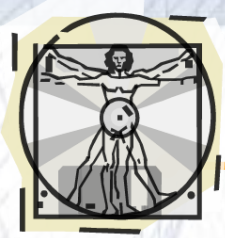
SENG 637
Dependability and Reliability of Software Systems

Chapter 7: GUI & Web Testing Automation

Department of Electrical & Software Engineering, University of Calgary

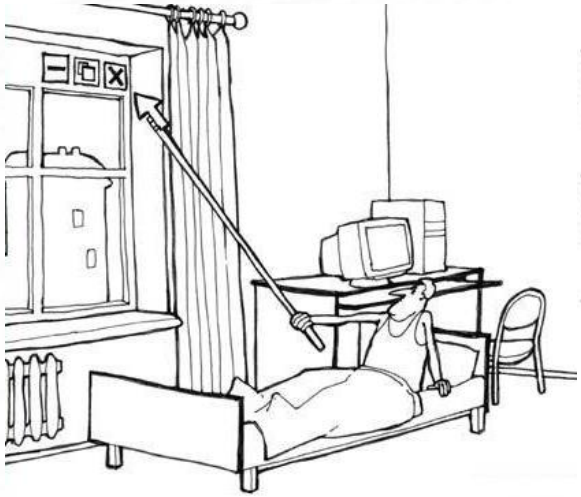
B.H. Far (far@ucalgary.ca)

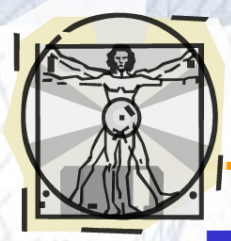
http://people.ucalgary.ca/~far



Contents

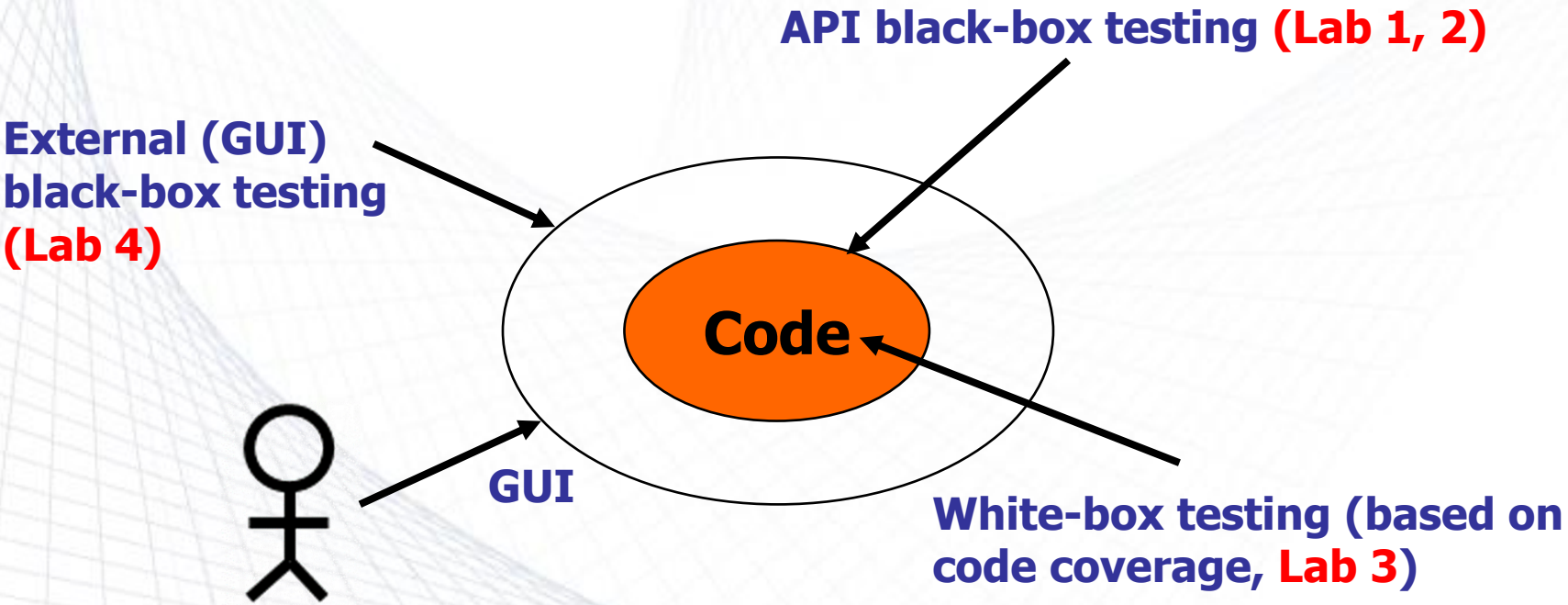
- Manual testing vs. test automation
- Graphical user interface (GUI) testing
  - Manual
  - Capture/Play
- Selenium WebDriver
- GUI testing tools
- Common myths, misses and hints about test automation

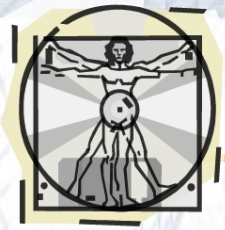




GUI Testing

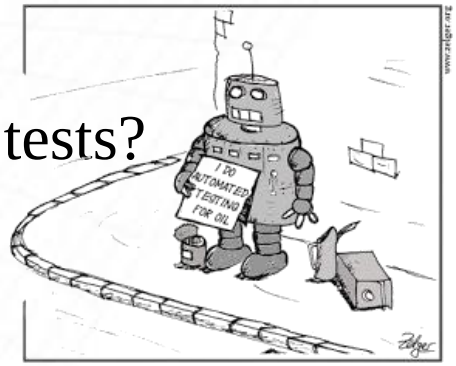
- GUI testing is the process of testing a product that uses a graphical user interface (GUI)
- In other words:



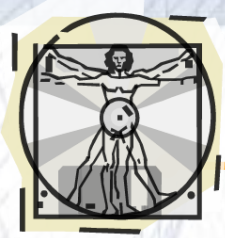


Problems with Manual Testing

- Not very repeatable
  - Either cannot remember exactly how it was last tested (what input value did I use a month ago to fail the software?)
  - Or, too expensive to document detailed procedures
    - Perhaps you have seen "thick" manual test scripts still some companies use
- Very effort intensive
  - How long will it take to re-run all the tests?
  - How often will you actually do it?

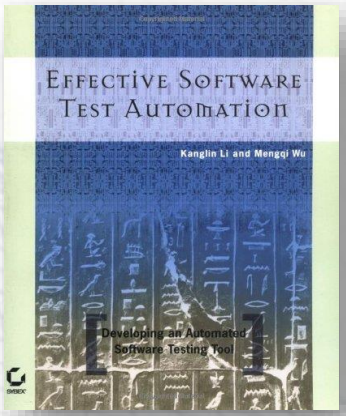


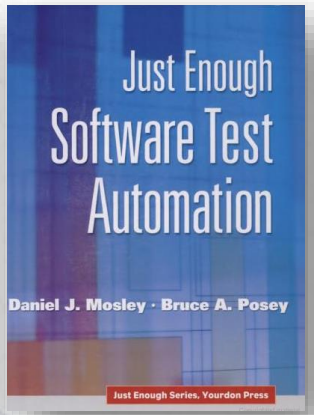
It was a shock to his master when he realized, robots need maintenance, too

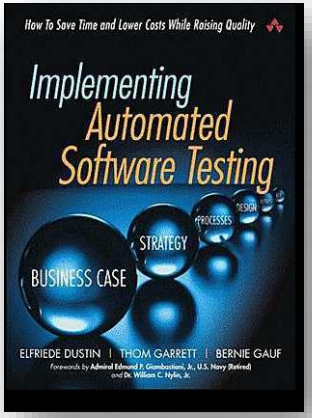


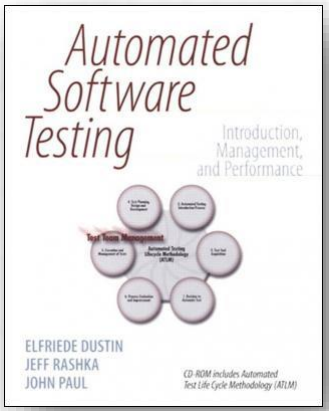
Automated Testing

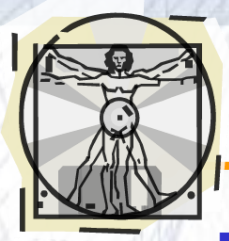
- Using a testing software (tool) to test another software (SUT)
- Developing test code (script) to test another piece of code (application code, SUT)











Why Automate Tests?

- It gives us **rapid** feedback (pass/fail)!
- It is **repeatable** and **reusable**
- Much faster than human testers, but needs **up-front investment**
- Automated tests and TDD can potentially drive design
- It is better documented

UNIVERSITY OF CALGARY

Automated vs. Manual Tests

	Pros	Cons
Manual	• Everyone can do it • Human oracles- more flexible • Less upfront investment • Less short term cost	• Boring • Might miss some tests • Time consuming • Not repeatable • Less visibility • Manual pass/fail documentation
Automated	• Faster • Works as documentation • Safety net for refactoring • Reusable • Fun and challenging to write • Can calculate coverage easily • Visibility of tests for all stakeholders	• Investment on tool • Investment on training • Always runs the same tests • Less flexible oracles • Need experts to write test scripts • Upfront time investment • May not capture actual user experience

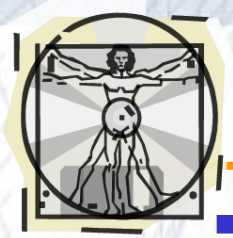
Also watch: https://www.youtube.com/watch?v=gOxcn8oAKcs

