GUI Testing and Automation with Sikuli

Bob Igo, Digital Arc Systems

The Master Plan

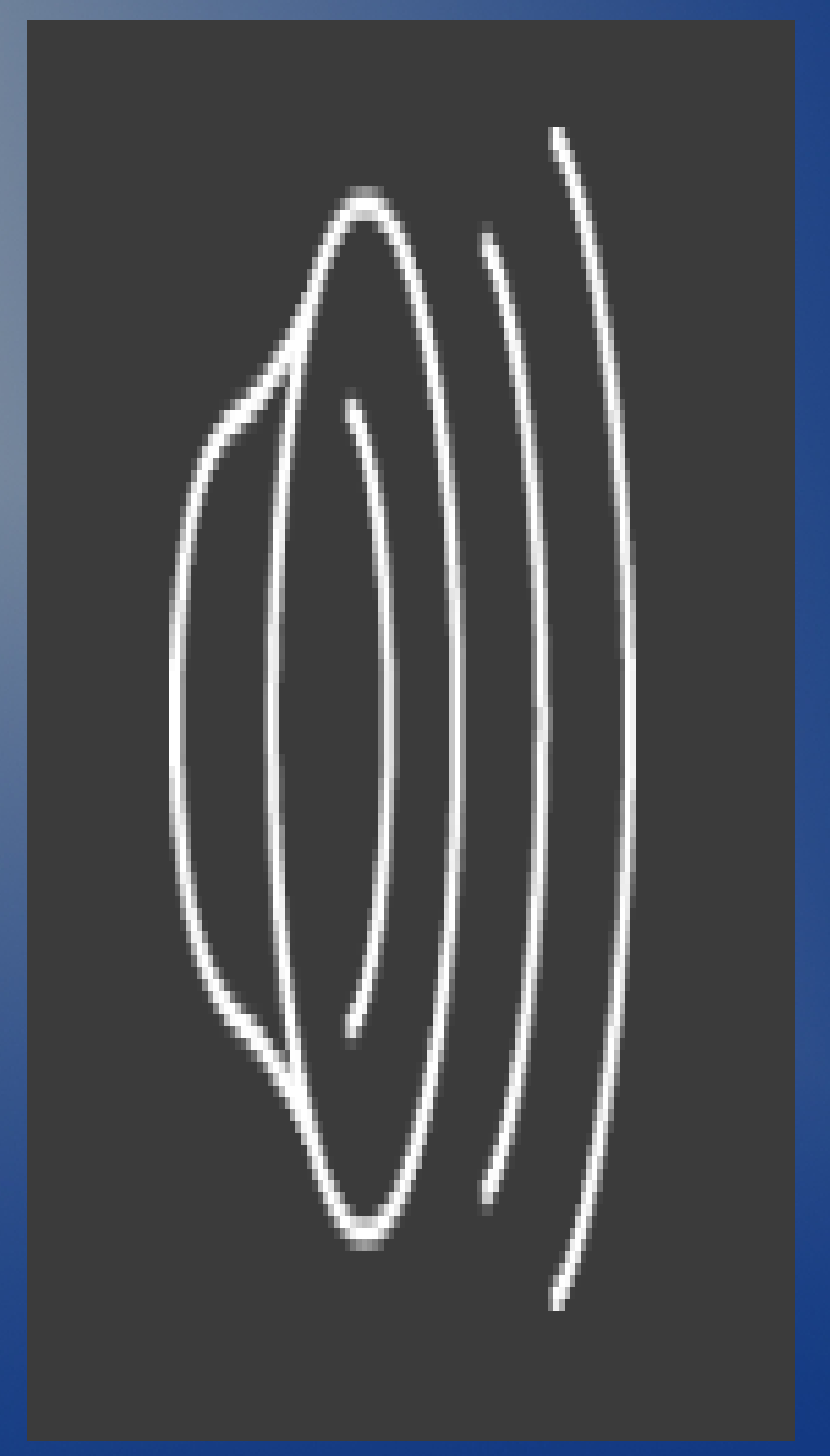
- Demonstration
- What is Sikuli?
- Sikuli Basics
- Basic Testing
- Intermediate Testing
- Advanced Testing
- Inherent Limitations

Running a Sikuli Script

- Invocation of Sikuli script, Example1:

```
sikuli-ide.sh -r
./unlock_emulator_and_run_browser.s
ikuli
```

Demonstration



Demonstration

How did that work?

