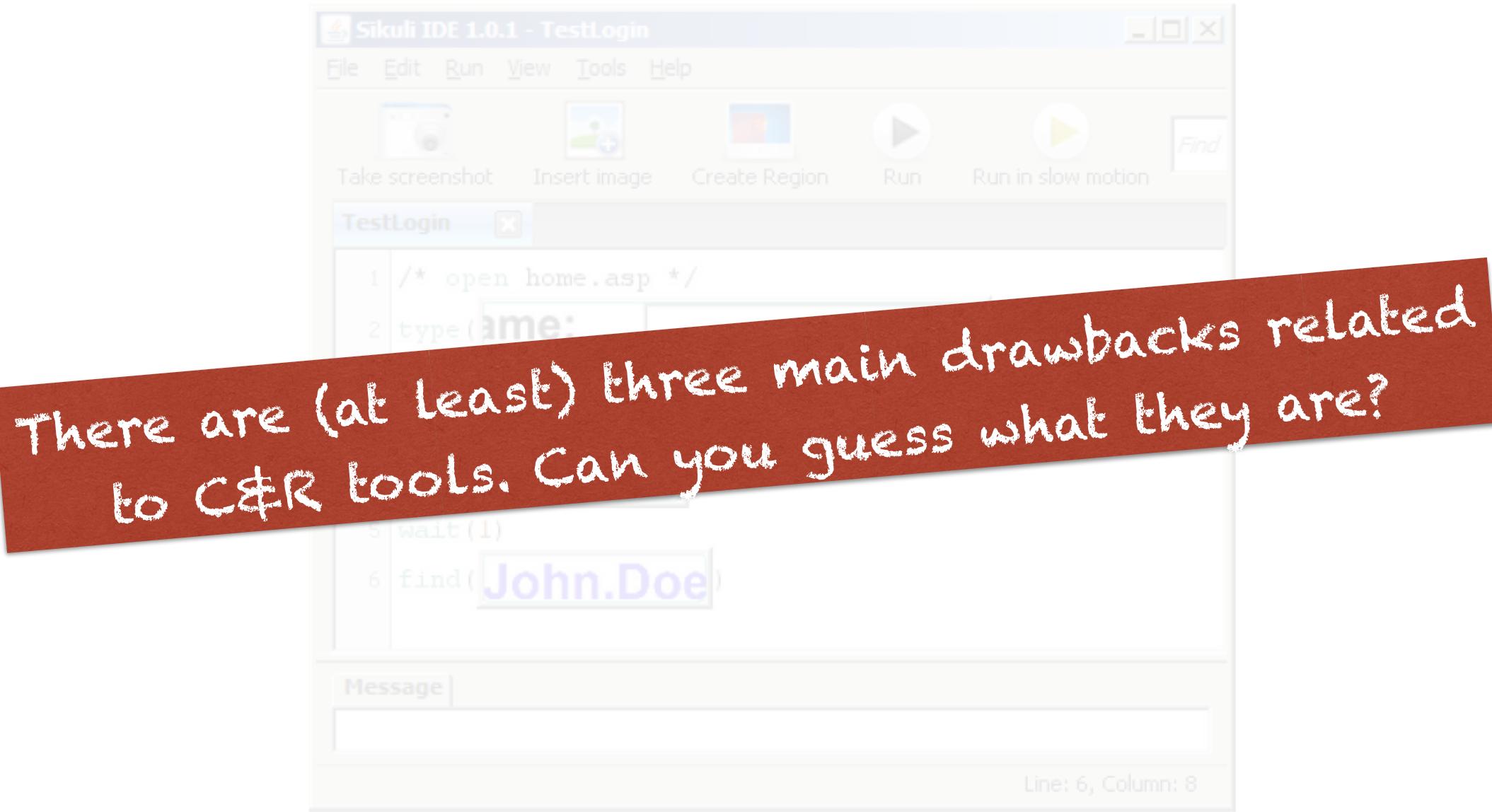
# (DOM) Capture & Replay



# (Visual) Capture & Replay



# (Visual) Capture & Replay



# Capture & Replay Drawbacks

**Hard-code values**

**Strong coupling**

**Duplicated code**

# Approaches to E2E Test Automation

## Element Localization

- coordinates (1st gen)
- DOM-based (2nd gen)
- Visual (3rd gen)

## Development of test cases

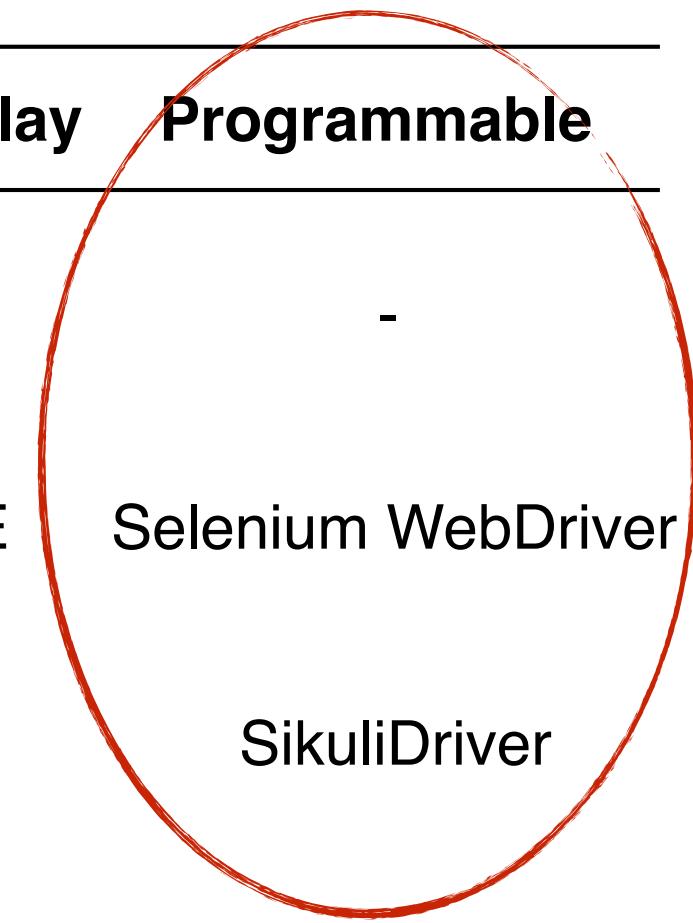
Capture & Replay      Programmable

-  
Selenium IDE

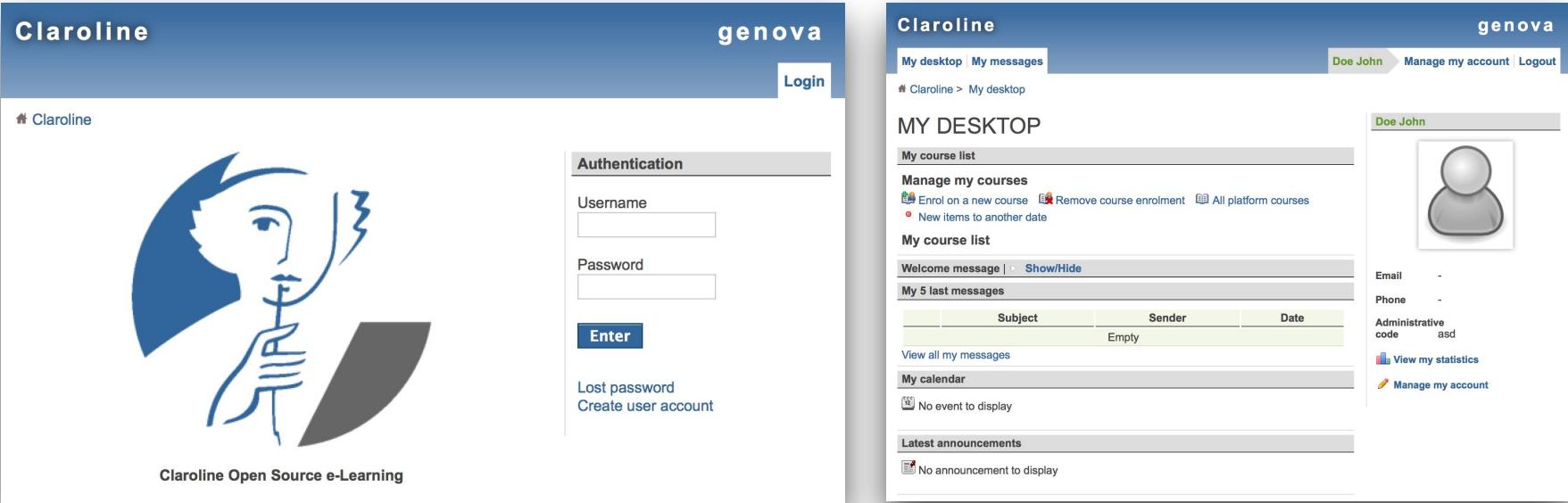
SikuliX,  
JAutomate

-  
Selenium WebDriver

SikuliDriver



# (DOM) Programmable

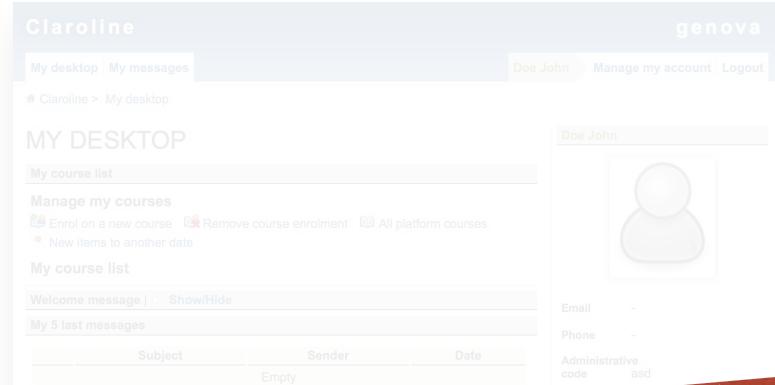
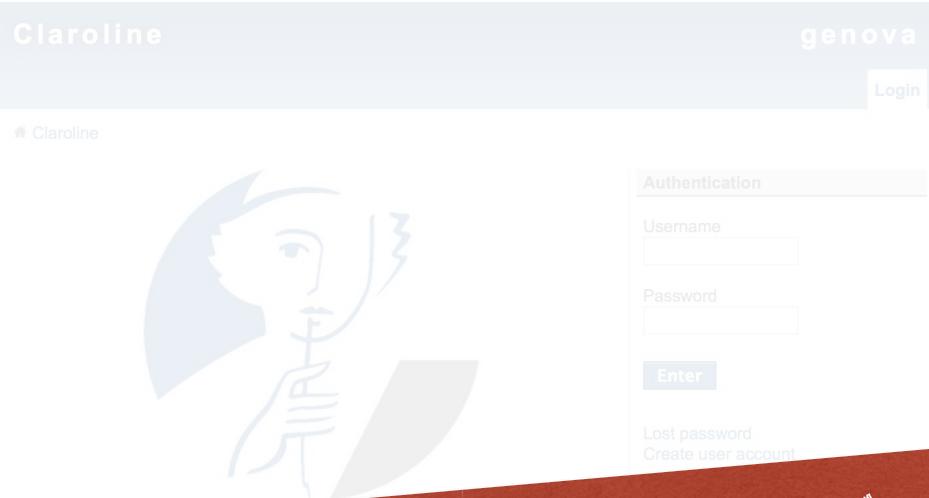


```
public class Login {  
    @Test  
    public void testLogin() {  
        WebDriver driver = new FirefoxDriver();  
        driver.get("http://localhost/claroline1115/index.php");  
        driver.findElement(By.id("user")).sendKeys("John Doe");  
        driver.findElement(By.XPath("//form/input[2]")).sendKeys("securepassword");  
        driver.findElement(By.LinkText(linkText="Enter")).click();  
        Assert.assertEquals(driver.findElement(By.Id("John Doe")), "Login was successful");  
    }  
}
```

Annotations on the code:

- A red box labeled "Browser specific" has an arrow pointing to the line `driver.get("http://localhost/claroline1115/index.php");`.
- A red box labeled "UI specific" has an arrow pointing to the line `driver.findElement(By.XPath("//form/input[2]")).sendKeys("securepassword");`.
- A red box labeled "Manipulation specific" has an arrow pointing to the line `Assert.assertEquals(driver.findElement(By.Id("John Doe")), "Login was successful");`.

