# Controlling apples with snakes!

Automating mobile apps with Appium

Nir Arad
@nirarad1 #pycon2018

www.waves.com

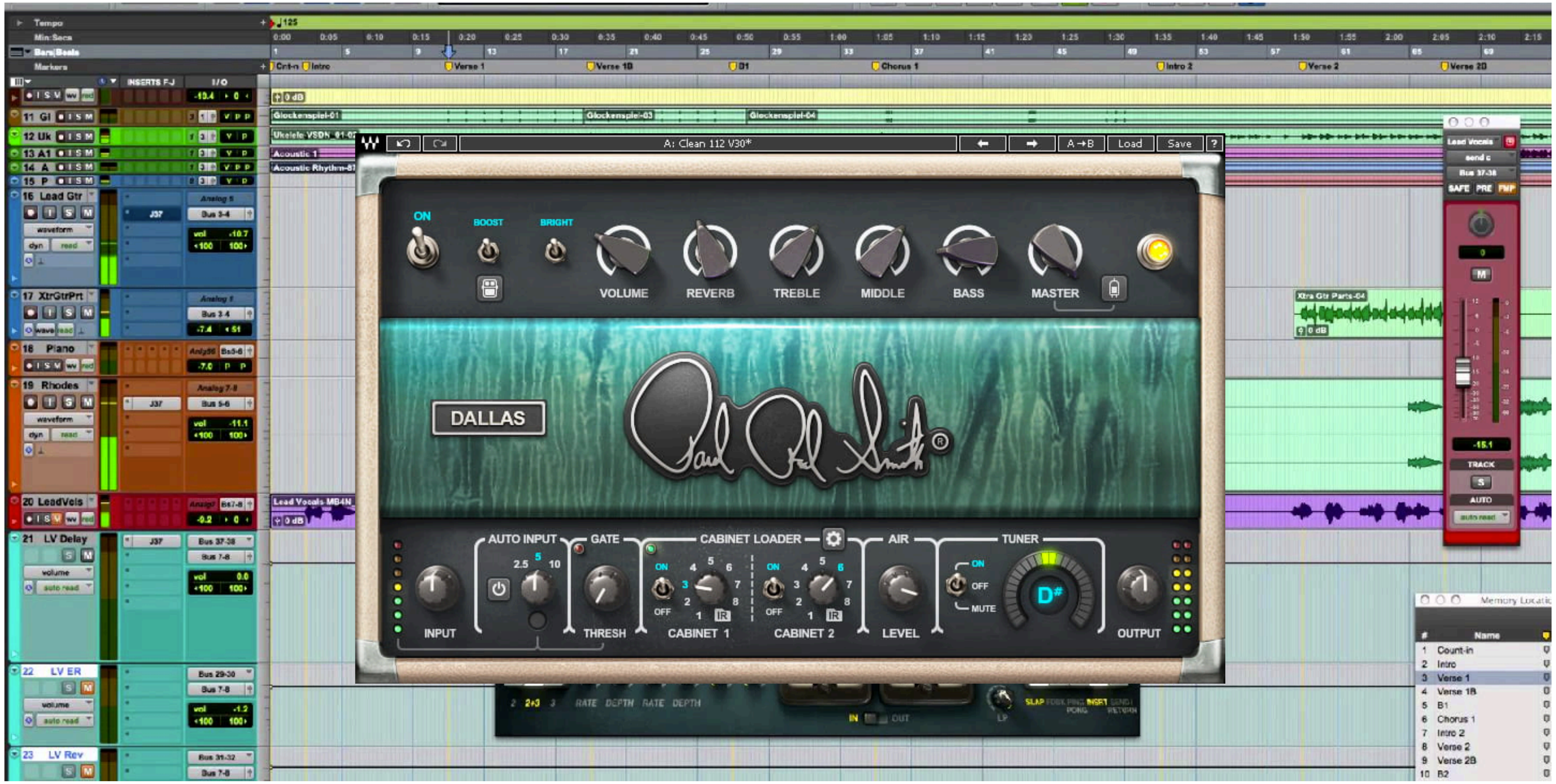
# MANUAL TESTING

WAVES

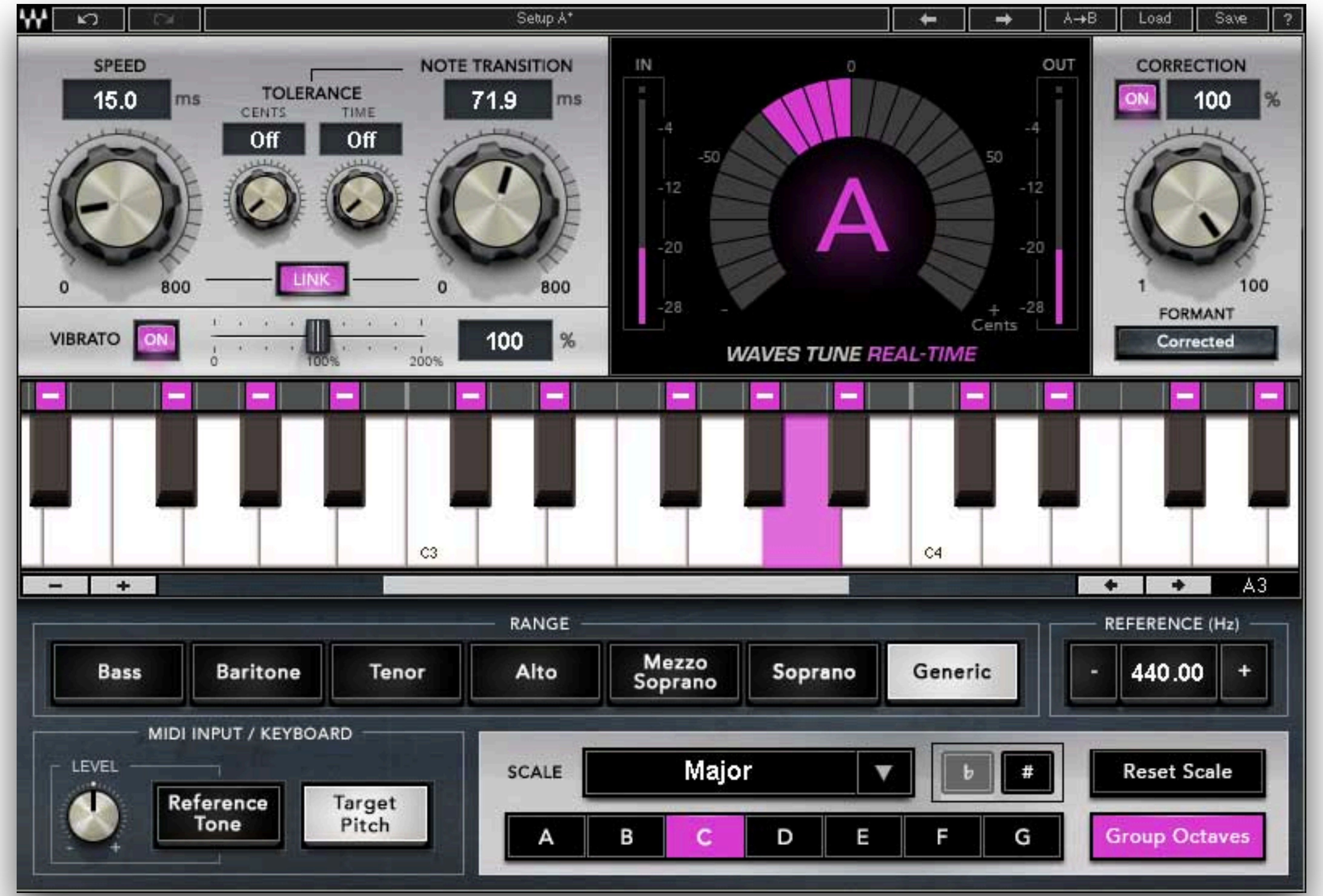
## ABOUT MYSELF

- Studied sound engineering
- Working at Waves Audio for the past 10 years
- Moved from manual QA testing to writing automated tests in Python