- Sikuli can *see*.
- Sikuli finds on-screen matches with a reference image
  -
- It can then perform any keyboard or mouse action at or near the matches
  - `mouseDown(Button.LEFT)`

What does Sikuli do, in general?

- It can use any GUI that you can use
  - native
  - Flash/Silverlight
  - cross-platform
- run anywhere, displayed locally
  - local program
  - via VNC
  - inside VMs
    - headless or not
  - ssh + X11 forwarding

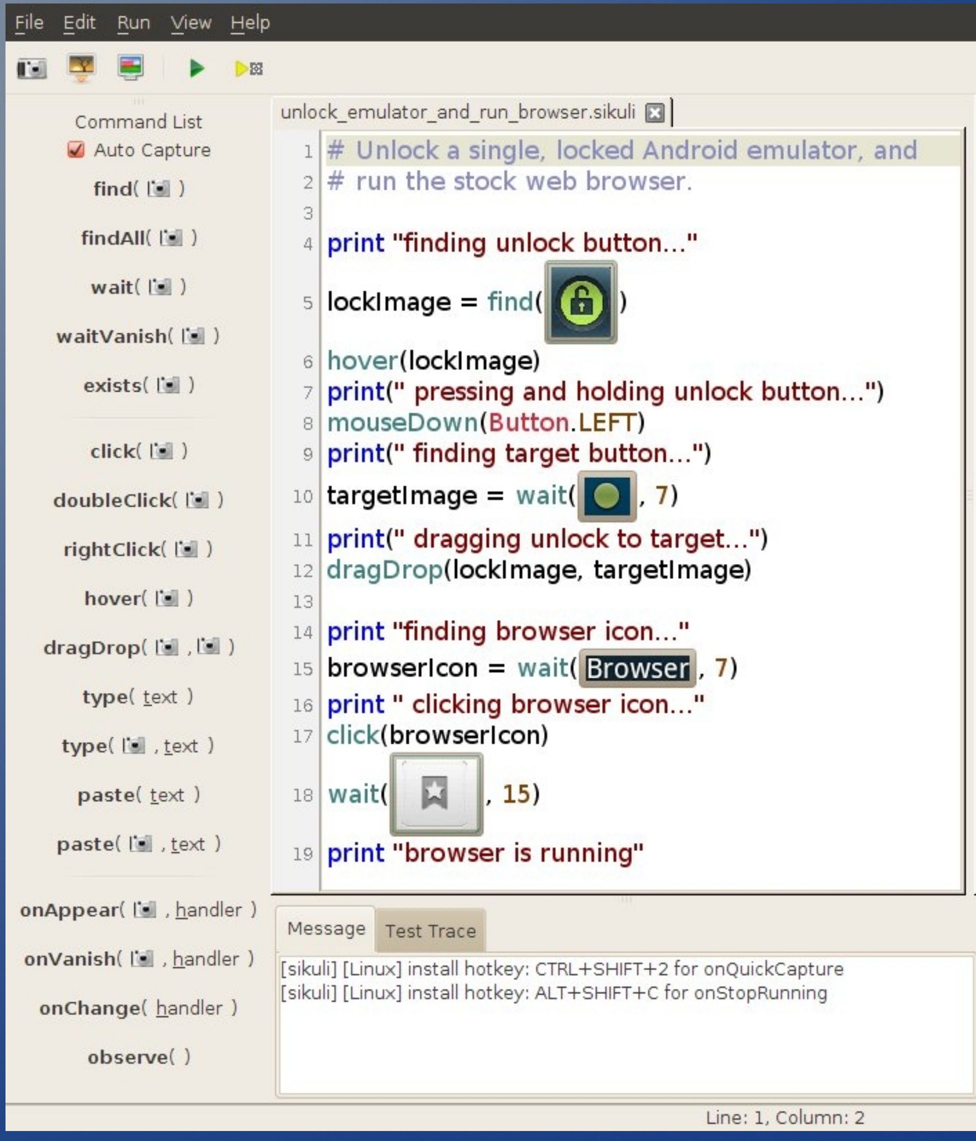
How do you tell it what to do?

