Appium Is Not The Problem Your Assumptions Are

Wim Selles | Director at TechChamps

TSC Member and Core Contributor to the WebdriverIO Project

One promise

Somewhere in this talk, something will **click**.

Leave With

- A **user-first testing mindset**
- Test what a **real person** actually experiences
- Use Appium/platform **understanding** to explain failures

IT STARTED SO WELL,

BUT THEN...

YOU, SIX MONTHS AGO

You have been automating web apps for years.

You know your **SELECTORS**. You know your **WAITS**.

You know your **BACKDOORS**: Jav 

est data without touching the UI.

You know which Stack Overflow answer to trust (yes, we are old school craftsmen).

Then your client launches a native app.

H·I·S Q·U·O·T·E

We want automated tests for it. You can do that, right? It is ***BASICALLY THE SAME AS WEB****.*

You did your homework

You watched **YouTube**. You attended **conferenc**

- **Cross-platfor**
- **Android and** **release?** Absolutely.
- **Emulators, simulators, real devices, cloud?** All of them.

Appium it is. You picked it for **all the right reasons**.

• YOU OPEN APPIUM INSPECTOR •

```
0" height="2856">

t=""
 false"
lse"
="true"
 id="451"


portant="true"
displayed="true">

table="true"
="true"

2]">
ext="Home"
772]" />
```

iOS

```
<AppiumAUT>
  <XCUIElementTypeApplication
      name="wdiodemoapp" bundleId="org.wdiodemoapp"
      enabled="true" visible="true" accessible="false"
      x="0" y="0" width="402" height="874">
    <XCUIElementTypeWindow
        enabled="true" visible="true" accessible="false"
        x="0" y="0" width="402" height="874">
      <XCUIElementTypeOther
          enabled="true" visible="true" accessible="fals
          x="0" y="0" width="402" height="874">
        <XCUIElementTypeWebView
            enabled="true" visible="true" accessible="fa
            x="0" y="0" width="402" height="784">
          <XCUIElementTypeButton
              name="Toggle navigation bar"
              enabled="true" visible="true" accessible="
              x="16" y="163" width="30" height="31"
```

Two trees: locator strategy

You know **XPath**. XPath knows you. This is going to be fine.

```
// Android: android.widget.TextView[@text='Login']
await $('//android.widget.TextView[@text='Login']').tap()

// iOS: XCUIElementTypeStaticText[@name='Login']
await $('//XCUIElementTypeStaticText[@name='Login']').tap()
```

Two platforms. Two completely **different** trees. Two **separate selectors** for every single element.

And it is slow. **S**

Then it gets personal

A modal appears out of nowhere. Your test cannot find it.

iOS asks for a permission. Your test freezes and stares at the dialog like it owes it money.

You need to reset app state between tests. There is no `localStorage.clear()`.
No cookies. **Nothing**.

You search for "backdoors". You find a forum post from 2019 that has no accepted answer.

• YOU HAVE SEEN ENOUGH. •

THE WRONG MAP

- YOU BROUGHT THE RIGHT SKILLS TO THE WRONG TERRITORY

What the web taught you

HTML is source of truth

A W3C specification. Readable markup, consistent structure. The same contract in every browser and every tool.

A live, updating tree

The DOM reflects the current state at all times. Query it any time. Changes happen, the tree follows.

Selectors that travel

Copy from the browser, paste into your test. It works. Every time. On every browsers.

DevTools on demand

The entire element tree, right in front of you. Every attribute, every style, easy debugging, easy manipulation.

Backdoors

`localStorage.clear()` . JS injection. API mocking. Seed data without touching the UI.