# (DOM) Programmable



Can you enumerate (at least) some of the main advantages that come with the use of a programmable tool w.r.t. a C&R tool?

```
    driver = new FirefoxDriver();
    driver.get("http://localhost/claroline1115/index.php");
    driver.findElement(By.Id(id="user")).sendKeys("John Doe");
    driver.findElement(By.XPath("//form/input[2]")).sendKeys("securepassword");
    driver.findElement(By.LinkText(linkText="Enter")).click();
    Assert.assertEquals(driver.findElement(By.Id("John Doe")), "Login was successful");
}
```

UI specific

Manipulation specific

# Programmable tools Advantages

~~Hard-code values~~

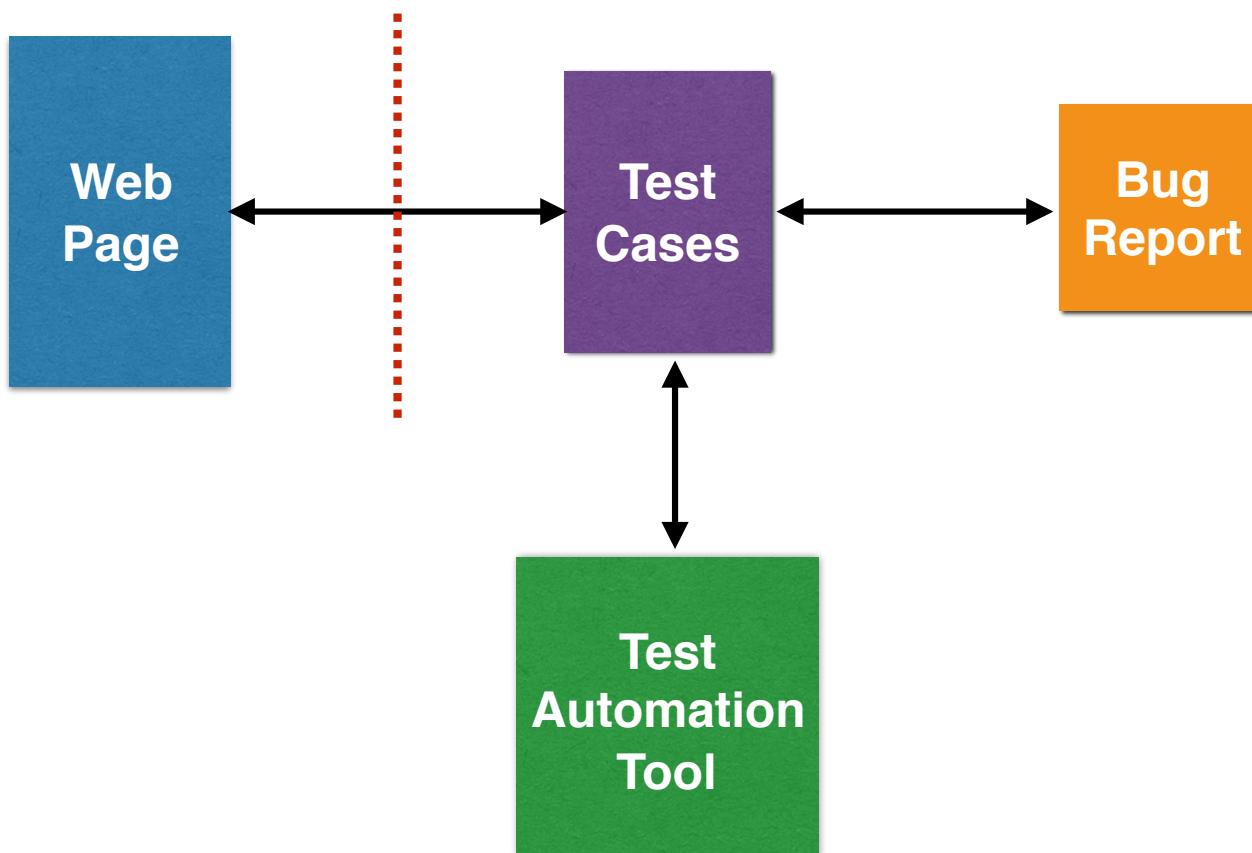
**Data-driven**

~~Strong coupling~~

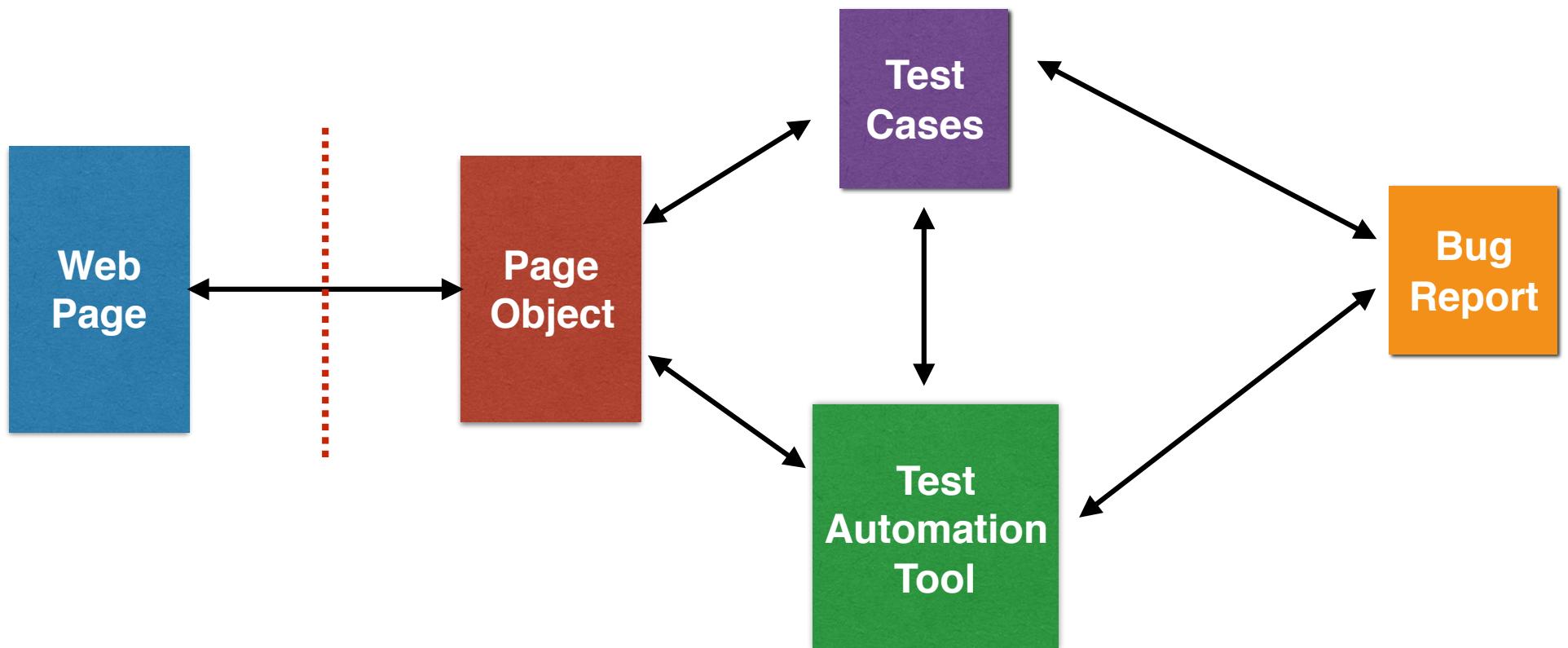
**Design Patterns**

~~Duplicated code~~

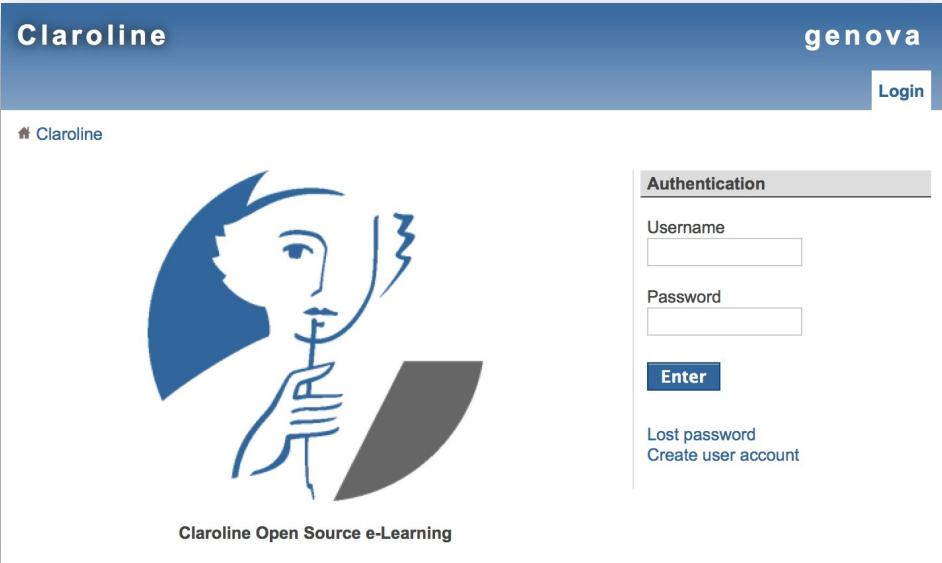
# Naive web testing



# Use Page Objects!



# Page Object Pattern: An Example



```
public class IndexPage {  
    private final WebDriver driver;  
  
    @FindBy(id="user")  
    private WebElement username;  
  
    @FindBy(xpath="//form/input[2]")  
    private WebElement password;  
  
    @FindBy(linkText="Enter")  
    private WebElement login;  
  
    @FindBy(id="loggedUser")  
    private WebElement loggedUser;  
  
    public UserAccount goToCreateAccount(){  
        createUserAccount.click();  
        return new UserAccount(driver);  
    }  
  
    public HomePage login(String usr, String pwd) {  
        username.sendKeys(usr);  
        password.sendKeys(pwd);  
        login.click();  
        return new HomePage(driver);  
    }  
}
```

# Page Object Pattern: An Example



Claroline

genova

Login

Claroline



Claroline Open Source e-Learning

Authentication

Username

Password

Enter

Lost password  
Create user account

WebElements

```
public class IndexPage {  
    private final WebDriver driver;  
  
    @FindBy(id="user")  
    private WebElement username;  
  
    @FindBy(xpath="//form/input[2]")  
    private WebElement password;  
  
    @FindBy(linkText="Enter")  
    private WebElement login;  
  
    @FindBy(id="loggedUser")  
    private WebElement loggedUser;  
  
    public UserAccount goToCreateAccount(){  
        createUserAccount.click();  
        return new UserAccount(driver);  
    }  
  
    public HomePage login(String usr, String pwd) {  
        username.sendKeys(usr);  
        password.sendKeys(pwd);  
        login.click();  
        return new HomePage(driver);  
    }  
}
```

# Page Object Pattern: An Example



Navigations

```
public class IndexPage {  
    private final WebDriver driver;  
  
    @FindBy(id="user")  
    private WebElement username;  
  
    @FindBy(xpath="//form/input[2]")  
    private WebElement password;  
  
    @FindBy(linkText="Enter")  
    private WebElement login;  
  
    @FindBy(id="loggedUser")  
    private WebElement loggedUser;  
  
    public UserAccount goToCreateAccount(){  
        createUserAccount.click();  
        return new UserAccount(driver);  
    }  
  
    public HomePage login(String usr, String pwd) {  
        username.sendKeys(usr);  
        password.sendKeys(pwd);  
        login.click();  
        return new HomePage(driver);  
    }  
}
```