# AUDIO PLUGINS



# AUDIO PLUGINS



SPEED 15.0 ms

TOLERANCE CENTS Off TIME Off

NOTE TRANSITION 71.9 ms

VIBRATO ON 100 %

CORRECTION ON 100 %

FORMANT Corrected

REFERENCE (Hz) 440.00

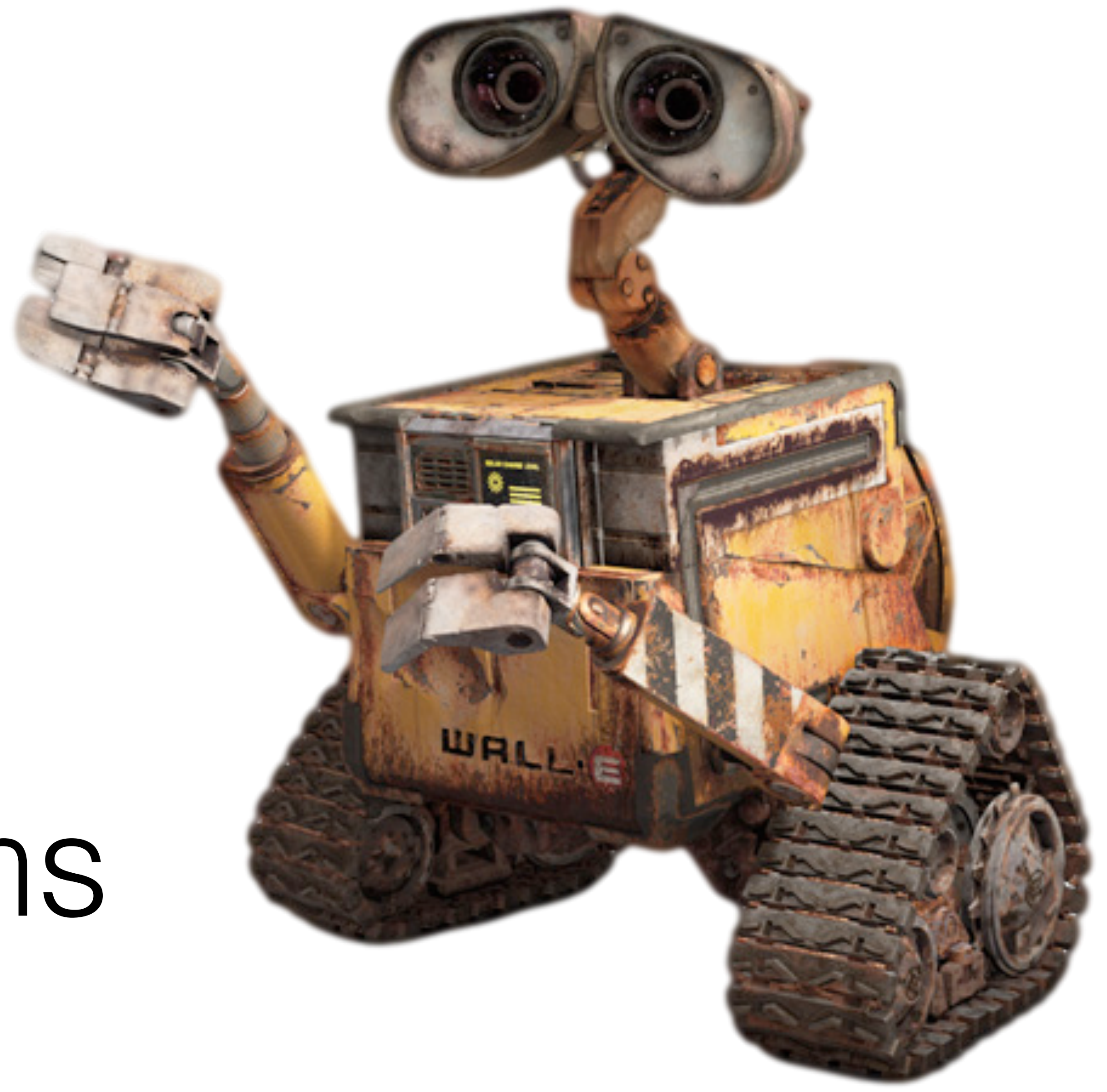
SCALE Major

## WHY USE AUTOMATED TESTING?

Test all of these every day!!

## BENEFITS OF AUTOMATED TESTING

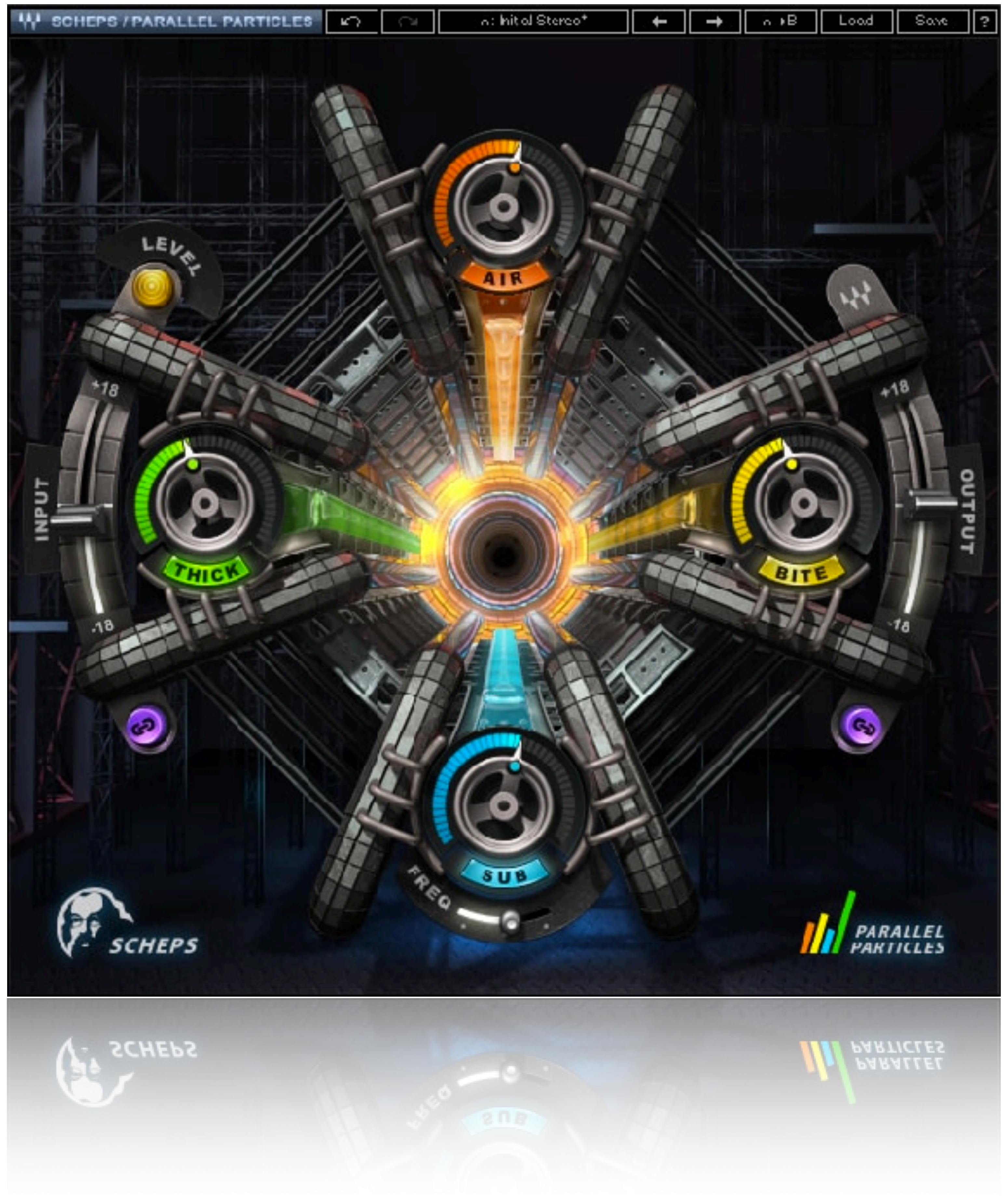
- **Save time** - immediate results, 24/7
- **Speed** - Runs faster than humans
- **Accuracy** - Tests run always the same
- **Reusable** - Same test across all platforms
- **Increase Coverage** - Faster releases

