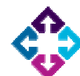
# Test Automation Framework Using Rational Functional Tester

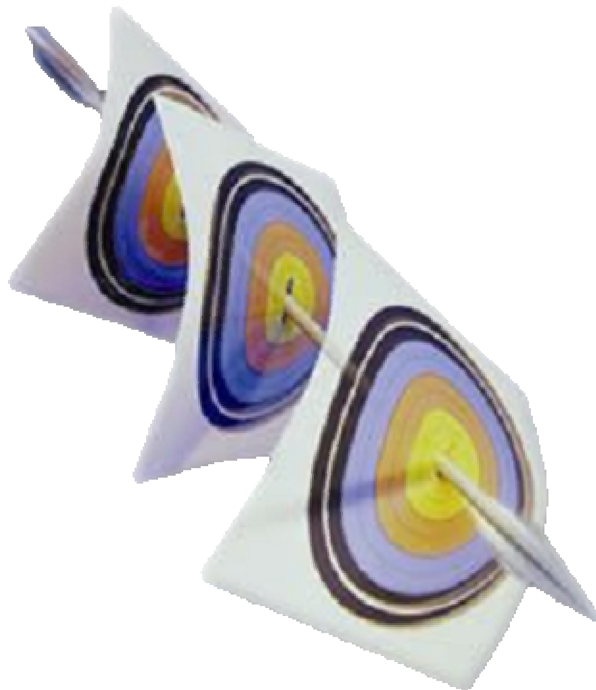
Shinoj Zacharias
(Shinoj.zacharias@in.ibm.com)