# Page Object Pattern: An Example



Claroline

genova

Claroline Open Source e-Learning

Actions

```
public class IndexPage {  
    private final WebDriver driver;  
  
    @FindBy(id="user")  
    private WebElement username;  
  
    @FindBy(xpath="//form/input[2]")  
    private WebElement password;  
  
    @FindBy(linkText="Enter")  
    private WebElement login;  
  
    @FindBy(id="loggedUser")  
    private WebElement loggedUser;  
  
    public UserAccount goToCreateAccount(){  
        createUserAccount.click();  
        return new UserAccount(driver);  
    }  
  
    public HomePage login(String usr, String pwd) {  
        username.sendKeys(usr);  
        password.sendKeys(pwd);  
        login.click();  
        return new HomePage(driver);  
    }  
}
```

# Page Object Pattern: An Example



The screenshot shows a web application interface for 'Claroline'. At the top, there's a navigation bar with links for 'My desktop' and 'My messages'. Below it, a breadcrumb trail shows 'Claroline > My desktop'. The main content area is titled 'MY DESKTOP' and contains sections for 'My course list', 'Manage my courses' (with links for 'Enrol on a new course', 'Remove course enrolment', and 'All platform courses'), 'New items to another date', 'My course list', 'Welcome message' (with 'Show/Hide' link), 'My 5 last messages' (empty), 'View all my messages', 'My calendar' (empty), and 'Latest announcements' (empty). On the right side, there's a user profile section for 'Doe John' with fields for 'Email' (empty), 'Phone' (empty), 'Administrative code' (set to 'asd'), and buttons for 'View my statistics' and 'Manage my account'.

Getters

```
public class HomePage {  
    private final WebDriver driver;  
  
    @FindBy(xpath="//li/a[1]")  
    private WebElement myDesktop;  
  
    @FindBy(xpath="//li/a[2]")  
    private WebElement myMessages;  
  
    @FindBy(linkText="Enrol")  
    private WebElement enrolACourse;  
  
    @FindBy(id="loggedUser")  
    private WebElement loggedUser;  
  
    public CoursePage goToEnrol(){  
        enrolACourse.click();  
        return new CoursePage(driver);  
    }  
    ...  
    public String getLoggedUser() {  
        return loggedUser.getText();  
    }  
}
```

# Test without abstraction



```
public class Login {  
    @Test  
    public void testLogin() {  
        WebDriver driver = new FirefoxDriver();  
        driver.get("http://localhost/claroline1115/index.php");  
        driver.findElement(By.Id("user")).sendKeys("John Doe");  
        driver.findElement(By.XPath("//form/input[2]")).sendKeys("securepassword");  
        driver.findElement(By.LinkText("Enter")).click();  
        Assert.assertEquals(driver.findElement(By.Id("John Doe")), "Login was successful");  
    }  
}
```

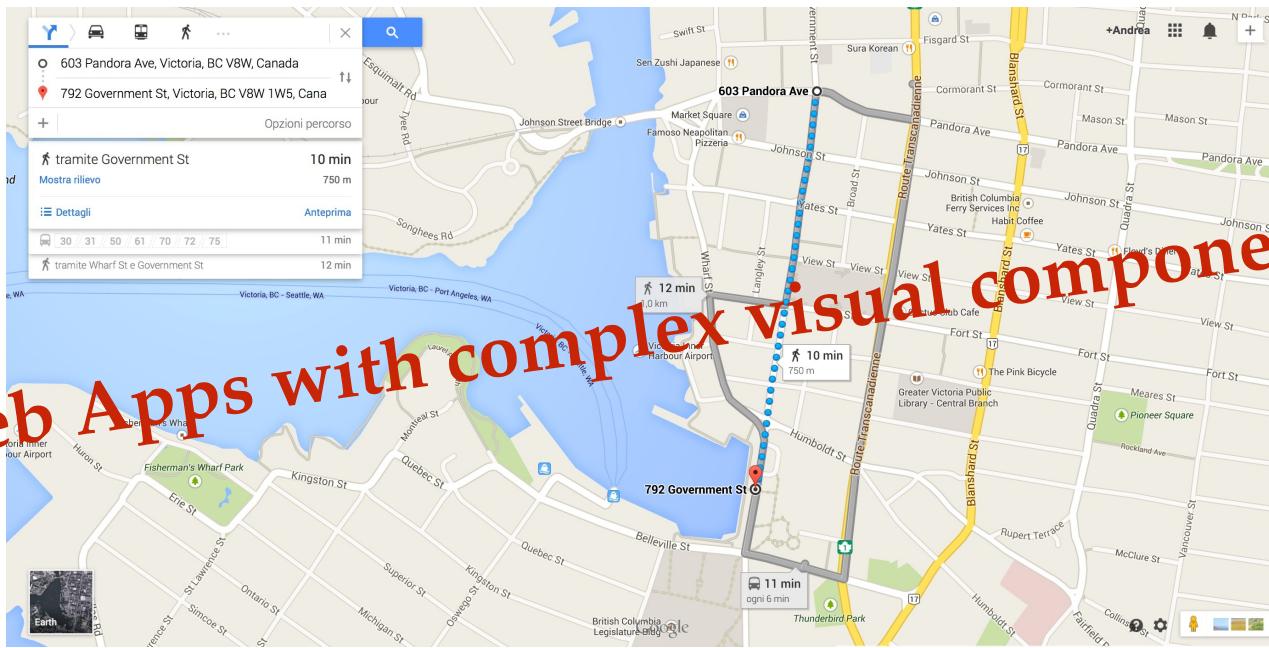
# Test with Page Object abstraction



```
public class Login {  
    @Test  
    public void testLogin {  
        Utils.connectToWebAppUnderTest();  
  
        IndexPage ip = new IndexPage(driver);  
  
        HomePage hp = ip.login("John Doe", "securepassword");  
  
        assertEquals(hp.getLoggedUser(), "Doe John");  
    }  
}
```

# Visual Test Automation

Web Apps with complex visual components



3rd generation: image-recognition



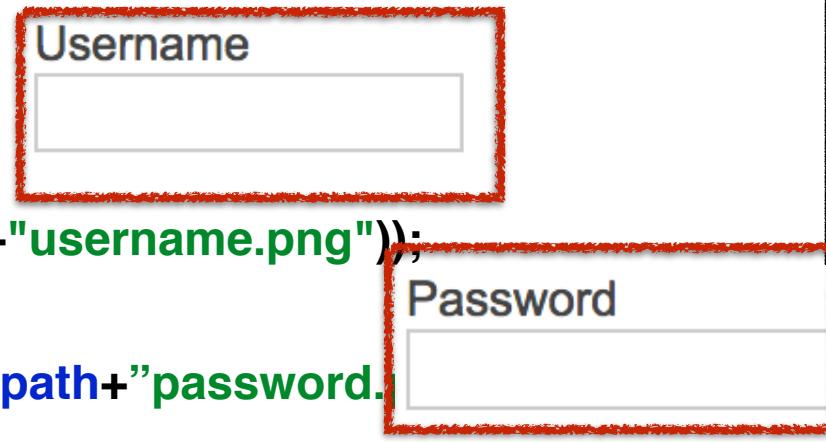
JAutomate

# (Visual) Programmable



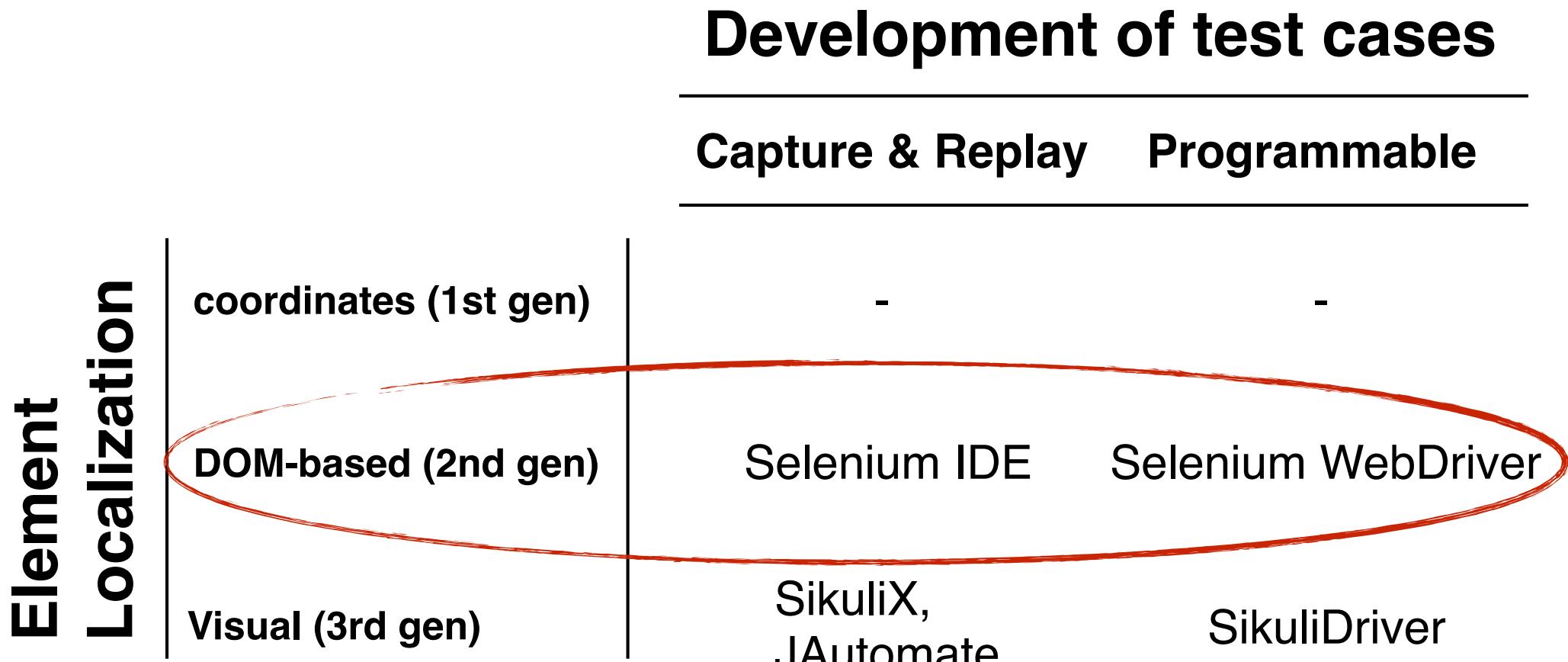
```
public class Login {  
    @Test  
    public void testLogin() {  
        WebDriver driver = new FirefoxDriver();  
        driver.get("http://localhost/claroline1115/index.php");  
  
        ImageTarget user = new ImageTarget(new File(path+"username.png"));  
        sendKeys(user, "John Doe");  
  
        ImageTarget password = new ImageTarget(new File(path+"password.png"));  
        sendKeys(password, "thenameofmycat");  
  
        ImageTarget login = new ImageTarget(new File(path+"login.png"));  
        click(login);  
  
        ...  
    }  
}
```