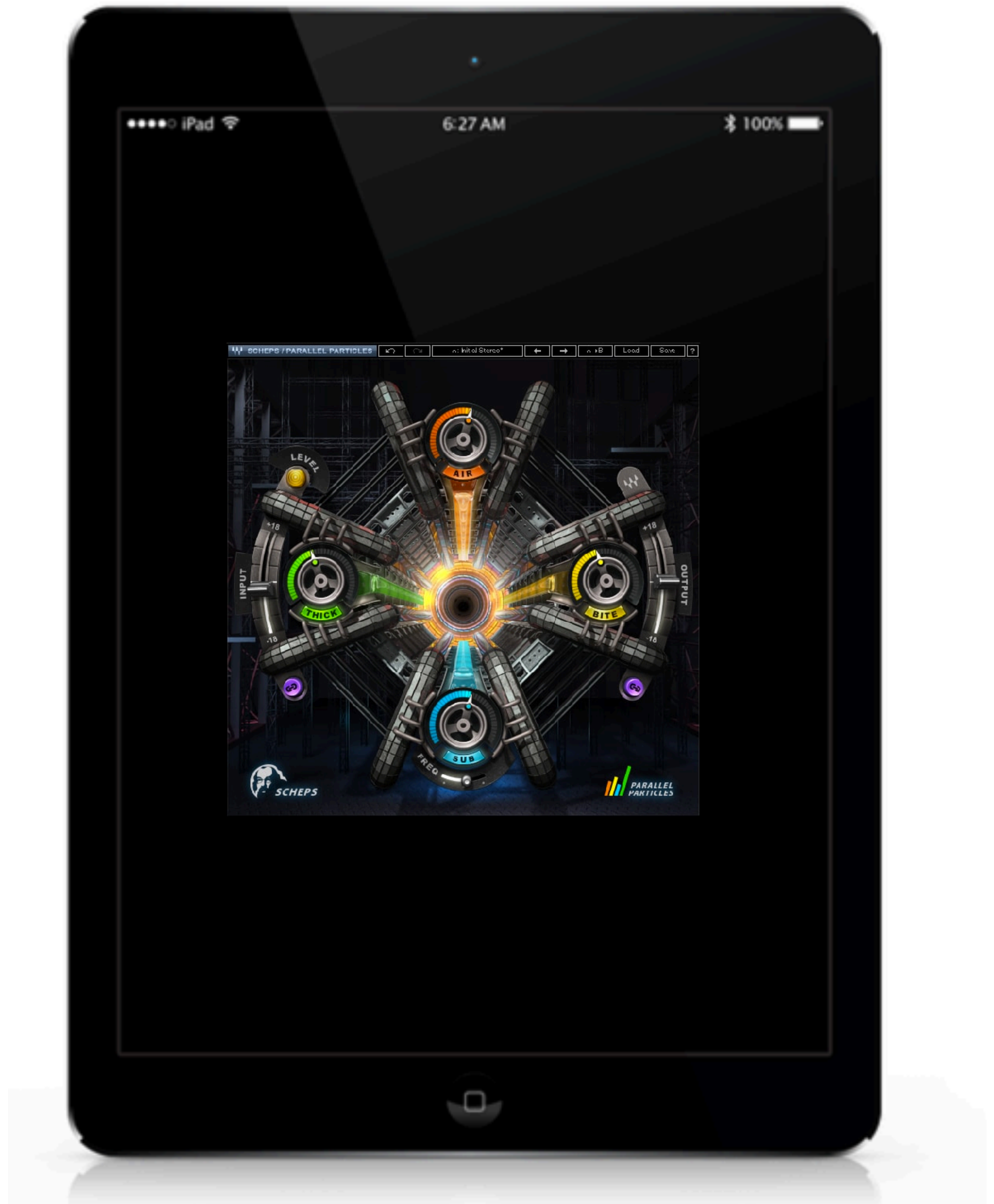


WAVES

# THE CHALLENGE

**Transfer our testing facilities from desktop to iOS**





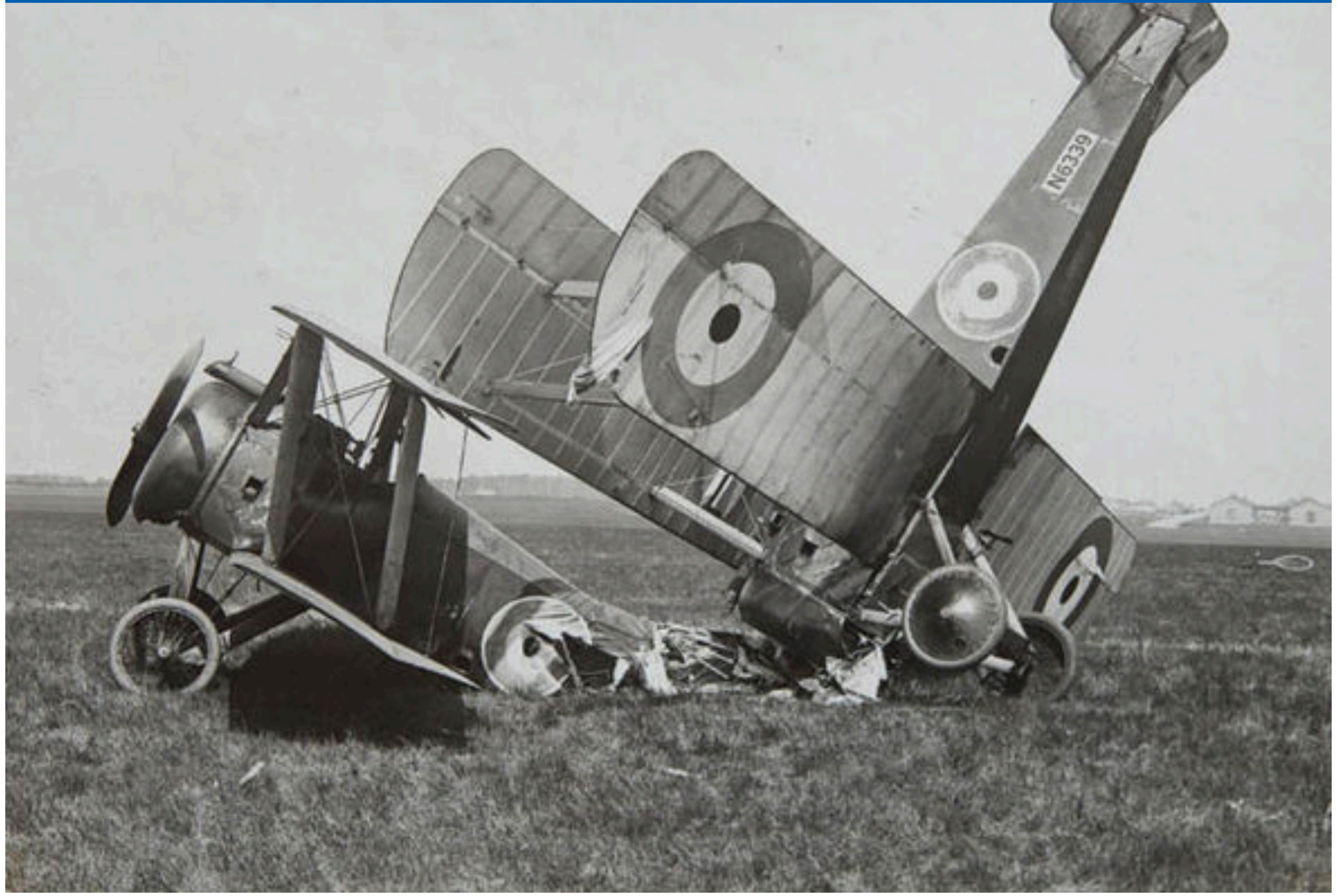
# TESTING AS A 3RD PARTY DEVELOPER

TestFlight Beta Testing

- Used for creating a distinct IDs for an App
- Manual testing
- Only supports real devices
- Doesn’t support Python