SCHULICH School of Engineering

UNIVERSITY OF CALGARY

Section 1

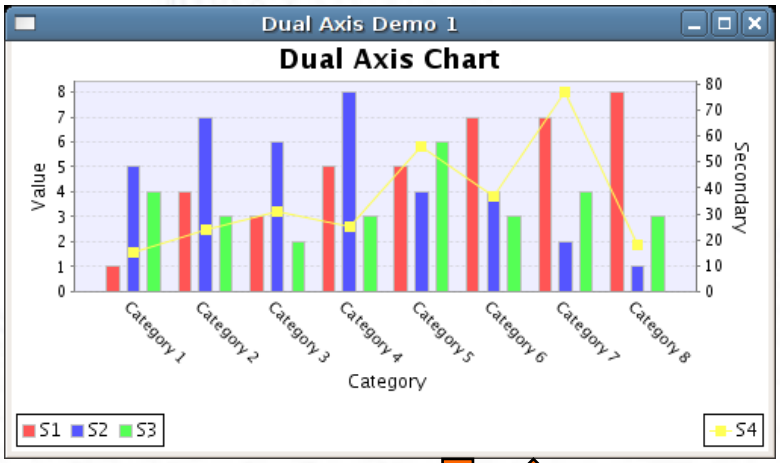
GUI Testing

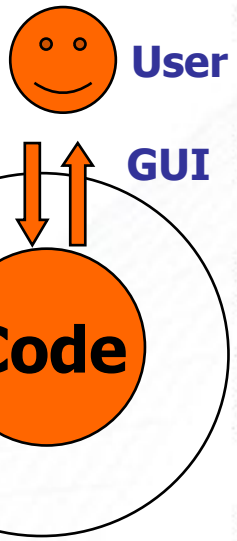


In the Previous Labs…

- Even if we conduct API-based black box and coverage-based white box testing, the SUT can have defects in the GUI logic (code) and the "glue" code

- E.g., how do we know that the charts are generated correctly graphically?
- Even if the values are correct, the visuals can be incorrect

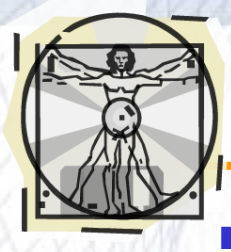




Interactions ("glue" code)

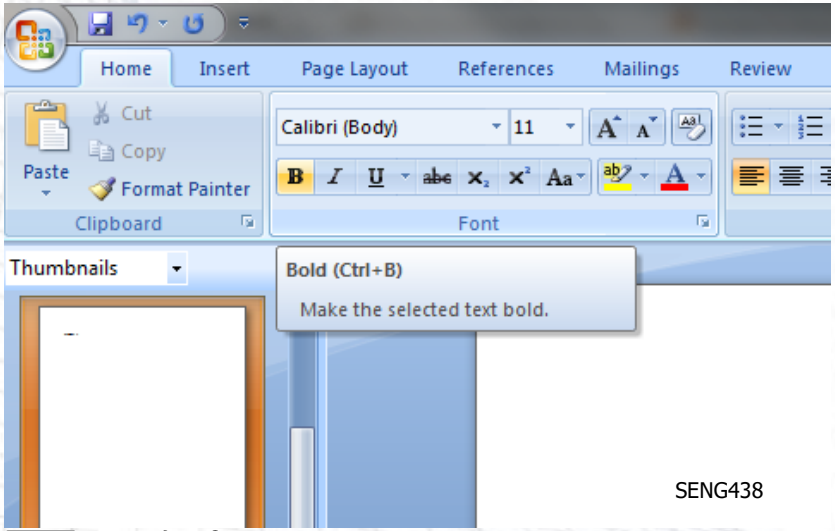
```
import java.util.LinkedList;

public class MyIntStack {
    private final LinkedList fStack;

    public MyIntStack() {
        fStack = new LinkedList();
    }

    public int pop() {
        return ((Integer) fStack.removeFirs
```

code

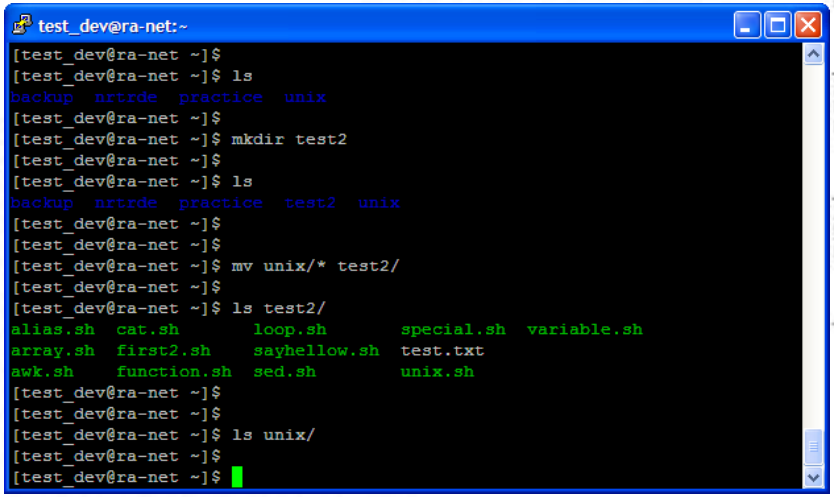


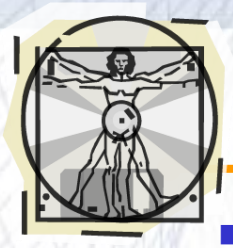
Graphical User Interface (GUI)

- GUI is a type of user interface item that allows people to interact with programs in more ways than typing (in command line interfaces)
- GUI software is an **event-driven** software
- User events (e.g. clicks) trigger event handlers



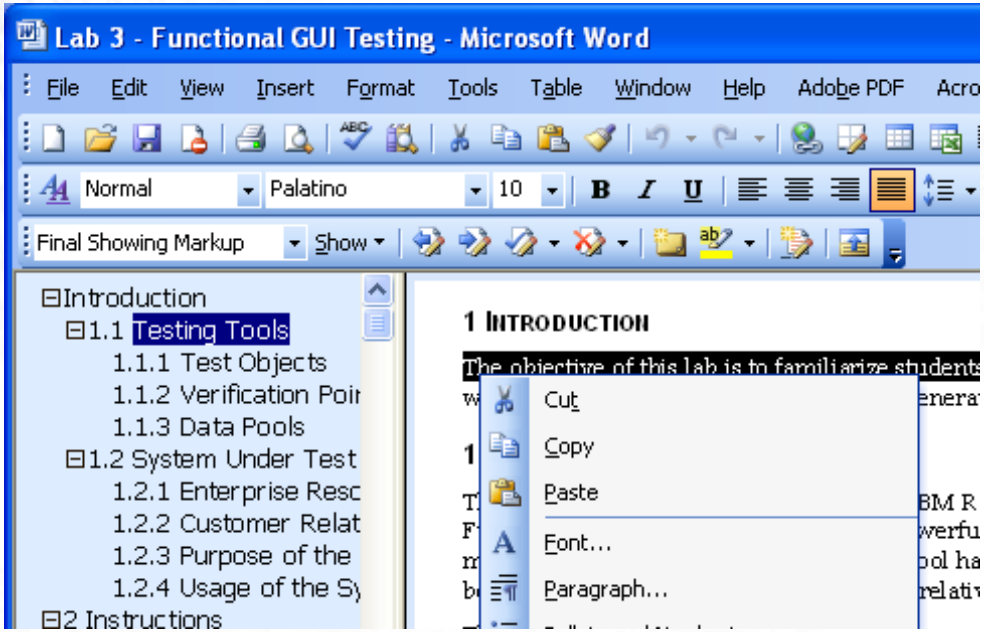
vs.



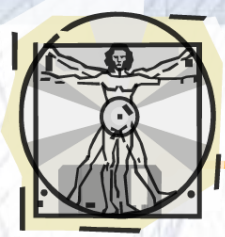


Challenges of GUI Testing

- Event-driven nature of GUIs leads to many challenges in testing them
- The user has extremely wide choice of actions compared to BB/WB testing

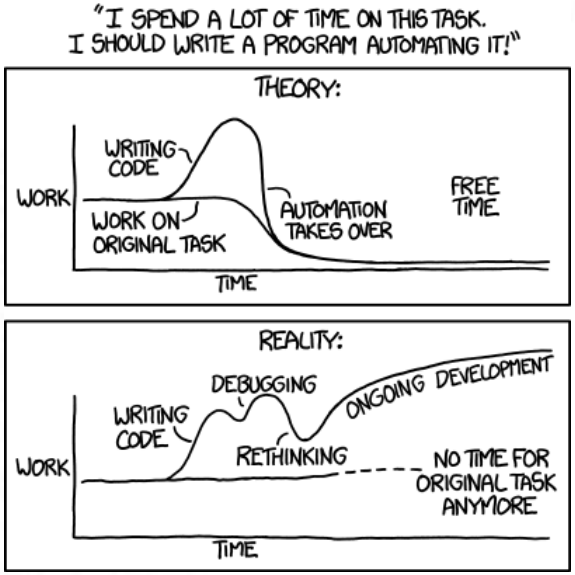


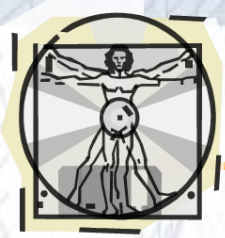
- Because users many click on any pixel on the screen, there are many, many more possible user inputs that can occur
- At any point in the application, the users may click on any field or object within a window



GUI Testing - Approaches

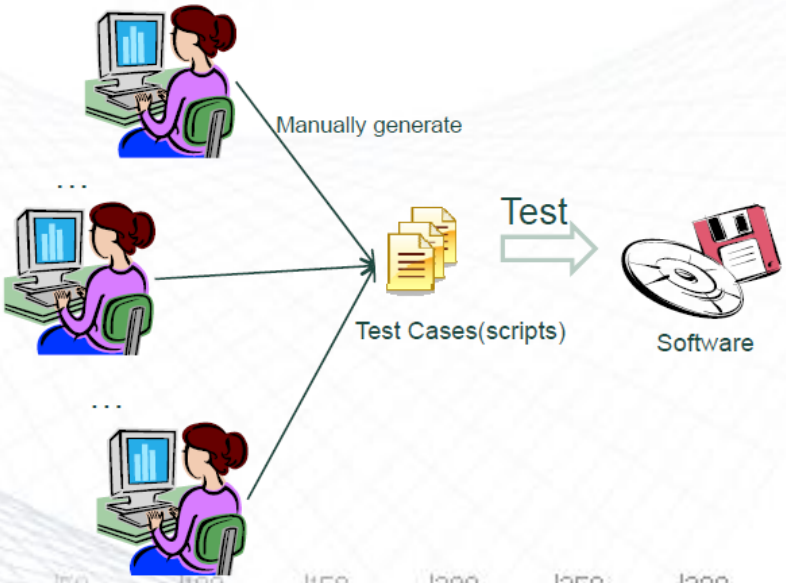
- Completely manual ← Focus of Lab 1
- Semi-automated generation ← Focus here (& Lab 4)
  - Capture/replay of user actions
- Automated generation
  - Model-based test generation (based on execution of user sessions using a GUI model, e.g. state transition model)