## Visual Locators



Enter

# Approaches to E2E Test Automation



# DOM-based locators



- **id**
- **css**
- **name**
- **linkText**
- **XPath**

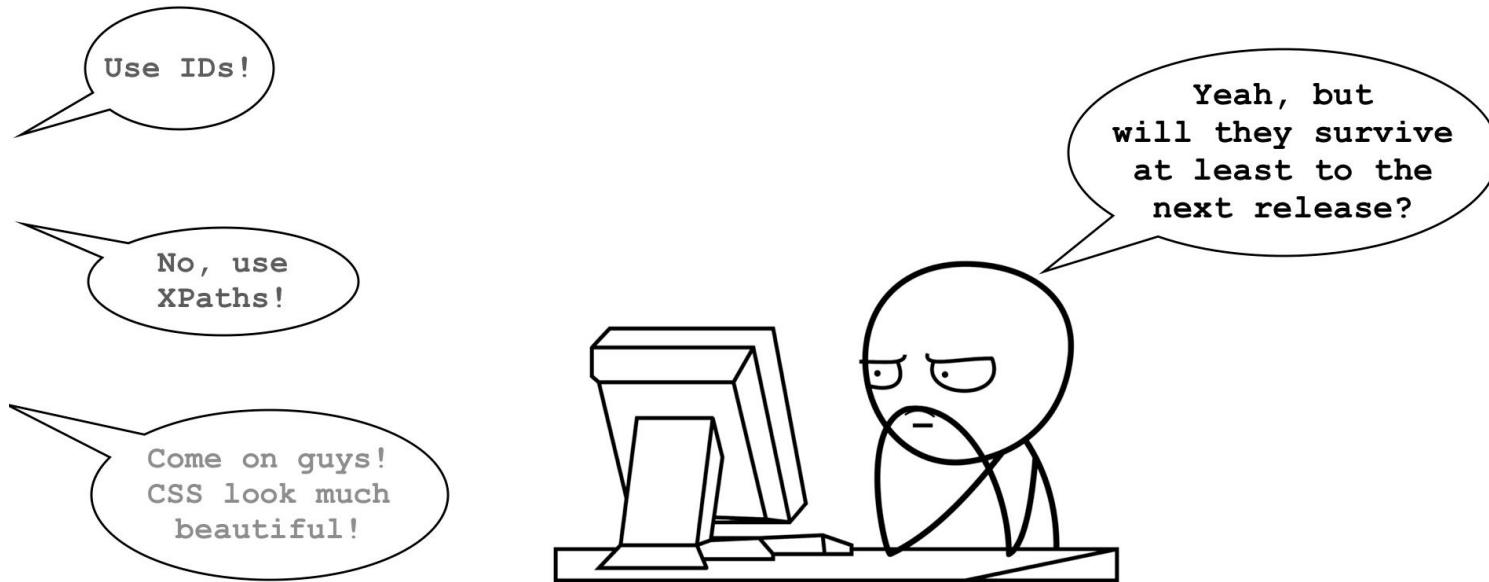
Username:   
Password:

```
<form name="loginform" action="homepage.asp" method="post">
  Username: <input type="text" id="UID" name="username"><br>
  Password: <input type="text" id="PW" name="password">
  <a href="javascript:loginform.submit()">Login</a>
</form>
```

Which one should I use?

```
public void testLogin() {
    WebDriver driver = new FirefoxDriver();
    driver.get("http://www.....com/login.asp");
    driver.findElement(By.id("UID")).sendKeys("John.Doe");
    driver.findElement(By.id("PW")).sendKeys(123456);
    driver.findElement(By.linkText("Login")).click();
    // we are in the 'HomePage.asp' page
    assertEquals("John.Doe", driver.findElement(By.id("uname")).getText());
    driver.close();
}
```

# DOM Locators and SW Evolution



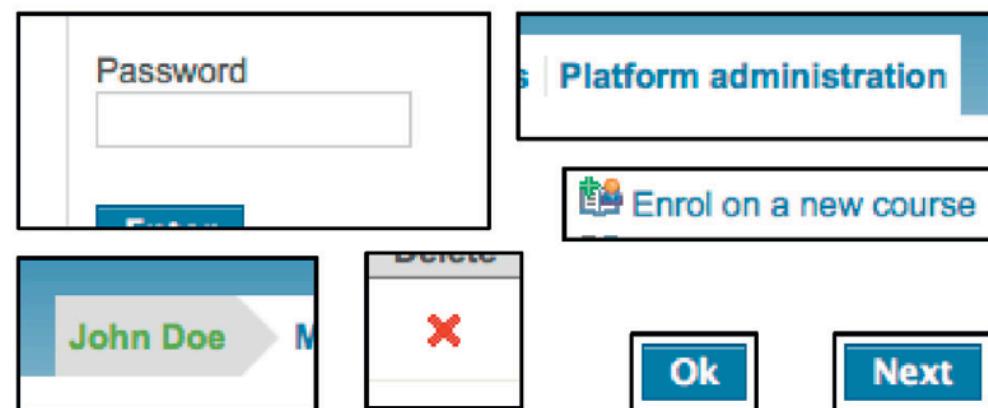
- **ids** are preferable but *(i)* not present *(ii)* auto-generated *(iii)* not unique
- other strategies are applicable **only** in **specific** contexts
- XPath **complex** and **brittle** but **powerful** and **flexible**
- a **careful** choice of locators can save you precious time (and headache)

# Approaches to E2E Test Automation

		<b>Development of test cases</b>	
		<b>Capture &amp; Replay</b>	<b>Programmable</b>
<b>Element Localization</b>	<b>coordinates (1st gen)</b>	-	-
	<b>DOM-based (2nd gen)</b>	Selenium IDE	Selenium WebDriver
	<b>Visual (3rd gen)</b>	SikuliX, JAutomate	SikuliDriver

# Visual locators

manually crafted  
(SikuliX and SikuliAPI)



autogenerated  
(JAutomate)

JAutomate Studio

File Edit Script Settings Reports Help

Record Run Run from Cursor Run Selection Stop

TestLogin.txt | StartWeb "http://www....com/home.asp"

Click Username: [ ] Click Password: [ ] Login

Wait

Write "John.Doe"

Click Username: [ ] Click Password: [ ] Login

Write "123456"

Click Username: [ ] Click Password: [ ] Login

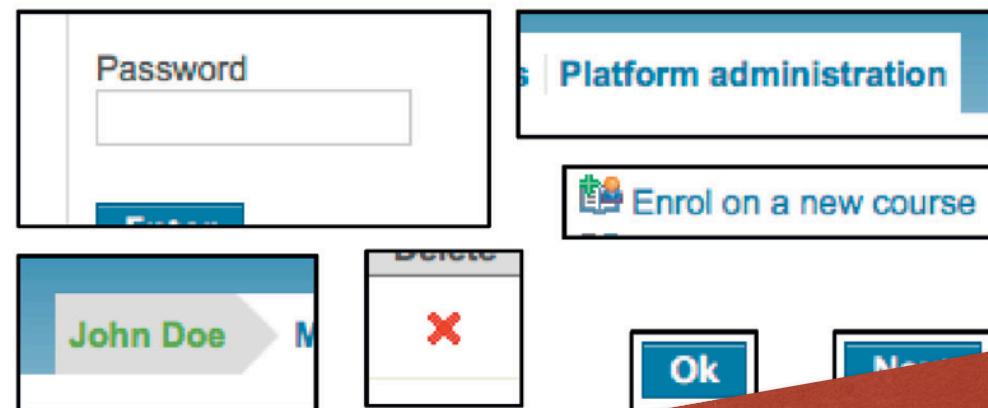
WaitVerify JohnDoe 186 94 16 17

Basic Commands

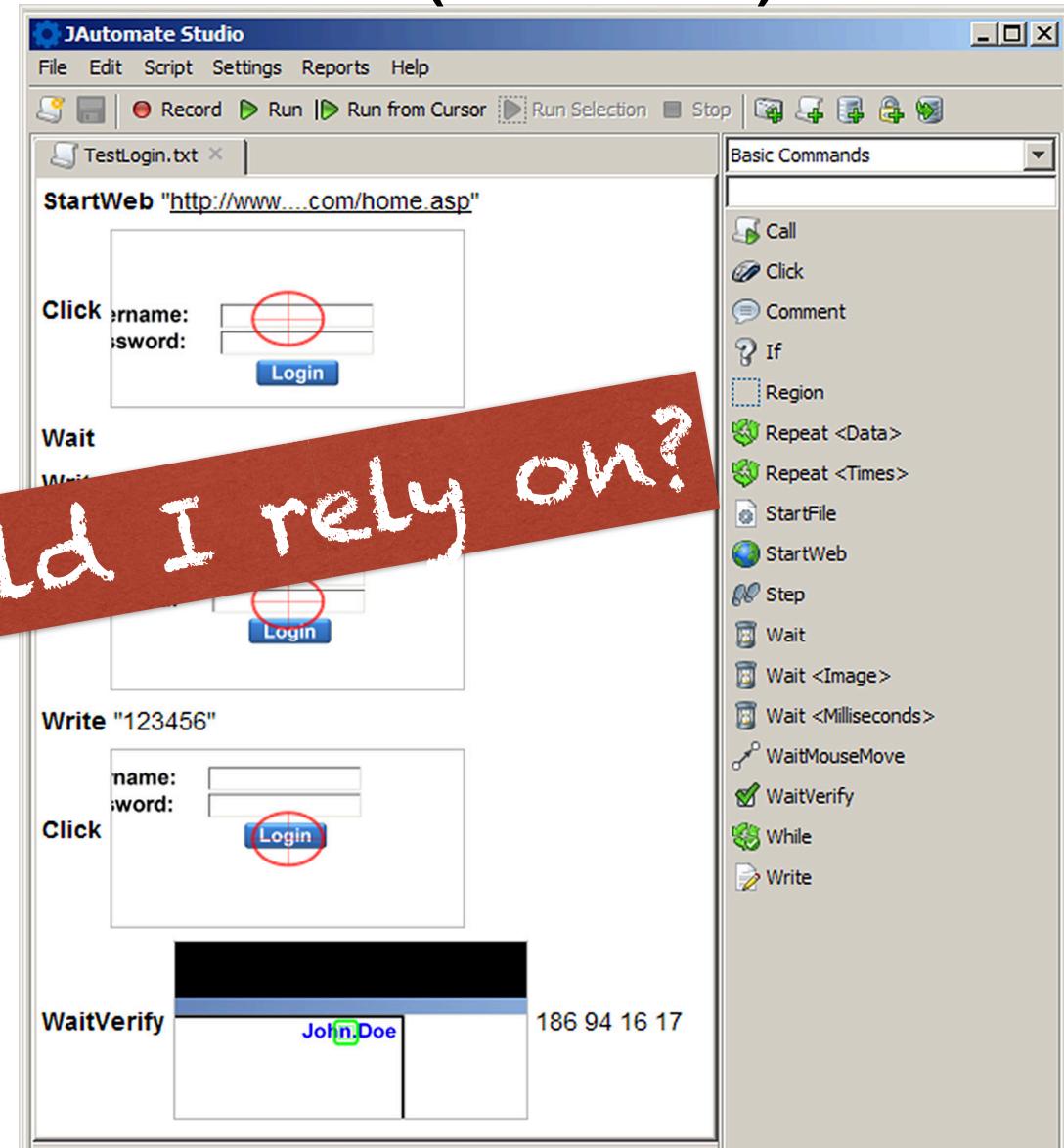
- Call
- Click
- Comment
- If
- Region
- Repeat <Data>
- Repeat <Times>
- StartFile
- StartWeb
- Step
- Wait
- Wait <Image>
- Wait <Milliseconds>
- WaitMouseMove
- WaitVerify
- While
- Write

# Visual locators

manually crafted  
(SikuliX and SikuliAPI)



autogenerated  
(JAutomate)