WAVES

## TestFlight Beta Testing



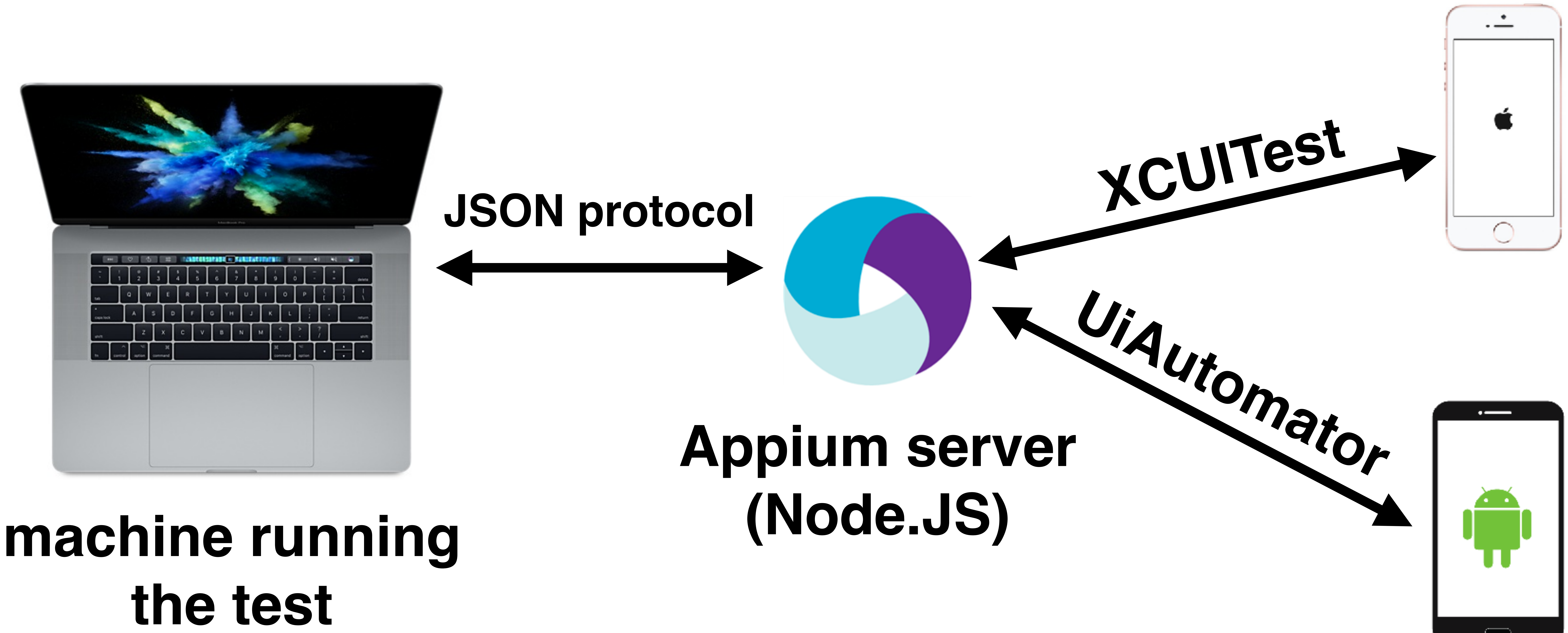
appium

by SAUCELABS

- Supports both real and emulated devices
- Can automate native, web, and hybrid apps
- Works in Python
- Open source

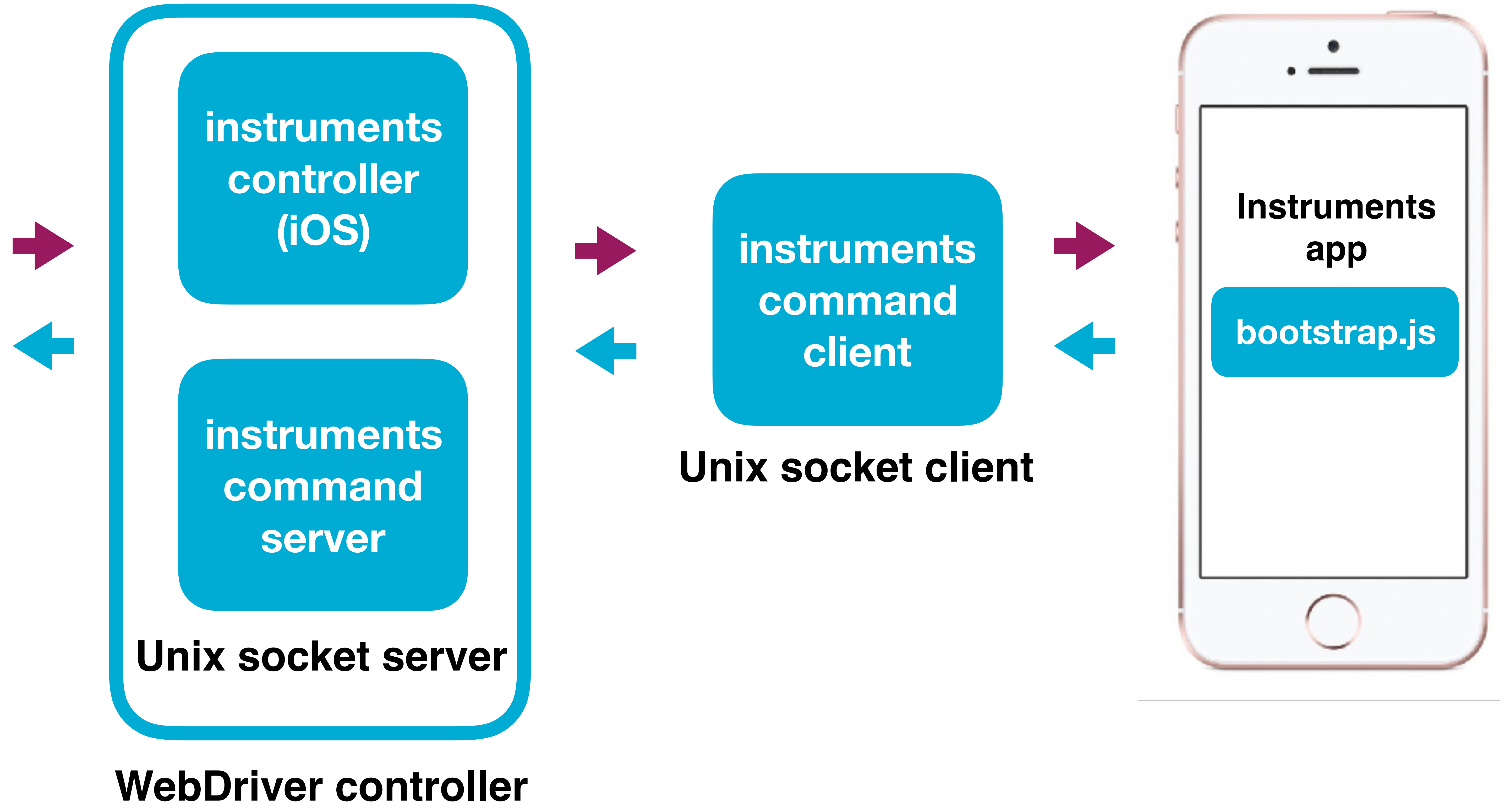
WAVES

# WRAPPING NATIVE MOBILE OS FRAMEWORKS



# iOS INSTRUMENTS

```
def login():
    driver.get(self.url)
    element =
driver.find_element_by_id("usrname")
    element.send_keys(self.user)
    element =
driver.find_element_by_id("psw")
    element.send_keys(self.password)
    driver.find_element_by_id("login-
btn").click()
```



## Real device

**Pros:**

More stable

Real device performance

**Cons:**

Expensive

Slower response time

Needs to be updated, can break

## Emulated device

**Pros:**

Faster (no data transfer)

