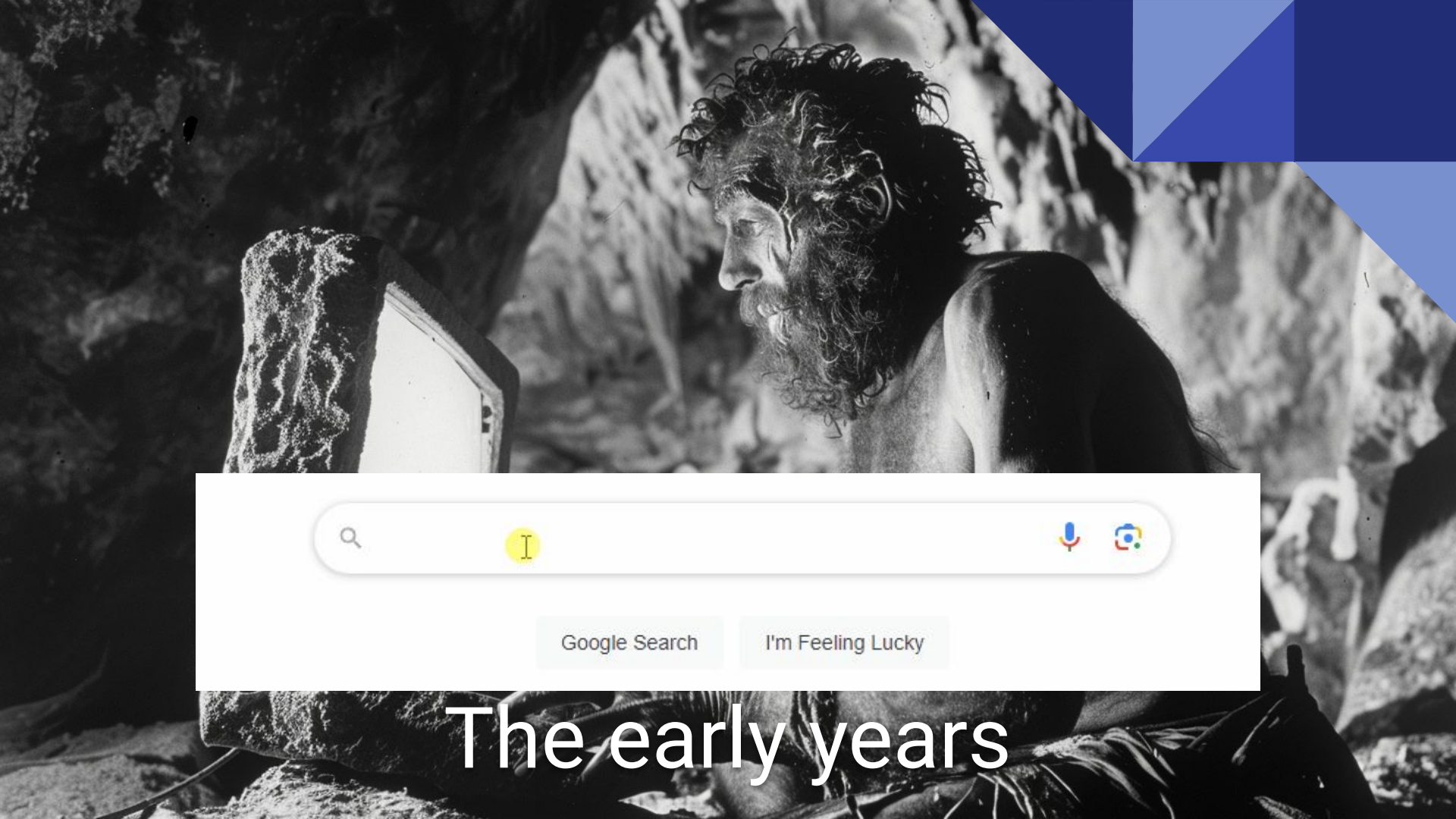
15 years UI test automation

Eyk Haneklaus
ehaneklaus@rosen-group.com

ROSEN
empowered by technology

Agenda

- How things started
- Writing test automation code
- Running tests with Azure Devops
- Current challenges
- Summary