- You write *Sikuli scripts* using its python API
  - by hand or in the Sikuli IDE
- You run Sikuli scripts in two ways:
  - Click in the IDE.
  - `sikuli-ide.sh -r ./`*your_script_name*`.sikuli`

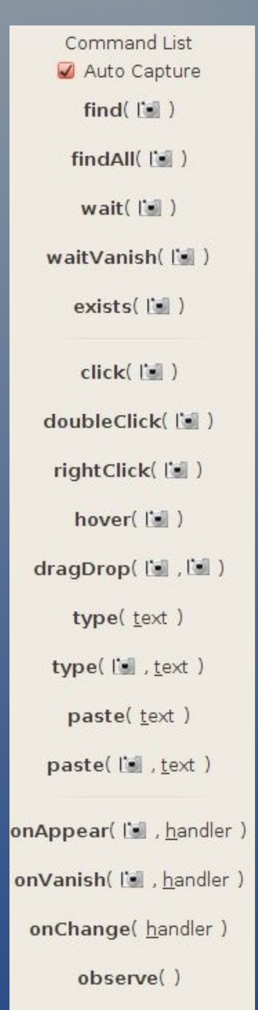
Anatomy of a Sikuli Script: Example1

```
# Unlock a single, locked Android emulator, and
# run the stock web browser.

print "finding unlock button..."
lockImage = find( )
hover(lockImage)
print(" pressing and holding unlock button...")
mouseDown(Button.LEFT)
print(" finding target button...")
targetImage = wait( , 7)
print(" dragging unlock to target...")
dragDrop(lockImage, targetImage)

print "finding browser icon..."
browserIcon = wait( Browser , 7)
print " clicking browser icon..."
click(browserIcon)
wait( , 15)
print "browser is running"
```