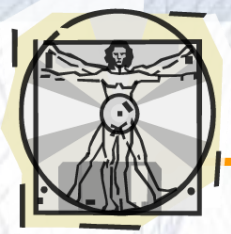




Conventional (Manual) Approach

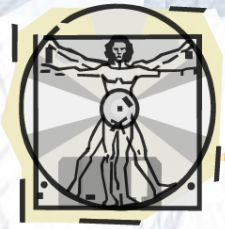
- Done by manually stepping through pages of test procedures, or by exploratory testing
- Problems?
  - Labor intensive, highly error prone
  - Needed to be repeated each time regression testing is required





Capture/Playback Approach

- A **capture and playback** testing tool **captures** user sessions (user inputs and events) and store them in scripts (one per session) suitable to be used to **replay** the user session
- Testing infrastructure (i.e. tool) is needed to intercept GUI events, GUI states, thus storing user sessions and also to be able to replay them
  - They can work at application or VM level



Capture/Playback Approach

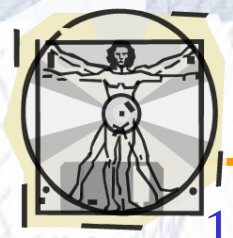
- What do we need?
  - A tool that can **capture** (and create script/code) of the steps performed for each manual test
  - **SUT** (or its emulated version)
  - A tool that can **replay** (run and verify) the script/ code and report the results

e.g. SikuliX

- How to do?
  - Mainly depends on the tool support

QTP

Selenium



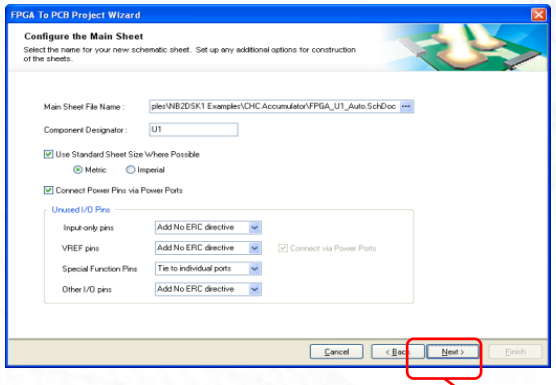
Capture/Playback Process

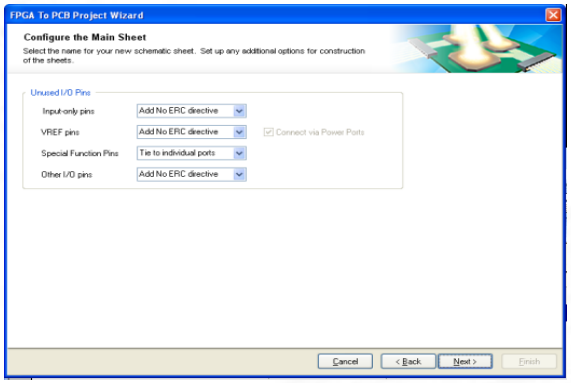
1. The tester interacts with the system GUI to run the system, thus generating sessions of sequence of mouse clicks, UI and keyboard events, voice commands, etc.
2. The tool captures and stores the user events and the GUI screenshots
   - A script is produced per each user session
3. The tester can replay the execution by running the script
   - Ideally, the script can be modified by the tester
   - The script can be enriched with expected output, checkpoints
   - The script can be replicated to generate many variants (e.g., changing the input values)
4. In case of GUI changes, the script must be updated

Capture/Playback Tools

- Source-code-based scripting of the GUI workflow

SUT





Manual steps:
1. Push "Next" button
2. Wait
3. "Next page" will show up

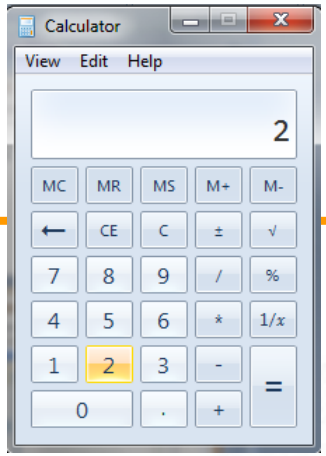
```
// in the 1st  page
first_page.directory.click();
keyboard.enter("c:\temp\");

first_page.option.click();
first_page.next_button.click();

wait(1); // 1 second

// verify that the 2nd page actually appears
assertExists(second_page.getObject());
```

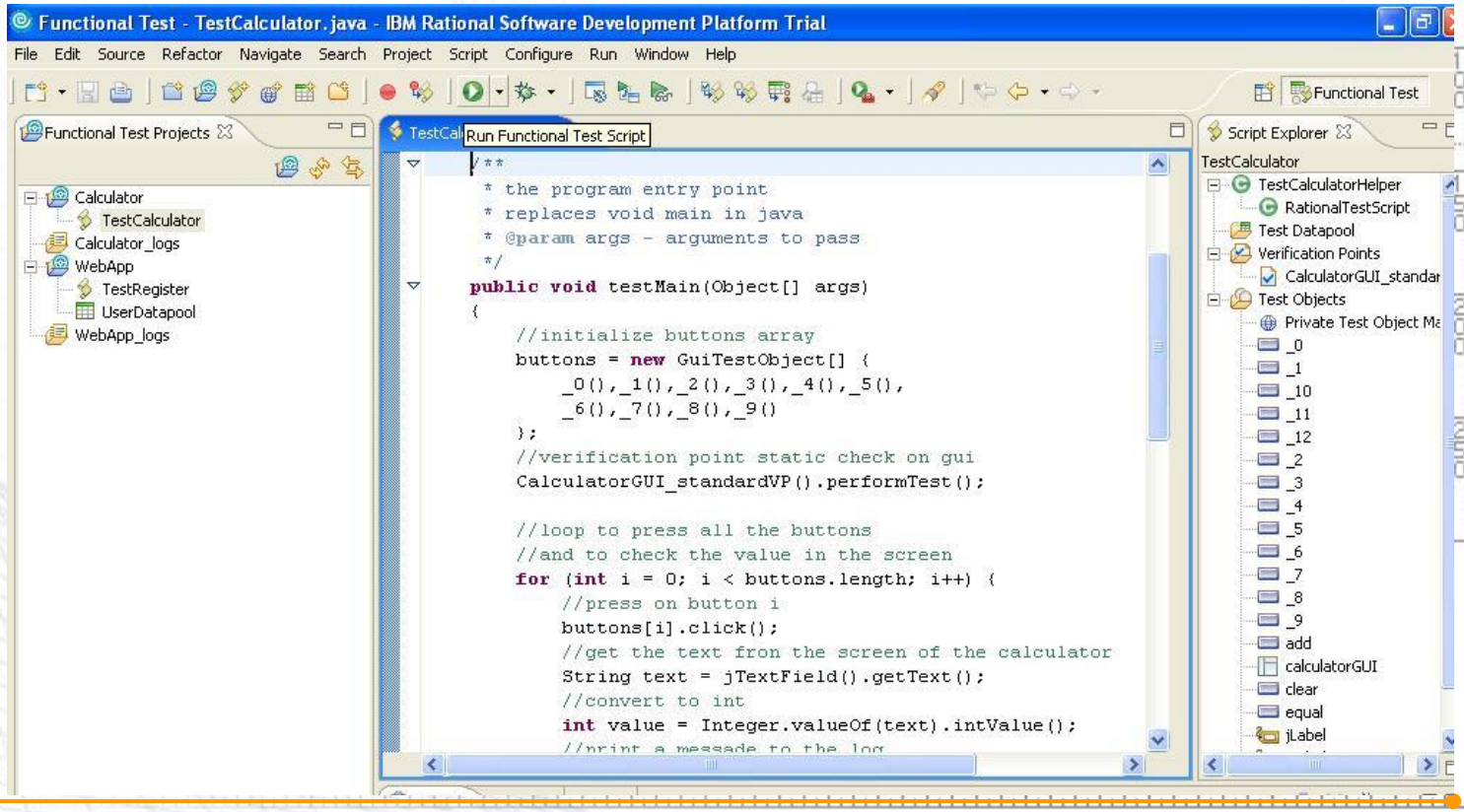
Recorded script

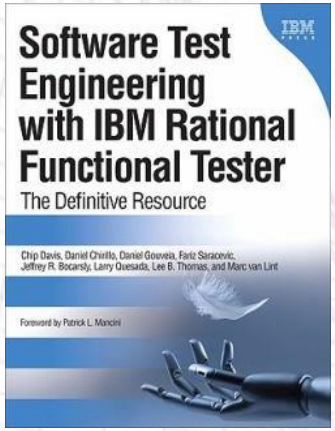
https://www.youtube.com/watch?v=wcIAh1DP3JQ&index=4&list=PLF20D996D82993E79

Capture/Playback Tools

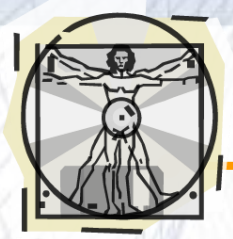
- **Example:** IBM Rational Functional Tester
- In advanced tools, we can also modify the test code after capture, and place complex code structure in the test code





