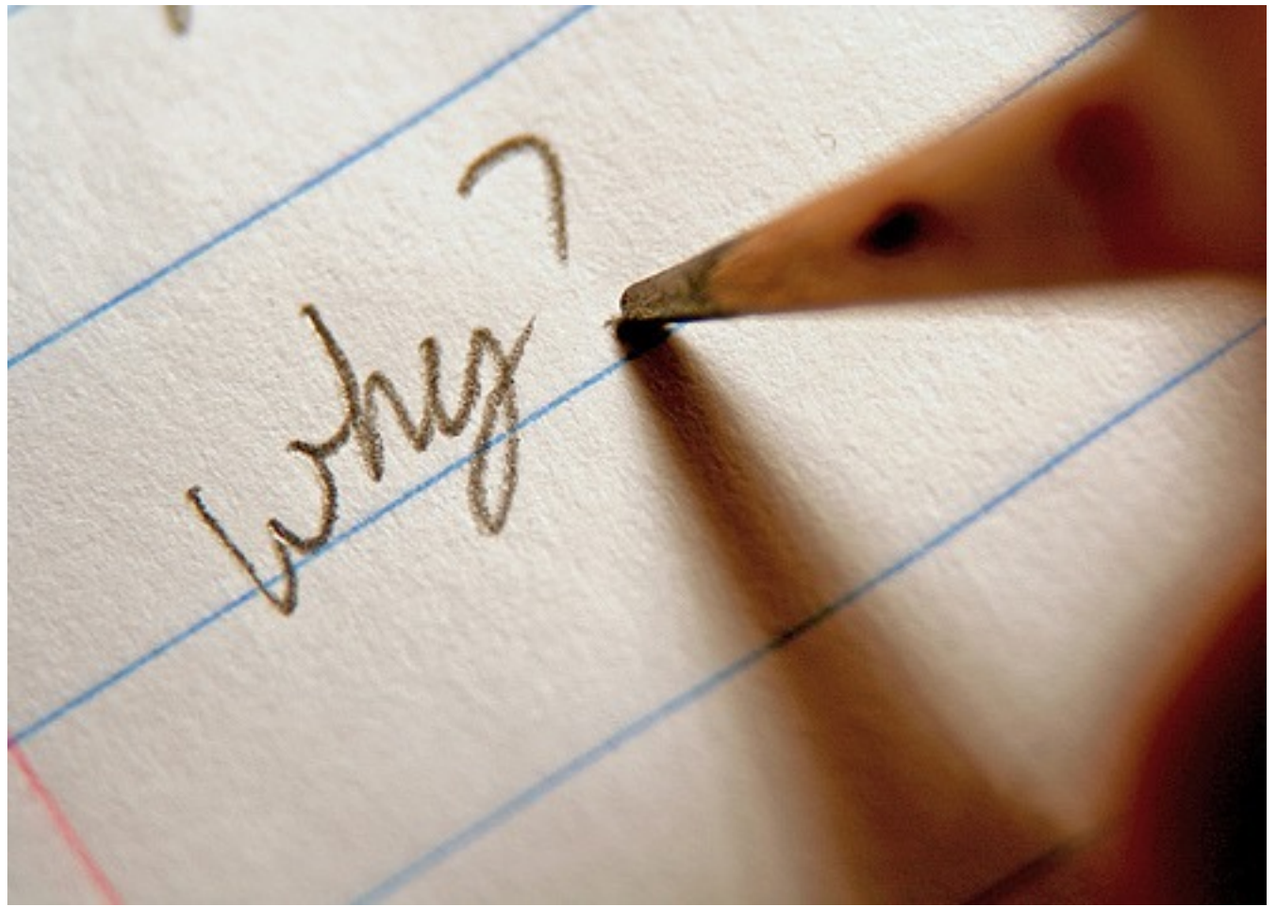
Functional UI testing of Adobe Flex RIA

Viktor Gamov
viktor.gamov@faratasystems.com
August, 12 2011

Agenda

- Why to test?
- How to test?
- What the automated testing means?
- Automated testing tools
- Automated testing architecture
- Loading automation classes
- Creating test-friendly applications
- Bunch of Demos


Why?

Software has to satisfy the user!

Demo: Killer RIA

How to test?

