

Writing Maintainable Test Automation



Marcel de Vries
Global MD & CTO Xebia|Xpirit

@marcelv | <https://fluentbytes.com>

**Microsoft
Partner**

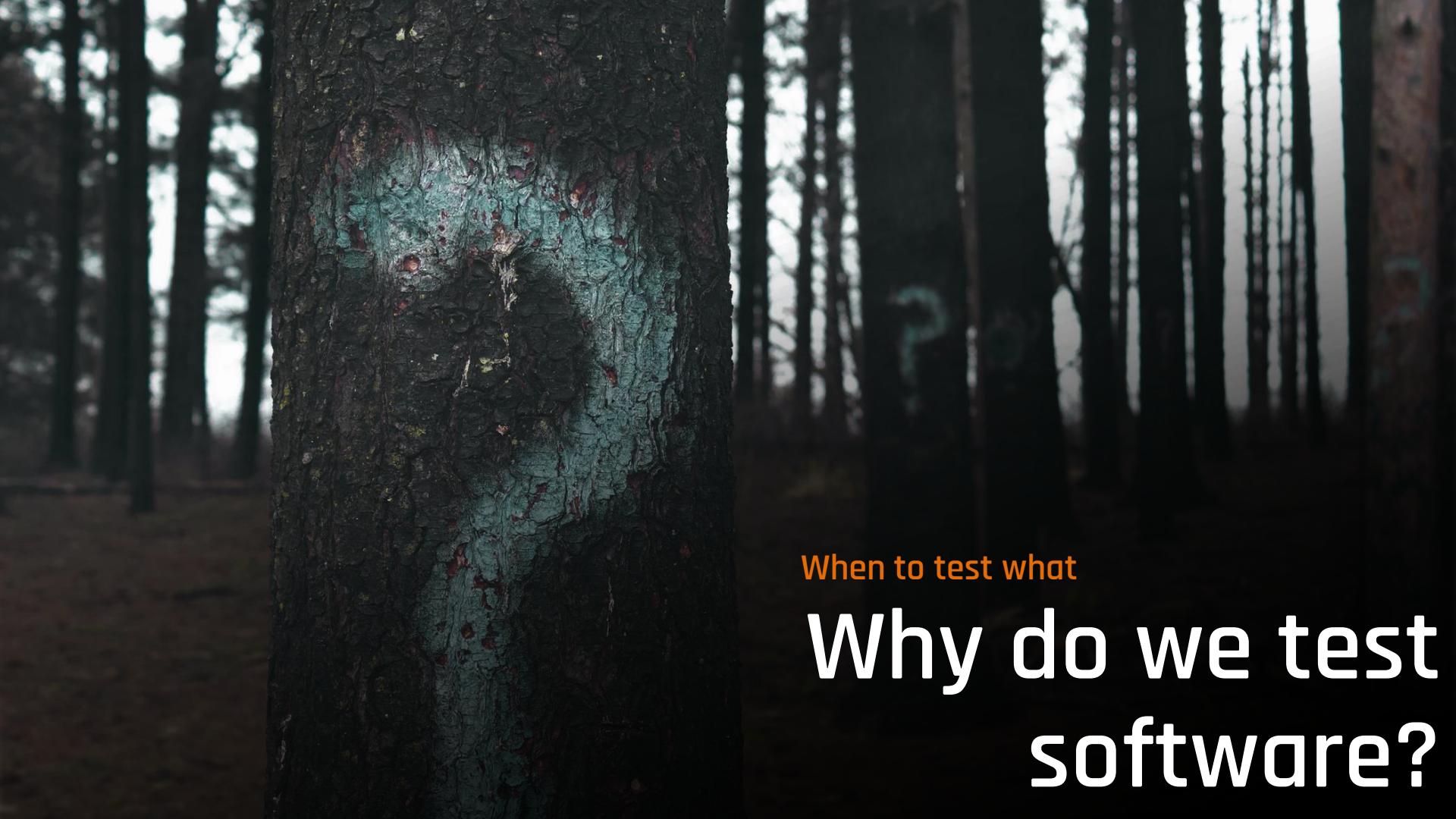
Gold DevOps
Gold Cloud Platform
Gold Data Analytics
Gold Application Integration
Gold Application Development

GitHub Verified Partner



A close-up photograph of a row of ornate chess pieces on a light-colored wooden board. The King piece is in sharp focus in the foreground, its head turned back over its shoulder to look over the other pieces. Behind it, the Queen, Rook, Bishop, Knight, and Pawns are visible in a slightly blurred perspective.