The early years

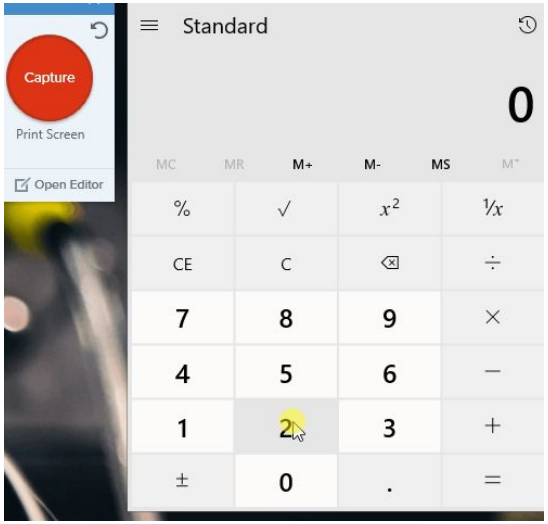
What does the industry offer?

- SmartBear TestComplete: *“With codeless [...] creation, you can run automated tests regardless of technical skill level.”*

- LeapWork: *“No-code, visual approach for testers and business users”*

- Ranorex: *“Tools for script-free automation”*

Record & play back: Let’s try it!



- We have made rapid progress…
- … initially
- But:
- sometimes “hack scripts” required
- Small changes to our application caused re-recording lots of test cases again…
- … and again!

Record & play back: What a bad idea…

- “Recordings” are hard to maintain
- too many “hack scripts” were necessary
- **Result: We ended up using an unstable, custom scripting language in an IDE that was NOT designed for writing code**

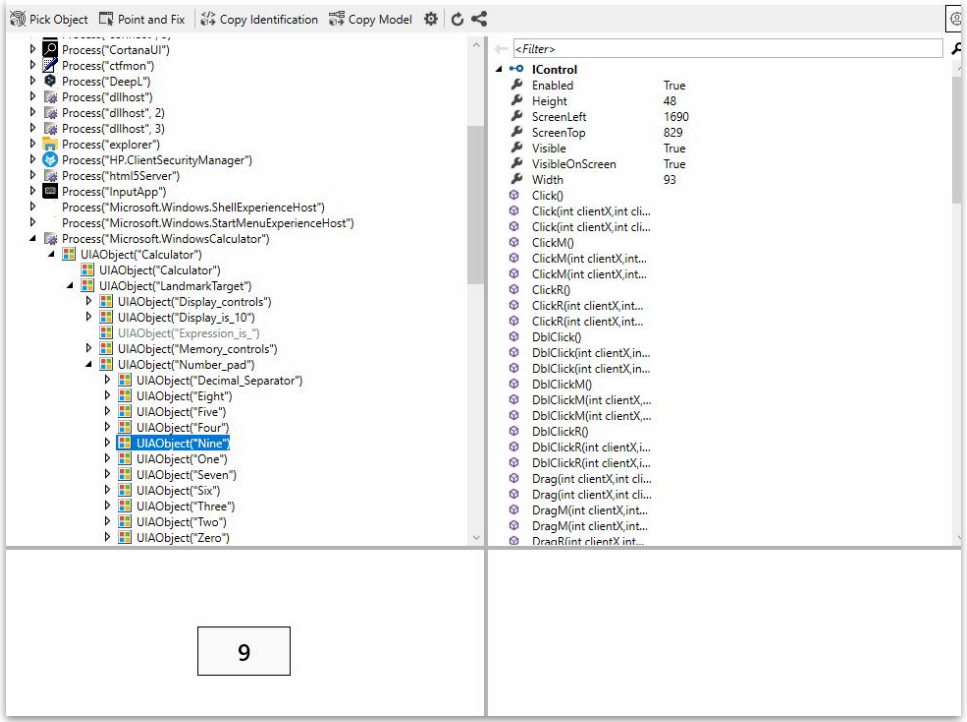
“Ok, now let’s write code”: CodedUI

- Microsoft introduced “Coded UI” VisualStudio 2010
- Yay, we could write UI tests in C#!
- Finding controls was sloooow -> custom “caching” strategy
- Finding controls was unreliable, “flaky” -> custom retry strategy
- Code complex and ugly
- Microsoft retired CodedUI in VisualStudio 2019