- Unit Testing
- Functional (UAT, QA) Testing
- Integration Testing
- Load Testing

Metaphor: building the house

- think of unit tests as having the building inspector at the construction's site
- think of functional testing as having the homeowner visiting the house and being interested in how the house looks, if the rooms have the desired size

What automated testing means?

Automated testing is the use of software to control the execution of tests, the comparison of actual outcomes to predicted outcomes, and other test control and test reporting functions.

General approaches

- Code-Driven Testing - the public (usually) interfaces to classes, modules, or libraries are tested with a variety of input arguments to validate that the returned results are correct.

- GUI Testing - a testing framework generates and records the UI events keystrokes and mouse clicks. The tester observes the changes reflected via the user interface to validate the observable behavior of the program.

Some Automated Testing tools

- HP QuickTest Professional (QTP)
- Selenium
- Ranorex
- FlexMonkey

HP QTP

- All-in-one suite for automation testing enterprise applications
- Supports the large pool of software development environments like SAP , Web , Oracle etc.
- Support libraries out-of-the-box with Flex (*qtp.swc, qtp_air.swc*)
- VBScript as scripting language

Selenium

- Web applications testing
- Open source and large community
- Cross platform (Selenium IDE - browser extension)
- Uses JavaScript as scripting language

Ranorex

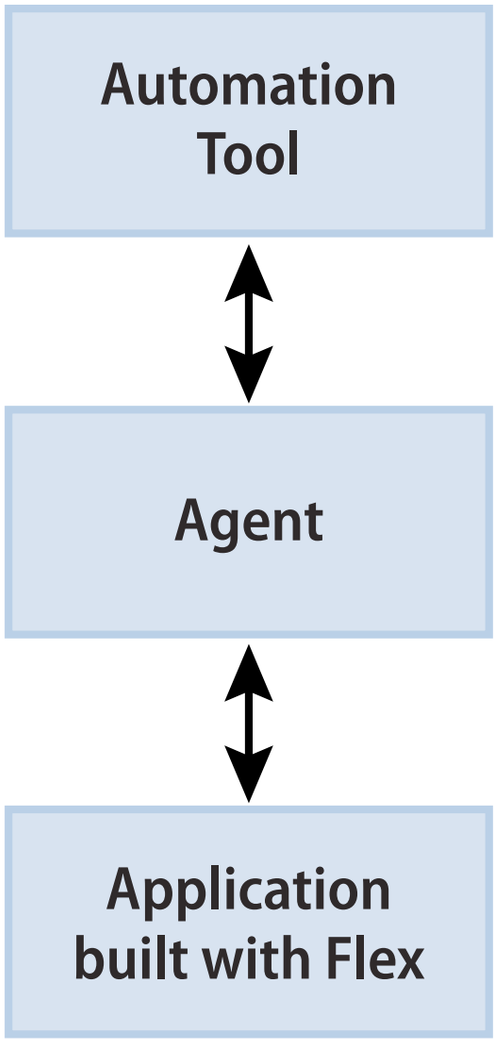
- Similar to QTP, but easier to use
- Support various technologies: .NET, WPF, Silverlight, Qt, Win32, WFC, Java SWT, Flash/ Flex
- Script languages C#, VB.NET, IronPython
- Integrates with Visual Studio 2010
- Platform: Windows only

Flex Monkey

- Open Source (commercial support available)
- Flex apps testing centric tool
- Cross platform (console - AIR application)
- Generated FlexUnit 4 test code
- Supports testing only Flex applications (no Flash)
- Integration with Selenium IDE (FlexMonkium project)