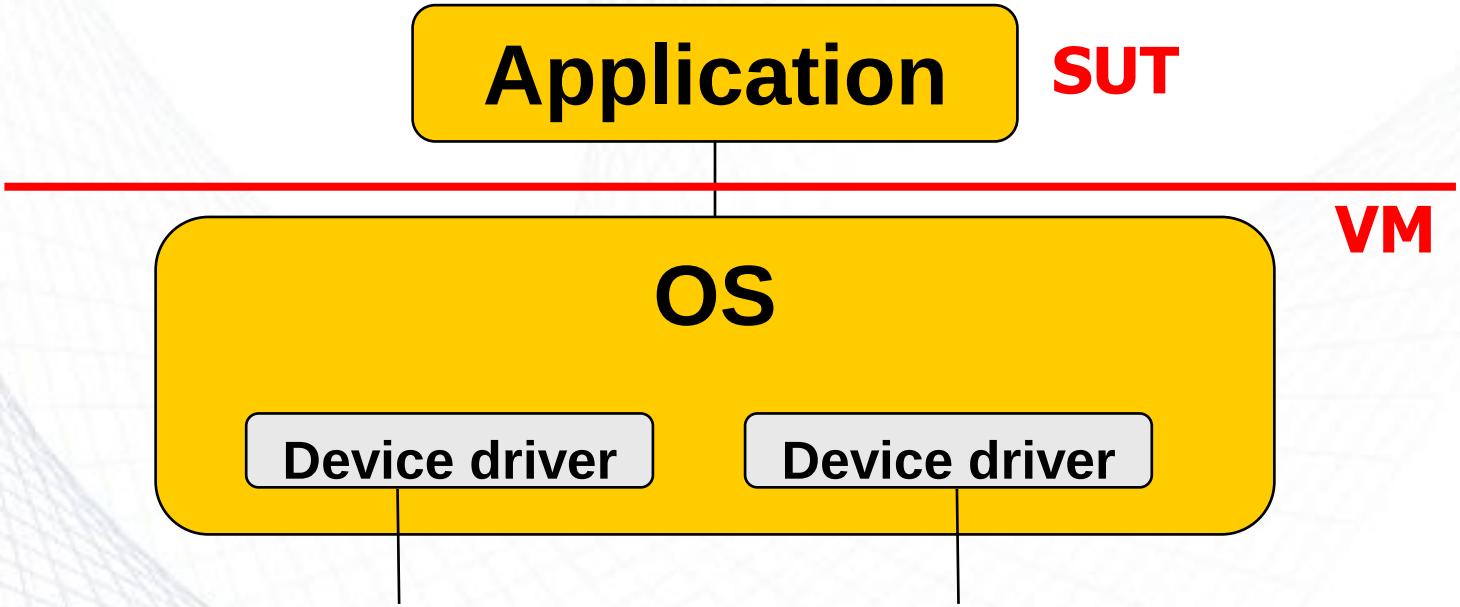


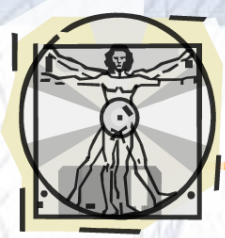
UNIVERSITY OF CALGARY



Capture/Playback – How They Work

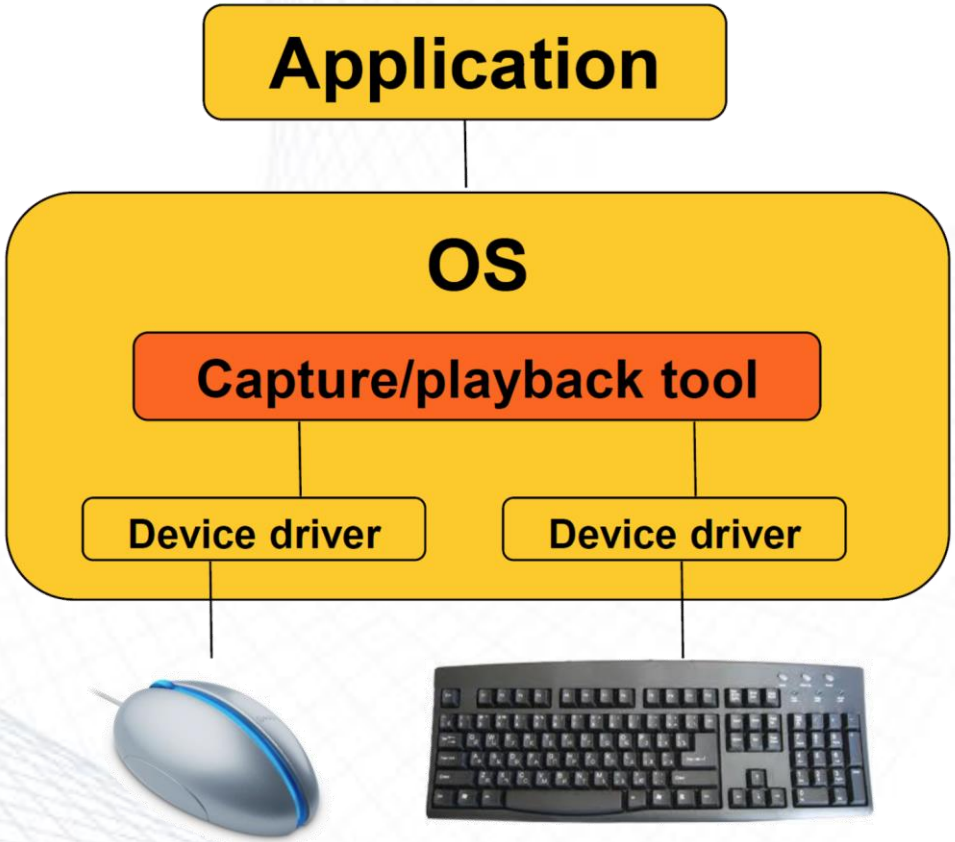
- First, let's see how an application works with input devices





Capture/Playback – How They Work

- Capture/playback tools intercept into the OS and device drivers



Challenges

- But! A major drawback in Capture/Playback tools is that when the GUI changes, input sequences previously recorded may no longer be valid

  ← Problem when regression testing

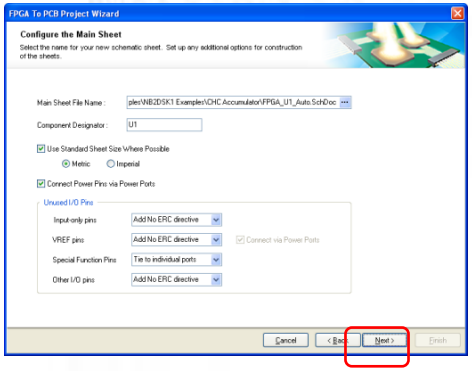
- Often, co-maintenance of GUI test code with the SUT code is needed
- The ability to generate many variations of a recorded script without having to manually edit the script itself

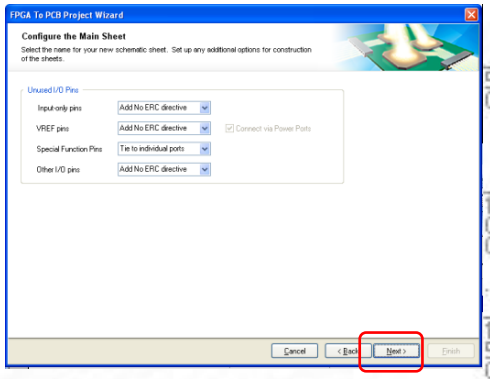
Example

GUI testing when program code is modified

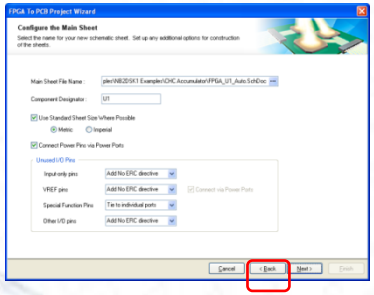
```
// in the 1st  page
first_page.directory.click();
keyboard.enter("c:\temp\");

first_page.option.click();
mouse.left_click();
first_page.next_button.click();

wait(1); // 1 second

// verify that the 2nd page actually appears
assertExists(second_page.getObject());

// in the 2nd page

second_page.installation_mode.open();
second_page.installation_mode.choose(2);

second_page.finish_button.click();

// the rest...
```

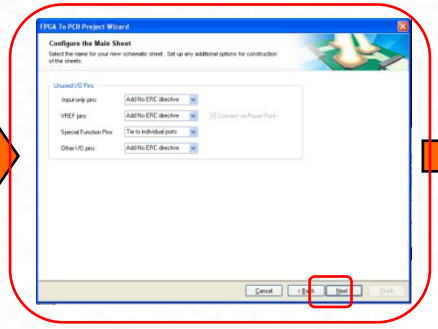
tests:

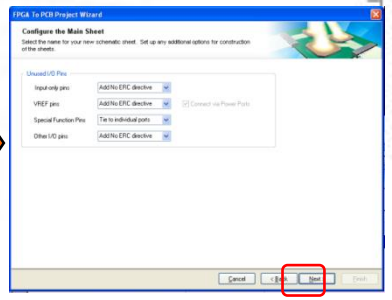




New page in the middle



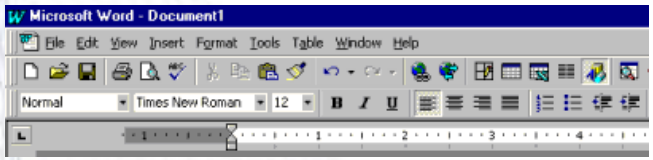




Test code will "break"
SHULD BE updated

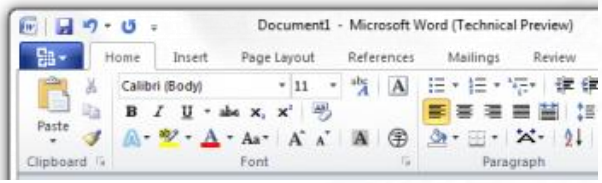
Test-Code Co-Maintenance/ Co-Evolution

- In all test levels, unit testing and GUI testing
- But specially for GUI testing