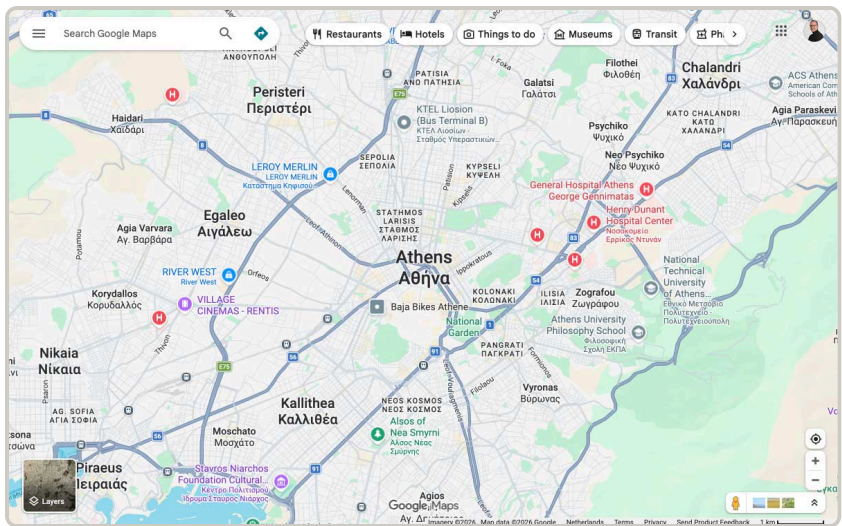
Abstracted platform

Chrome on your machine behaves the same on CI. The browser abstracts the OS. Your tests do not care what runs underneath.

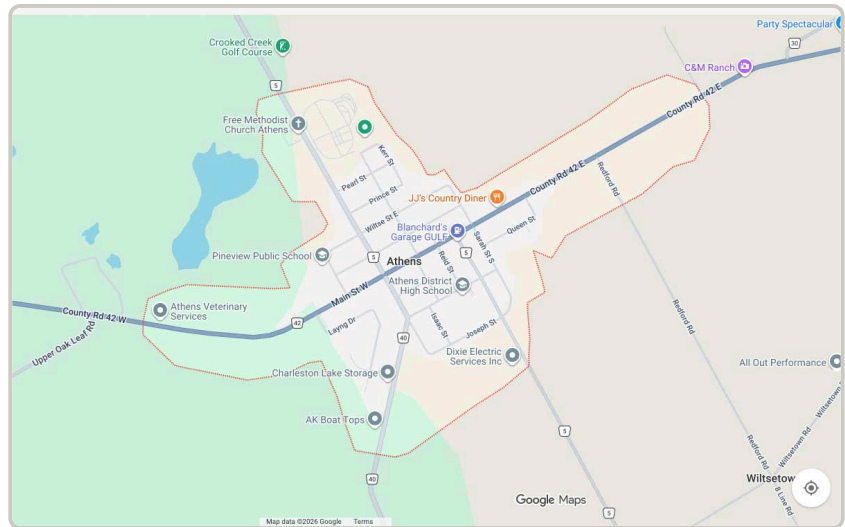
You are using the wrong (mind)map

Web	vs	Native
HTML readable markup, W3C spec, always there		**UI hierarchy** XML snapshot, platform-specific, generated on demand
DOM live tree, reflects state, never lies		**Snapshot** taken at the moment you ask, may already be stale
Selectors copy from browser, paste in test, done		**Locators** tied to the driver, the OS, and the version
DevTools live inspection, every attribute on demand		**Appium Inspector** a snapshot tool, not a live view
Backdoors JS injection, localStorage, API mocking		**No backdoors** platform APIs only, no browser shortcuts
Platform abstracted away by the browser		**Platform** Android and iOS are two completely different worlds

• YOU ARE USING THE WRONG MAP •



A•T•H•E•N•S• •G•R•E•E•C•E



A•T•H•E•N•S• •O•N•T•A•R•I•O• •C•A•N•A•D•A

The assumptions that follow

XPath works everywhere, so I will just use that.

It does work. You will **regret** it.

The element tree is always up to date.

It is a **snapshot**. Snapshots **lie**.

Android and iOS are basically the same underneath.

They are **not even close**.