Challenges: Automation Maintainability When to test what

The background of the slide is a dark, atmospheric photograph of a forest. In the foreground, a large tree trunk is shown in sharp focus, its bark dark and textured. Behind it, many other tree trunks are visible as vertical, lighter-colored silhouettes against a darker background, creating a sense of depth and density.

When to test what

Why do we test
software?



When to test what

WHAT do we test?

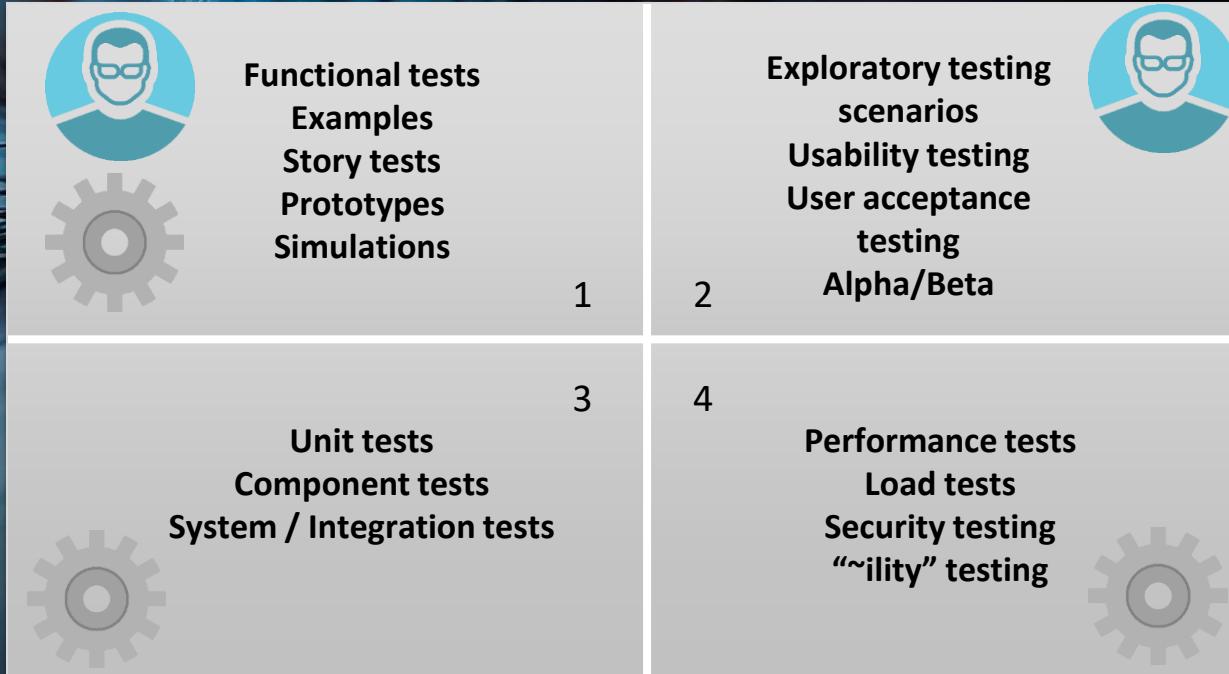
When to test what

Categorizing tests

Business facing

Supporting the Team

Critique Product



Technology Facing



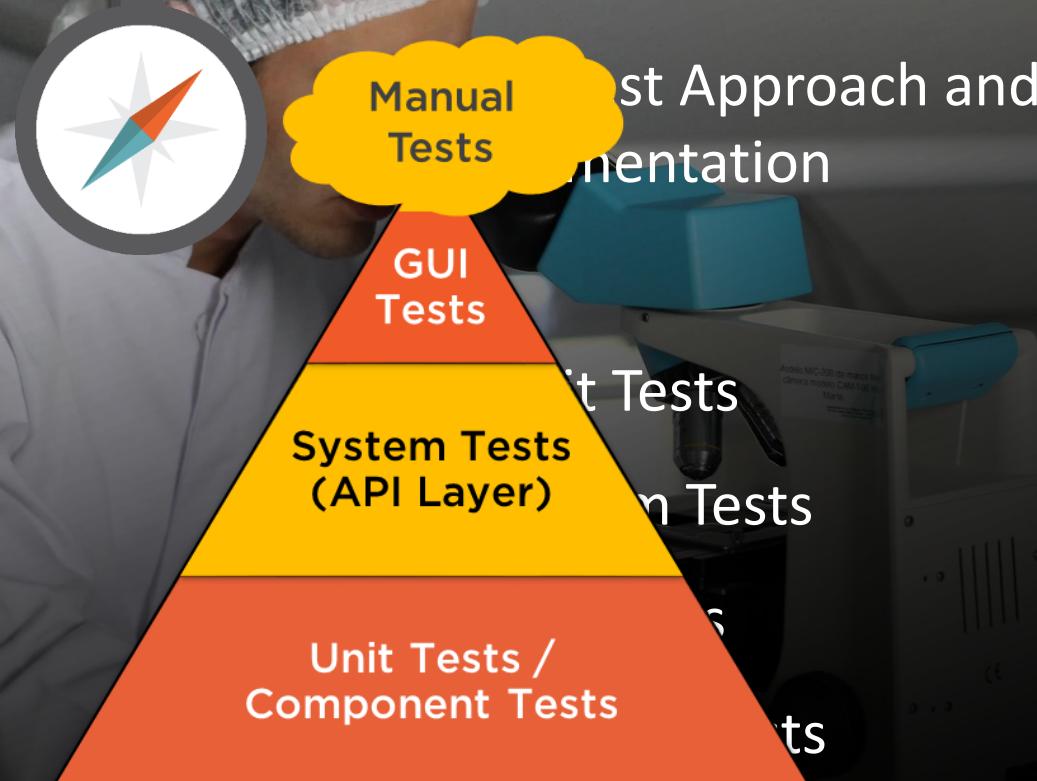
Automation

HOW do we test?

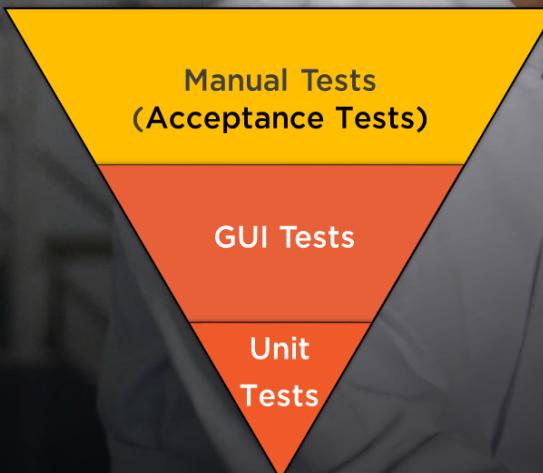


Demo: How we test

Ideal Test Automation Pyramid



The harsh reality!