IBM Software
Innovate2011
The Premier Event for Software and Systems Innovation



Software. Everyware. **August 9-11,** Bangalore | **August 11,** Delhi

# Agenda

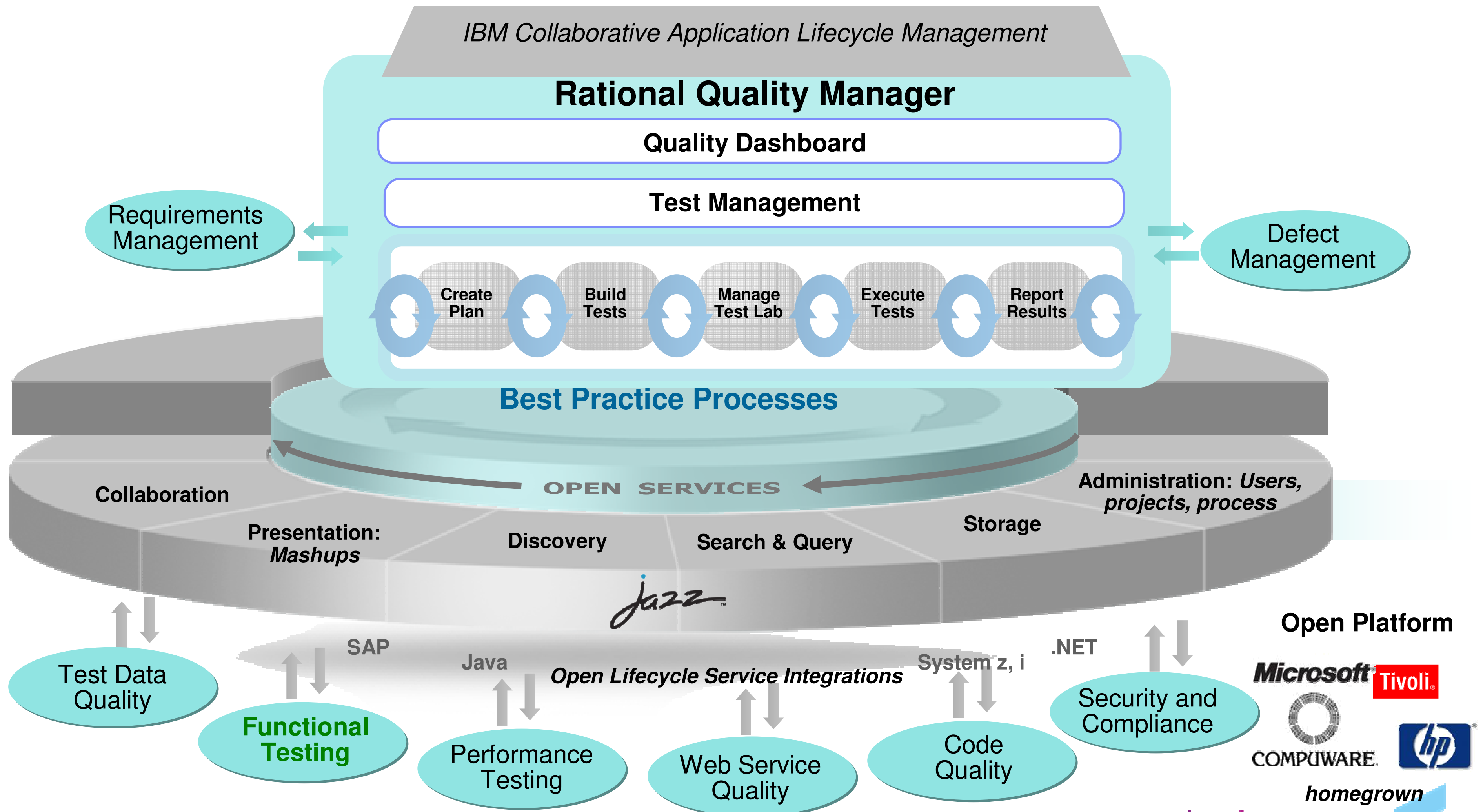


- Collaborative Application Life Cycle Management
- Introduction to Rational Functional Tester
- Why Automation Framework
- RFT and Automation Framework
- Types of Automation Framework
- Automation Framework Using Find
- Keyword driven framework
- Q & A



© 2011 IBM Corporation

# Collaborative Application Lifecycle Management






© 2011 IBM Corporation

# Maximize your investment in test automation
*With IBM Rational Functional Tester*

- Automated functional and regression testing tool
- Achieve success quickly and minimize maintenance
  - Simplified natural language scripting with Storyboard testing
  - Eclipse based or Visual Studio .net
  - Easy to learn, maximize reuse
- Powerful scripting language
- Complete test coverage
  - Supports testing for Java, Web, Visual Basic .Net, SAP, Siebel, Web 2.0, Power Builder and Terminal Based applications
  - Ability to support custom controls