“Ok, now let’s write code”: Appium WinAppDriver

- Microsoft recommends using Appium WinAppDriver (still today: https://techcommunity.microsoft.com/t5/testingspot-blog/winappdriver-and-desktop-ui-test-automation/ba-p/1124543)
- No proper control inspection tool
- Ugly code
- “Druid skills” required
- Writing UI tests takes much too long and is way too complicated
- **Microsoft does not update WinAppDriver anymore (since ~3 years)**
- The community is not amused https://github.com/microsoft/WinAppDriver/issues/1550

But finally!!!



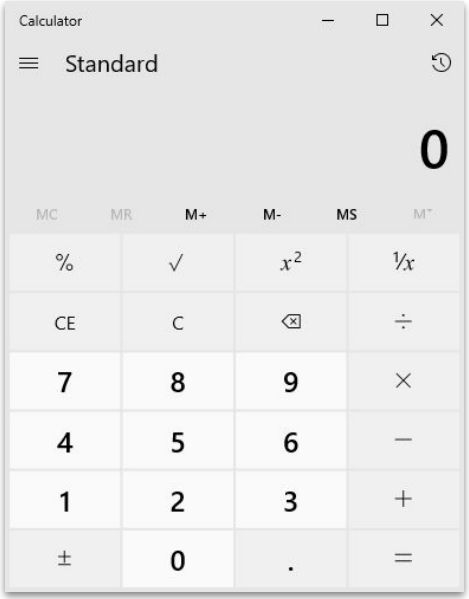
Finally we found a solution that

- has a Control Finder with **code generation(!)**
- fast and reliable
- can be used from C# and Visual Studio (proper refactoring, debugging, version control, …)
- is not bound to VS, can also be used from prototyping tools (LINQPad)
- **Easy to learn!**

Let’s dip our toes into some code

The Task

- Start the calculator
- type 2 * 3 =
- check that the result equals 6
- close the calculator



The code

```
var twoButton = driver.Find<IImmersiveProcess>(new ProcessPatter
{
    ProcessName = "Microsoft.WindowsCalculator"
}).Find<ITopLev
{
    FrameworkId
    ClassName =
    LocalizedCo
    ObjectIdent
    ObjectGroup
}).Find<IContro
{
    FrameworkId
    ClassName =
    LocalizedCo
    ObjectIdent
    ObjectGroup
}).Find<IContro
{
    FrameworkId = "XAML",
    ClassName = "NamedContainerAutomationPeer",
    LocalizedControlType = "group",
    ObjectIdentifier = "Number_pad"
}).Find<IControl>(new UIAPattern()
{
    FrameworkId = "XAML",
    ClassName = "Button",
    LocalizedControlType = "button",
    ObjectIdentifier = "Two"
});
```

```
twoButton.Click();
multiplyButton.Click();
threeButton.Click();
equalsButton.Click();

var result = resultView.GetProperty<string>("Text");
```

```
var threeButton = driver.Find<IImmersiveProcess>(new ProcessPattern()
{
    ProcessName = "Microsoft.WindowsCalculator"
    FrameworkId = "XAML",
    ClassName = "NamedContainerAutomationPeer",
    LocalizedControlType = "group",
    ObjectIdentifier = "Number_pad"
}).Find<IControl>(new UIAPattern()
{
    FrameworkId = "XAML",
    ClassName = "Button",
    LocalizedControlType = "button",
    ObjectIdentifier = "Three"
});
```

Let’s clean up: The Page Object Pattern

- https://martinfowler.com/bliki/PageObject.html
- a simple but effective software design pattern
- separates test code from automation code