Easier to maintain

Free

Concurrent run (Android only)

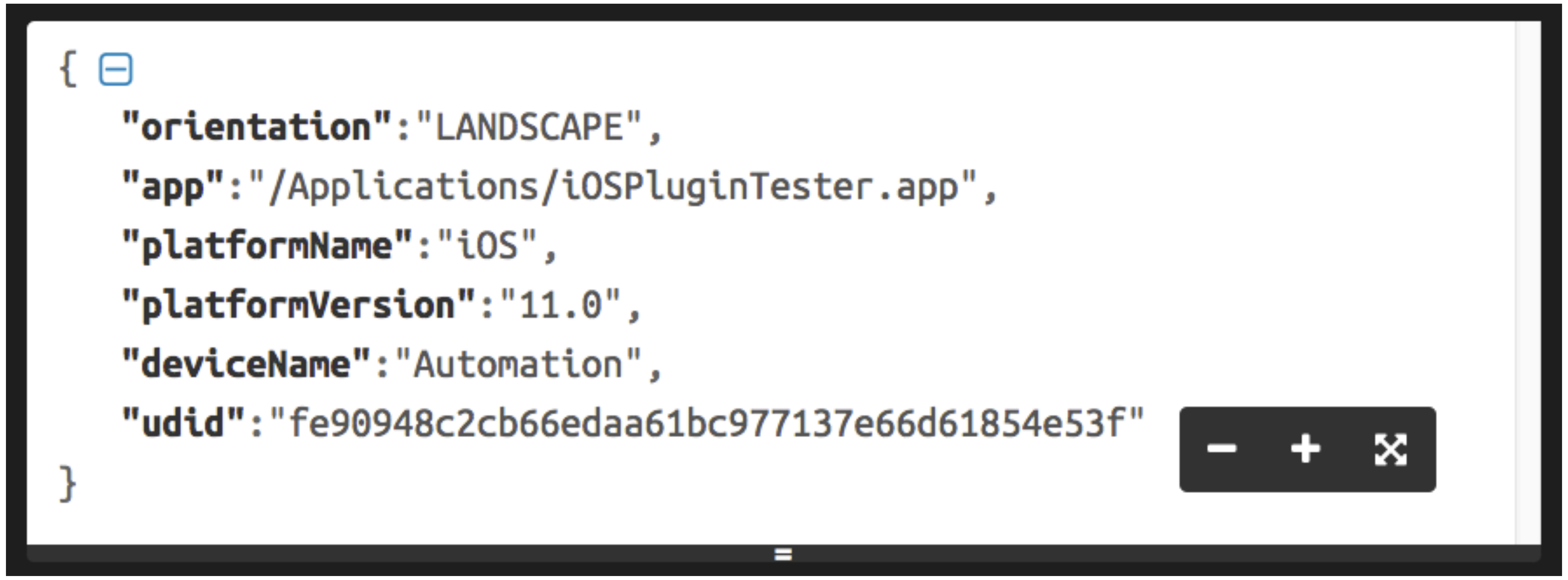
**Cons:**

Less stable

Not the real thing

# SESSION CONTEXT

- Sessions are defined by desired capabilities
- Each test uses a different session
- When starting a session Appium will copy the application on the the device and launch it



```
{
    "orientation":"LANDSCAPE",
    "app":"/Applications/iOSPluginTester.app",
    "platformName":"iOS",
    "platformVersion":"11.0",
    "deviceName":"Automation",
    "udid":"fe90948c2cb66edaa61bc977137e66d61854e53f"
}
```

# Instantiating a WebDriver object

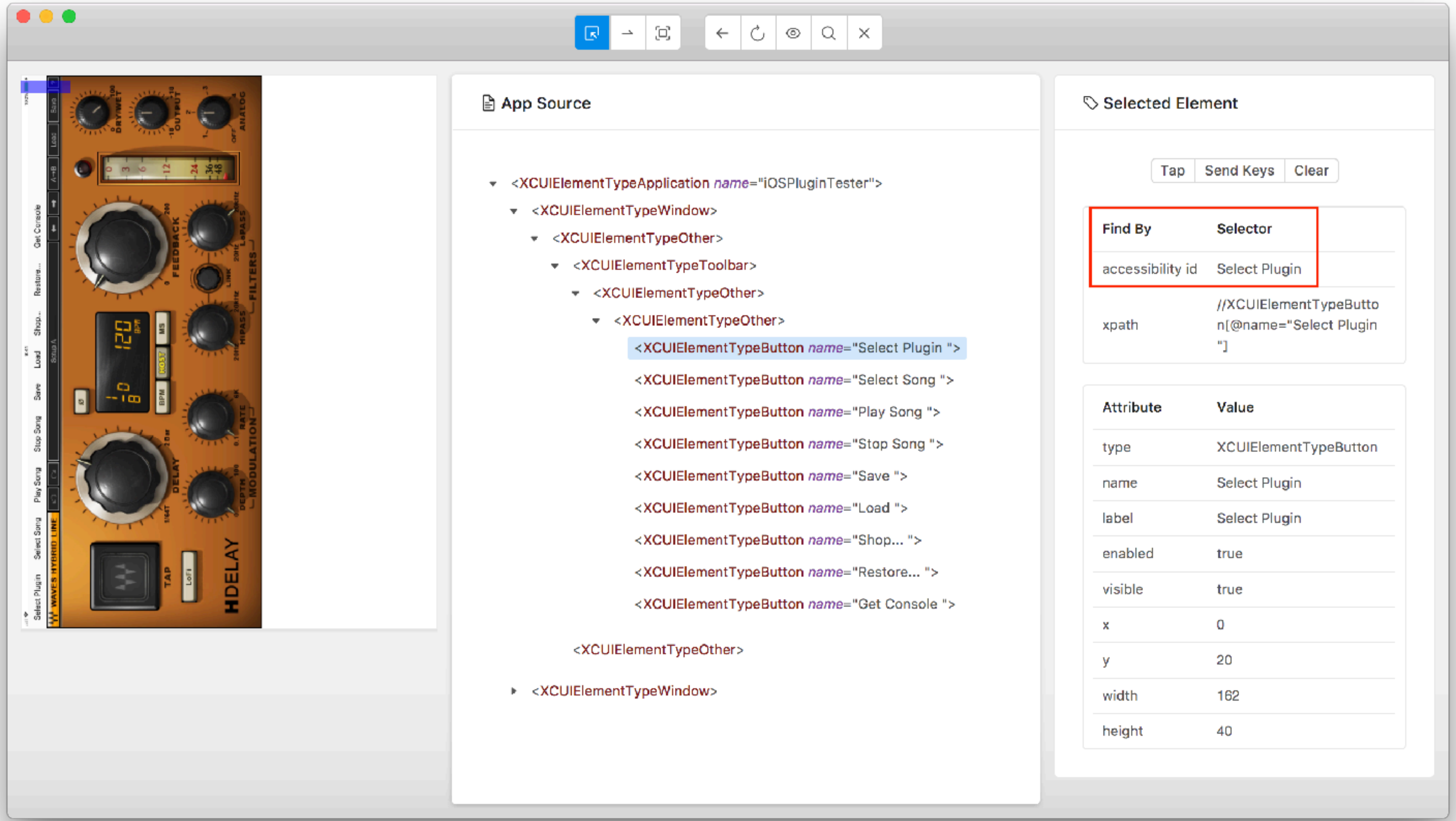
```
# First we define basic information for the test
APPIUM_PORT = '4723'
udid = 'fe90948c2cb66edaa61bc977137e66d61854e53f'
app_path = '/Applications/iOSPluginTester.app'

command_executor = 'http://127.0.0.1:%s/wd/hub' % APPIUM_PORT


# Then we define a dictionary for Appium
desired_capabilities = {'orientation': 'LANDSCAPE',
                        'app': app_path,
                        'platformName': "iOS",
                        'newCommandTimeout': 240,
                        'platformVersion': "11.0",
                        'deviceName': "Automation",
                        'udid': udid}

# Finally we create a web driver instance
from appium import webdriver
driver = webdriver.Remote(command_executor,
                          desired_capabilities)
```