It is basically just like web, right?

You have heard this before. You know how it **ends**.

The cost

Every one of these assumptions is quietly making your tests **slower**, **flakier**, and **harder to debug**.

So what IS actually there?

WEB VS. NATIVE

WHAT ACTUALLY EXISTS

You expect HTML elements. You get this.

Web

```
div
button
input
span
p
a
```

Android

```
android.widget.FrameLayout
android.view.ViewGroup
android.widget.ScrollView
android.widget.TextView
android.widget.Button
android.view.View
```

iOS

```
XCUIElementTypeApplication
XCUIElementTypeWindow
XCUIElementTypeOther
XCUIElementTypeWebView
XCUIElementTypeStaticText
XCUIElementTypeButton
```

What This Means

These are not simplified names. These are the **actual element types** you work with **every single day**.

You expect attributes. You get properties.

Web

```
<button
  id="login-btn"
  class="btn btn-primary"
  data-testid="login"
  role="button"
  aria-label="Login">
  Login
</button>
```

Yours to **define**.

Android

```
<android.view.View
  content-desc="Login"
  clickable="true"
  enabled="true"
  bounds="[427,2616][640,2772]">
  <android.widget.TextView
    text="Login" />
</android.view.View>
```

iOS

```
<XCUIElementTypeButton
  name="Login" label="Login"
  enabled="true" visible="true"
  accessible="true"
  x="16" y="163"
  width="106" height="31"
  traits="Button"
/>
```

Defined by the **platform** and translated by **Appium**.

What this means

On the web, you **wrote the attributes**.

On native, the platform wrote them and you are **reading what is there**.

Expect stable structure. Get a snapshot.

Web

your test

↓

`$('[aria-label="Login"]')`

↓

DOM: always current

↓

elementId ✓

Reality Check

Element references can go **stale** if the framework re-renders the node, but re-querying the **live DOM** is instant.

VS

Android

your test

↓

`$('//android.widget.TextView[@text="Login"]')`

↓

AccessibilityNodeInfo: snapshot

↓

elementId?

Reality Check

UIAutomator2 reads the `AccessibilityNodeInfo` tree directly for most locators. **XML** is only generated for **XPath**. Cost still grows with tree depth, and results can go **stale** when the UI changes.

There is no ...

Reality Check

iOS does **NOT** use **XML**.

Appium generates it from the XCUITest hierarchy.

iOS: Appium builds the tree from scratch

Web

your test

↓

`$('[aria-label="Login"]')`

↓

DOM: always current

↓

elementId ✓

Reality check

Element references can go **stale** if the framework re-renders the node, but re-querying the **live DOM** is instant.