Demo

Automated Testing Architecture

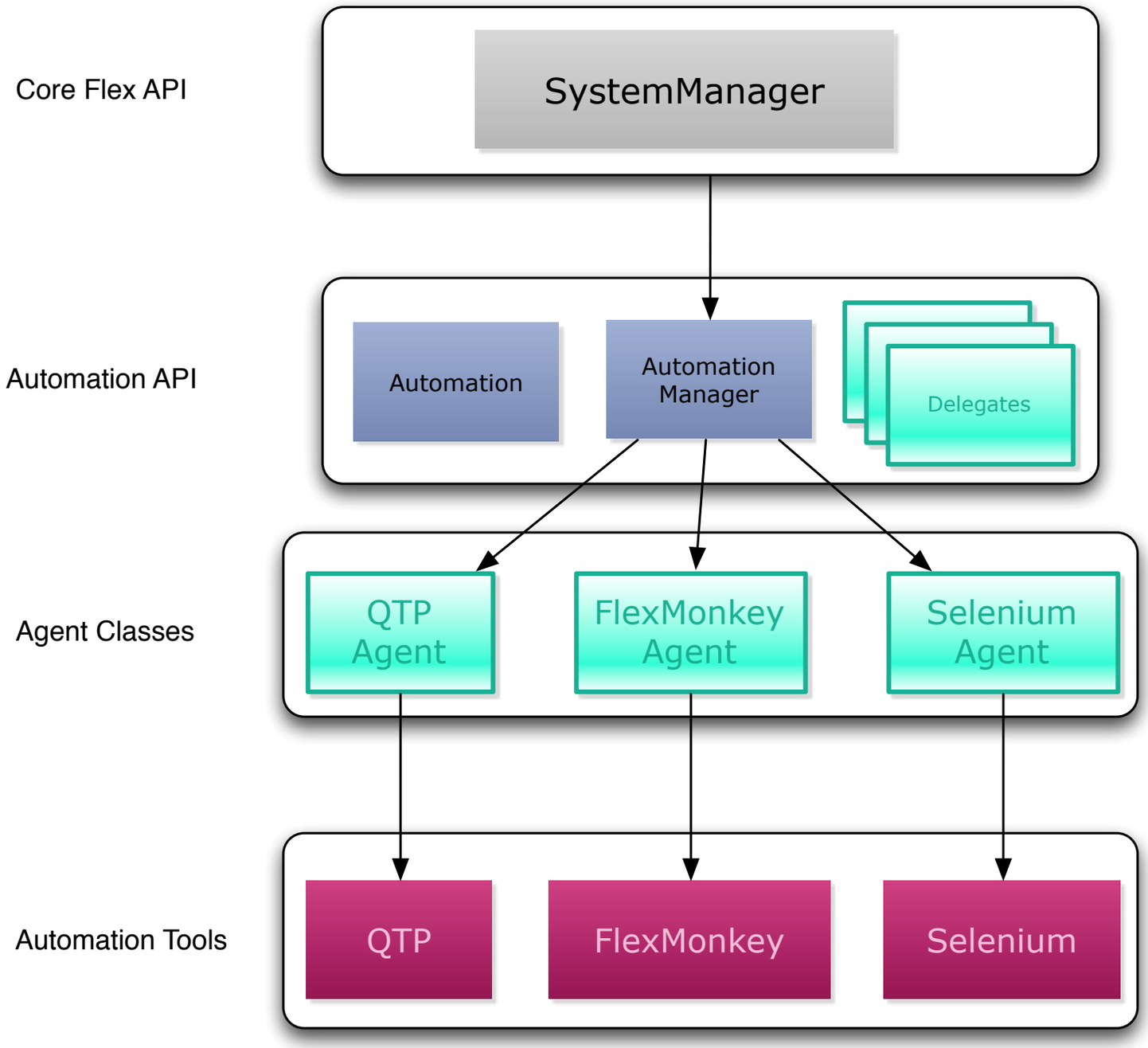


- *automation tool* are applications that use the data that provided by the agent.