Let’s clean up: The Page Object

```
public class CalculatorApp
{
    private LocalDriver Driver { get; } = new LocalDriver();

    private ITopLevelWindow MainWindow => this.Driver.Find<IImmersiveProcess>(new ProcessPattern
                        .Find<ITopLevelWindow>(new UIAPattern() {FrameworkId = "XAML", ClassName

    private IControl NumberPad => this.MainWindow.Find<IControl>(new UIAPattern() { FrameworkId
                        .Find<IControl>(new UIAPattern() { FrameworkId = "XAML", ClassName = "Na

    private IControl StandardOperators => this.MainWindow.Find<IControl>(new UIAPattern() { Fram
                        .Find<IControl>(new UIAPattern() { FrameworkId = "XAML", ClassName = "Na

    public void Start()
    {
        System.Diagnostics.Process.Start("calc");
    }

    public void Close()
    {
        this.MainWindow.Close();
    }

    public void ClickNumberButton(string numberName)
    {
        var numberButton = this.NumberPad.Find<IControl>(new UIAPattern() { FrameworkId = "XAML"
        numberButton.Click();
    }
```

Let’s clean up: The test code

```
var calculator = new CalculatorApp();
calculator.Start();
calculator.ClickNumberButton("Two");
calculator.ClickMultiplyButton();
calculator.ClickNumberButton("Three");
calculator.ClickEqualsButton();

var result = calculator.GetResultText();

calculator.Close();
```

```
var calculator = new CalculatorApp();
calculator.Start();
calculator.ClickNumberButton("Three");
calculator.ClickPlusButton();
calculator.ClickNumberButton("Four");
calculator.ClickEqualsButton();

var result = calculator.GetResultText();

calculator.Close();
```

- New tests can be added easily!
- If the application changes, only the “page object” needs to be changed, not dozens (hundreds?) of tests!

Some additional tips

- Don’t do error handling in your tests!
  - Messes up your code, wastes time (development)
  - The errors happen in places that you do NOT expect
  - The exception message is sufficient to quickly find the cause
- Don’t write “retry logic”, when a control can not be found
  - messes up your code, wastes time (development & execution)
  - there are better APIs to wait for a control until it is enabled
  - … if not, **get a better tool!**
- Don’t implement control caching for performance reasons
  - Messes up your code, wastes time (development)
  - Very unstable
  - If your automation framework is too slow, **get a better tool!**

Some general tips

- Write simple and “flat” test code
- Don’t create a company testing framework. Get a tool that works.
- Don’t use inheritance, generics, … sure, you know your stuff

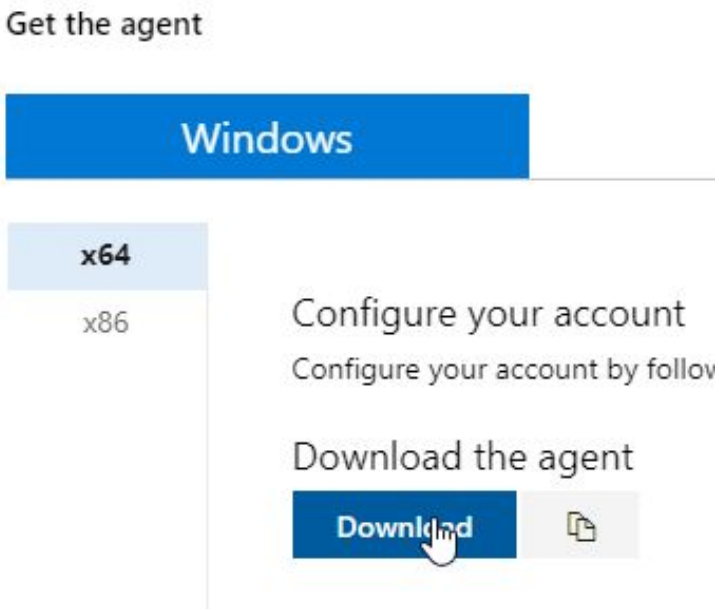
Let’s automate… the automation

Azure DevOps

- Developers can run the tests directly from any source branch!
- The test code is associated to “Test cases” in order to organize them and for documentation purposes
- The team (including managers) has convenient access to the latest test results (and also an archive of previous test results)
- Testers are able to create bugs from failed test runs (and attach logs, screenshots, videos)