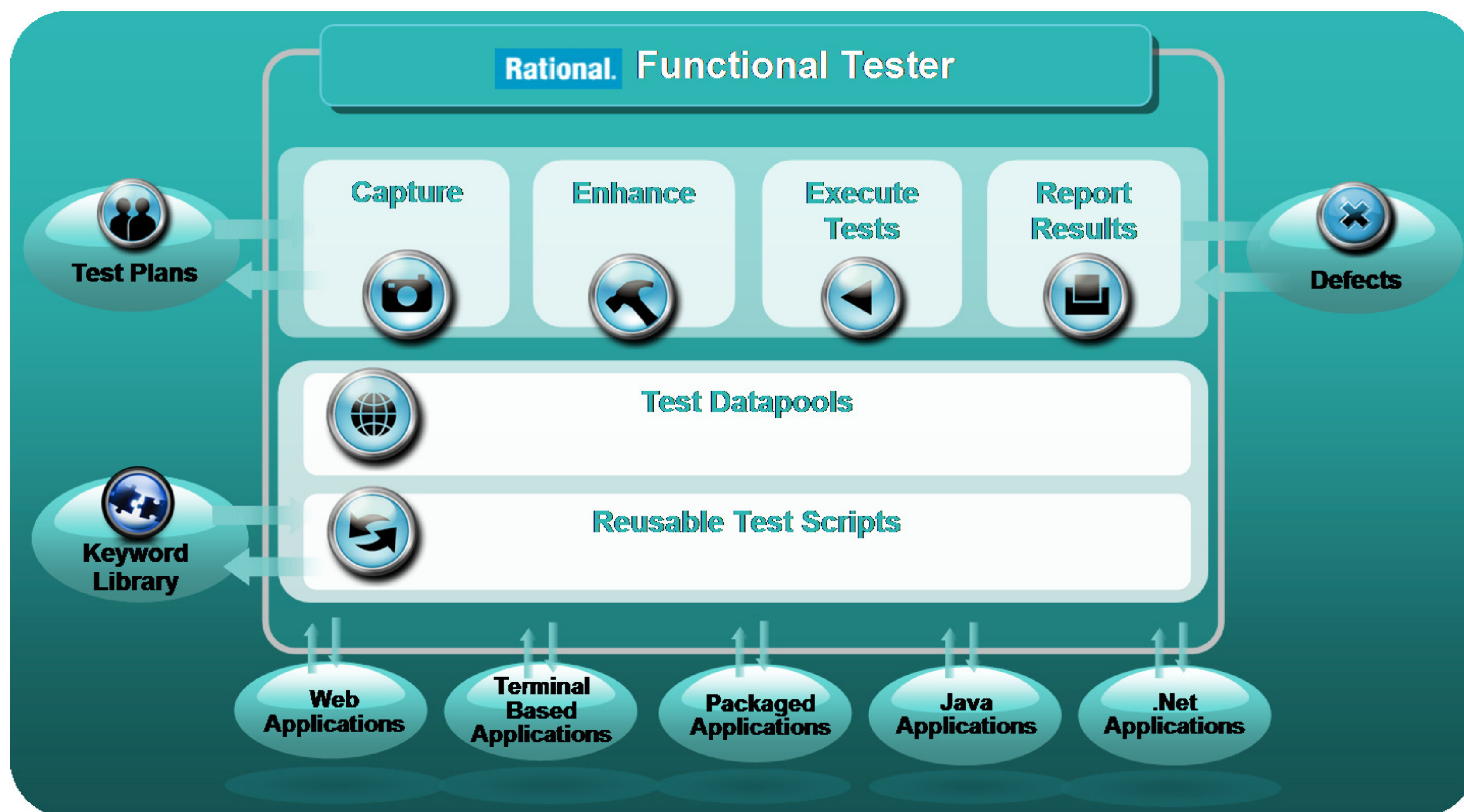
© 2011 IBM Corporation

# Introduction – Rational Functional Tester (RFT)

- **RFT Recorder**
  - Test scripts are recorded on the fly, as user navigates application
  - Verification points are inserted to validate system response
- **RFT Scripts**
  - Java code or VB.net is added to perform a variety of functions
  - Typical Modifications: Conditional branching, datapooling, refactoring
- **RFT Playback**
  - Scripts are executed. Discrepancies are logged


© 2011 IBM Corporation

# Rational Functional Tester





© 2011 IBM Corporation

# RFT Scripts - Java

Private Test Object Map for Script Script1

File Edit Find Test Object Preferences Applications Display Help

- Java: Frame: logFrame1: javax.swing.JFrame
  - Java: Button: ok-orderlogon: javax.swing.JButton
  - Java: RadioButton: radioButtonA: javax.swing.JRadioButton
- Java: Dialog: IncompleteOrder: javax.swing.JDialog
  - Java: Panel: JOptionPane: javax.swing.JOptionPane
    - Java: Button: OK: javax.swing.JButton
- Java: Frame: ClassicsCD: ClassicsJava
  - Java: Button: placeOrderButton2: javax.swing.JButton
  - Java: Tree: tree2: javax.swing.JTree
- Java: Frame: orderForm: javax.swing.JFrame
  - Java: Button: placeOrder: javax.swing.JButton
  - Java: Text: .cardNumberField: javax.swing.JTextField
  - Java: Text: .expireField: javax.swing.JTextField

Recognition | Administrative

| Property | Value | Weight |
|---|---|---|
| .class | javax.swing.JButton | 100 |
| .classIndex | 0 | 50 |
| accessibleContext.accessibleName | OK | 100 |

```
cardNumberIncludeTheSpaces().click(atPoint(23,5));
placeAnOrder().inputChars("123456");
expirationDate().click(atPoint(6,5));
placeAnOrder().inputChars("12/12/2010");
placeOrder2().click();
```

Functional Test — File Edit Source — Functi
- Project1
  - Script1
  - Script2
  - Script3
  - Script4
  - Script5
- Project1_log
- Project2
- Project2_log

Functional Test

Script1
- Script1Helper
  - RationalTestScript
- Test Datapool
- Verification Points
- Test Objects
  - Private Test Object Map
  - cardNumberIncludeTheSpace
  - classicsCD
  - expirationDate
  - newCustomer
  - ok
  - ok2
  - placeAnOrder
  - placeOrder
  - placeOrder2
  - tree2


© 2011 IBM Corporation

# Script Assure™

Version 1.0

Version 2.0

Account # / Log In ID

Password

Log In

Click here to save your start pa

Tester Sees

Customer Log On

User Name:

Password:

Log On

| Property | Value |
| --- | --- |
| .class | Html.I |
| .classIndex | 0 |
| .id | button |
| .name | userLo |
| .type | submit |
| .value | Log In |

| Value |
| --- |
| Html.INPUT |
| 0 |
| button1 |
| userLogin |
| submit |
| Log On |