# Test Case Evolution

# Issues with Test Automation

## Test Scripts

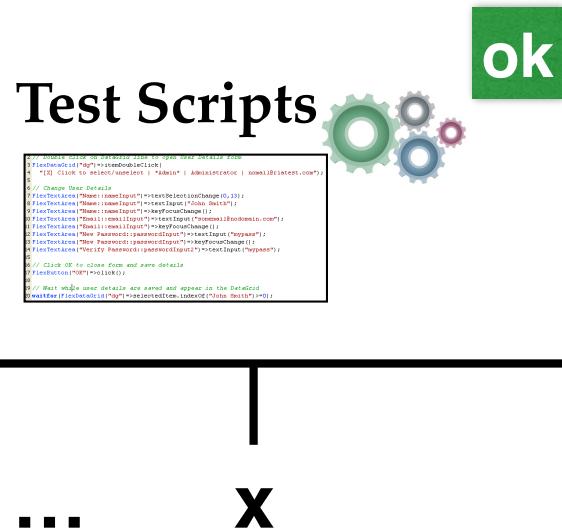


```
#FindMeOnTheWeb("http://www.testautomation.com")
# Click on "User Details" link
# Change User Details
# Enter New Name
# Enter New Email
# Enter New Password
# Verify New Password
# Click OK to close form and save details
# Enter New Name
# Enter New Email
# Enter New Password
# Verify New Password
# Click OK to close form and save details
# Wait for results
# Wait for results
```

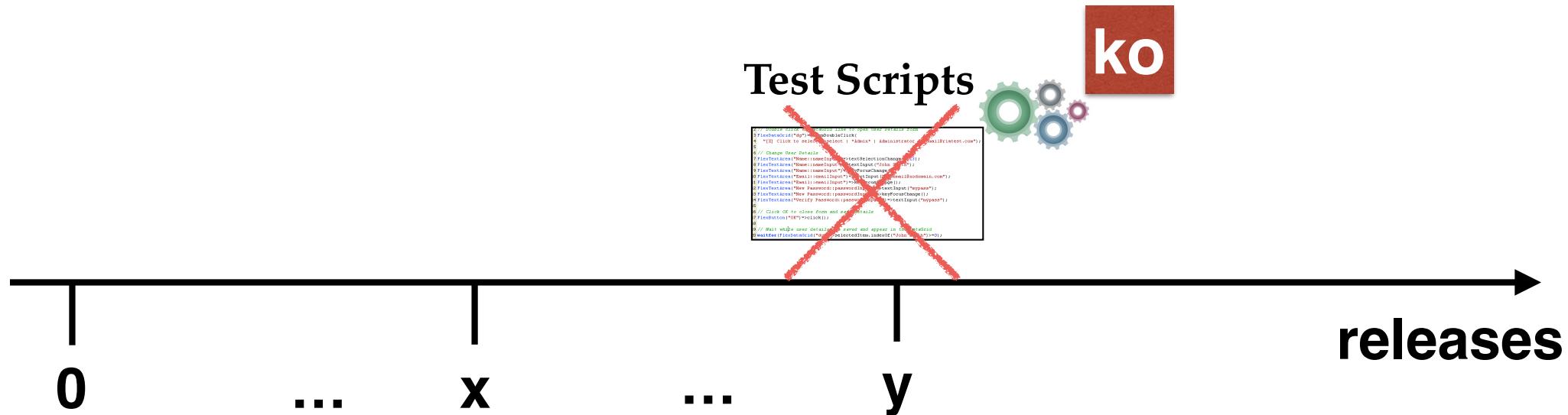
0

→  
releases

# Issues with Test Automation

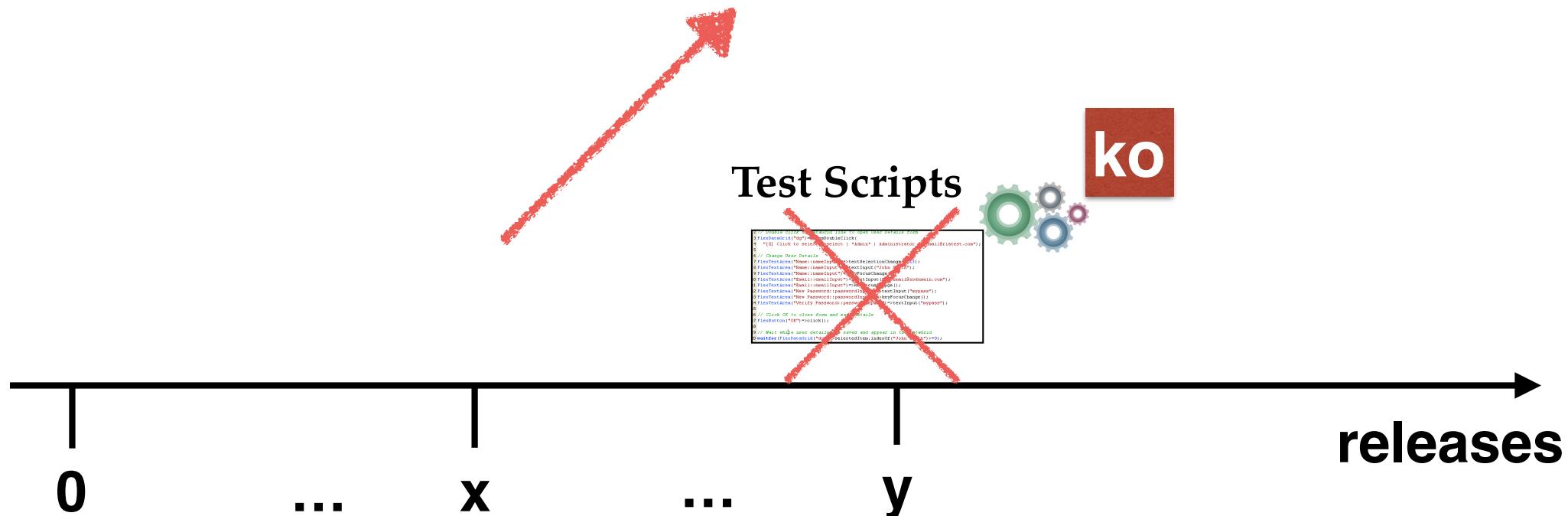


# Issues with Test Automation



# Issues with Test Automation

**in web development, the releases are frequent!**



# Test Scripts Maintenance

- Even **minor changes of the GUI** layout can lead most of the test cases to fail. The repair activity costs e.g., the Accenture company \$120 million per year

[Grechanick et al., "Maintaining and evolving GUI-directed test scripts," ICSE 2009]

- A study on Adobe's Acrobat Reader found that **74%** of its test scripts get broken between **two successive releases**

[Memon and Soffa, "Regression Testing of GUIs," SIGSOFT Software Engineering Notes 28, 2003]

- The **main source** for web test cases **fragility** are the web elements **locators**

[Leotta et al., "Comparing the Maintainability of Selenium WebDriver Test Suites Employing Different Locators: A Case Study.," JAMAICA 2013]

[Christophe et al., "Prevalence and Maintenance of Automated Functional Tests for Web Applications," ICSME 2014]

# Test Evolution Patterns

## Structural Change

### Release N

**Username:**  
**Password:**

**Login**

```
<form name="loginform" action="homepage.asp" method="post">
    Username : <input type="text" id="userid" name="user"><br>
    Password : <input type="text" id="pw" name="password"><br>
    <a href="javascript:loginform.submit()" id="login">Login</a>
</form>
```

---

### Release N+1

**Username:**  
**Password:**

**Login Now!**

```
<form name="loginform" action="homepage.asp" method="post">
    Username : <input type="text" id="userid" name="user"><br>
    Password : <input type="text" id="pw" name="password"><br>
    <a href="javascript:loginform.submit()" id="login">Login Now!</a>
</form>
```

# Test Evolution Patterns

## Logical Change

### Release N

Auth1.php

Username

Password

Login!

### Release N+1

Auth1.php

Username

Birthday

Submit

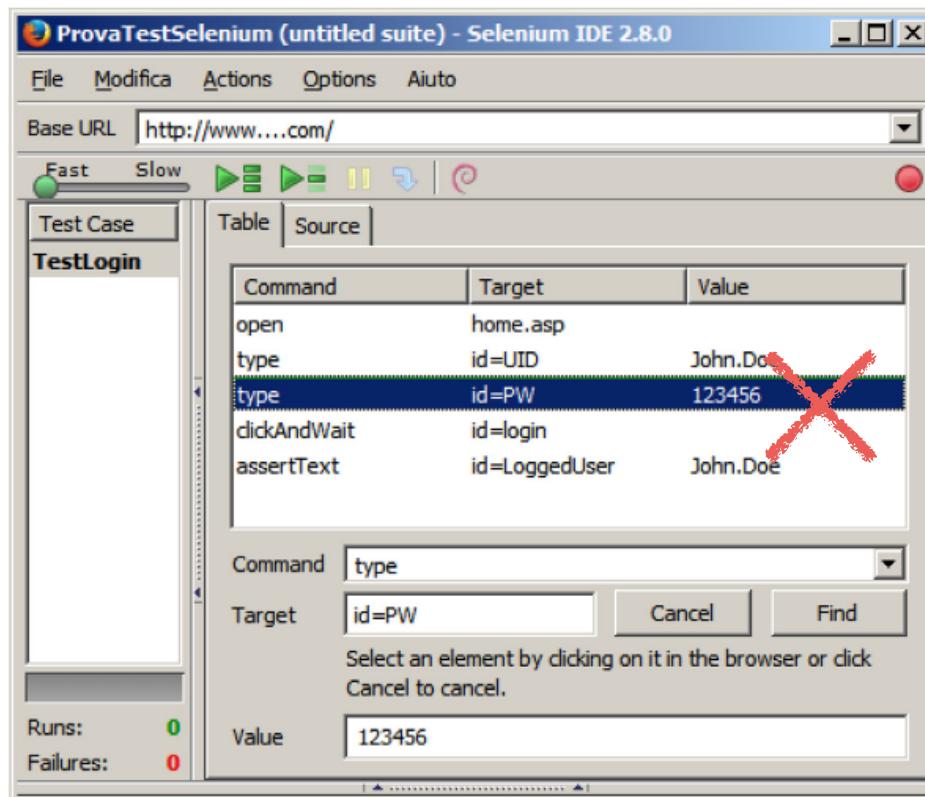
Auth2.php

Password

Login!