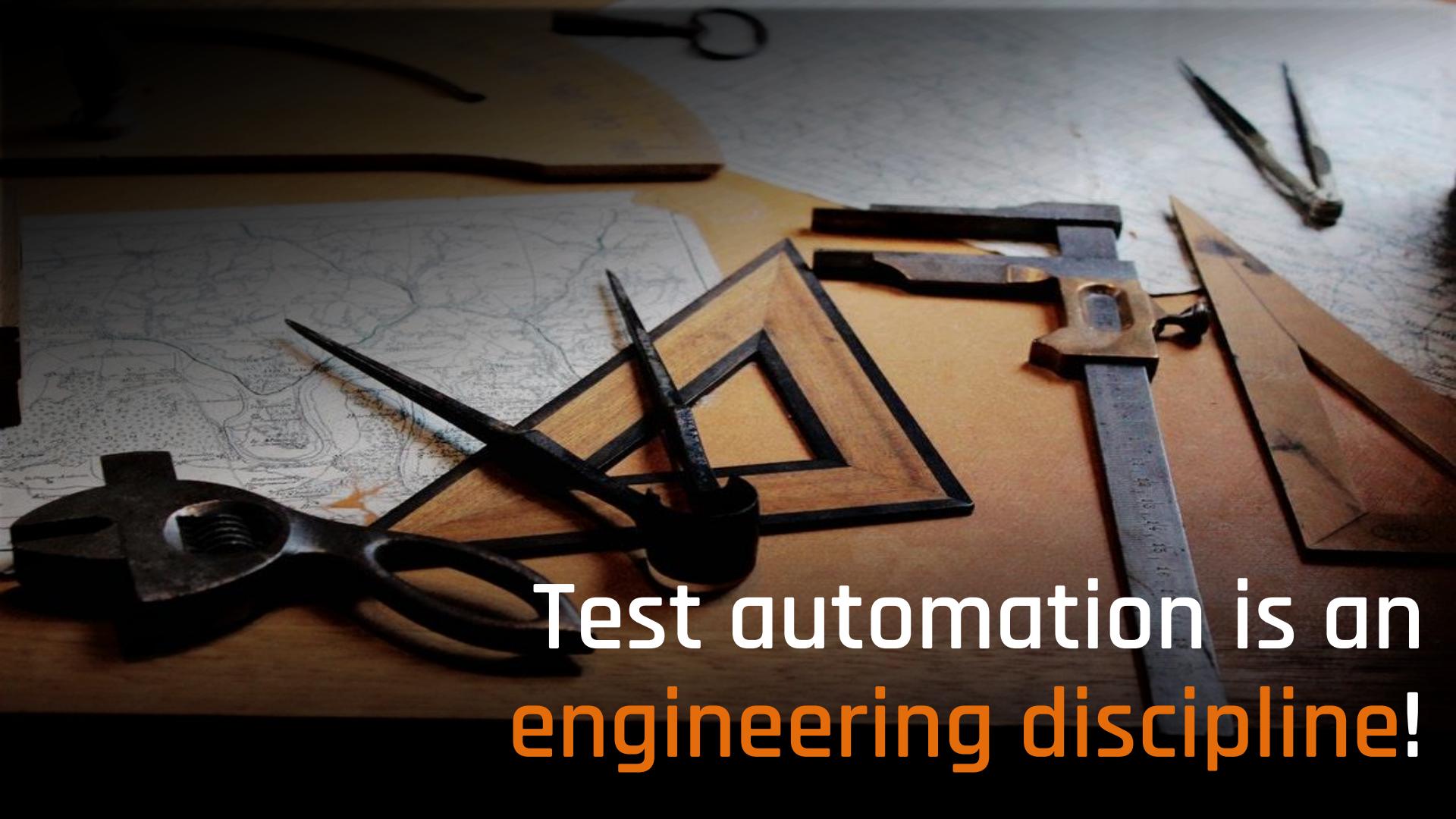


- Many tests manually executed
- Often end-to-end scenarios
- Captured as documents
- No automation or brittle automation through UI testing



Demo: How we automate

DONE!



Test automation is an
engineering discipline!