Setup an agent to run UI tests

Download the agent zip, and unpack



Set up an agent to run UI tests

Run config.cmg to configure the agent (interactive mode!)

```
config.cmd --url https://<your-azure-devops>
--auth pat --token <your-pat>
--unattended --replace
--pool <your-pool>
--agent <your-agent-name>
--runAsAutoLogon --overwriteAutoLogon
--windowsLogonAccount <your-test-user>
--windowsLogonPassword <test-user-password>
```

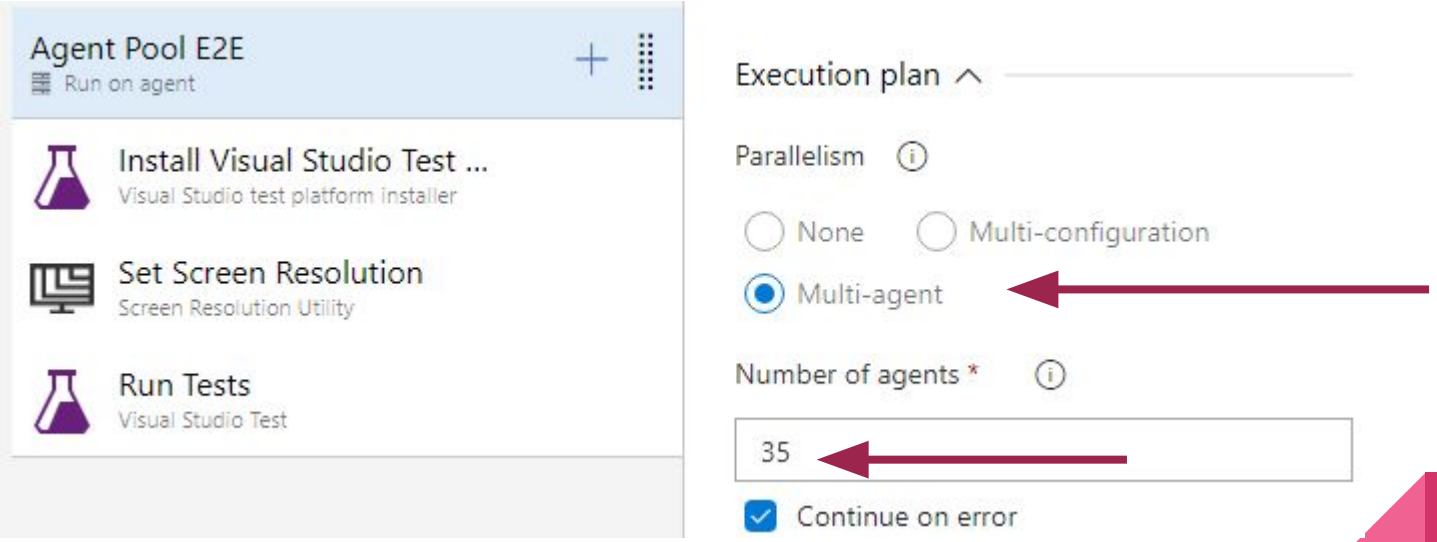
needs Agent Pool permissions

no questions, overwrites existing agent

required to run UI tests (“interactive”)

Parallel test execution

Configure the Agent Job as “Multi-agent”



Parallel test execution

ALWAYS run the VS Test Platform Installer!

choose a specific version (faster!)