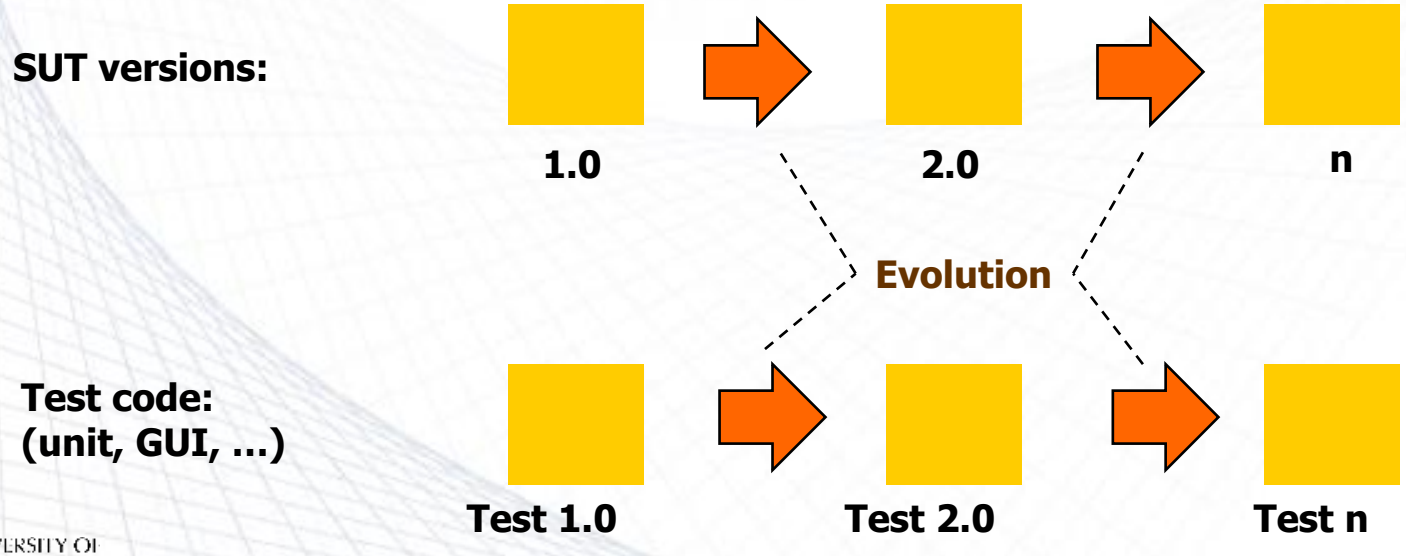


Word 1995

...



Word 2016

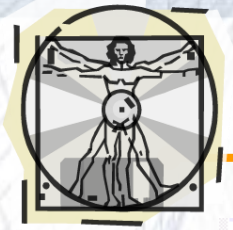


GUI Testing tools – Support Test Lifecycle

- Capture/playback tools should have the following capabilities:
  - (must) record scripts of user/system interactions
  - (must) replay the recorded test code (script)

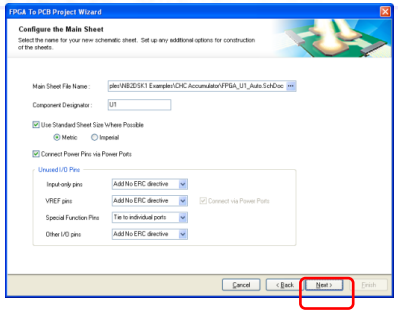
Nice to have

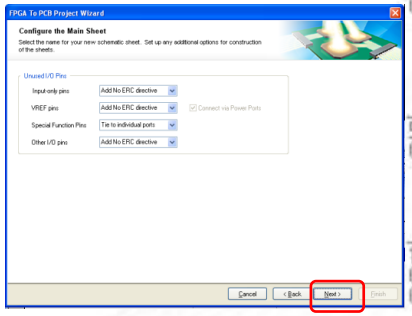
  - Tester's access to scripts for editing/maintenance
  - Tester's ability to insert validation points in the test script
  - Test script editing using higher-level abstractions such as icons, etc.
  - High level view of what functionality is being tested
  - Ability to generate many variations of a recorded script without having to manually edit the script itself → Data-driven testing

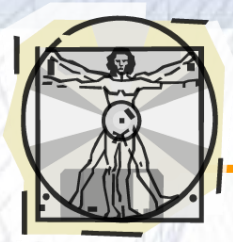


Insert Validation Points in the Test Script

```
// in the 1st  page
first_page.directory.click();
keyboard.enter("c:\temp\");

first_page.option.click();
mouse.left_click();
first_page.next_button.click();

wait(1); // 1 second

// verify that the 2nd page actually appears
assertExists(second_page.getObject());

// in the 2nd page
assertIncludes(second_page.installation_mode.getItems(), "Standard");
assertIncludes(second_page.installation_mode.getItems(), "Advanced");
assertIncludes(second_page.installation_mode.getItems(), "Premium");

second_page.installation_mode.open();
second_page.installation_mode.choose(2);

second_page.finish_button.click();

// the rest...
```

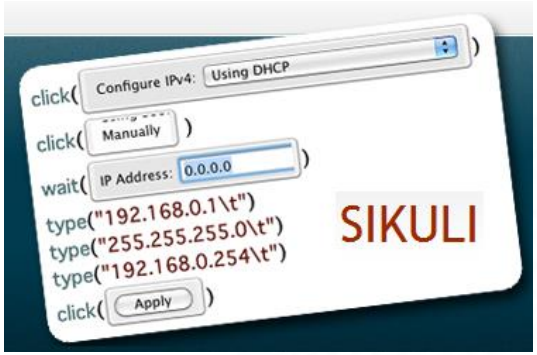






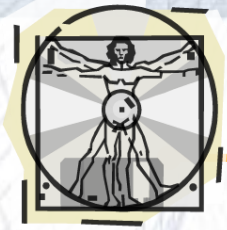
Script Editing Using Higher-level Abstractions

- Test script editing using higher-level abstractions such as icons, etc.
- Editable test script: in a known language, e.g. Python, Java, etc.

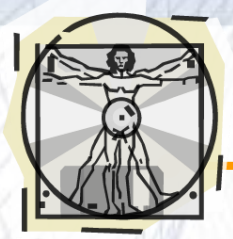



```
def test_textarea_add_del_by_key(self):
    type("hello world")
    assertExist( hello world. )
    type("a",KEY_CTRL)
    type("\n")
    assertNotExist( hello world. )
```

Coverage Criteria for GUI-based Testing



- Conventional code-based coverage is not adequate
- Possible coverage criteria:
  - **Event-coverage:** all events of the GUI need to be executed at least once
  - **State-coverage:** all states of the GUI need to be exercised at least once
  - **Functionality-coverage:** using a functional (operational) point of view, i.e. logical units of job
    - (e.g. log-in) are tested a least once

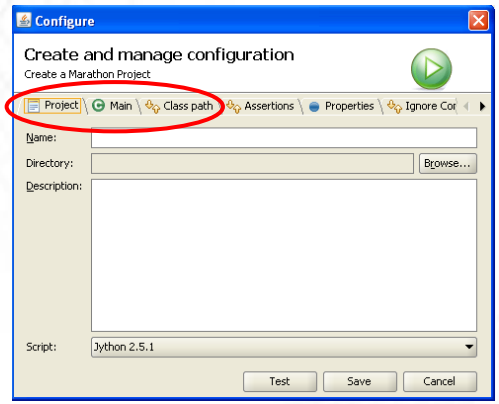


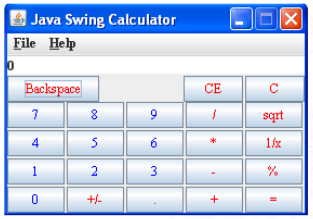
Example: Using Marathon

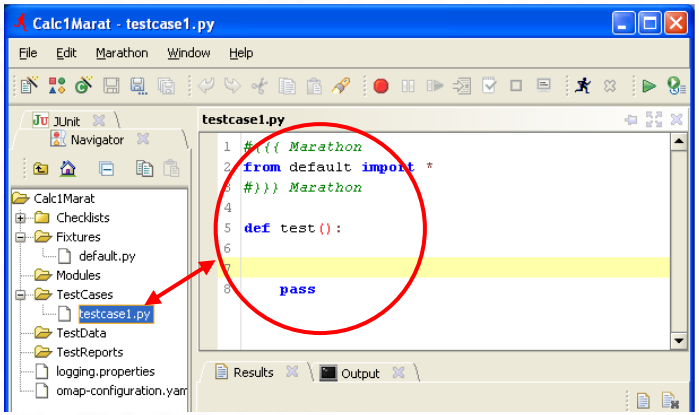
- Marathon is a commercial tool that helps writing System/ GUI tests for Java applications
- Consists of a recorder, player, and an editor
- Records tests script in Python
  - It is possible to use it without knowing Python …
- Allows to insert assertions in the script easily
  - Using a specific GUI
- Shows testing results using JUnit's control bar
  - Red/Green bar
- http://www.marathontesting.com
  - download Marathon and its user manual (trial license)

Example: Using Marathon

- Calculator application
- Write tests for the calculator, e.g. add two values
- Steps:
  - Create a new Marathon project
  - Create a new empty test (a test is a Python script file)
  - Build the test (recording)



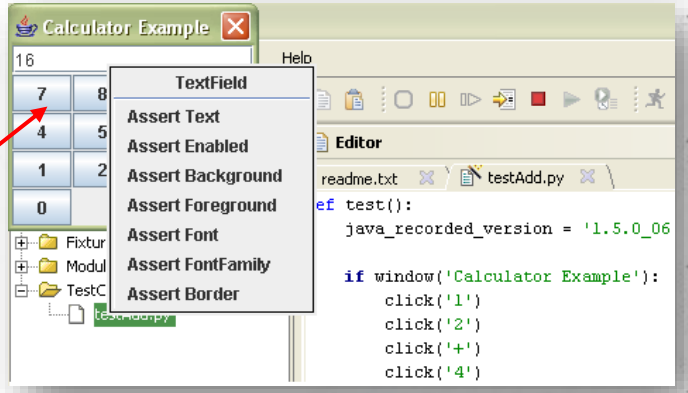


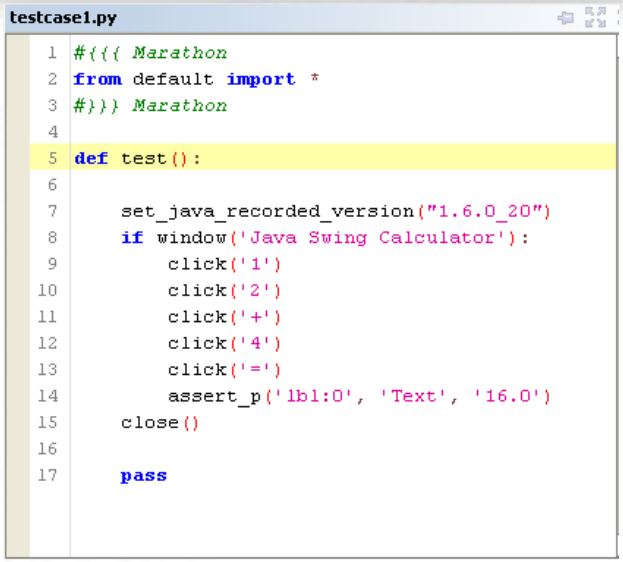


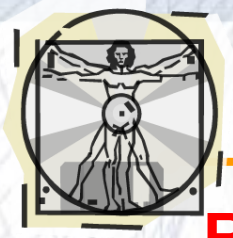
Example: Using Marathon

Record the script:

- Click on the record button in the toolbar
  - the red circle
- Enter “12 + 4 =”, the result is displayed (16)
- Press **control + right-click** in the text area, to open the contextual menu
- Select Assert Text
- Stop recording (the button with a red square), the script now looks like this:
- Save the script





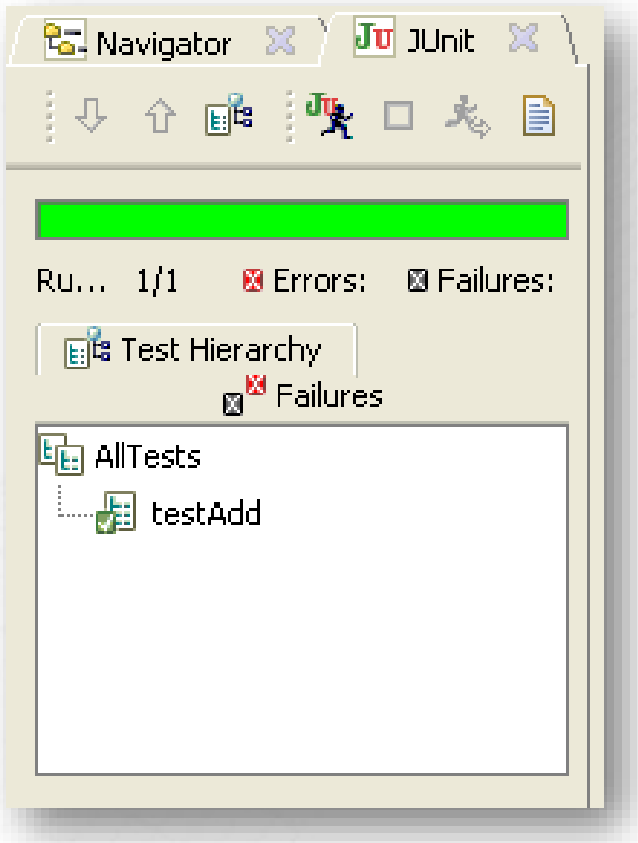


Example: Using Marathon

Run the script:

- Select the “JUnit” tab
- Click the test view, you should see your new created test
- Click on the “Run all tests” button, in the JUnit’s control bar

  and get the *green bar*

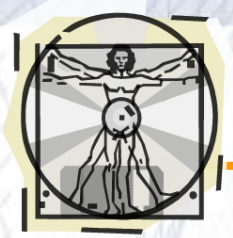


SCHULICH School of Engineering

UNIVERSITY OF CALGARY

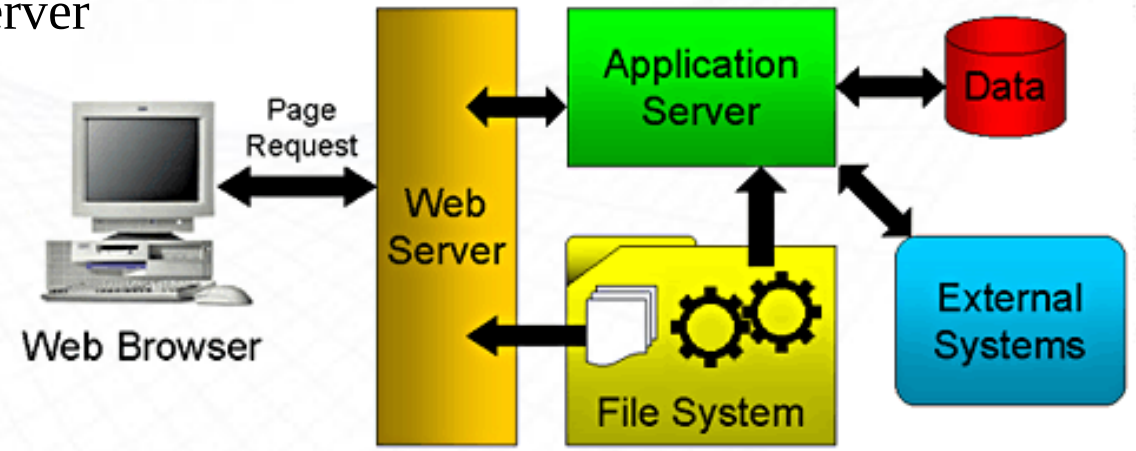
Section 2

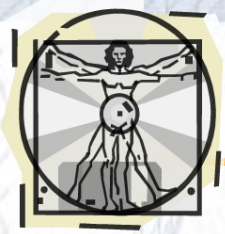
Selenium IDE & WebDriver



Web Application Architecture

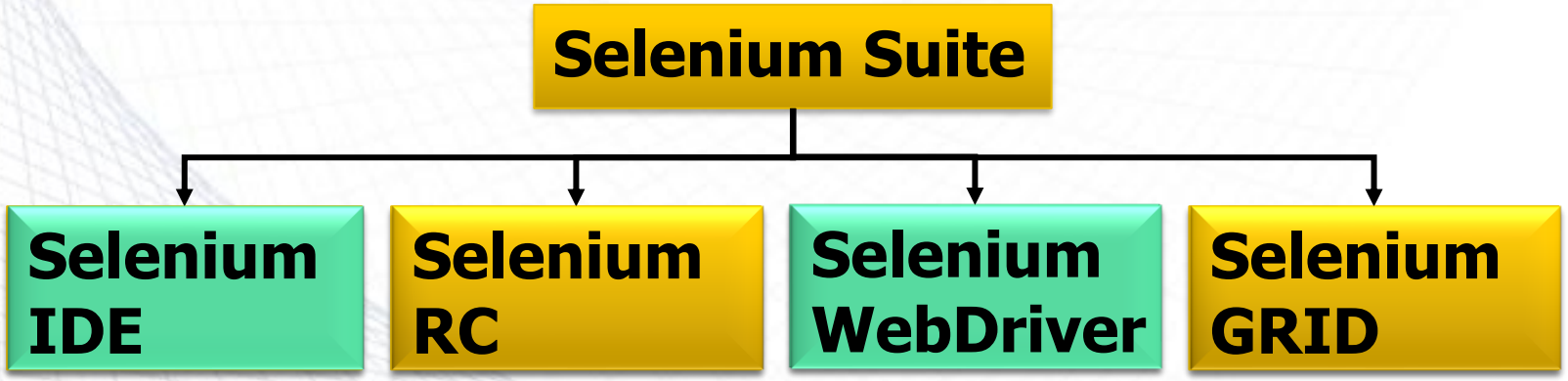
- Heterogeneous system
  - Front end
    - Browser: IE, Edge, Firefox, Chrome, Safari…
  - Server side
    - Application Server
    - Database Server
    - File System
    - ……
    - ……
    - Single-page (Angular)



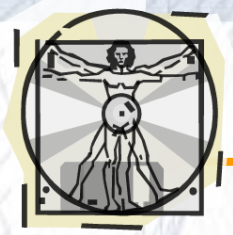


What is Selenium?

- Selenium is an open-source automated testing suite each catering to different testing needs
- Used for automating web-based applications across different platforms and browsers
- Supports C#, Java, Perl, PHP, Python, Ruby, Javascript
- Selenium has four components

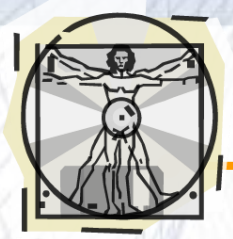


Tutorial: https://www.guru99.com/selenium-tutorial.html



Selenium Suite Components

- Selenium IDE
  - Supports FireFox and Chrome
  - Conditional operations are not supported
- Selenium remote control RC
  - Needs separate RC server
  - API has redundant and confusing commands
  - No direct browser interaction
  - Slow execution

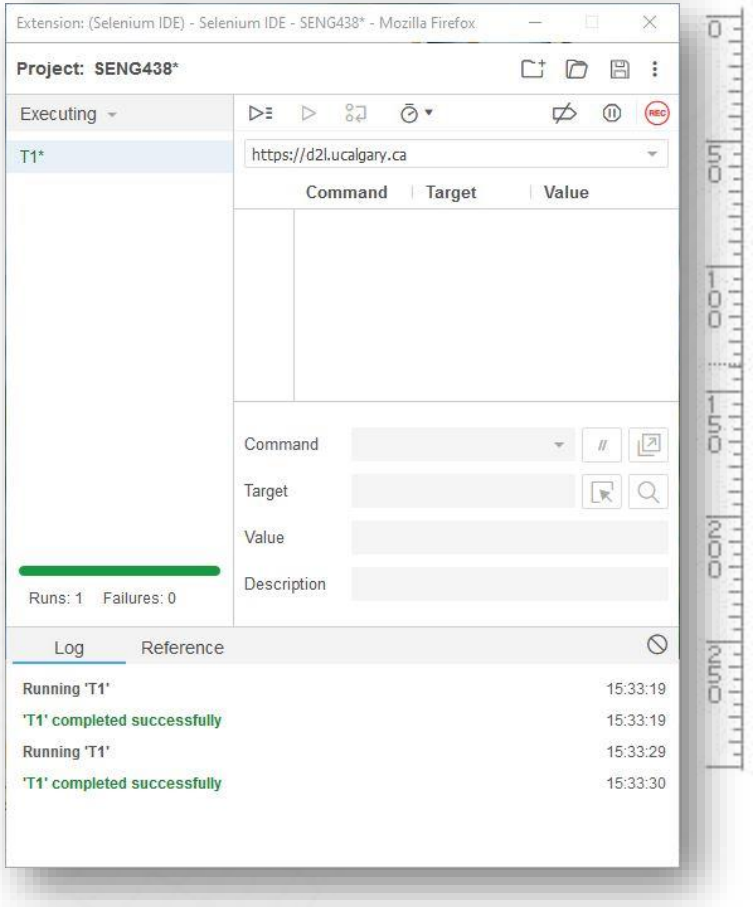


Selenium Suite Components

- Selenium Grid
  - Similar architecture as RC
  - Requires RC server to run in multiple browser and environments
- Selenium WebDriver (aka. Selenium 2)
  - Direct communication with browser
  - No need separate server
  - Simple commands
  - Faster execution