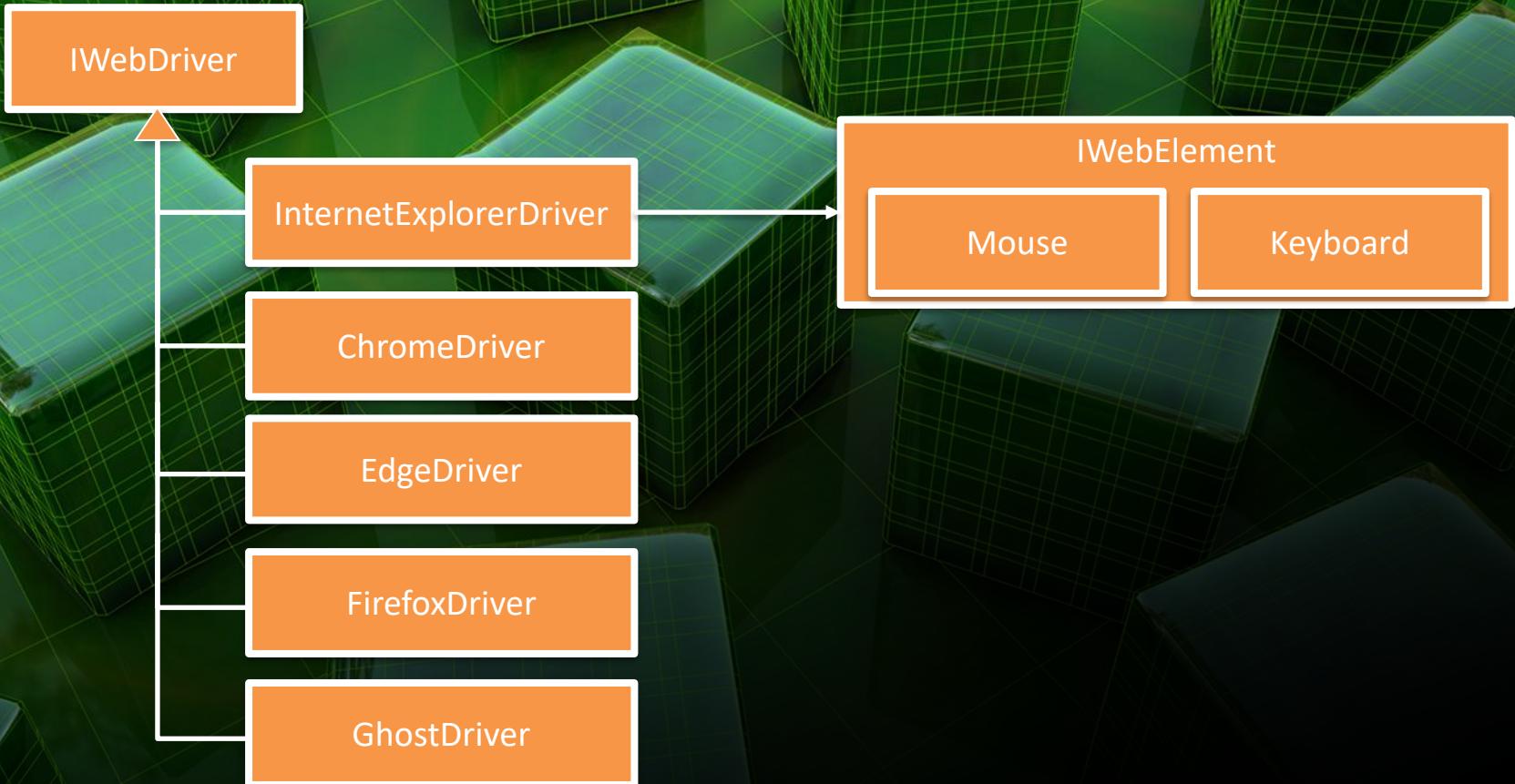
A blacksmith is shown working in his forge. He is wearing a cap and safety glasses, focused on his work. The forge is filled with various tools hanging from the ceiling and a large window in the background. The scene is dimly lit, with strong light coming from the window, creating a dramatic effect.