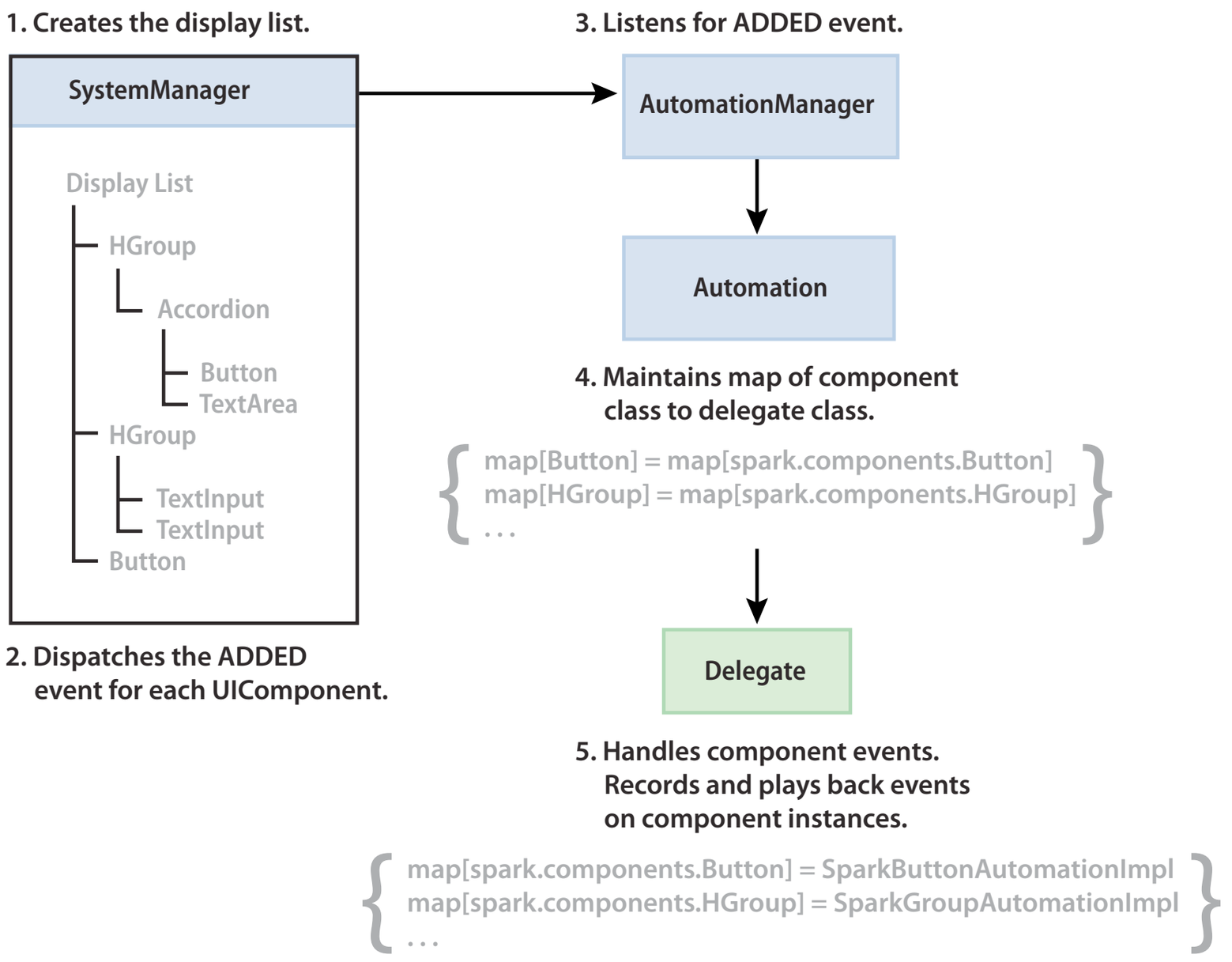
- *automation agent* facilitates communications between an application and an automation tool.

- *delegates* are Flex framework components are instrumented by attaching a delegate class to each component at run time. The *delegate class* defines the methods and properties required to perform automation.