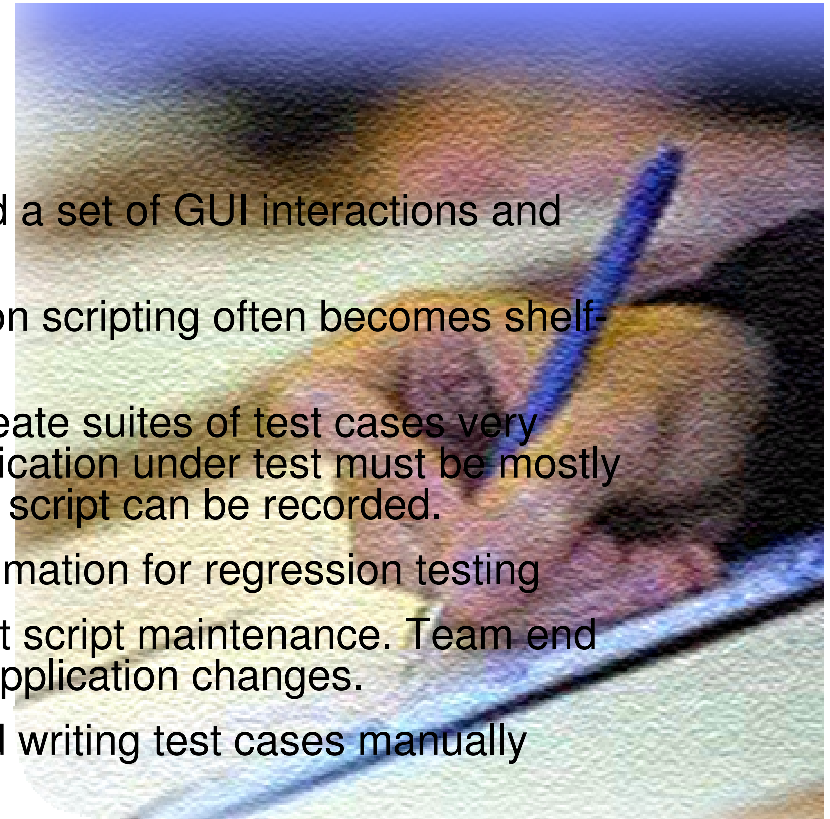
No User Intervention Required With ScriptAssure™

Determi



© 2011 IBM Corporation

# Why automation framework



- Most automation tools enable testers to simply record a set of GUI interactions and play them back against the application under test.
- Even with adequate training on the tool the automation scripting often becomes shelf-ware.
- Although record-playback features can be used to create suites of test cases very quickly, the limitation of this approach is that the application under test must be mostly complete and functioning correctly before a workable script can be recorded.
- Testers are often effectively limited to using GUI automation for regression testing
- The record-playback approach leads to extensive test script maintenance. Team end up recording and re-recording scripts each time the application changes.
- Test teams abandons the record-playback model and writing test cases manually instead.
- The object recognition algorithms are complex and inaccessible, making updates to the scripts extremely tedious and in some cases impossible.
- Though tools exposes the recognition algorithms, which makes updates much more manageable, but which also has the unfortunate side effect of making object recognition less reliable.
- Need reusable assets