We need to apply good
coding practices to
Test Automation



Very short intro to the object
models (Selenium)

Object Model Selenium



How Controls Are Located

```
IWebDriver driver = new EdgeDriver();
driver.Url = "http://localhost:5266/";
driver.Navigate();

var btn = driver.FindElement(
    By.CssSelector("tr:nth-child(2) .btn-primary"))

btn.Click();
```

Some notes on finding controls

Finding elements by ID is most robust

Finding on CSS or Class

Issue when redesign of the website

CssSelector is ok as long as you search for Unique stuff

Finding on Text or Partial Text

Issue when different languages

Not all drivers are the same!

Firefox might find by ID and chrome needs Name or vice versa...

```
31     def __init__(self, path=None):
32         self.file = None
33         self.fingerprints = set()
34         self.logduplicates = True
35         self.debug = debug
36         self.logger = logging.getLogger(__name__)
37         if path:
38             self.file = open(os.path.join(path, "fingerprint.log"), "a")
39             self.file.seek(0)
40             self.fingerprints.update(f.readline() for f in self.file)
41
42     @classmethod
43     def from_settings(cls, settings):
44         debug = settings.getbool("superfluous_requests")
45         return cls(job_dir(settings), debug)
46
47     def request_seen(self, request):
48         fp = self.request_fingerprint(request)
49         if fp in self.fingerprints:
50             return True
51         self.fingerprints.add(fp)
52         if self.file:
53             self.file.write(fp + os.linesep)
54
55     def request_fingerprint(self, request):
56         return request_fingerprint(request)
```

maintainability

Maintainable tests are:

DRY

maintainability

```
31     def __init__(self, file=None, logduplicates=True, debug=False):
32         self.file = None
33         self.fingerprints = set()
34         self.logduplicates = True
35         self.debug = debug
36         self.logger = logging.getLogger(__name__)
37         if path.exists(self.file):
38             self.file = open(self.file, "r")
39             self.file.seek(0)
40             self.fingerprints.update([line.strip() for line in self.file])
41
42     @contextmanager
43     def from_settings(settings):
44         debug = settings.getboolean("SUPERFLUOUS_DEBUG")
45         return self(job_dir(settings), debug)
46
47     def request_seen(self, request):
48         fp = self.request_fingerprint(request)
49         if fp in self.fingerprints:
50             return True
51         self.fingerprints.add(fp)
52         if self.file:
53             self.file.write(fp + os.linesep)
54
55     def request_fingerprint(self, request):
56         return request_fingerprint(request)
```

Maintainable tests are:

SOLID

maintainability

```
31     self.file = None
32     self.fingerprints = set()
33     self.logduplicates = True
34     self.debug = debug
35     self.logger = logging.getLogger(__name__)
36     if path:
37         self.file = open(path, "w")
38         self.file.seek(0)
39         self.fingerprints.update(self.read())
40
41     @classmethod
42     def from_settings(cls, settings):
43         debug = settings.getboolean("SUPERFLUX_DEBUG")
44         return cls(job_dir(settings), debug)
45
46     def request_scrape(self, request):
47         fp = self.request_fp(request)
48         fp.write(self.fingerprints)
49         return True
50         self.fingerprints.add(fp)
51     if self.file:
52         self.file.write(fp + os.linesep)
53
54     def request_fingerprint(self, request):
55         return self.request_scrape(request)
```

Maintainable tests are:

DAMP

maintainability

```
31     def __init__(self, *args, **kwargs):
32         self.file = None
33         self.fingerprints = set()
34         self.logduplicates = True
35         self.debug = debug
36         self.logger = logging.getLogger(__name__)
37         if path:
38             self.file = open(path, "w")
39             self.file.seek(0)
40             self.fingerprints.update(self.read())
41
42     @classmethod
43     def from_settings(cls, settings):
44         debug = settings.getbool("SUPERFLUX_DEBUG")
45         return cls(settings["superflux_dir"], debug)
46
47     def request_fingerprint(self, request):
48         fp = self.fingerprints.get(request.ip)
49         if fp:
50             self.fingerprints.add(fp)
51         if self.file:
52             self.file.write(fp + os.linesep)
53
54     def request_fingerprints(self, request):
55         return self.fingerprints
```

A wide-angle photograph of a desert landscape. In the foreground, there are numerous small, dark, wavy patterns on the sand, possibly from animal tracks or wind. The middle ground shows rolling sand dunes stretching towards the horizon. The sky is filled with wispy, light-colored clouds against a darker blue background.