Sikuli IDE, Showing Example1



Using the Sikuli IDE

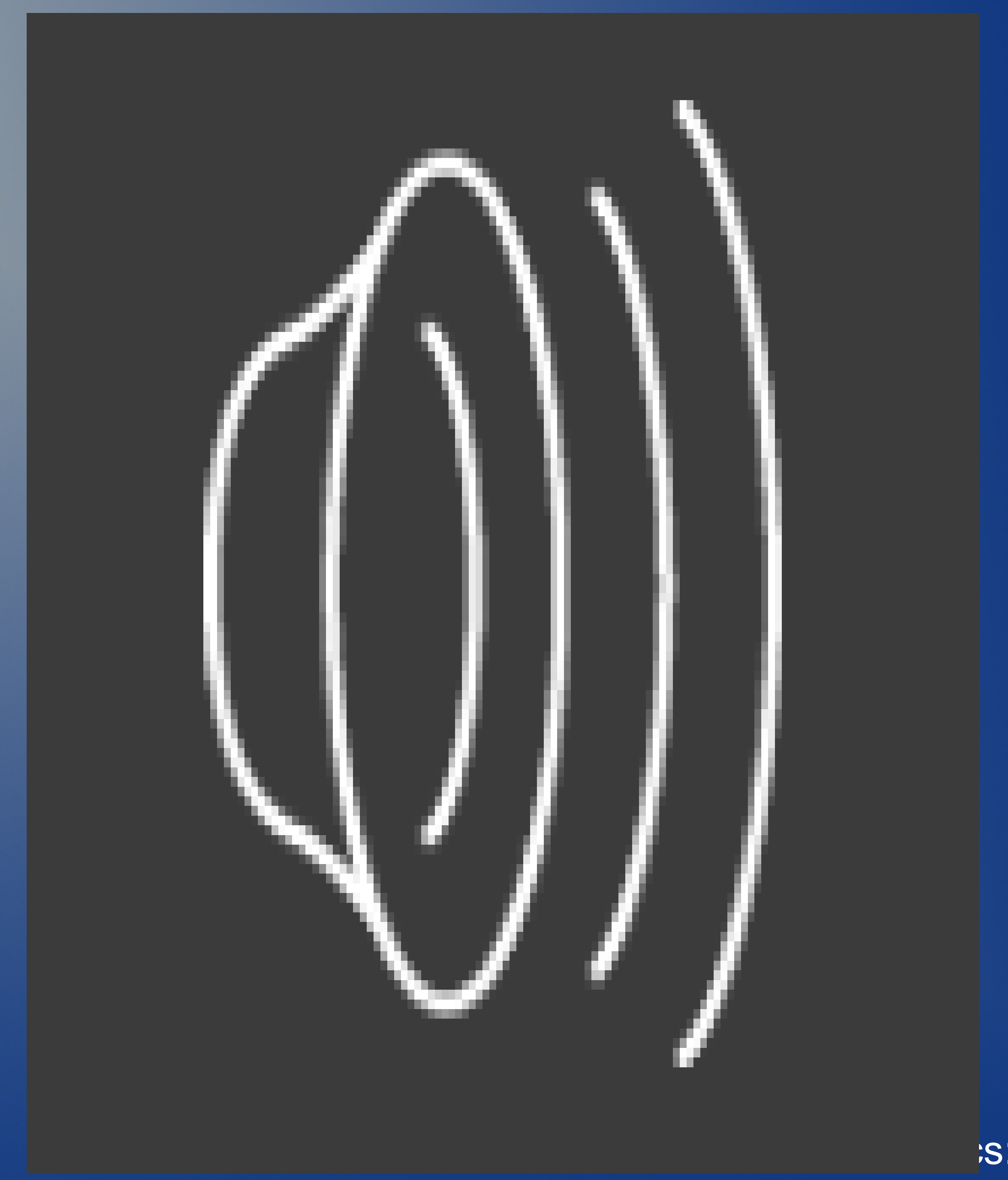


- Left column: Sikuli commands, coupled with optional image capture
- Most buttons do this:
  - help capture a reference image
  - insert the image and the Sikuli command into the script

Using the Sikuli IDE

- Right column: Your Sikuli script, in a special editor
- Use as a text editor
- Left column's buttons inject code at the cursor

```
# Unlock a single, locked Android emulator, and
# run the stock web browser.

print "finding unlock button..."
lockImage = find( )
hover(lockImage)
print(" pressing and holding unlock button...")
mouseDown(Button.LEFT)
print(" finding target button...")
targetImage = wait( , 7)
print(" dragging unlock to target...")
dragDrop(lockImage, targetImage)

print "finding browser icon..."
browserIcon = wait( Browser , 7)
print " clicking browser icon..."
click(browserIcon)
wait( , 15)
print "browser is running"
```



Questions?

- Next up, some Quirks and Gotchas

Basic Quirks and Gotchas

- Sikuli: awesome, but has rough edges
- IDE is best for rapid prototyping
  - not all Sikuli commands are in the IDE's left column
  - python indent issues
  - no parser warnings/errors
  - no UNDO
  - can be unstable

Basic Quirks and Gotchas

- Sikuli ”files” are actually directories named *scriptname*`.sikuli`, containing your `.png` images, the *scriptname*`.py` and *scriptname*`.html`
- When loading/saving in the IDE, load/save the directory name, **not** *scriptname*`.py`

Basic Quirks and Gotchas

- To run the IDE and load a script in one action:
  - `sikuli-ide.sh ./scriptname.sikuli`