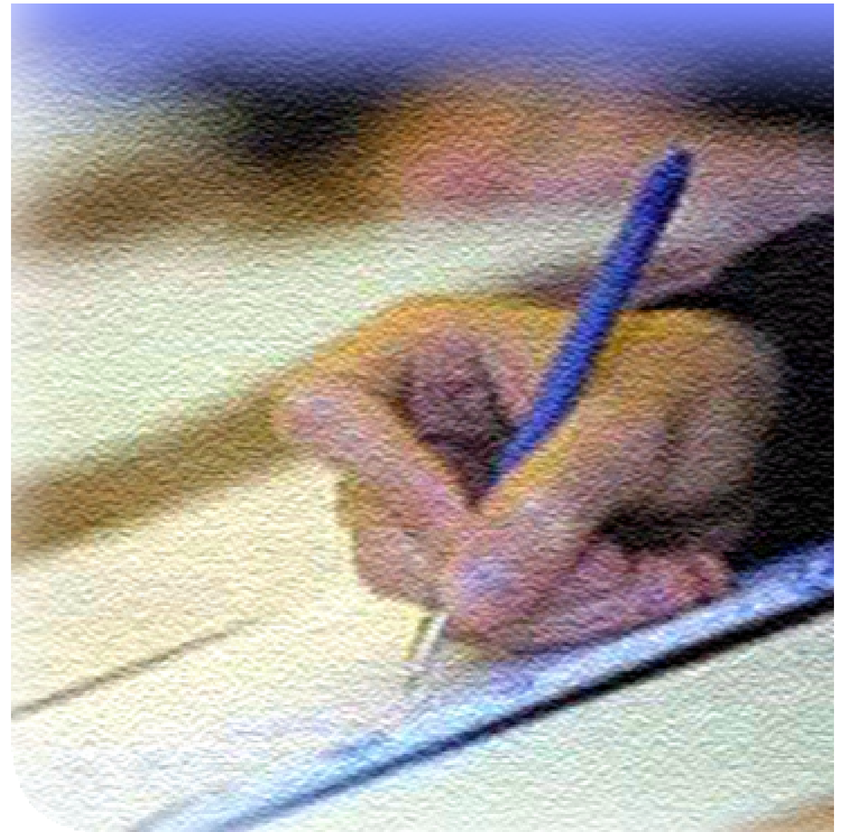

© 2011 IBM Corporation

# Rational Functional Tester and automation framework

- Powerful scripting capability. Can leverage on Java and VB programming language
- Object Oriented approach to test automation
- Object recognition algorithm and Object Maps
- Find() algorithm to dynamically find test objects.
- Rich set of RFT APIs for scripting.
- Extensibility mechanism using proxy SDK.


© 2011 IBM Corporation

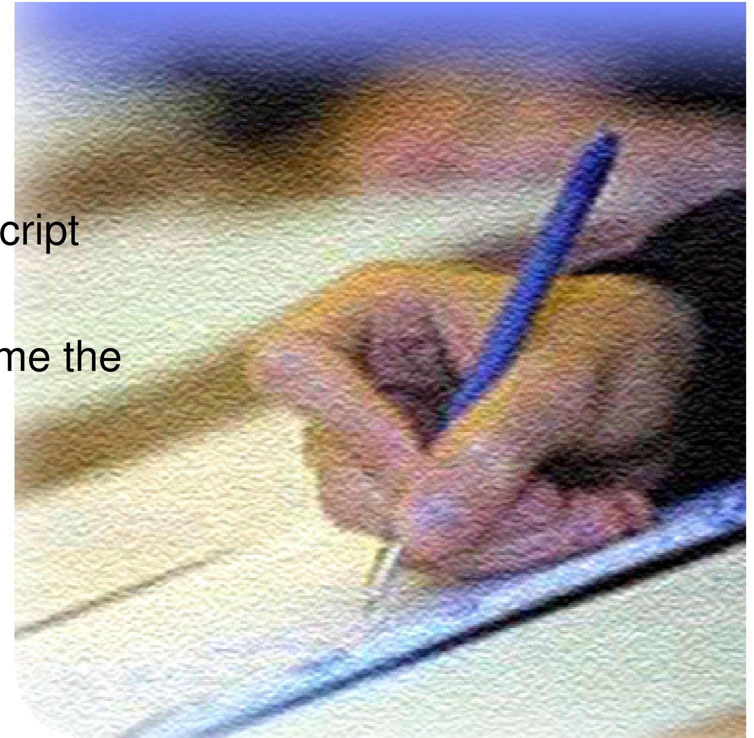
# Types of framework



- Framework using Find API
- Keyword driven framework.


© 2011 IBM Corporation

# Automation Framework Using Find



- The record-playback approach leads to extensive test script maintenance.
- Team end up recording and re-recording scripts each time the application changes
- Hierarchical changes affect script playback


© 2011 IBM Corporation

# Automation Framework Using Find

## Find() API

- Easy to use API for finding a control in Application Under Test
- Need not use Object Map
- Can use Hybrid approach : Object Map Vs non-Object Map
- Ensures resilience of the script
- Exposed on any test object (e.g.: GuiTestObject, BrowserTestObject etc)
- Exposed on RootTestObject


© 2011 IBM Corporation

# Automation Framework Using Find

## Find() Available in Script

- Find(Subitem Properties)
- Find(Subitem Properties, Boolean mappableOnly)

```
public void testMain(Object[] args)
{

    startBrowser("http://www.google.com");
    find()
}
```

find(Subitem properties) : TestObject[] - RationalTestScript
find(Subitem properties, boolean mappableOnly) : TestObject[] - RationalTestScript


© 2011 IBM Corporation

# Automation Framework Using Find

## Find(Subitem Properties)

**Subitem can be either atChild() or atDescendant() or atList()**

**atChild()**

One or more properties that must be matched against the direct child of the starting TestObject

**atDesendant()**

One or more properties that can be matched against any child of the starting TestObject

**atList()**

A sequential list of properties to match against. atList valid subitems are atChild, atDescendant, and atProperty. The first list item will be matched against to get a list of candidates, and out of those candidates their descendants will be matched against for the next list item, and so on.