# Approaches vs Test Case Evolution

## Scenario1 : C&R Approach + logical change

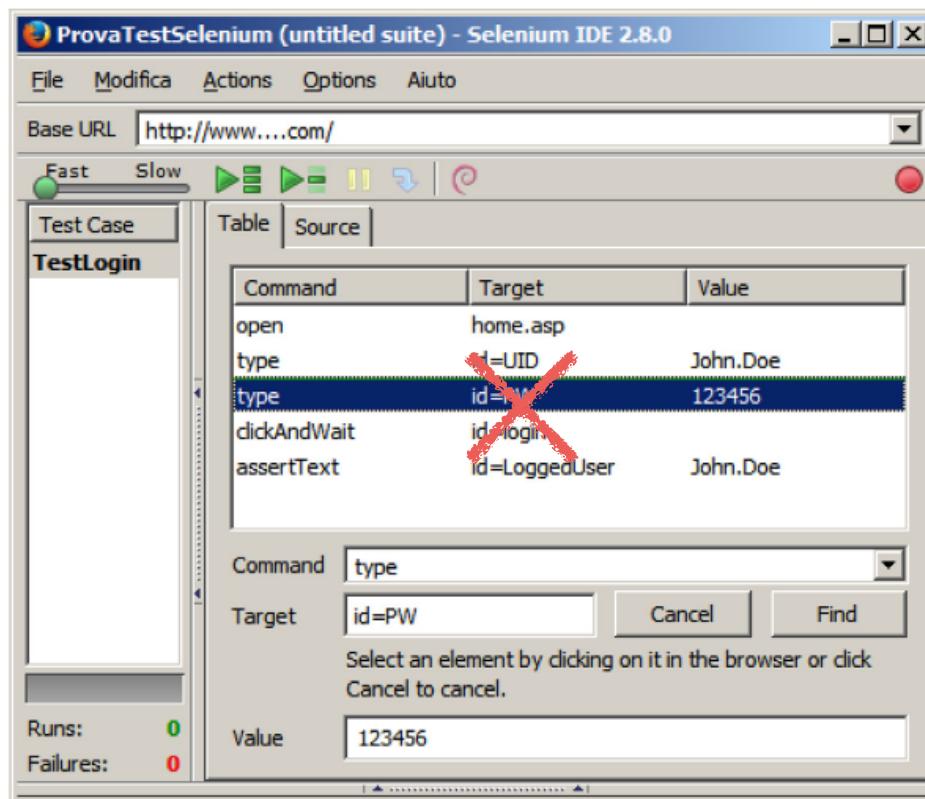


**Keep the portion of the script up to the broken command.**

**Re-capture the execution scenario by starting from the last working command.**

# Approaches vs Test Case Evolution

## Scenario2 : C&R Approach + structural change



**Modify all the broken locators, actions, or values.**

# Approaches vs Test Case Evolution

## Scenario3 : Programmable Approach + logical change

```
public class Login {  
    @Test  
    public void testLogin() {  
        WebDriver driver = new FirefoxDriver();  
        driver.get("http://localhost/claroline1115/index.php");  
        driver.findElement(By.Id(id="user")).sendKeys("John Doe");  
        driver.findElement(By.XPath("xpath="//form/input[2]")).  
        sendKeys("securepassword");  
        driver.findElement(By.LinkText(linkText="Enter")).click();  
        Assert.assertEquals(driver.findElement(By.Id("Jonn Doe")),  
        "Login was successful");  
    }  
}
```

```
public class IndexPage {  
    private final WebDriver driver;  
    @FindBy(id="user")  
    private WebElement username;  
    @FindBy(xpath="//form/input[2]")  
    private WebElement password;  
    @FindBy(linkText="Enter")  
    private WebElement login;  
    @FindBy(id="loggedUser")  
    private WebElement loggedUser;  
  
    public UserAccount goToCreateAccount(){  
        createUserAccount.click();  
        return new UserAccount(driver);  
    }  
  
    public HomePage login(String usr, String pwd) {  
        username.sendKeys(usr);  
        password.sendKeys(pwd);  
        login.click();  
        return new HomePage(driver);  
    }  
}
```

**Modify the page object actions  
and, eventually, the tests.**

# Approaches vs Test Case Evolution

## Scenario4 : Programmable Approach + structural change

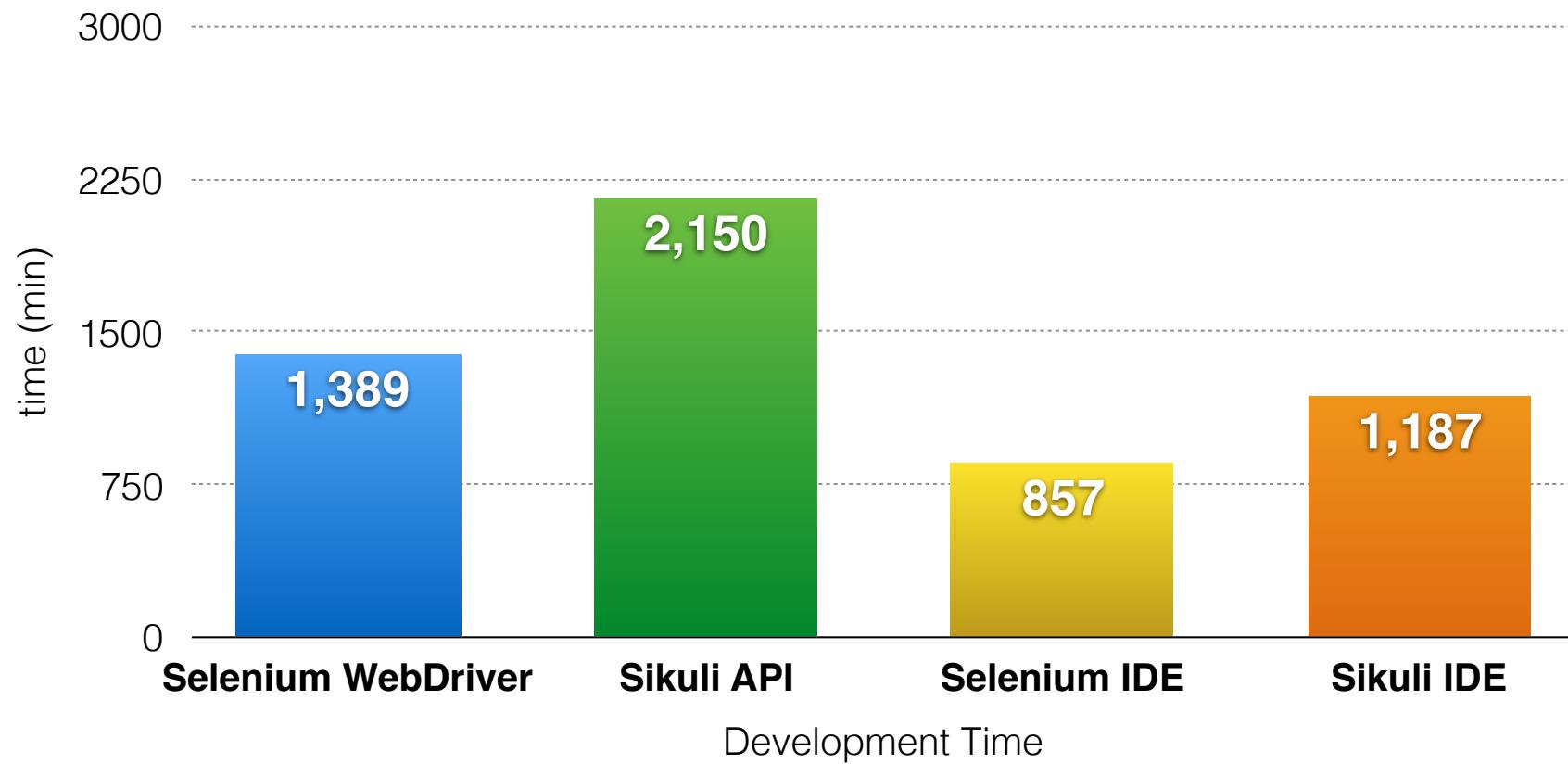
```
public class Login {  
    @Test  
    public void testLogin() {  
        WebDriver driver = new FirefoxDriver();  
        driver.get("http://localhost/claroline1115/index.php");  
        driver.findElement(By.Id(id="user")).sendKeys("John Doe");  
        driver.findElement(By.XPath("xpath="//form/input[2]"))  
            .sendKeys("securepassword");  
        driver.findElement(By.LinkText(linkText="Enter")).click();  
        Assert.assertEquals(driver.findElement(By.Id("John Doe")),  
            "Login was successful");  
    }  
}
```

```
public class IndexPage {  
    private final WebDriver driver;  
    @FindBy(id="user")  
    private WebElement username;  
    @FindBy(xpath="//form/input[2]")  
    private WebElement password;  
    @FindBy(linkText="Enter")  
    private WebElement login;  
    @FindBy(id="loggedUser")  
    private WebElement loggedUser;  
  
    public UserAccount goToCreateAccount(){  
        createUserAccount.click();  
        return new UserAccount(driver);  
    }  
  
    public HomePage login(String usr, String pwd) {  
        username.sendKeys(usr);  
        password.sendKeys(pwd);  
        login.click();  
        return new HomePage(driver);  
    }  
}
```

**Modify only the page object(s).**

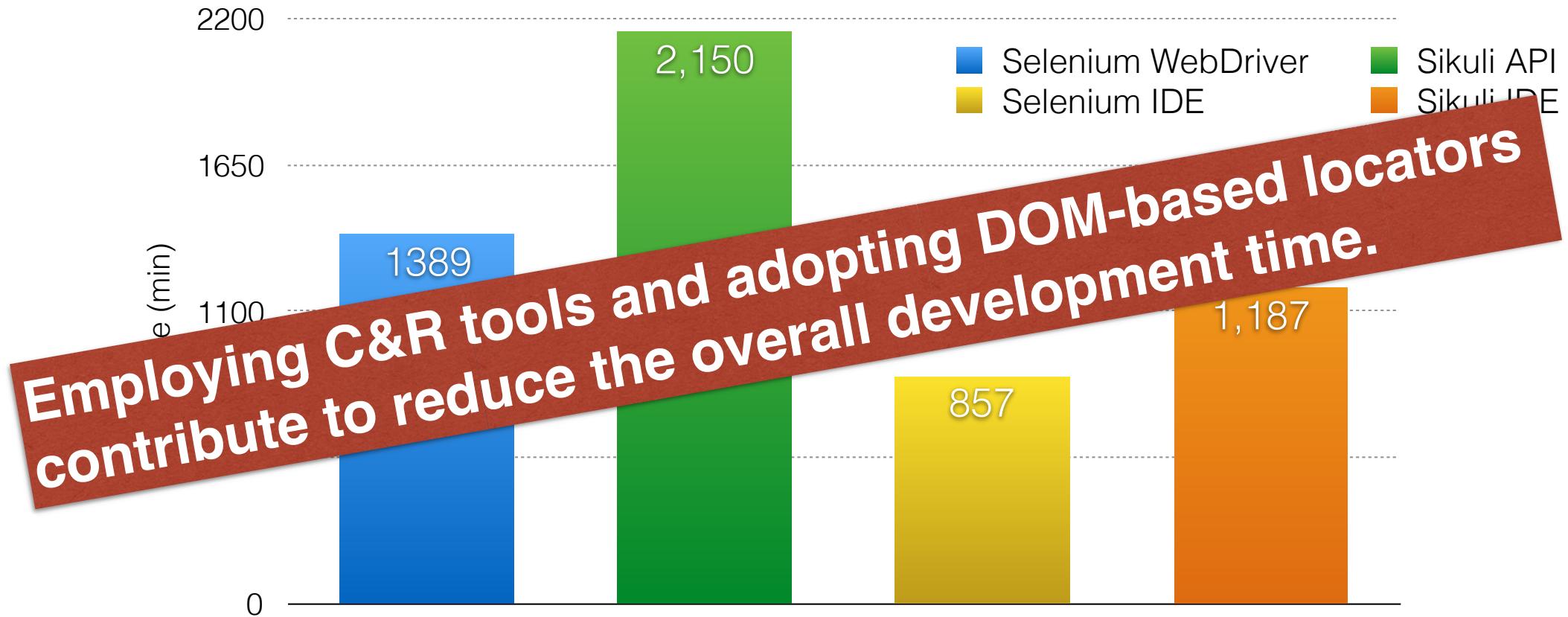
**No modifications to the tests needed!!!**