Will actually run the tests

Parallel test execution

Configure the VS Test Runner Task

Advanced execution options

Batch tests

Based on number of tests and agents

Utilizes all available agents

Batch options

Automatically determine the batch size

Specify a batch size

Number of tests per batch *

1

Ensures no agent is waiting (most efficient for long running tests)

Replicate tests instead of distributing when multiple agents are used in the job

Add test attachments

Use TestContext::AddResultFile to attach files

```
C#

public abstract void AddResultFile (string fileName);
```

Where to store attachments

ResultsDirectory	Gets base directory for results from the test run. Typically a subdirectory of Test nDirectory.
TestDeployment Dir	for files deployed for t ically a subdirectory of Same as Depl hat property instead.
TestDir	ed files and result property instead.
TestLogsDir	Gets d ly a subdirectory of ResultsD irectory. Use that property for test run specific result files instead.
TestName	uted
TestResults Directory	st result fi
TestRunDirectory	Ge e directory for the test run, under deployed files and result files are stored.

Do NOT use any of these directories for logs!!!

Will be DELETED after test run

… before they are actually attached.

Where to store attachments

Instead use something like

`%TEMP%\MyTests\...`

Cleanup %TEMP% on every reboot

Viewing test results

1 Run(s) Completed (0 Passed, 1 Failed) 81 unique failing tests in the last 14 days

87
Total tests

84 Passed
3 Failed
0 Others

96.55%
Pass percentage

30m 50s
Run duration

Bug Link

Filter by test or run name

Test	Duration
Automated Tests for [redacted] (3/87)	0:30:50.260
[redacted]	0:03:04.740
[redacted]	0:03:01.740
[redacted]	0:02:57.817

Viewing test attachments

Result Details

Failed Yesterday on LINATESTE2E204		Duration	0:12:15.090
Owner	not available	Date started	6/12/2024, 11:18:09 PM
Date completed	6/12/2024, 11:30:24 PM	Failing since	Yesterday
Failing since release	[redacted]		

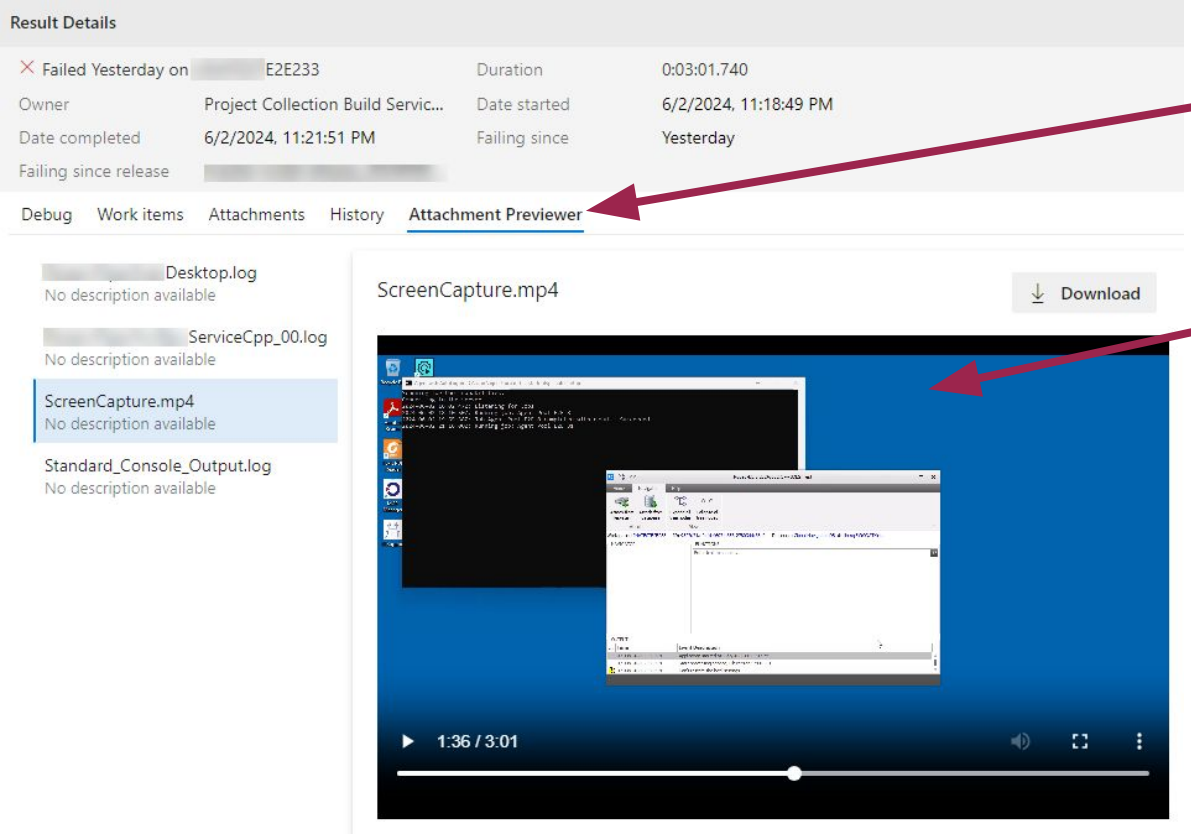
Debug | Work items | **Attachments** | History | Attachment Previewer