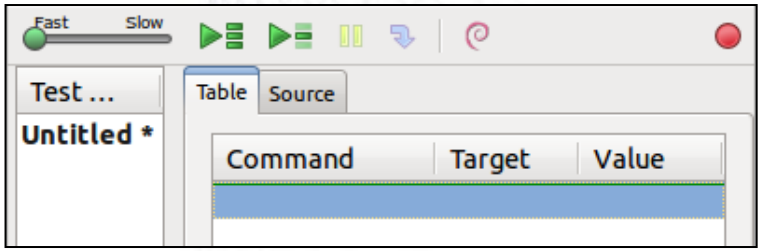
Selenium IDE

- Firefox (and Chrome) extension
- Easy record and replay
- Debug and set breakpoints
- Save tests in HTML, WebDriver and other formats

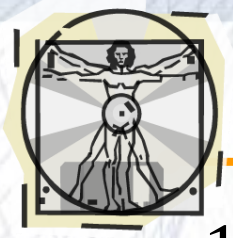


Selenium IDE Test Cases

- Selenium saves all information in an HTML table format

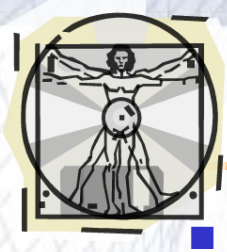


- Each record consists of:
  - **Command** – tells Selenium what to do (e.g. “open”, “type”, “click”, “verifyText”)
  - **Target** – tells Selenium which HTML element a command refers to (e.g. textbox, header, table)
  - **Value** – used for any command that might need a value of some kind (e.g. type something into a textbox)



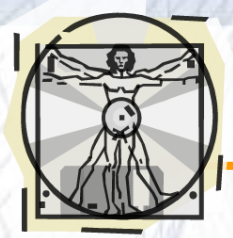
Record/Play with Selenium IDE

1. Start Selenium IDE
2. Point it to the URL of web SUT
3. Start recording in Selenium IDE
4. Execute a scenario on running web SUT
5. Stop recording in Selenium IDE
6. Verify / Add assertions
7. Save test
8. Replay the test



Selenium WebDriver

- Selenium-WebDriver
  - Write a program to control the test execution of a web SUT
  - More flexible and powerful than IDE
- Selenium-WebDriver supports multiple browsers in multiple platforms
  - Google Chrome 12+
  - Internet Explorer 6+
  - Firefox 3.0+
  - Opera 11.5+
  - Android – 2.3+ for phones and tablets
  - iOS 3+ for phones
  - iOS 3.2+ for tablets

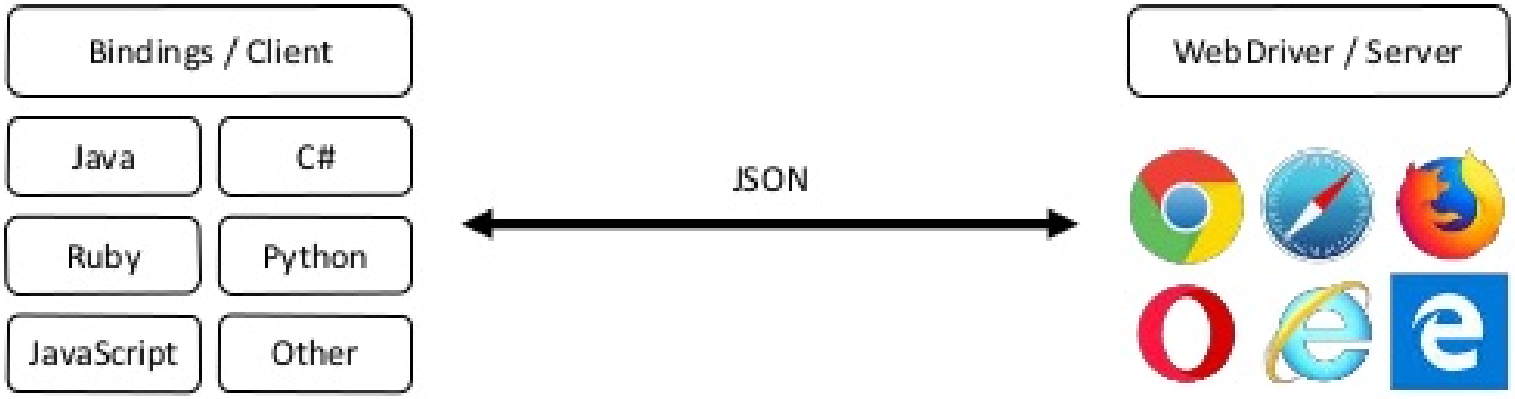


Selenium WebDriver

- WebDriver is designed to providing a simpler and uniform programming interface
  - Same WebDriver script runs for different platforms ← testing SUT on multiple browsers
- Support multiple programming languages:
  - Java, C#, Python, Ruby, PHP, Perl…
- It's efficient
  - WebDriver leverages each browser's native support for automation

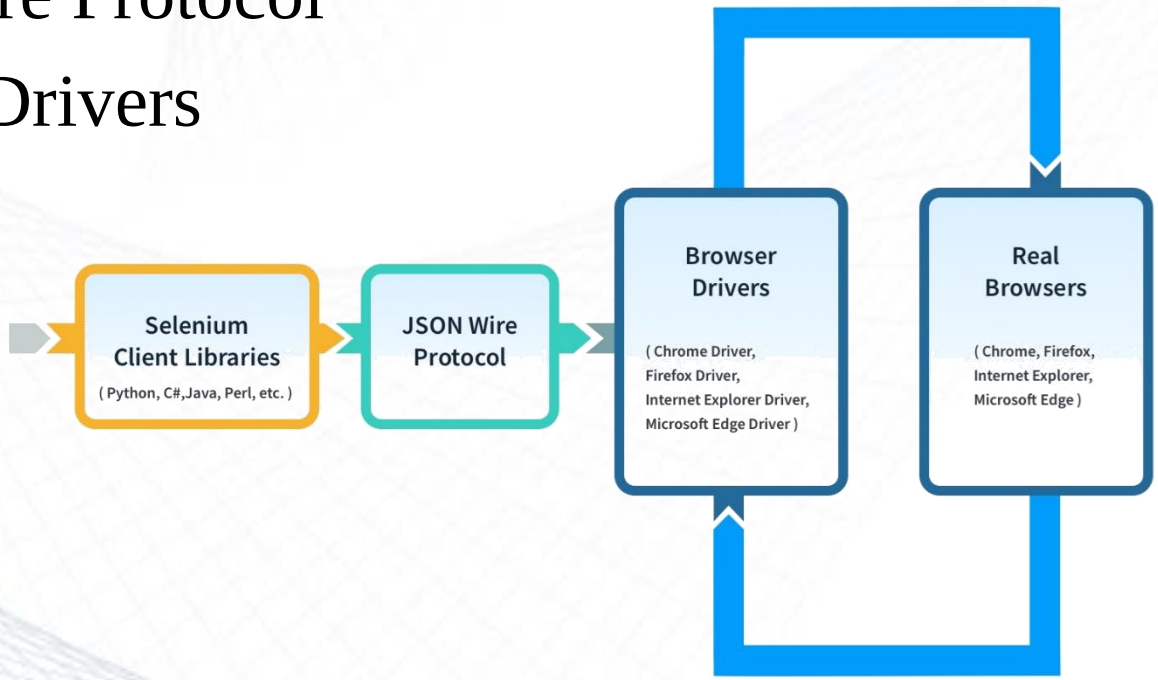
Selenium WebDriver

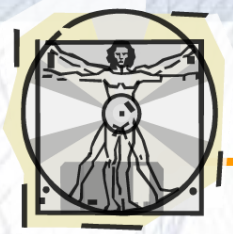
- Selenium provides ***bindings*** for different programming languages libraries that is used to write the tests with and ***WebDriver*** that can manage (and control) a specific browser, for which it is designated
- The two components communicate over HTTP by exchanging ***JSON*** payload, using JSON Wire protocol