© 2011 IBM Corporation

# Automation Framework Using Find
## RootTestObject

- Represents a global view of the system being tested
- Provides access to system-wide functionality, such as
  - Finding an arbitrary testobject based on properties, location etc
  - DomainTest Object

```
RootTestObject root = RootTestObjec
root.find(Subitem properties)
```

```
ProcessTestObject p1 = StartApp("Notepad") ;
Integer pid = new Integer((int)p1.getProcessId()) ;
foundTOs = root.find(atList(atProperty(".processId", pid,atDescendant(".class", ".text"))) ;
```


© 2011 IBM Corporation

# Automation Framework Using Find

## RootTestObject Special properties for find

There are special properties that apply to a RootTestObject find. These include:

*.processName*

- dynamically enable the process for testing
- constrain the find to only look in processes with that name.

*.processId*

- dynamically enable the processes testing.
- constrain the find to only look in  processes with that process id (pid).

*.domain*

- only search in toplevel domains matching the .domain property

*.hWnd*

- If the .domain "Win" is also specified the matching window will be enabled for testing.

*Handle* - if the .domain "Net" is also specified the matching window will be enabled for testing.



© 2011 IBM Corporation

# Automation Framework Using Find

## Find(atChild)

One or more properties that must be matched against the direct child of the starting TestObject

find(atChild(Property[] properties))
find(atChild(String propName, Object propValue))
find(atChild(String propName1, Object propValue1, String propName2, Object propValue2 ))

```
Property prop1 = new Property(".class", "JButton");
Property prop2 = new Property(".name", "ok");
Property prop3 = new Property("text", "OK");
Property[] props = {prop1, prop1, prop3};
TestObject[] tobs = find(atChild(props));

TestObject[] tobs = find(atChild(“.class” , “JButton”))

TestObject[] tobs = find(atChild(“.class”, “JButton”, “text”, “OK”)
```


© 2011 IBM Corporation

# Automation Framework Using Find

## Find atDesendant

One or more properties that can be matched against any child of the starting TestObject

find(atDesendant(Property[] properties))
find(atDesendant(String propName, Object propValue))
find(atDesendant(String propName1, Object propValue1, String propName2, Object propValue2 ))

```
Property prop1 = new Property(".class", "JButton");
Property prop2 = new Property(".name", "ok");
Property prop3 = new Property("text", "OK");
Property[] props = {prop1, prop1, prop3};
TestObject[] tobs = find(atDesendant(props));

TestObject[] tobs = find(atDesendant(“.class” , “JButton”))

TestObject[] tobs = find(atDesendant(“.class”, “JButton”, “text”, “OK”)
```