# FINDING ELEMENTS USING WebDriver

Appium supports a subset of the WebDriver locator strategies:

- Class name
- accessibility ID
- Name
- XPath

# APPIUM DESKTOP



App Source

```
<XCUIElementTypeApplication name="iOSPluginTester">
  <XCUIElementTypeWindow>
    <XCUIElementTypeOther>
      <XCUIElementTypeToolbar>
        <XCUIElementTypeOther>
          <XCUIElementTypeOther>
            <XCUIElementTypeButton name="Select Plugin ">
            <XCUIElementTypeButton name="Select Song ">
            <XCUIElementTypeButton name="Play Song ">
            <XCUIElementTypeButton name="Stop Song ">
            <XCUIElementTypeButton name="Save ">
            <XCUIElementTypeButton name="Load ">
            <XCUIElementTypeButton name="Shop... ">
            <XCUIElementTypeButton name="Restore... ">
            <XCUIElementTypeButton name="Get Console ">

      <XCUIElementTypeOther>

  <XCUIElementTypeWindow>
```

Selected Element

| Find By | Selector |
|---|---|
| accessibility id | Select Plugin |
| xpath | //XCUIElementTypeButton[@name="Select Plugin "] |

| Attribute | Value |
|---|---|
| type | XCUIElementTypeButton |
| name | Select Plugin |
| label | Select Plugin |
| enabled | true |
| visible | true |
| x | 0 |
| y | 20 |
| width | 162 |
| height | 40 |