Automation API

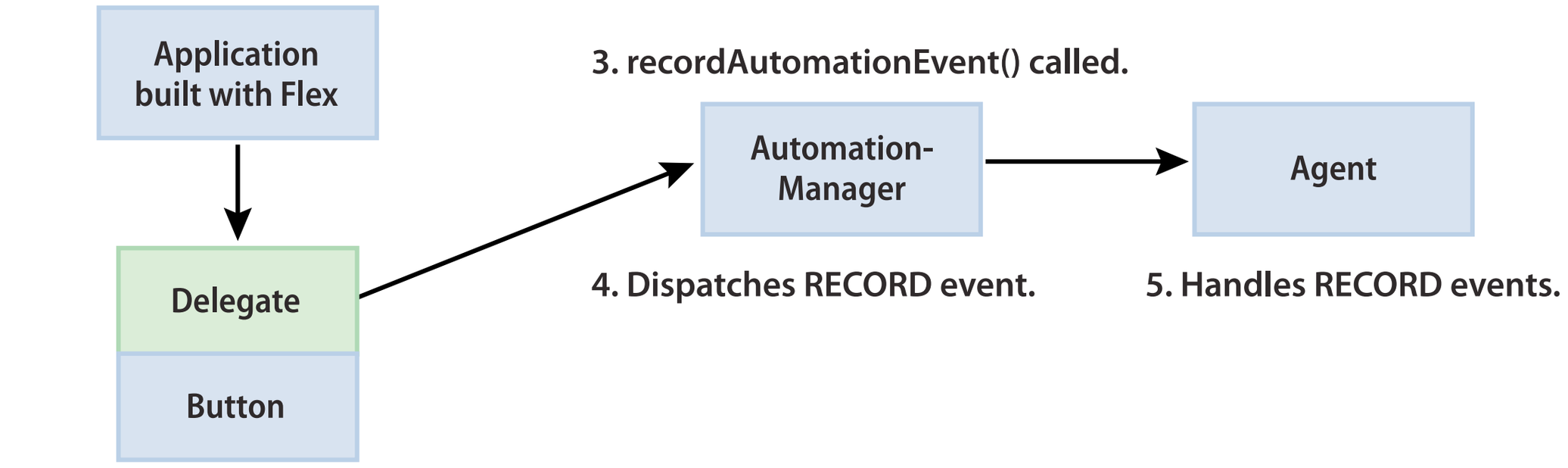


Automation flow



Automation flow

1. User clicks on a component.



2. Listens for component events such as CLICK.

Loading automation classes

- Compile time - To compile an application that includes the automation classes, include automation libraries at compile time by using the compiler's *include-libraries* option.

- Run time - Compile your app without any additions. At run time, load your application into a wrapper SWF file that has built in automation libraries. This wrapper SWF file uses SWFLoader to load your application SWF file to be tested.

Creating test-friendly applications

- providing meaningful identification of objects

  "*submitPanel*" rather than "*myPanel*" or "*p1*"

- avoiding renaming objects

  if (!automationName) return label;

- adding and removing containers from the automation hierarchy

  showInAutomationHierarchy = false;

- instrumenting events
- instrumenting custom components

Instrumenting events

- **instrumenting existing events -** the tools are not generally interested in recording all the events : *MOUSE_OVER*

- **instrumenting custom events -** when you extend components, you often add events that are triggered by new functionality of that component

- **blocking and overriding events -** in some cases, you might want to block or override the default events that are recorded for a component

Instrumenting existing events

- override the `replayAutomatableEvent()` method of the *IAutomationObject* interface
- call the *AutomationManager*'s `recordAutomatableEvent()` method when a button dispatch *MOUSE_MOVE* event
- define the new event for the agent

Instrumenting custom events

- call the `Automation.automationManager2` class's `recordCustomAutomationEvent()` method
- define the new event for the agent
- listen for the `AutomationCustomReplayEvent`, get the details about it and replay the event

What is a custom component

- A component extending from a standard component (No additional events, no additional properties)
- A component extending from a standard component – Additional events/ additional properties
- A component extending from a non container class, but the current class is a container.

Instrumenting custom components

- create a delegate class that implements the required interfaces (`IAutomationObject`)
- add testing-related code to the component (not recommended)

Create a delegate class

- use a pattern for delegate class name `[ComponentClassName]AutomationImpl`
- extend the `UIComponentAutomationImpl` or implement `IAutomationObject` interface
- mark the delegate class as a mixin by using the [Mixin] metadata keyword
- register the delegate with the *AutomationManager* by calling the *Automation.registerDelegateClass()*

Instrument component with a delegate class

- override the getters of the *automationName* and the *automationValue* properties
- add listeners for events that the automation tool records
- override the *replayAutomatableEvent()* method
- add the new component to custom class definition XML file (*FlexMonkeyEnv.xml*)

Instrument composite component

- override the getter of the *numAutomationChildren* property and the methods to expose children:
  - *getAutomationChildren()*
  - *getAutomationChildrenAt()*