© 2011 IBM Corporation

# Automation Framework Using Find

## Find(atList(Subitem))

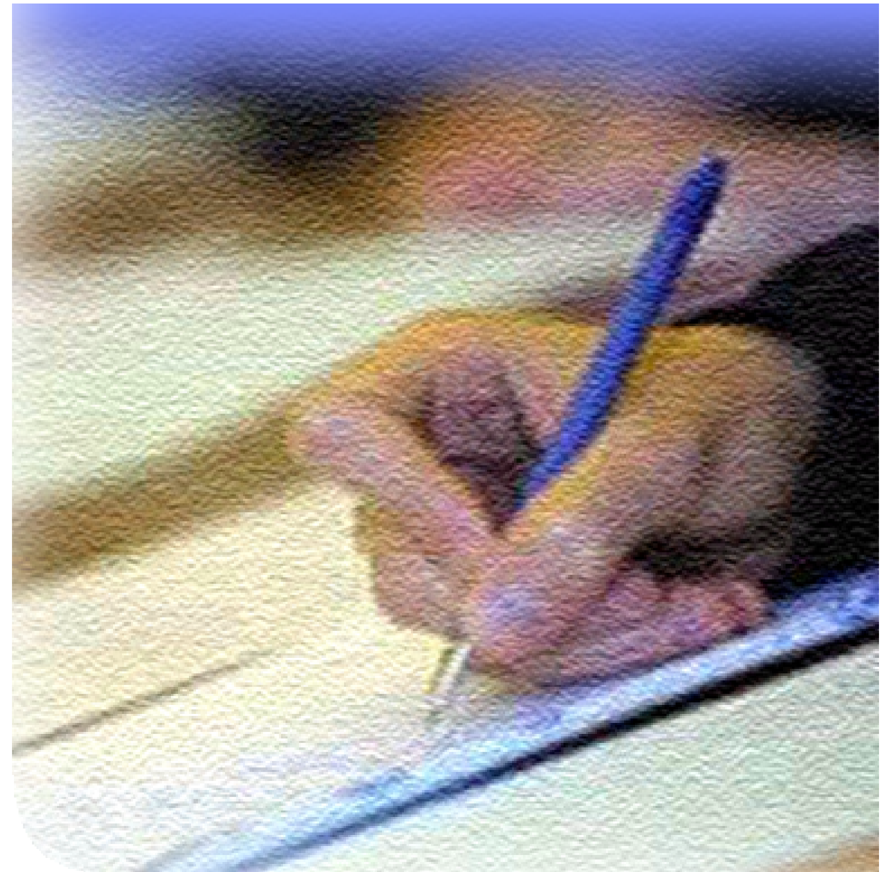
A sequential list of properties to match against. Valid subitems for atList are atChild, atDescendant, and atProperty. The first list item will be matched against to get a list of candidates, and out of those candidates their descendants will be matched against for the next list item, and so on.

```
find(atList(atChild(), atDescendant(), atProperty());
```

```
TestObject[] tobs = find(atList(atDescendant(".class", dialogRE),
                                atChild(".class", buttonRE, ".value", "OK"))) ;
tobs[0].click();
```


© 2011 IBM Corporation

# Automation Framework Using Find
## Creating library using find



- ClickButton()
- SelectComboBoxItem()
- SelectTreeItem()
- SetText()


© 2011 IBM Corporation

# Automation Framework Using Find example

```
ClickButton(String buttonName)
{
        TestObject to = find(atChild(“.class”, “JButton”, “name”, buttonName”)
}

selectComboBox(String ComboBoxName, String comboItem)
{
        TestObject to = find(atChild(“.class”, “JCombobox”, “name”, ComboBoxName)
        to.select(comboItem);
}
```


© 2011 IBM Corporation

# Keyword Driven Framework

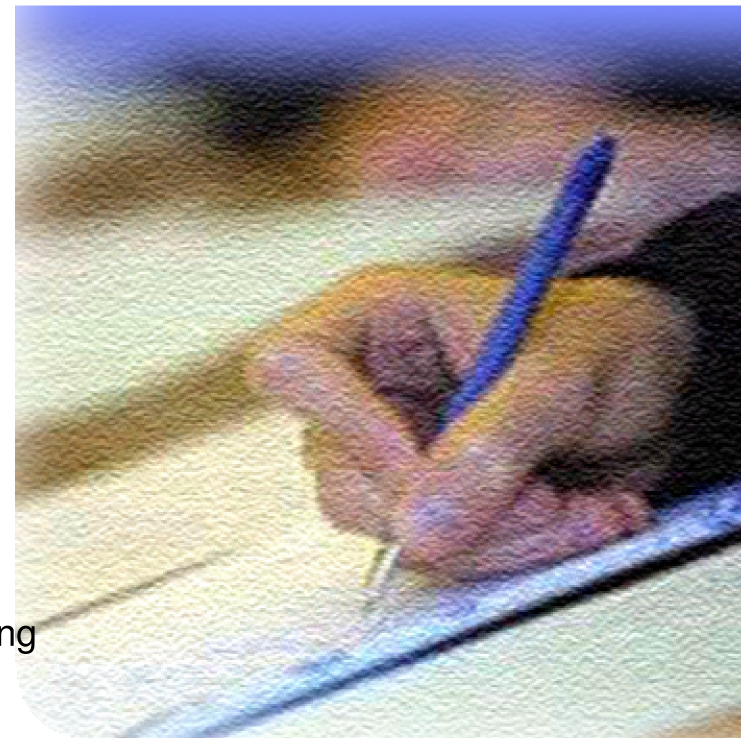
Keywords are reusable assets that can be created

Enables creating tests at more high-level/abstract manner using Keyword approach

Enables non technical SMEs and Business user to design test without knowing the complexities of the tool


© 2011 IBM Corporation

# Keyword Driven Framework
## Roles in Keyword Framework



**Test Architect/Designer :**

- Role played mostly by a non technical SME or Business user.
- Domain Expert, such as an SAP Consultant.
- Creates Keyword driven test cases with parameters

**Automation Expert:**

- Role played by the technical person.
- Understand the automation tools and has the knowledge of scripting languages.
- Implements the framework (set of libraries)
- Responsibility to enhance the framework for application changes.