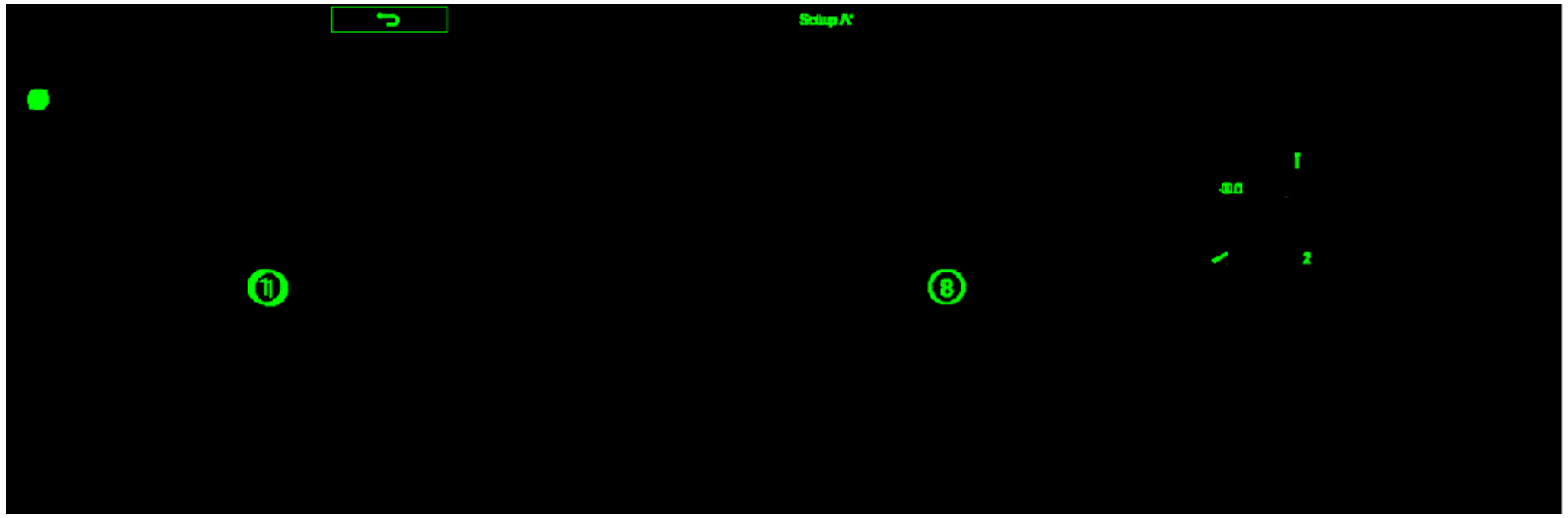
# IMAGE COMPARISON TEST

Reference build

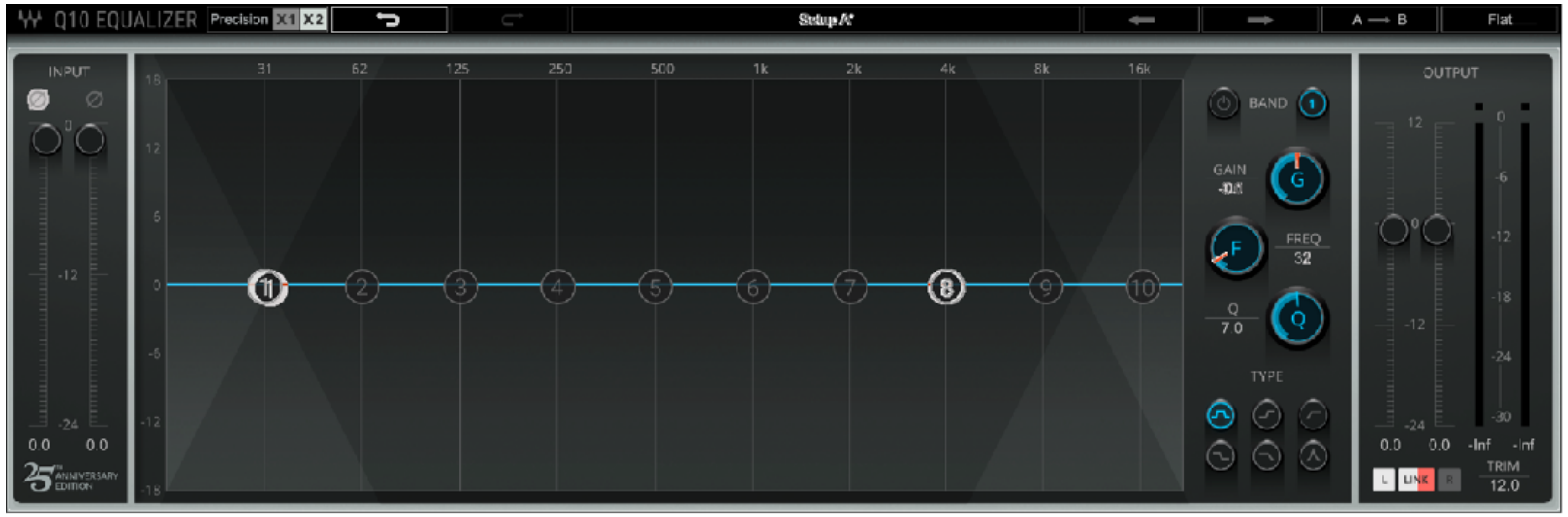
New build

# IMAGE COMPARISON TEST

Diff image

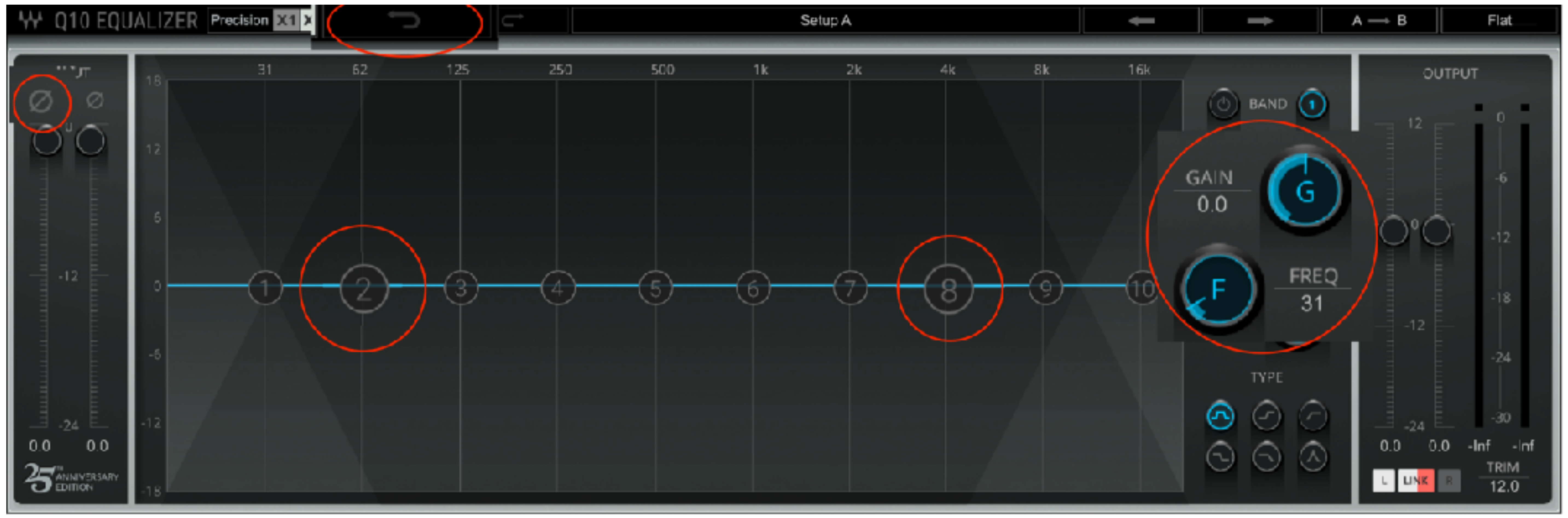


Inverted diff

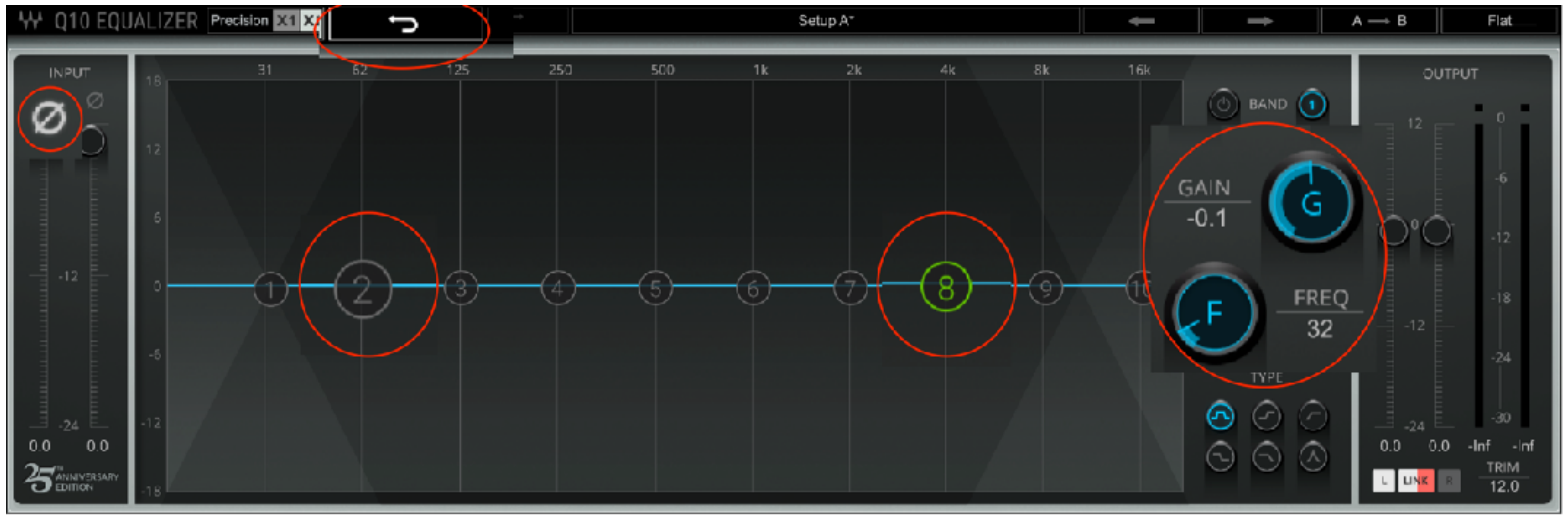


# IMAGE COMPARISON TEST

Reference build



New build



# THE TEST CODE

```python
# Find the button element for the plugins menu
plugin_menu = driver.find_element_by_accessibility_id("Select Plugin")


# Open the menu
plugin_menu.click()


# Find all plugin elements
element_type = "//XCUIElementTypeButton"
plugins = driver.find_elements_by_xpath(element_type)


# Iterate through all plugins and take a screenshot
for plugin_button in plugins:
    plugin_button.click()
    driver.get_screenshot_as_file(image_path)
    compare_images(image_path, ref_image_path)

    # Open the menu again for the next iteration
    plugin_menu.click()


# Close the web driver
driver.quit()
```

```python
from PIL import Image

def compare_images(ref_img_file_path, new_img_file_path, diff_img, comb_img):
    ref_img_obj = Image.open(ref_img_file_path)
    new_img_obj = Image.open(new_img_file_path)
    ref_img_mtx = ref_img_obj.load()
    new_img_mtx = new_img_obj.load()
    # Creating a new black image
    diff_img_obj = Image.new('RGB', (ref_img_obj.size[0], ref_img_obj.size[1]))

    diff_pixels = 0
    for col in range(ref_img_obj.size[0]):  # Iterating over rows
        for row in range(ref_img_obj.size[1]):  # Iterating over columns

            if ref_img_mtx[col, row][0:3] != new_img_mtx[col, row][0:3]:
                diff_pixels += 1

                # Painting over the tested image with an inverted color
                p = ref_img_mtx[col, row]  # Pixel tuple (R, G, B)
                new_img_obj.putpixel((col, row), (255 - p[0], 255 - p[1], 255 - p[2]))

                # Painting a green pixel on the black image
                diff_img_obj.putpixel((col, row), (0, 255, 0))

    # Saving the inverted colors image
    new_img_obj.save(comb_img)
    diff_img_obj.save(diff_img)
    return diff_pixels
```