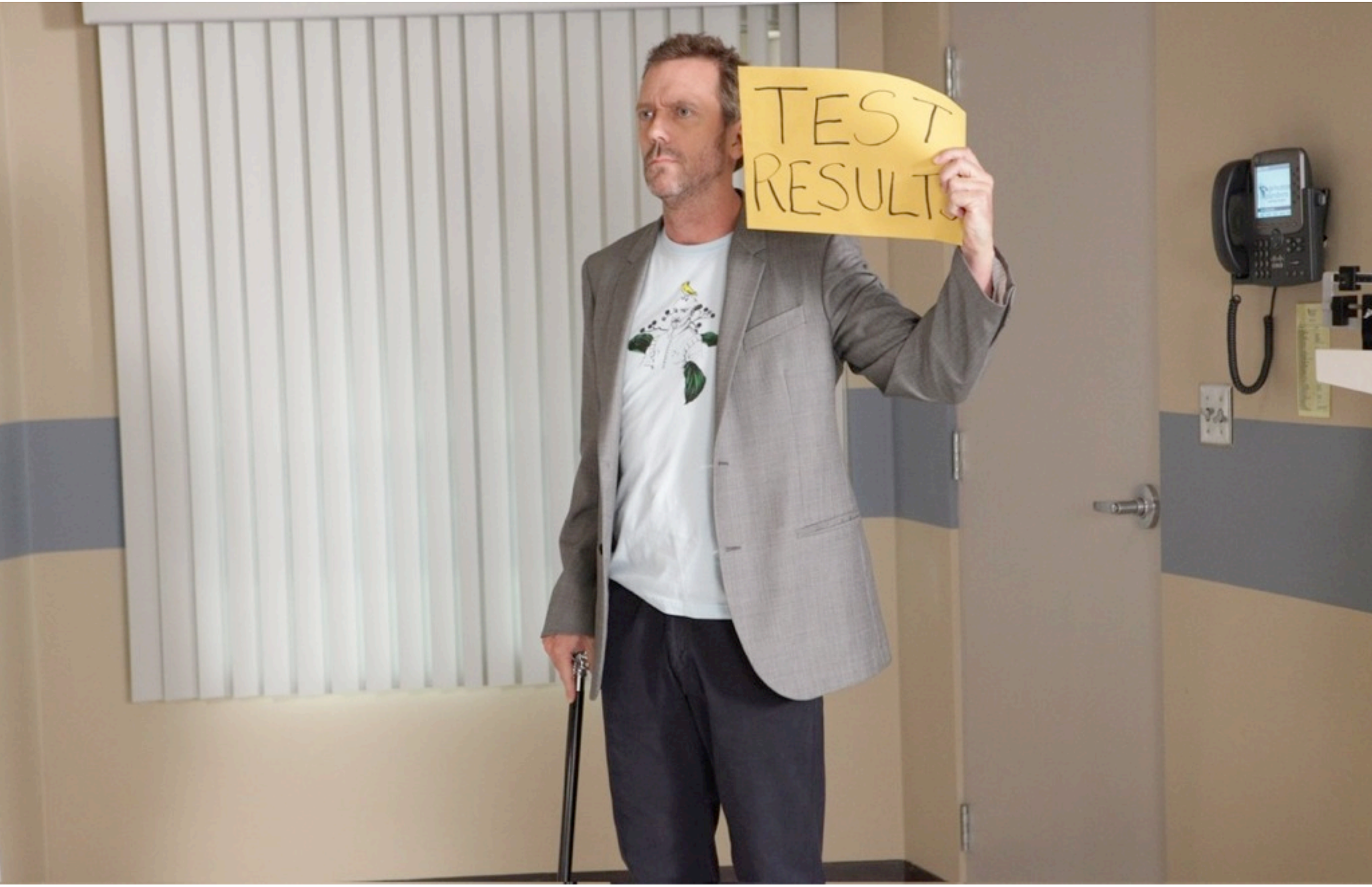
And one more thing...

PROBLEM

- Unless tests are run automatically, testing is ineffective

Solution: Use Jenkins

- Will automatically run tests for you
- Will tell you if they fail
- Provide you reports, result and metrics
- Can be downloaded from http://jenkins-ci.org/
- Very easy to setup and configure


TEST
RESULTS

Demo

Instead of an epilogue

«90% of coding is debugging. The other 10% is writing bugs»
Bram Cohen, author of the BitTorrent protocol

«If debugging is the process of removing bugs then programming must be the process of putting them in»
Edsger Dijkstra

Be brave!
Just test and no more bugs!

Resources

- GorillaLogic http://goo.gl/Ly6dd
- FlexMonkey resources page http://goo.gl/eKKaZ
- Ranorex Flex Testing http://goo.gl/sJI1L
- Automation Docs http://goo.gl/PwuZc
- Blog on Flex Automation http://goo.gl/CCQ80

p.s. Sample code

https://github.com/gAmUssA/flex_flexmonkey_flexunit_ci