vs

iOS

your test

↓

`$('//XCUIElementTypeStaticText[@name="Login"]')`

↓

XML: generated from scratch

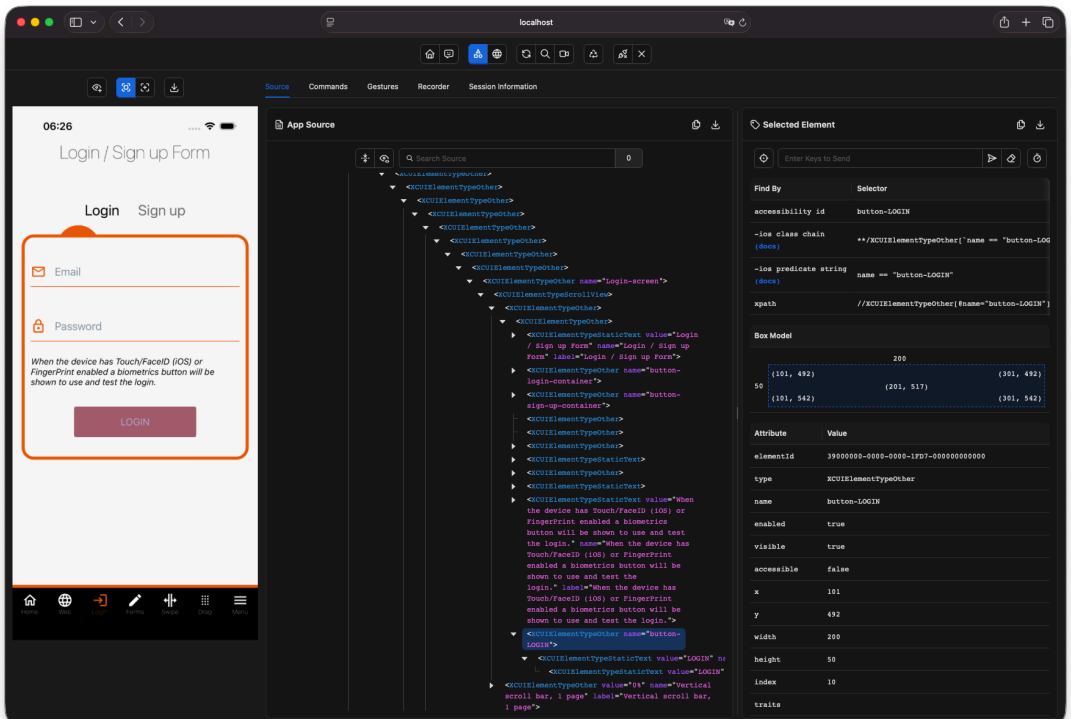
↓

elementId (maybe)

Reality check

iOS has **no native XML**. Appium rebuilds it from the full XCUIElement tree on each query.

• THIS IS WHAT YOU ARE WORKING WITH •



Appium Inspector is not DevTools

Expectation		Reality
Live DOM	→	**Frozen** snapshot
One **standard** tree	→	One tree per **platform**
Copy **selector** to code	→	**Translate locator** per driver

So when you ask Appium for an element...

You think you know what happens when you call `$('~Login').click()`.

You are about to find out you ***DO NOT***.

How Appium really works

- What happens between your test and the device

The journey of a single element find

When you write `$('//XCUIElementTypeStaticText[@name="Login"]').click()`, this happens:

```
Your test
  -> WebdriverIO       sends an HTTP request to Appium
  -> Appium            routes it to the right platform driver
    -> UIAutomator2      captures the current UI hierarchy  (Android)
    -> XCUITest          captures the current UI hierarchy  (iOS)
  -> Appium            serialises that hierarchy to XML
  -> Appium            runs your selector query against the XML
  -> Appium            deserialises matched node into an element reference
  -> your test         receives that element reference back
  -> your action       (repeat)
```

The Catch

Every `findElement` based on **XPath** pays this **cost** again.

One **bad** locator repeated multiple times becomes your **flaky framework**.

Drivers are not optional magic

Your test sends a **WebDriver** command → Appium **routes** it to the platform driver → The driver executes it via **native** APIs

UiAutomator2 (Android)

- Google's official UI testing framework
- Reads the `AccessibilityNodeInfo` tree
- Returns what is in the **current view** hierarchy
- Best locator fields: `resource-id`, `content-desc`, `text`
- Runs on-device as an APK

XCUITest (iOS)

- Apple's official UI testing framework (Xcode)
- Traverses the full XCUIElement tree
- Returns the **full accessibility** tree
- XML is generated from scratch on each query, there's **no caching**
- Best locator fields: `name`, `label`, `value`
- WebDriverAgent (WDA) runs on the device

Why this matters for performance

The **bigger** the hierarchy, the **bigger** the snapshot, the **slower** the find.

At **200 views**, every element lookup pays that cost, not once at load time, **every time**.

What makes it faster:

- **Flat** UI hierarchies
- **Direct** locators

What makes it slower:

- **Deep**, nested view trees
- Full-tree **XPath** queries

P·E·R·F·O·R·M·A·N·C·E R·U·L·E

Every extra lookup has a **cost**, so fewer, more specific finds **win**.