# TEST RESULTS

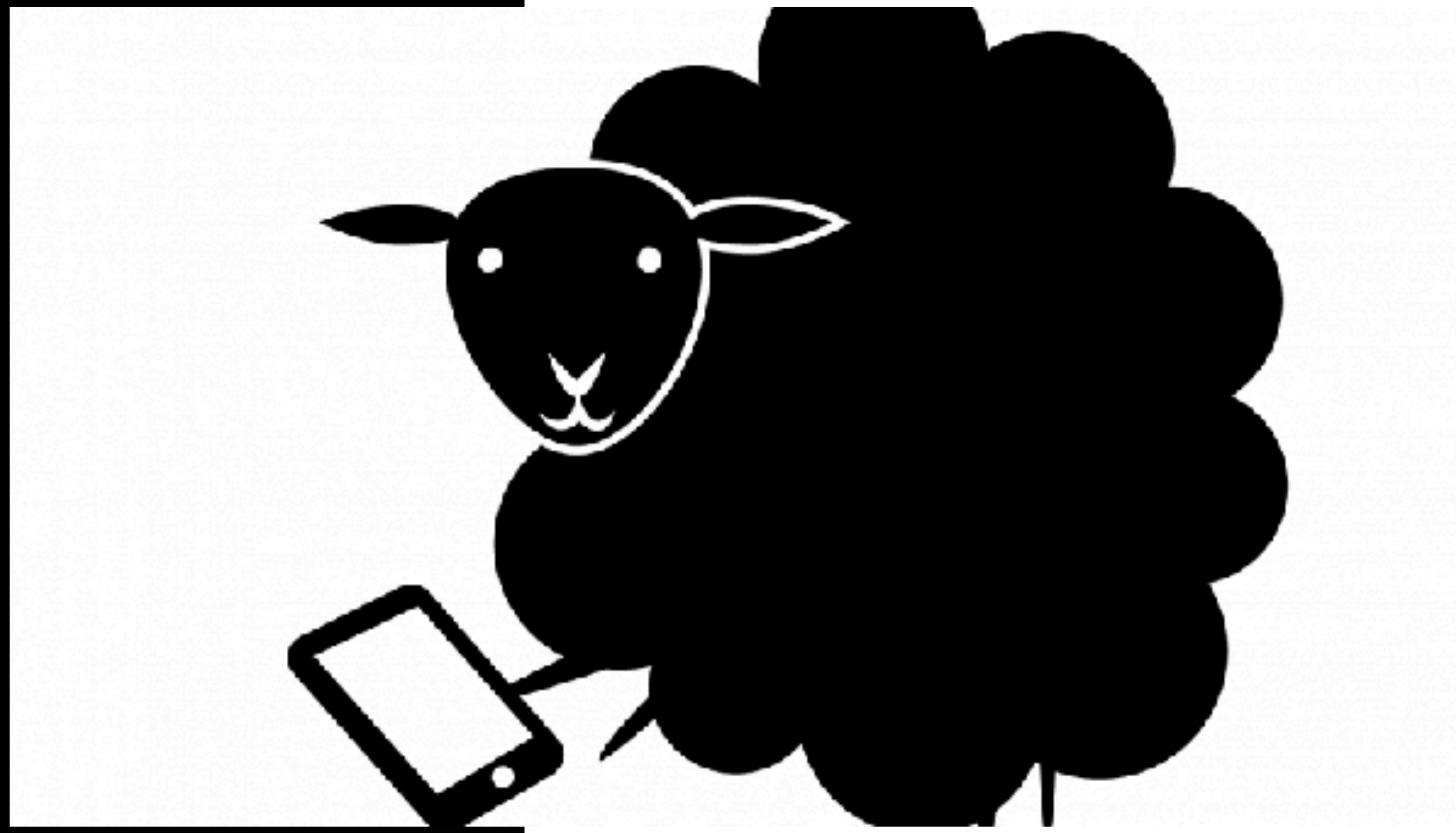
**iOS PluginTester 05/05/18 16:48**

Test summary (see log file for more info): 1 FAILURE(s)

**GUI Verification**

| # | Plugin name | Component | Status | Diff image | Combined image |
|---|---|---|---|---|---|
| 1 | Q10 | Stereo | 9389 pixels different | Diff | Comb |
| 2 | AudioTrack | Stereo | Pass | | |
| 3 | L1+Ultramaximizer | Stereo | Pass | | |

# DEMO



# AUTOMATING MOBILE GESTURES

- TouchAction objects contain a chain of events

- They simulate user actions on the touch screen

**Element based actions**

```
TouchAction().press(el0).moveTo(el1).release()
```

**Positional based actions**

```
.press(100,100) # Start at 100,100
.moveTo(100,100) # Increase X & Y by 100 each, ending up at 200,200
```

## TouchAction MOBILE GESTURES

**The available events from the spec are:**

- press
- release
- moveTo
- tap
- wait
- longPress
- cancel
- perform

WAVES

# MultiAction MOBILE GESTURES

- MultiTouch objects are collections of TouchActions.
- MultiTouch gestures only have two methods, add, and perform.

```
action0 = TouchAction().tap(el)
action1 = TouchAction().tap(el)
MultiAction().add(action0).add(action1).perform()
```

## DEMO

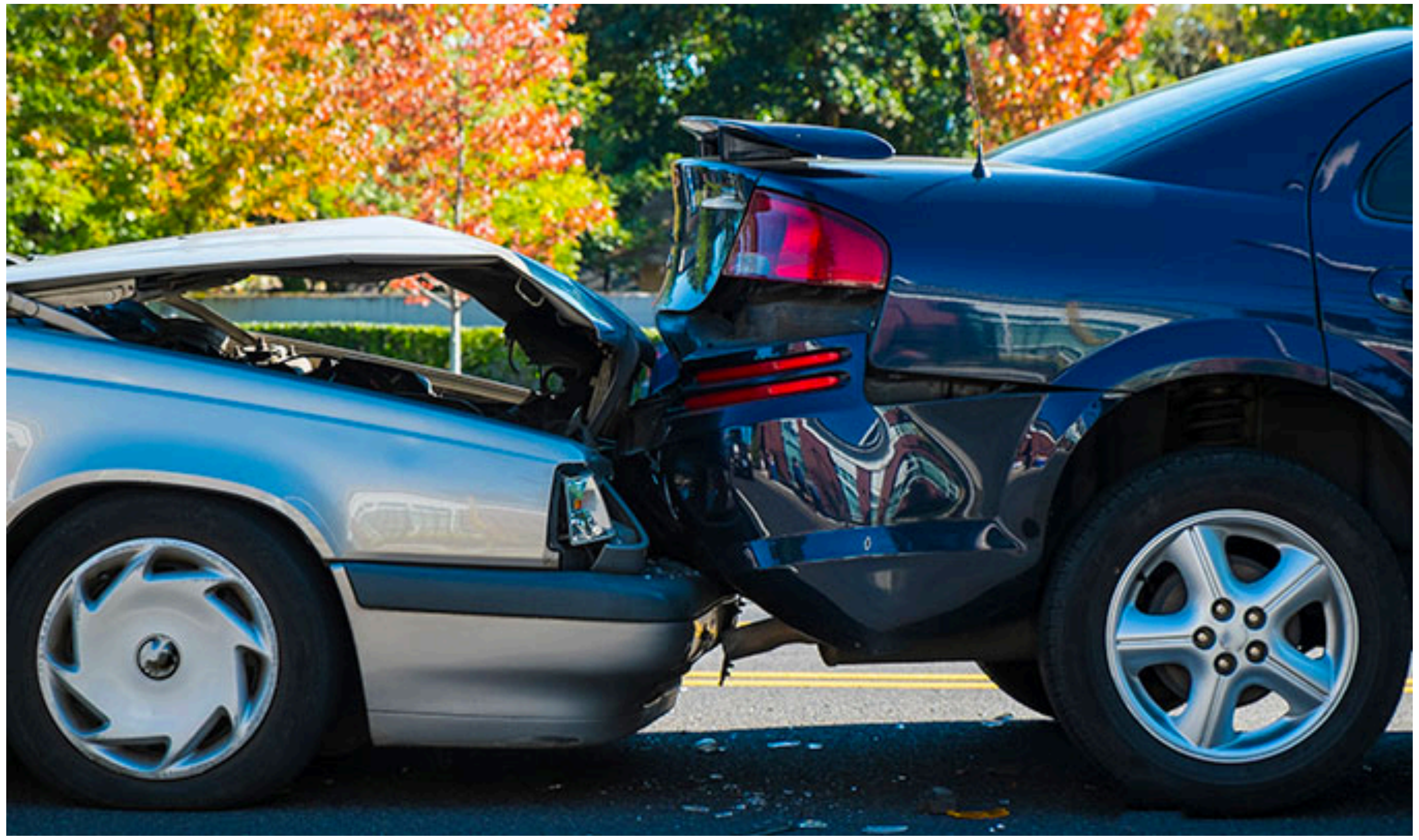
# APPIUM PHILOSOPHY

- You shouldn’t have to recompile your app or modify it in any way in order to automate it
- You shouldn’t be locked into a specific language or framework to write and run your tests
- A mobile automation framework shouldn’t reinvent the wheel when it comes to automation APIs
- A mobile automation framework should be open source, in spirit and practice as well as in name

## FINAL TIPS

- Divide your tests into small chunks
- Choose tests that take a long time to complete manually
- Test reference data (images, audio files) have to be updated with the product
- Not every test is suitable as automated test

# Automated testing is an insurance policy



## Links and sources:

**Nir Arad**

nir.arad@gmail.com

@nirarad1 #pycon2018

http://appium.io

https://nishantverma.gitbooks.io/appium-for-android - Nishant Verma

https://medium.com/@dmathewwws/steps-to-put-your-app-on-testflight-and-

then-the-ios-app-store-10a7996411b1 - Daniel Mathews

http://www.waves.com

Thank you

WAVES