DRY

Don't Repeat yourself!



Demo: Apply DRY



SOLID

SOLID

Single responsibility principle

There should never be more than one reason for a class to change.



SOLID

Open/closed principle

Software entities ... should be open for extension but closed for modification



SOLID

Liskov substitution principle

Functions that use pointers or references to base classes must be able to use objects of derived classes without knowing it

SOLID

Interface segregation principle

Clients should not be forced to depend upon interfaces that they do not use



SOLID

Dependency inversion principle

Depend upon abstractions, [not]
concretions.

SOLID

System under test

VS

The test itself



DATE	NAME	ARTIST	PRICE	
	12/22/2023 John Egbert Live	John Egbert	\$65	PURCHASE DETAILS
	3/22/2024 The State of Affairs: Michael Live!	Michael Johnson	\$85	PURCHASE DETAILS
	2/22/2024 To the Moon and Back	Nick Sailor	\$135	PURCHASE DETAILS

abstraction

HomePage

`NavigateToHomePage()`

`PutProductWithNameInBasket(string name)`



SHOW

John Egbert Live

John Egbert

12/22/2023

join John for his farewell tour across 15 continents. John really needs no introduction since he has already mesmerized the world with his banjo.

\$65

PER TICKET

QUANTITY:

1

PLACE ORDER

abstraction

ProductPage

SetQuantity(**int** quantity)
PlaceOrder()



A Globomantics Company

EVENT NAME	DATE	PRICE PER TICKET	QUANTITY	TOTAL
john Egbert Live	12/22/2023	\$65	<input type="button" value="1"/> <input type="button" value="Update"/>	\$65 <input type="button" value="X"/>
The State of Affairs: Michael Live!	3/22/2024	\$85	<input type="button" value="1"/> <input type="button" value="Update"/>	\$85 <input type="button" value="X"/>
« Back to event catalog				\$150

abstraction

ShopingBasketPage

```
SetQuantity(string productName, int quantity)
RemoveItem(string productName)
BackToEventCatalog()
GotoCheckout()
```



LOBOTICKET
A Globomatics Company

Checkout

Name

Email

Address

Town

Postal Code

Credit Card

Expiry Date

SUBMIT ORDER

abstraction

CheckoutPage

SubmitOrder(CustomerInfo customer)



LOBOTICKET
A Globomatics Company

Thank you for your order!

Please check your email for confirmation.

abstraction

ThankYouPage

IsThankYouMessageShown()



Demo: Apply the S in SOLID

The background of the image is a dark, moody forest. Bare, silhouetted tree branches reach out from the right side. A dirt path or road cuts through the center-left, leading towards a bright, hazy area in the distance. The overall atmosphere is mysterious and damp.

DAMP

Descriptive And Meaningful Phrases!

CONSIDER THE FOLLOWING TEST, WHAT ARE WE TESTING?

```
driver.Navigate().GoToUrl("http://localhost:5266/");
driver.Manage().Window.Maximize();

driver.Navigate().GoToUrl("http://localhost:5266/");
driver.Manage().Window.Size = new System.Drawing.Size(1616, 1034);
driver.FindElement(By.LinkText("PURCHASE DETAILS")).Click();
driver.FindElement(By.CssSelector(".btn")).Click();
driver.FindElement(By.LinkText("Back to event catalog")).Click();
driver.FindElement(By.CssSelector("tr:nth-child(2) .btn-primary")).Click();
driver.FindElement(By.Name("TicketAmount")).Click();
{
    var dropdown = driver.FindElement(By.Name("TicketAmount"));
    dropdown.FindElement(By.XPath("//option[. = '3']")).Click();
}
driver.FindElement(By.CssSelector(".btn")).Click();
driver.FindElement(By.CssSelector("tr:nth-child(2) .cancelIcon")).Click();
driver.FindElement(By.CssSelector(".navbar-brand > img")).Click();
var element =
driver.FindElement(By.XPath("/html/body/header/div/div[2]/div/div[2]/span[1]"));
Assert.IsTrue(element != null);
Assert.IsTrue(element.Text == "1");

driver.Close();
```