# Test Suite Development Cost



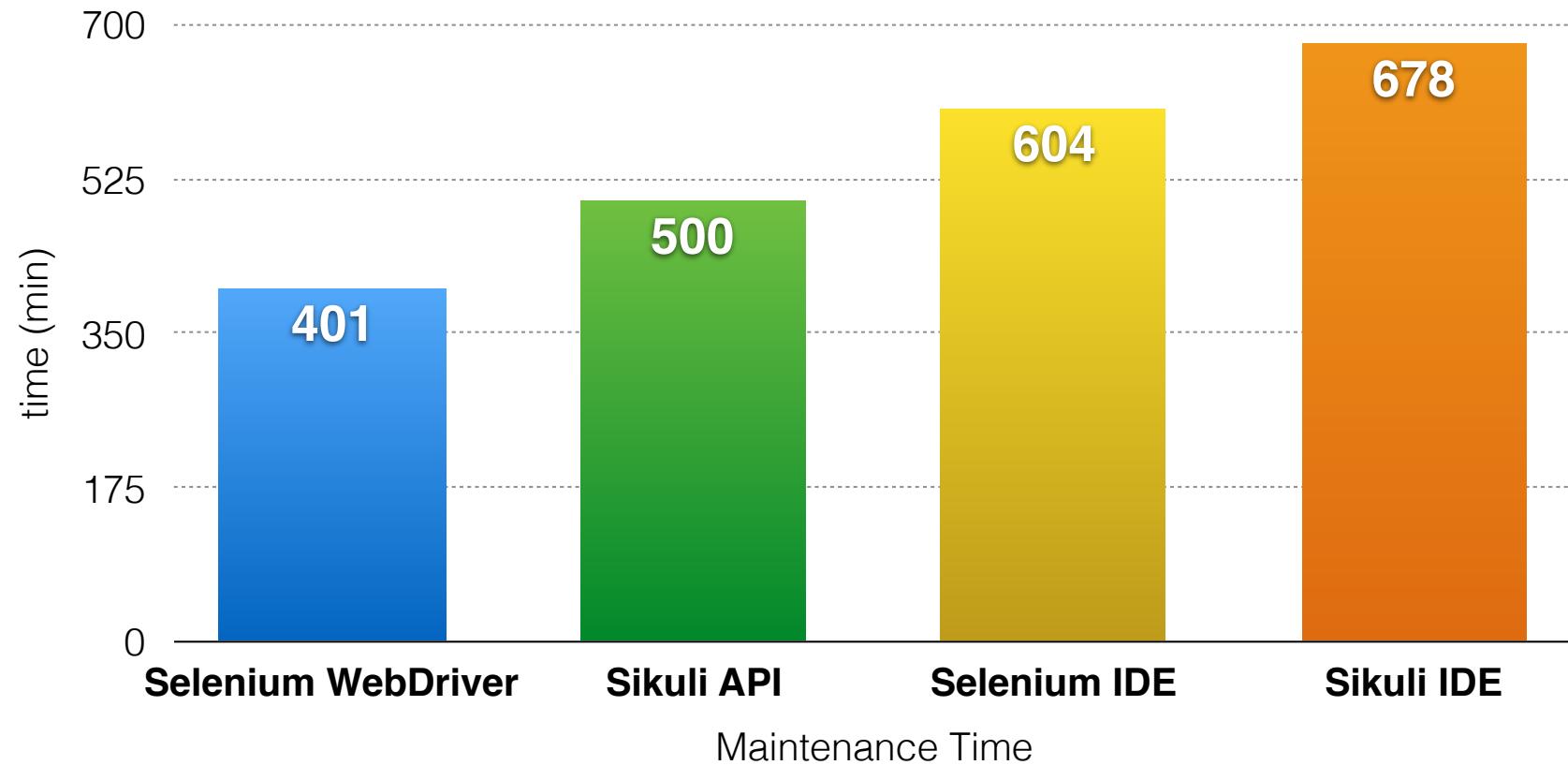
[Leotta et al., “*Approaches and Tools for Automated End-to-End Web Testing*,” Advances in Computers, Volume 101:5 (2016)]

# Test Suite Development Cost



[Leotta et al., “*Approaches and Tools for Automated End-to-End Web Testing*,” Advances in Computers, Volume 101:5 (2016)]

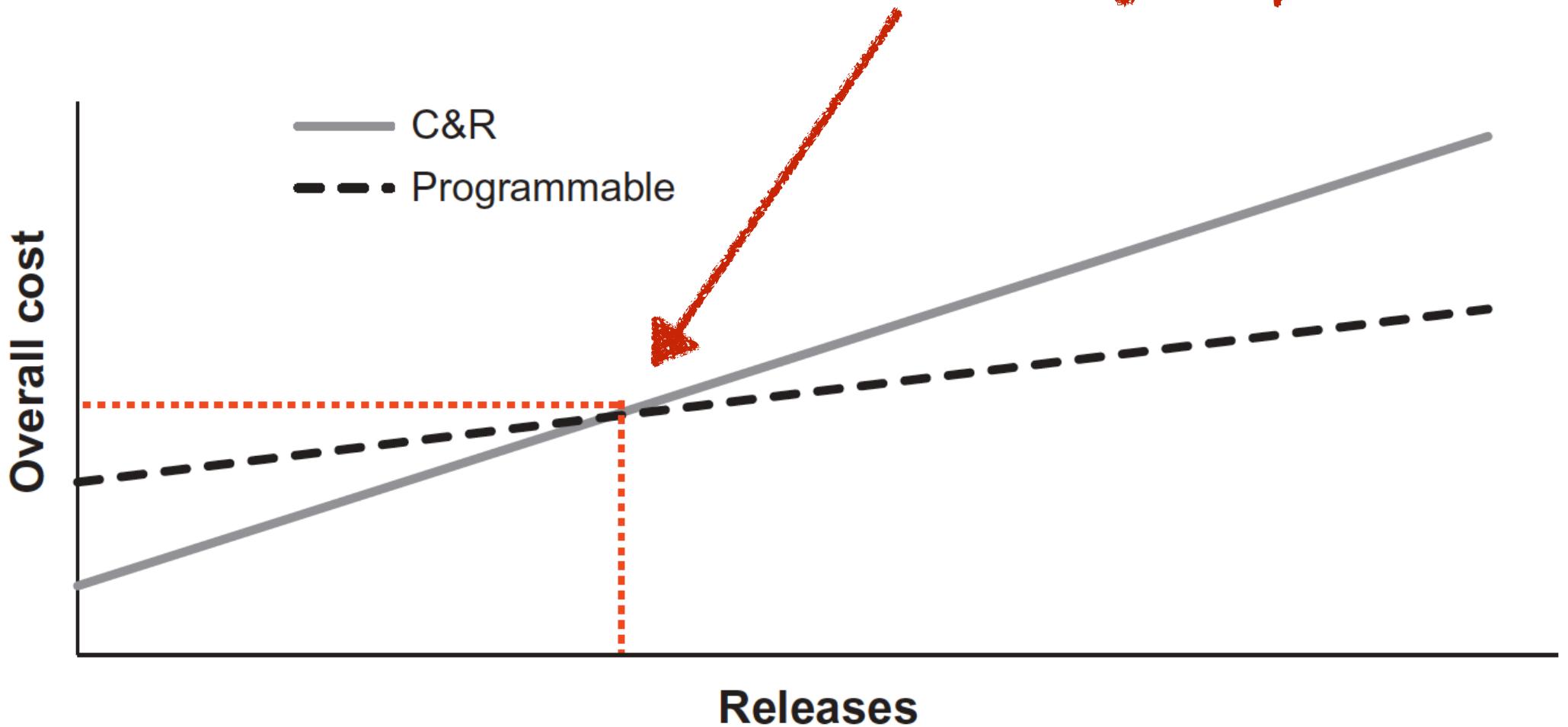
# Test Suite Maintenance Cost



[Leotta et al., “*Approaches and Tools for Automated End-to-End Web Testing*,” Advances in Computers, Volume 101:5 (2016)]

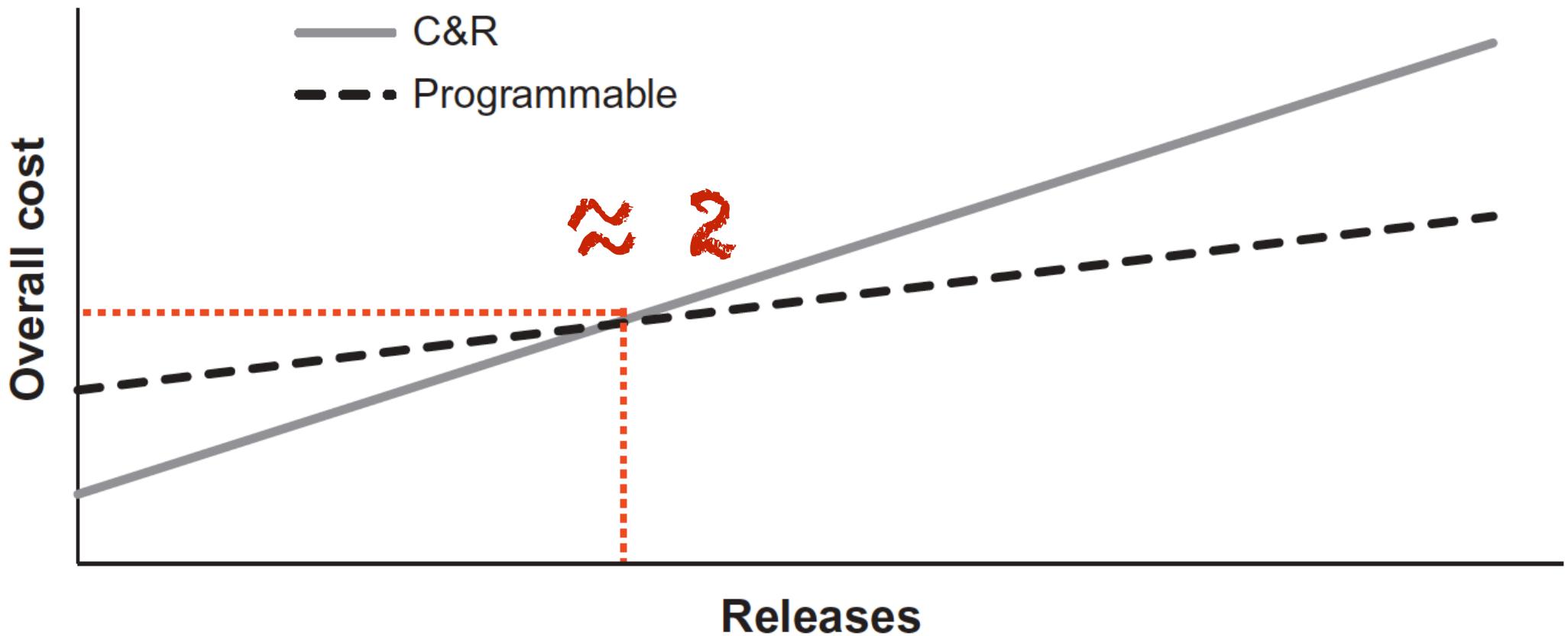
# When do they become convenient?

What number would you put here?



[Leotta et al., "Approaches and Tools for Automated End-to-End Web Testing," Advances in Computers, Volume 101:5 (2016)]

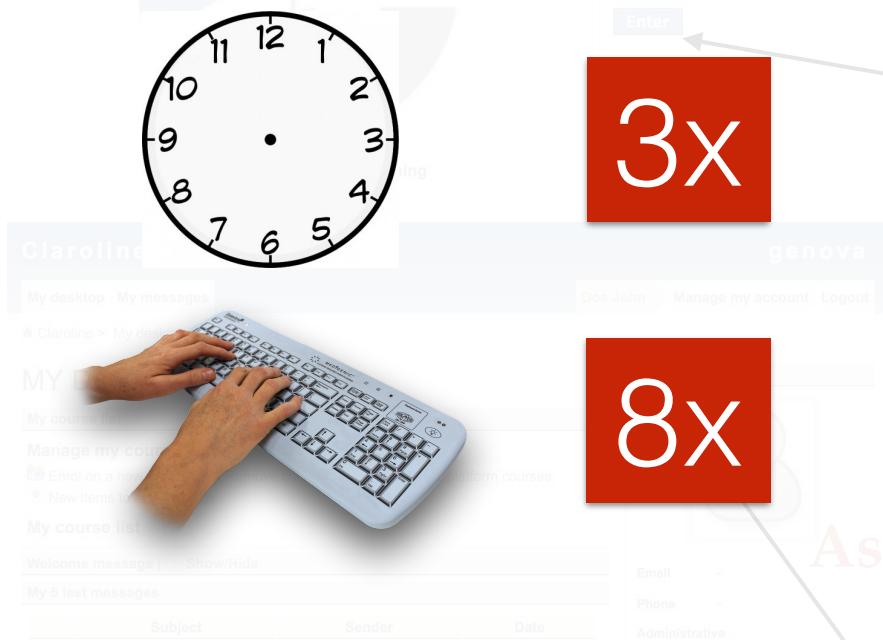
# When do they become convenient?