Android locator strategy: order matters

Android strategy	Sample code
1 **Accessibility ID** Fastest and clearest intent	`$('~Login')`
2 `//xpath` Often fine, slower on deep trees	`$('//android.widget.TextView[@text="Login"]')`
3 `UiSelector` Use when XPath gets messy	`$(`android=new UiScrollable(` ` new UiSelector().scrollable(true)` `).scrollIntoView(` ` new UiSelector().text("Login")` `)` `)`

Android rule

Start with **Accessibility ID**. XPath can work, but every broad query costs you.

iOS locator strategy: order matters

iOS strategy	Sample code
1 **Accessibility ID** First choice every time	`$('~Login')`
2 `predicate string` Expressive live queries	`$('-ios predicate string:name == "Login"')`
3 `class chain` Structural query, no XML walk	`$(`-ios class chain: **/XCUIElementTypeButton[\`name == "Login"\`] `)`
4 `xpath` fallback Most expensive option	`$('//XCUIElementTypeButton[@name="Login"]')`

iOS rule

On iOS, keep XPath as **last resort**. Prefer predicates and class chains first.

The single thread challenge

Main thread

network response arrives

↓

BUSY

re-render · layout pass · a11y tree refresh

↓

idle

Appium

POST /element

↓

waiting for main thread...

↓

200 OK, element found

The catch

When the **main thread is busy**, your driver **cannot get an answer**.
This is **not Appium being slow**. This is the **platform**.

WRONG CHOICES.

EVERY ONE OF THEM MADE SENSE WITH THE WRONG MENTAL MODEL.

YOU KNOW A LOT. YOU STILL SLIP.

Something feels **OFF.**

What worked **NO** longer fits.

You **IMPROVISE** a fix.

Now it is **WORSE**.

This is what the **WRONG CHOICES** look like in practice.

Gestures exist. Most tests ignore them.

What users do

- Swipe to navigate
- Scroll through content
- Pinch to zoom
- Long press for context menus
- Drag and drop

What your tests do

```
await $('~button').click()
await $('TableRow1Column2Cell').getText()
```

That is it.

When tests have gestures, this happens

```
await driver.action('pointer')
  .move({
    x: 200,
    y: 640
  })
  .down()
  .pause(200)
  .up()
  .perform()
```

A **fixed** coordinate. Written once on the device in front of you.

P·L·O·T T·W·I·S·T

Coordinates are a screenshot of **one device**.

Change screen size, and your "**working click**" turns random.

The keyboard is UI. Test it as UI.

Your test

```
await $('~email').setValue('test@example.com')
await $('~password').setValue('secret')
await $('~login').click()
```

Keyboard?

What keyboard?

User experience

- Keyboard slides up,
  covering **half the screen**
- The focused field (hopefully)
  scrolls into view
- Where is my **next** field/button?
- There is **no Escape key**
- Dismissal is **platform-specific**

Automating the Impossible

Outside Appium's reach

- Biometric authentication **flows**
- Face ID and fingerprint **prompts**
- Camera input and AR **features**

What vendors offer

- **Inject code** into your app
- Simulate the interaction **programmatically**
- The test **passes**

The Catch

Be aware that you are testing their **injection**, not your **app**.

Same OS, different runtime behavior

What Android emulators miss

- No OEM-specific behavior (Samsung, OnePlus, ...)
- No true hardware signals
- No full user-grade security model
- No realistic timing under device load

What Android real devices reveal

- Whether the OEM skin breaks your layout
- Whether memory pressure changes app behavior
- Whether OS security restrictions block flows
- Whether performance feels right to a real user

Your test setup is not your user's reality

Your test

- **Freshly** booted emulator
- **Full** battery
- **Perfect** network
- **Nothing** else running

Your user

- 4G dropping to 3G
- Battery saver throttling the **CPU**
- 40 other apps in **memory**
- **Notification** interruptions
- 3-year-old **mid-range** device
- An OS version you **never** tested

The gap

You are not really testing your app.
You are testing it under **ideal conditions** your users **never have**.