Automation → Test

- Recall Example1:

```
# Unlock a single, locked Android emulator, and
# run the stock web browser.

print "finding unlock button..."
lockImage = find( [image] )
hover(lockImage)
print(" pressing and holding unlock button...")
mouseDown(Button.LEFT)
print(" finding target button...")
targetImage = wait( [image] , 7)
print(" dragging unlock to target...")
dragDrop(lockImage, targetImage)

print "finding browser icon..."
browserIcon = wait( Browser , 7)
print " clicking browser icon..."
click(browserIcon)
wait( [image] , 15)

print "browser is running"
```

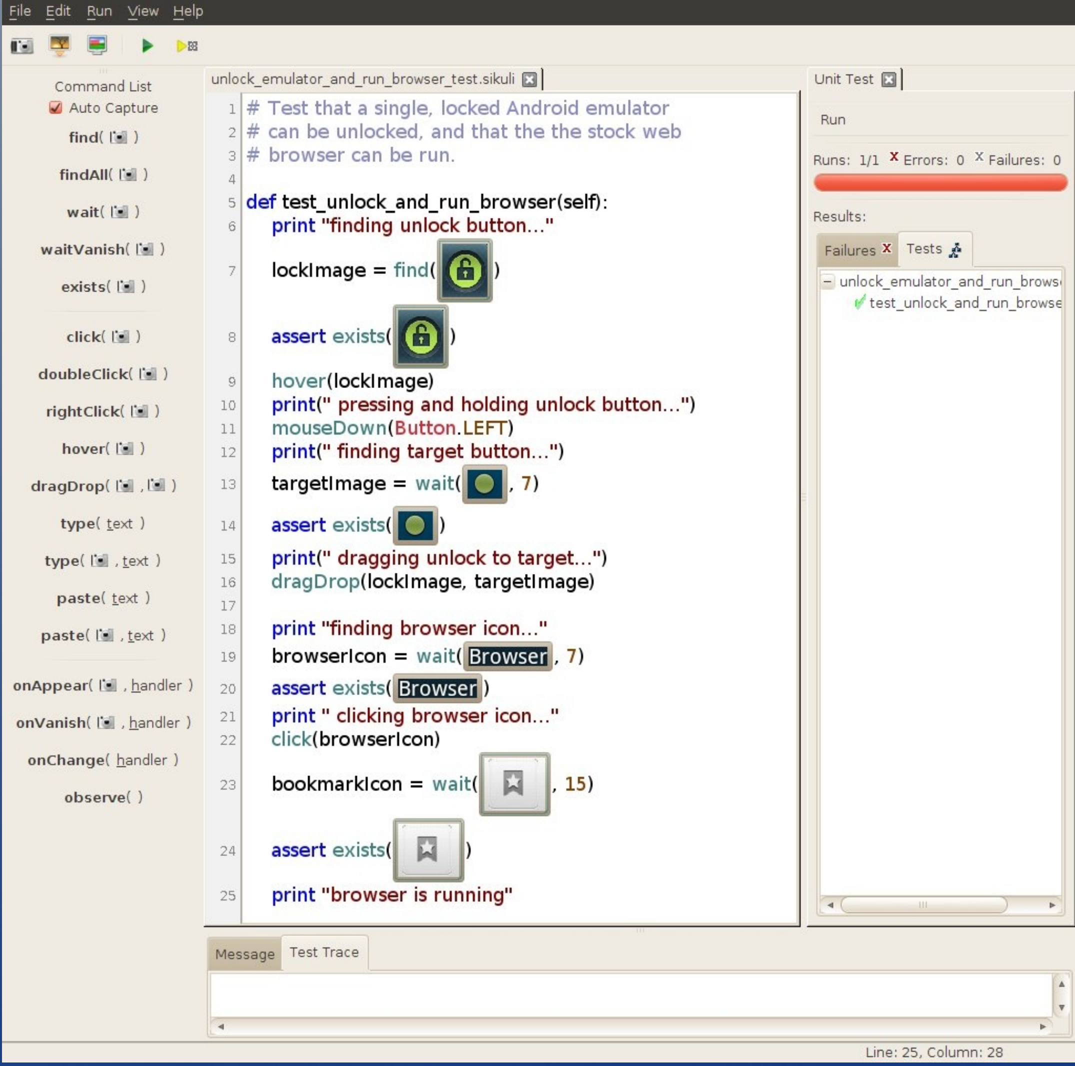
Automation → Test

- Test 1: Trivial conversion to JUnit test:

```
# Test that a single, locked Android emulator
# can be unlocked, and that the the stock web
# browser can be run.

def test_unlock_and_run_browser(self):
    print "finding unlock button..."
    lockImage = find( )
    assert exists( )
    hover(lockImage)
    print(" pressing and holding unlock button...")
    mouseDown(Button.LEFT)
    print(" finding target button...")
    targetImage = wait( , 7)
    assert exists( )
    print(" dragging unlock to target...")
    dragDrop(lockImage, targetImage)

    print "finding browser icon..."
    browserIcon = wait(Browser, 7)
    assert exists(Browser)
    print " clicking browser icon..."
    click(browserIcon)
    bookmarkIcon = wait( , 15)
    assert exists( )
    print "browser is running"
```

Automation → Test

- Select View → Unit Test in Sikuli IDE



Automation → Test

- Or run like so:
  - `sikuli-ide.sh -t ./unlock_emulator_and_run_browser_test.sikuli`

Why JUnit?

- Regular Sikuli script:
  - exit code is pass/fail
  - if the script fails at any point, no more code is called
- A JUnit Sikuli script:
  - exit code is pass/fail
  - adds text detailing the test process, e.g.:

```
Time: 12.156
OK (1 test)
```

  - if the script fails inside a test, other tests still run

Test Expansion

- Adding a timing test
- You can import any python module
  - e.g.

```
import time
```

Test Expansion