[Leotta et al., "Approaches and Tools for Automated End-to-End Web Testing," Advances in Computers, Volume 101:5 (2016)]

# Why?

## Augmented Maintainability



[Leotta et al., “*Improving Test Suites Maintainability with the Page Object Pattern: An Industrial Case Study*,” ICSTW 2013]

// A simple Page Object class with Selenium WebDriver

```
public class IndexPage {  
    private final WebDriver driver;
```

```
@FindBy(id="user")
```

```
private WebElement username;
```

```
@FindBy(xpath="//form/input[2]")
```

```
private WebElement password;
```

```
@FindBy(linkText="Enter")
```

```
private WebElement login;
```

```
@FindBy(id="loggedUser")
```

```
private WebElement loggedUser;
```

```
public HomePage login(String usr, String pwd) {
```

```
    username.sendKeys(usr);
```

```
    password.sendKeys(pwd);
```

```
    login.click();
```

```
    return new HomePage(driver);
```

```
}
```

```
}
```

```
@Test
```

```
public void testLogin {
```

```
    Utils.connectToWebAppUnderTest();
```

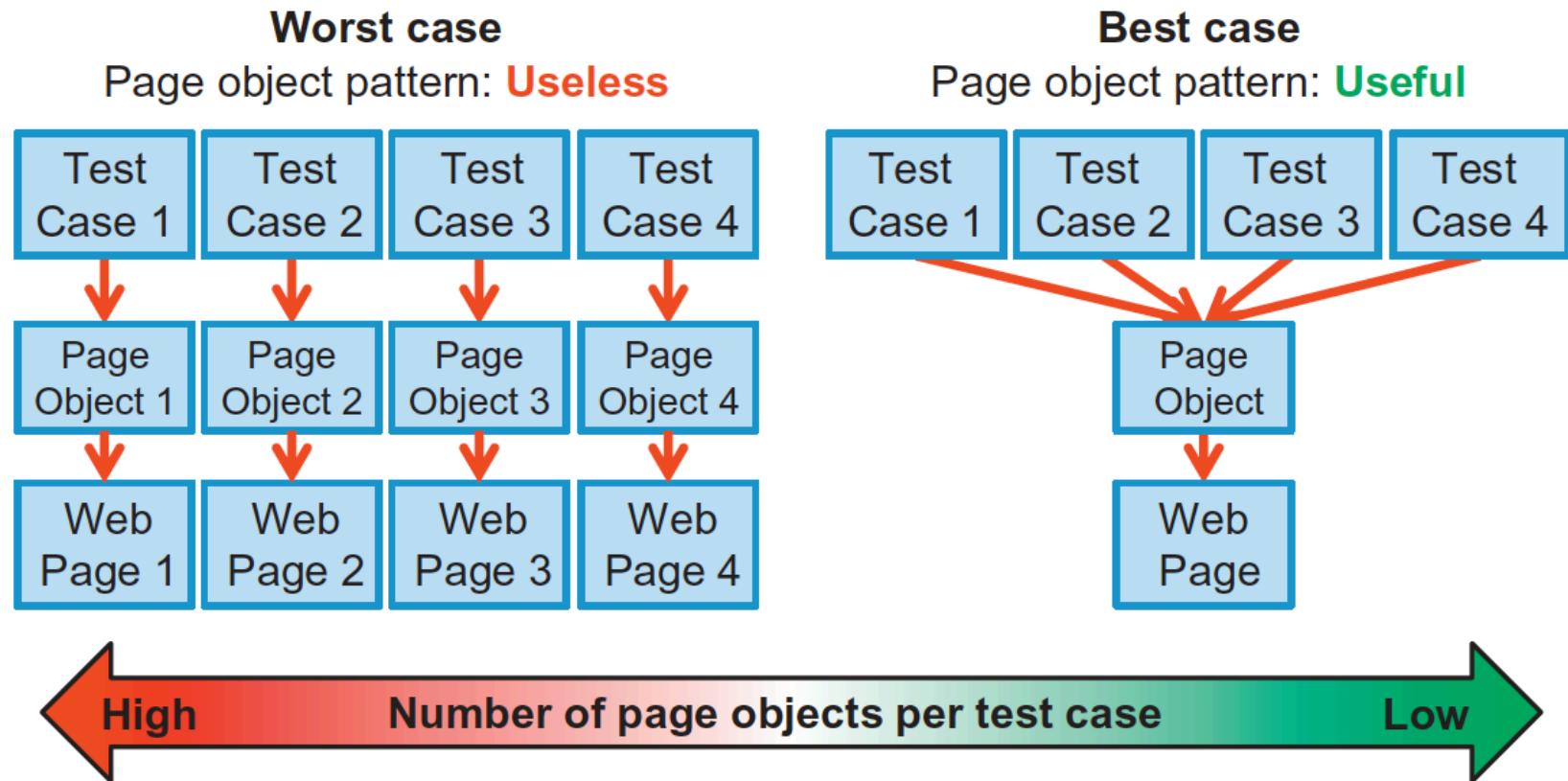
```
    IndexPage IP = new IndexPage(driver);
```

```
    HomePage hp = IP.login("John Doe", "securepassword");
```

```
    assertTrue(hp.checkLoggedUser("Doe John"));
```

```
}
```

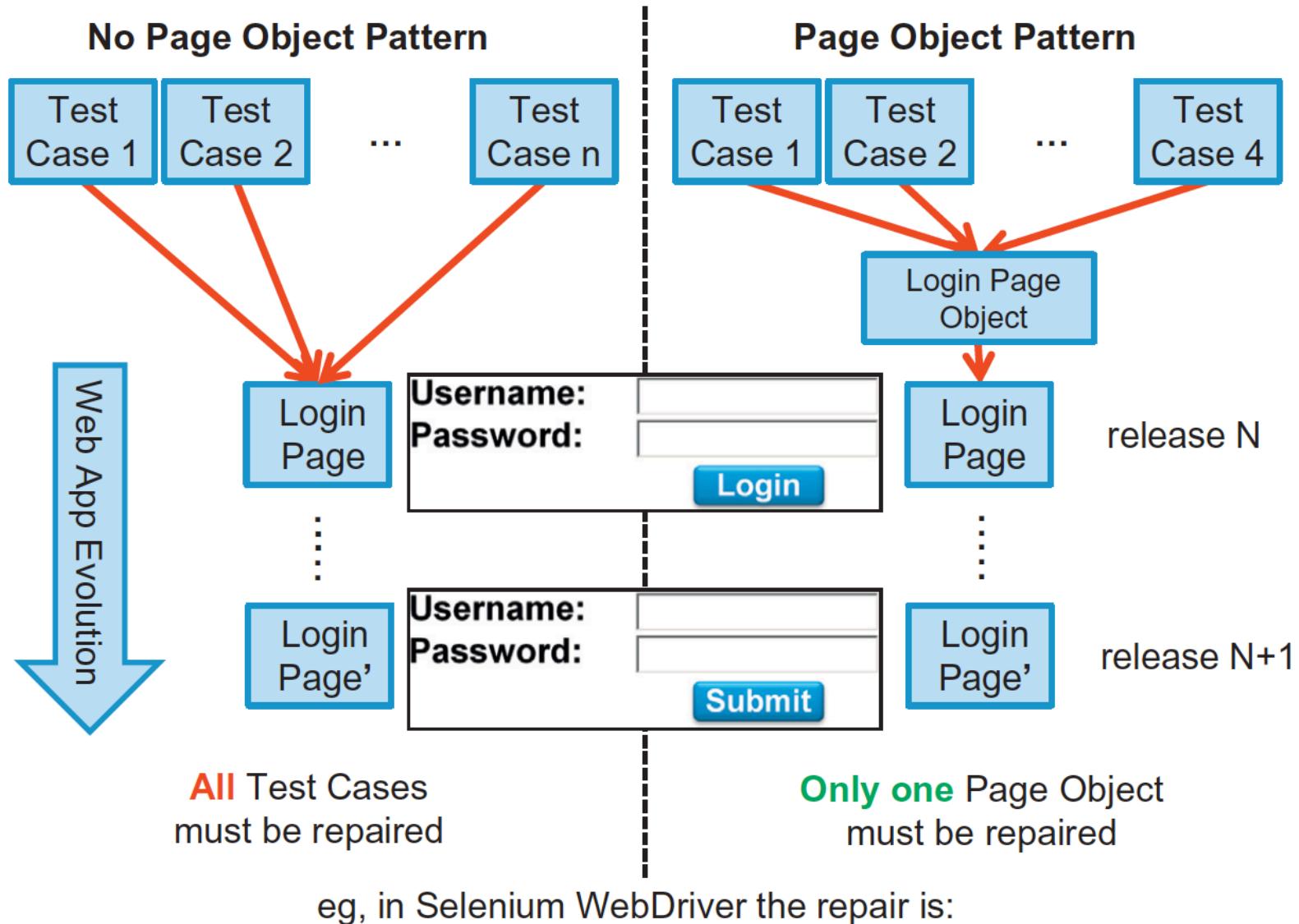
# Factor 1 : Test cases vs Page objects



**The web page modularity of the web application under test affects the benefits of programmable test cases.**

**Reusable pages objects are maintained less during software evolution.**

# Factor 2 : Page objects and repair effort



@FindBy(linkText="Login") becomes @FindBy(linkText="Submit")

# Factor 3 : Number of Locators

## Elements with complex visual interaction

**Add Car Test Case:**

Select the car manufacturer

**DOM-based approach**  
one locator needed

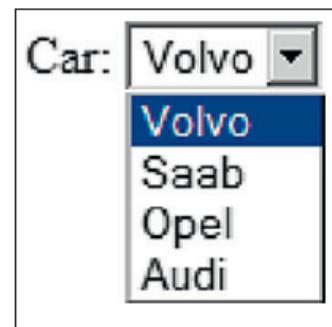
```
(driver.findElement(By.id("carList")))  
    .selectByVisibleText("Saab");
```

**Visual approach**  
two locators needed

Locate the Select  
Web Element  
+  
Click to Open



Locate the  
Option Value



# Factor 3 : Number of Locators

## Multistate Elements

### Add Car Test Case

Check the Box and Submit

### Remove Car Test Case

Uncheck the Box and Submit

I have a car

I have a car

The Target Web Element is always the same  
(ie, the Box)

### Visual approach

two locators needed



### DOM-based approach

one locator needed

`//input[@name="vehicle"]`

# Factor 3 : Number of Locators

## Data Driven Test Cases



### Examples of DOM-based Locators

```
//input[2]
linkText = Platform administration
//td[contains(text(),'John Doe')]
//img[@src="/web/img/edit.png?1232379976"]
//*[@id='body']/table[2]/tbody/tr/td[4]/a/img[4]
```

# Other Factors

**Robustness** favours the DOM based

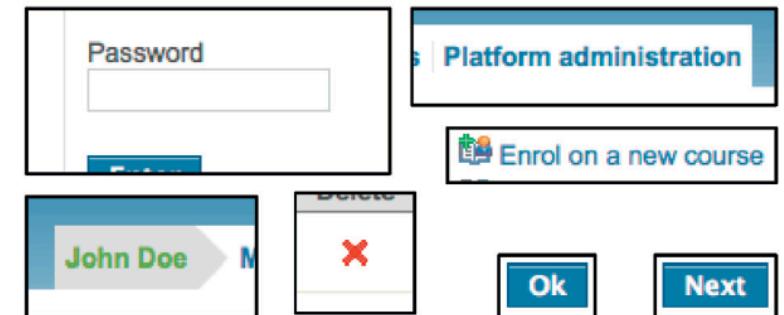
**Execution Time**: almost the same

**Multiple Instances** of the same web element

**Invisible/Hidden elements**

Web page loading/**synchronization**

**Comprehensibility**



Examples of DOM-based Locators

```
//input[2]
linkText = Platform administration
//td[contains(text(),'John Doe')]
//img[@src="/web/img/edit.png?1232379976"]
//*[@id='body']/table[2]/tbody/tr/td[4]/a/img[4]
```

# Can we do better?

## **Hybrid Testing**

join the advantages of visual and DOM-based tools  
in a single testing framework

## **Contextual Clues**

write the test script in natural language

## **Much still needs to be done!!!**