PIVOT

A DIFFERENT WAY OF THINKING

- KNOW YOUR PASSENGER, YOUR ROAD, AND HOW TO DRIVE

• GOING NOWHERE. •

We talked about the **CAR**.

We talked about the **ENGINE**.

We talked about the driver's **LICENSE**.

What about our **DESTINATION**?

Your destination is the person.

Test script	VS	Real person
Stable **scripted** flow		**Thumb/finger-first**, one-handed navigation
Desktop-like control assumptions		No hover, no right-click, no CMD+F
Gestures treated as **edge cases**		Swipe, scroll, and long-press are **default**
Assumes **uninterrupted** execution		Calls, notifications, and lock screens **interrupt**

The destination

Build for the **real person**, not the perfect script.

ONE QUESTION. TWO VERSIONS.

"Does this work?"

↓

"Would a **REAL PERSON** *do this* **WITHOUT THINKING TWICE?***"*

What that shift changes.

What to test

Would a **real person** notice if this broke?

How to test it

Shortcuts test something your user never does.

What passing is

A **workaround that passes** is a gap wearing a green checkmark.

How to read it

Not "why is Appium broken?" but "**what did my user experience** here?"

• License acquired •

User Thinking still absent.

KNOW HOW TO DRIVE.

SNAPSHOTS

Your **MIRRORS**. They show a **MOMENT IN TIME**. Check before you move.

LOCATORS

Your **NAVIGATION**. The **WRONG ROUTE** costs you on every single find.

GESTURES

How you **STEER**. Calling `.click()` on everything is **NOT DRIVING**.

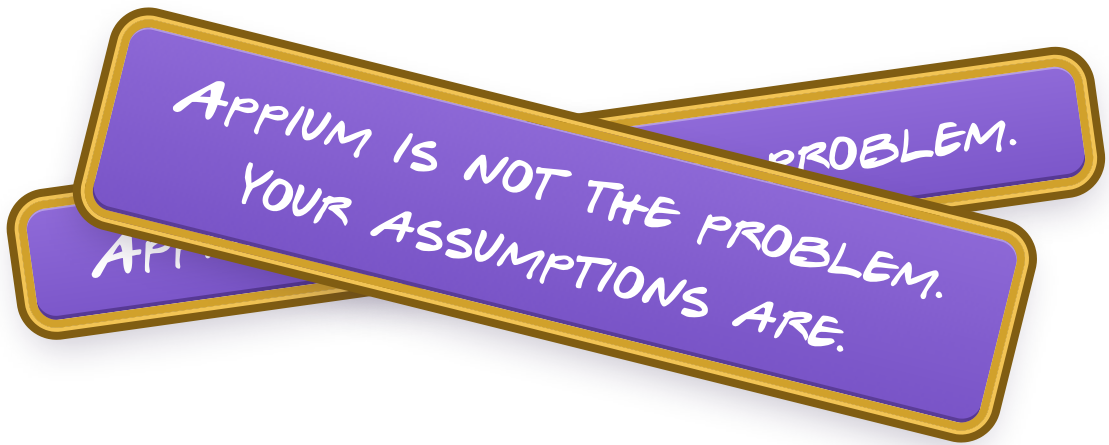
PASSENGER

Every decision is in service of **THEM**. Not the test. Not the tool. The **PERSON**.

A CALMER WAY FORWARD

- LESS MAGIC, FEWER SURPRISES

One last stamp.



AND THAT IS GOOD NEWS.

If the **TOOL** was broken, you would be **STUCK**.

If it is your **ASSUMPTIONS**, you can **CHANGE** them. **TODAY.**

That is the click.

Now take this model into your next failing test.

• Thank you •

W•i•m• •S•e•l•l•e•s

Director at TechChamps

TSC member and Core Contributor to the WebdriverIO project