- Test1A: Make it fail if it takes longer than 15 seconds.

```
def test_unlock_and_run_browser(self):
    startTime = time.clock()
    print "finding unlock button..."
    lockImage = find( )
    assert exists( )
    hover(lockImage)
    print(" pressing and holding unlock button...")
    mouseDown(Button.LEFT)
    print(" finding target button...")
    targetImage = wait( , 7)
    assert exists( )
    print(" dragging unlock to target...")
    dragDrop(lockImage, targetImage)

    print "finding browser icon..."
    browserIcon = wait( Browser , 7)
    assert exists( Browser )
    print " clicking browser icon..."
    click(browserIcon)
    bookmarkIcon = wait( , 15)
    assert exists( )
    print "browser is running"
    endTime = time.clock()
    print("runtime: ", endTime - startTime)
    assert (endTime - startTime < 15)
```

Limitations of Test1A

- Does not play well with other tests (or itself, run more than once)
- Monolithic
- No branching logic
- Does not share code with other tests

Intermediate Testing

- Let's address some of the limitations of Test1A
- Start with setUp() and tearDown()
  - addresses first two limitations

Playing Well With Others

- JUnit makes use of setUp() and tearDown() methods.
  - setUp() is called before each test_*() method
  - tearDown() is called after each test_*() method

```
def test_unlock_and_run_browser(self):
    startTime = time.clock()
    print "finding unlock button..."
    lockImage = find()
    assert exists()
    hover(lockImage)
    print(" pressing and holding unlock button...")
    mouseDown(Button.LEFT)
    print(" finding target button...")
    targetImage = wait(, 7)
    assert exists()
    print(" dragging unlock to target...")
    dragDrop(lockImage, targetImage)

    print "finding browser icon..."
    browserIcon = wait(Browser, 7)
    assert exists(Browser)
    print " clicking browser icon..."
    click(browserIcon)
    bookmarkIcon = wait(, 15)
    assert exists()
    print "browser is running"
    endTime = time.clock()
    print("runtime: ", endTime - startTime)
    assert (endTime - startTime < 15)
```