**Execution Engineer**

- Role played by a non technical tester.
- Has the knowledge of how to use the framework to run the regression tests suite.



© 2011 IBM Corporation

# Keyword Driven Framework
## Sample keywords

- LogonToSAPGUI
- CreateSalesOrder
- ShipTheProduct
- LogOut()


© 2011 IBM Corporation

# LogonToSAPGUI()

```
StartApplication("LogON")
EnterText("uernametxt", "John")
EnterText("passwordtxt", "xxxxx")
clickButton("logon");
```

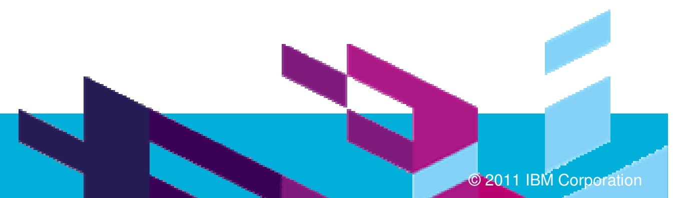
```
Entertext(String controlName, String text)
{
        TestObject to = find(atChild(".class",
"SAPText",
                ".name", controlName)
        to.setText(text);
}

clickButton(String buttonName)
{
        TestObject to = find(atDesendant(".class",
                "SAPButton", "name", buttonName"
        to.click()
}
```



© 2011 IBM Corporation

www.ibm/software/rational




© 2011 IBM Corporation

**www.ibm/software/rational**

**© Copyright IBM Corporation 2011. All rights reserved.** The information contained in these materials is provided for informational purposes only, and is provided AS IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, these materials. Nothing contained in these materials is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software. References in these materials to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. Product release dates and/or capabilities referenced in these materials may change at any time at IBM's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way. IBM, the IBM logo, Rational, the Rational logo, Telelogic, the Telelogic logo, and other IBM products and services are trademarks of the International Business Machines Corporation, in the United States, other countries or both. Other company, product, or service names may be trademarks or service marks of others.

- Back up


© 2011 IBM Corporation

# Layered approach to organize the tests

- The framework consists of three tiered architecture implemented through appobjects, tasks and test cases.
  - **AppObjects**: where you will store information about your application's GUI elements. It is also where you will write your Getter Methods, which return objects enabling the Caller to query and manipulate these GUI elements. Typically, these methods are called within the *Task* layer.
  - **Tasks**: where you will write reusable methods that exercise common functions in your application. It is also where you will write methods to manipulate and query complex, application-specific controls. Methods in the Task are called by *Test Cases.*
  - **Test Cases**: methods that navigate through an application, verify its state, and log results.


© 2011 IBM Corporation

# AppObjects.

- Separate the object maps from the test cases. Object maps are located in scripts whose sole purpose is to return the objects to the caller. In this way, the object maps can be private within the script that holds them, and it is possible to store the information about each object in the application in one and only one map
- The purpose of this folder is to hold scripts that return the GUI objects contained in the application. Each script in this folder includes a private object map and several methods that simply returns the object that will be used by other scripts.


© 2011 IBM Corporation

# Tasks

- Tasks methods invoke appobjects methods in order to gain access to GUI elements in the application. In turn, tasks methods are invoked by test cases. The strengths of the tasks folder are that it promotes code reuse and shields the test cases from low-level implementation details. A robust and well-designed tasks layer, along with well-constructed object maps in the appobjects layer, is critical to the success of the entire automation effort.


© 2011 IBM Corporation

# TestCases

- The testcases provides the most general view of the test effort. Generally speaking, test cases invoke tasks (passing data if necessary), verify conditions, and log results. Test cases should contain only the simplest logic and flow of control; all else is reserved for the tasks folder. If sufficient time and thought has been devoted to the construction of the tasks folder, it should be quick and easy to generate test cases, even for a relatively inexperienced testers.
- In the testcases , each class should represent a feature area or other logical, intuitive grouping, and each method should represent a test case. This design makes it very easy to write data-driven testing, thus increasing test coverage with minimal effort.


© 2011 IBM Corporation