NOW, CONSIDER THE FOLLOWING TEST, WHAT ARE WE TESTING?

```
string productName = "John Egbert";
var homepage = new HomePage(driver);
```

```
Assert.IsTrue(
    homepage.PutProductNameInBasket(productName)
    .PlaceOrder()
    .SetQuantity(productName, 3)
    .GotoCheckout()
    .SubmitOrder(new CustomerMarcel())
    .IsThankYouMessageShown());
```



DESCRIPTIVE AND MEANINGFUL PHRASES!

A close-up photograph of Steve Carell as Michael Scott from 'The Office'. He has dark hair and is looking upwards and to the right with a wide-eyed, open-mouthed expression of surprise or realization. The background is blurred, showing what appears to be office cubicles.

But how???

Page Object Pattern



The Loboticket website displays three distinct pages:

- Search Results Page:** Shows a grid of three events with thumbnails, names, dates, artists, prices, and "PURCHASE DETAILS" buttons.
- Product Details Page:** Shows a single event (John Egbert Live) with a thumbnail, title, date, price (\$65), and a "PLACE ORDER" button.
- Shopping Basket Page:** Shows a summary of items in the basket, including John Egbert Live at \$65 and The State of Affairs: Michael Lien at \$85, with a total of \$150 and a "CHECKOUT" button.

abstraction

HomePage

```
NavigateToHomePage()
PutProductWithNameInBasket(string name)
```

abstraction

ProductPage

```
SetQuantity(int quantity)
PlaceOrder()
```

abstraction

ShopingBasketPage

```
SetQuantity(string productName, int quantity)
RemoveItem(string productName)
BackToEventCatalog()
GotoCheckout()
```

Each Method Returns a Page as a result.

DAMP Tests

Page Object Pattern

- For each **page** create a **class** that abstracts the **interaction** with the page
- For each **action** create an **action method** that returns a **page object**
- For each **verification** create a **method** that executes the verification and returns true or false
- Never expose page internals, like controls, etc.!

Why Is This Good for Maintainability?

- Each **page object** only has knowledge of **only one functional part** of the system
 - S in SOLID
- **Changes** in the application **don't affect** the test **scenario**
 - Only the implementation behind the action and verification methods
- **One change** in the application **will not** ripple through my whole test automation framework

Applying This Principal to Your Development Team

- You can further improve maintainability of the tests
- Make the page object part of the *definition of done* of the web pages they build or modify
 - Developers are now responsible for making controls on the page better identifiable
 - Thus they will create better HTML structures & better identifiable objects
 - No test scenario will break anymore





Demo: Apply DAMP

Given, when, then

Given: Marcel wants to buy a ticket

And : We have a clean database

When: Marcel selects the "John Engelbert" product

And: He sets the quantity to 3

And: He adds them to the basket

And: He is going to the checkout

Then: We should see a "thank you!" message



What about data
and test automation?

Leverage containers!

Containers provide immutable infrastructure

For production you should not store data in a container but.....

For testing purposes, having the data in the container is a bliss!
Start a container for a given scenario and you can always run it over and over again

All side effects are discarded by starting a new container!

Wrap up

UI Test automation

Avoid it as much as possible!

Automation is an engineering discipline!

Maintainable tests are:

Solid

Dry

Damp

How to manage data?

Leverage CONTAINERS!





thanks!

@marcelv

Fluentbytes.com



Marcel de Vries

CTO

Xpirit Netherlands

@marcelv <http://fluentbytes.com>

Attributions

Unsplash.com, pictures used for backgrounds
giphy.com for animated gifs