First step: Break out the setup behavior into setUp()

```
def setUp(self):
    print "setUp"
    print "finding unlock button..."
    lockImage = find()
    assert exists()
    hover(lockImage)
    print(" pressing and holding unlock button...")
    mouseDown(Button.LEFT)
    print(" finding target button...")
    targetImage = wait(, 7)
    assert exists()
    print(" dragging unlock to target...")
    dragDrop(lockImage, targetImage)
def test_unlock_and_run_browser(self):
    startTime = time.clock()
    print "finding browser icon..."
    browserIcon = wait(Browser, 7)
    assert exists(Browser)
    print " clicking browser icon..."
    click(browserIcon)
    bookmarkIcon = wait(, 15)
    assert exists()
    print "browser is running"
    endTime = time.clock()
    print("runtime: ", endTime - startTime)
    assert (endTime - startTime < 15)
```

Second step: add teardown behavior to tearDown()

Now, We Have Test1B

```
import time

def setUp(self):
    print "setUp"
    print "finding unlock button..."
    lockImage = find( )
    assert exists( )
    hover(lockImage)
    print(" pressing and holding unlock button...")
    mouseDown(Button.LEFT)
    print(" finding target button...")
    targetImage = wait( , 7)
    assert exists( )
    print(" dragging unlock to target...")
    dragDrop(lockImage, targetImage)

def tearDown(self):
    type(Key.ESC)
    type(Key.F7)
    wait( , 15)
    type(Key.F7)
```

```
def test_unlock_and_run_browser(self):
    startTime = time.clock()
    print "finding browser icon..."
    browserIcon = wait(Browser, 7)
    assert exists(Browser)
    print " clicking browser icon..."
    click(browserIcon)
    bookmarkIcon = wait( , 15)
    assert exists( )
    print "browser is running"
    endTime = time.clock()
    print("runtime: ", endTime - startTime)
    assert (endTime - startTime < 15)
```

Limitations of Test1B

- ~~Does not play well with other tests (or itself, run more than once)~~
- ~~Monolithic~~
- No branching logic
- Does not share code with other tests

Intermediate Testing

- Let's add some branching logic.
- Maybe the emulator is already unlocked
  - perhaps another test crashed and wasn't able to call tearDown()
  - we don't want to fail our test because another test misbehaved

Test1C

- We can use python try/except blocks to know when Sikuli fails to find an image

```
def setUp(self):
    print "setUp"
    try:
        find( )
    except FindFailed:
        print "finding unlock button..."
        try:
            lockImage = find( )
            assert exists( )
            hover(lockImage)
            print(" pressing and holding unlock button...")
            mouseDown(Button.LEFT)
            print(" finding target button...")
            targetImage = wait( , 7)
            assert exists( )
            print(" dragging unlock to target...")
            dragDrop(lockImage, targetImage)
        except FindFailed:
            assert False
```

Questions?

- Next up, Advanced Testing

Limitations of Test1C

- ~~Does not play well with other tests (or itself, run more than once)~~
- ~~Monolithic~~
- ~~No branching logic~~
- Does not share code with other tests

More Quirks and Gotchas

- You can import any python module (e.g. `time`) using `import`, but this doesn't work if the module contains calls to the Sikuli API.
  - basic issue is subtle and involves how Sikuli turns a fairly normal .py script into a Sikuli script
- We need a workaround to share one Sikuli script's methods with another one.