- Rosen.[redacted].log — 1K — Added Yesterday
- Rosen.[redacted].Service... — 1K — Added Yesterday
- Rosen.[redacted].Service... — 1K — Added Yesterday
- Rosen.[redacted].log — 1K — Added Yesterday
- ScreenCapture.mp4 — 6019K — Added Yesterday
- Standard_Console_Output.... — 5K — Added Yesterday

No preview available for the selected file type

Download

Viewing test attachments (custom extensions)



Custom extension that shows attached files directly in the browser (screenshots, PDFs, Videos)

Custom “VideoRecorder” implementation that creates a screen capture (much better than MS version)

Current challenge: Image Comparison

Current challenges: Image Comparison

Use cases:

- Custom view of ultrasonic sensor data
- 3rd party map controls (openstreetmap, gmaps, yahoo, …)

Problem: Image comparison has false alerts

- Copyright 2023/2024
- Slight differences Win10 / Win11

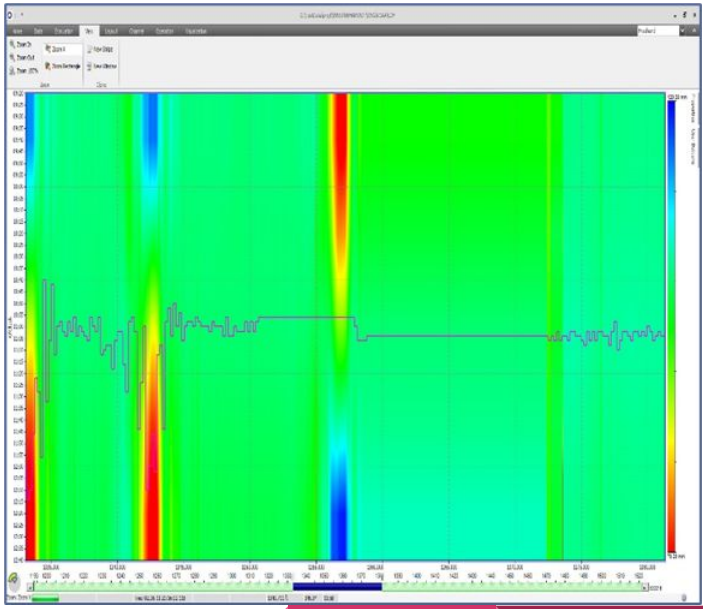


Image Comparison: Change perspective

- assume that control is already tested
  - test the application, NOT the control!
  - 3rd party controls are tested by others
  - custom controls should be tested at an earlier stage
- test the underlying data
  - “hack” into the tested process
  - supported by test tool
  - … also with code generation

Example:

- Select an item in a list control
- check if the scale/pan of a custom control is as expected

“Hacking”?, seriously?

Cons

- Needs developer knowledge about application
- hurts “black box principle”
- breaks when DataModel changes

Pros

- no reference images to maintain
- “rock solid”

Summary

- write simple code!
- choose the right tool! (time is money)
- automate!
- be pragmatic!

Thanks for your attention!
Questions?

Eyk Haneklaus
ehaneklaus@rosen-group.com
www.linkedin.com/in/eykh

These slides online

ROSEN
empowered by technology