Sharing Sikuli Script Methods

- Use execfile() to trick Sikuli
- Our scenario
  - Let's say we have 100 test scripts that involve unlocking and re-locking the Android emulator
  - for maintenance, setUp() and tearDown() need to be in one place, not 100 places

Test1C → Test1D

- advanced_test_utils.py

```
def unlockEmulator(self):
        try:
                print "seeing if emulator is already unlocked..."
                find("/home/igo/Documents/CPOSC_2010_Sikuli/sikuli_scripts/advanced_test_utils.sikuli/1286990500931.png")
                return True
        except FindFailed:
                print "finding unlock button..."
                try:
                        lockImage = find("/home/igo/Documents/CPOSC_2010_Sikuli/sikuli_scripts/advanced_test_utils.sikuli/unlock_button.png")
                        hover(lockImage)
                        print(" pressing and holding unlock button...")
                        mouseDown(Button.LEFT)
                        print(" finding target button...")
                        targetImage = wait("/home/igo/Documents/CPOSC_2010_Sikuli/sikuli_scripts/advanced_test_utils.sikuli/unlock_target.png", 7)
                        print(" dragging unlock to target...")
                        dragDrop(lockImage, targetImage)
                        return True
                except FindFailed:
                        return False

def lockEmulator(self):
        try:
                type(Key.ESC)
                type(Key.F7)
                wait("/home/igo/Documents/CPOSC_2010_Sikuli/sikuli_scripts/advanced_test_utils.sikuli/unlock_button.png", 15)
                type(Key.F7)
                return True
        except FindFailed:
                return False
```

- All image names must be full paths

Test1C → Test1D

- Warning: The IDE will delete those image files if you load the previous .py file into Sikuli.
- Once you add full paths to images, keep the script out of the IDE forever.

Test1C → Test1D

- `advanced_test_1.py`

```
execfile("/home/igo/Documents/CPOSC_2010_Sikuli/sikuli_scripts/advanced_test_utils.sikuli/advanced_test_utils.py")

import time

def setUp(self):
    print "setUp"
    assert self.unlockEmulator()

def tearDown(self):
    print "tearDown"
    assert self.lockEmulator()

def test_unlock_and_run_browser(self):
    startTime = time.clock()
    print "finding browser icon..."
    browserIcon = wait(Browser, 7)
    assert exists(Browser)
    print " clicking browser icon..."
    click(browserIcon)
    bookmarkIcon = wait( , 15)
    assert exists( )
    print "browser is running"
    endTime = time.clock()
    print("runtime: ", endTime - startTime)
    assert (endTime - startTime < 15)
```

- It's ok to load this into the IDE.

Questions?

- Next up, Inherent Sikuli Limitations

Inherent Sikuli Limitations

- Very CPU-intensive
  - mitigators: `setROI()`, `wait(s)/sleep(s)`
- Can't reliably find transient imagery
  - a fixed image in motion
  - a frame of a 30fps video

Inherent Sikuli Limitations

- Compared with human testers, unable to report that something unexpected happened, if everything expected also happened
- Translucency can make matches difficult

More Quirks and Gotchas

- Sikuli creates and populates a `tmplib` directory anywhere you run a Sikuli script
  - you'll need to clean up after it

Things I Didn't Cover

- `findAll()` is useful if you want to test several GUIs at the same time
- Sikuli provides `openApp()` and `closeApp()` to run and kill processes, but it works strangely in Linux.
  - You're better off writing a wrapper that runs (or verifies as running) the program you want to test, then runs the Sikuli script.
- Managing your image file names

Things I Didn't Cover

- Speed issues
  - sometimes much slower than human testers
  - sometimes much faster than human testers
- writing a wrapper to launch scripts and handle return values
- setting the match threshold for imagery
  - default is 70%
- installing Sikuli

References

- Sikuli Project Home
  - http://groups.csail.mit.edu/uid/sikuli/
- Sikuli API reference
  - http://sikuli.org/trac/wiki/reference-0.10
- Slides from this and my other talks
  - http://bob.igo.name/?s=slides
- Catherine Devlin's Sikuli talk
  - http://catherinedevlin.pythoneers.com/presentations/sikuli/sikuli.html
- Sikuli import workaround discussion
  - https://answers.launchpad.net/sikuli/+faq/1114
- Sikuli tickets
  - https://bugs.launchpad.net/sikuli
- OpenCV
  - http://opencv.willowgarage.com/wiki/