WebDriver Architecture

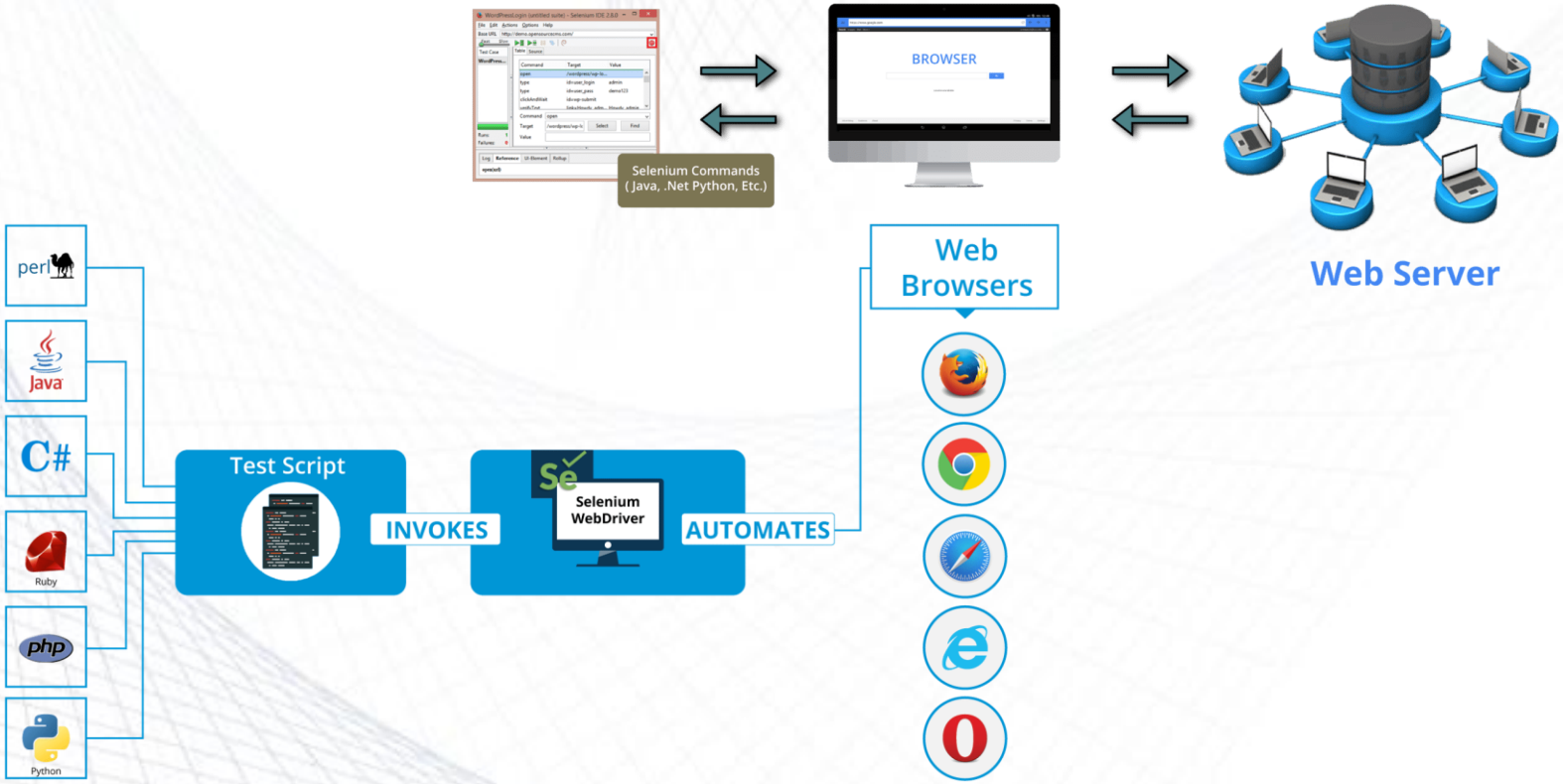
- **Four major components**
  - Client Libraries
  - JSON Wire Protocol
  - Browser Drivers
  - Browsers

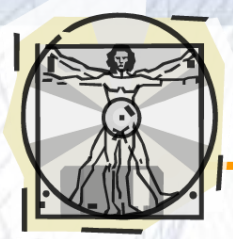




How WebDriver Works?

- This is how WebDriver works

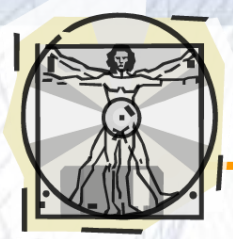




Browser Elements

- ***Browser Elements*** are the different components that are present on web pages
- Common elements are:
  - Text boxes
  - Call-to-action (CTA) buttons
  - Images
  - Hyperlinks
  - Radio buttons/ Check boxes
  - Text area/ Error messages
  - Drop down box/ List box/ Combo box
  - Web table/ HTML table
  - Frame

Testing browser elements means that we check whether they are working fine and responding the way we want them to



Using Selenium WebDriver

(1) Go to a page

(2) Locate an element

(3) Do something with that element

.....

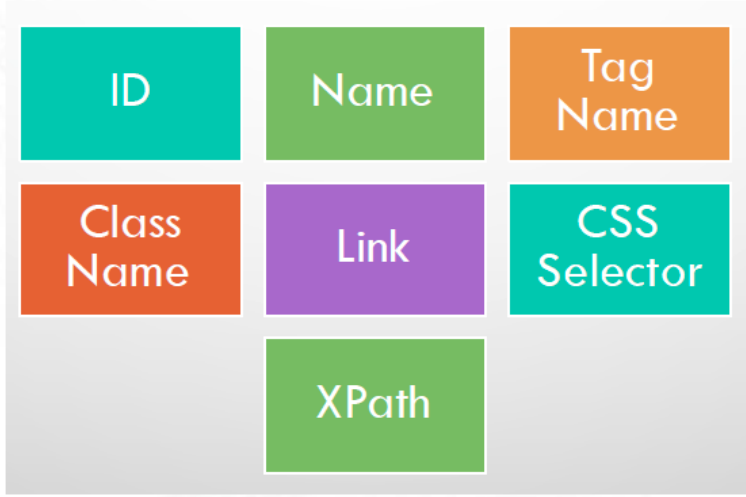
(i) Locate another element

(i+1) Do something with that element

(i+2) Verify / Assert the result

Element Locators

- When selenium test runs, first it will locate the target element from the SUT page and then it will perform given action on it (e.g. click, type, select, etc.)



How to Locate an Element

- By id
  - HTML: `<div id="coolestWidget">...</div>`
  - WebDriver:

```
driver.findElement( By.id("coolestWidget") );
```

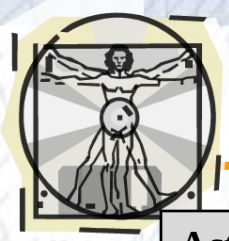
- By name
  - HTML: `<input name="donald" type="text"/>`
  - WebDriver: `driver.findElement( By.name("donald") );`
- By Xpath
  - HTML

```
<html>
    <input type="text" name="example" />
  <input type="text" name="other" />
</html>
```

  - WebDriver: `driver.findElements( By.xpath("//input") );`
  - There are plug-ins for firefox/chrome to automatically display the Xpath

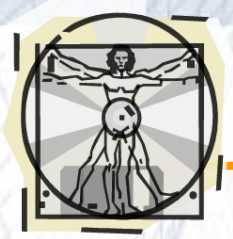
Locator Types

Locator Type	Syntax	Description
id	driver.findElement (By.id("ID_of_Element"))	Locate by value of the "id" attribute
className	driver.findElement (By.className("Class_of_Element"))	Locate by value of the "class" attribute
linkText	driver.findElement (By.linkText("Text"))	Locate by value of the text of the hyperlink
partialLinkText	driver.findElement (By.partialLinkText("PartialText"))	Locate by value of the sub-text of the hyperlink
name	driver.findElement (By.name("Name_of_Element"))	Locate by value of the "name" attribute
xpath	driver.findElement (By.xpath("Xpath"))	Locate by value of the xpath
cssSelector	driver.findElement (By.cssSelector("CSS Selector"))	Locate by value of the CSS selector
tagName	driver.findElement (By.tagName("input"))	Locate by value of its tag name



Action Commands

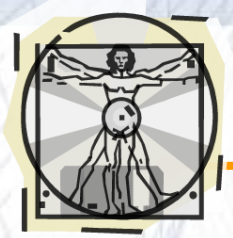
Actions	Commands
Open URL in a browser	driver.get(“anyurl”)
Clicking any element on the page	driver.findElement(By.id(“anycontrol")).click
Typing text in any textbox or text area	driver.findElement(By.name(“txt")).sendKeys("Name");
Clear text in any textbox or text area	driver.findElement(By.name(“txt")).clear();
Get Page title	driver.getTitle();
Get current url of the page	driver.getCurrentUrl();
Navigate to ul/back/forward	driver.navigate().to(“anyurl"); driver.navigate().back(); driver.navigate().forward(); -
Get text of any control	driver.findElement(By.name(“txt")).getText();
Wait	Thread.sleep(5000);
Dropdown selection /deselection	Select dropdown = new Select(driver.findElement(By.id(“dropdownlist"))); dropdown.selectByVisibleText("Audi");



Using Selenium

- Scenario
  - Open “https://d2l.ucalgary.ca”.
  - Enter a valid username and password and submit the details to login
  - Verify that the user credentials (email and password) is correct
- **First using SeleniumIDE**
- **Then using WebDriver**

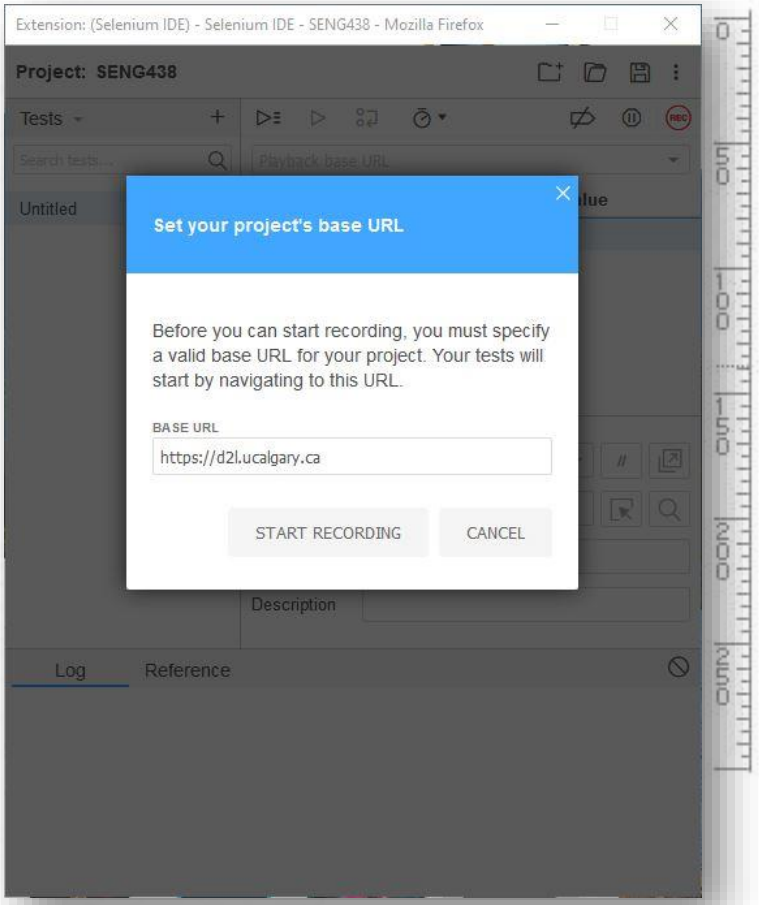
Tutorial: https://www.guru99.com/selenium-tutorial.html

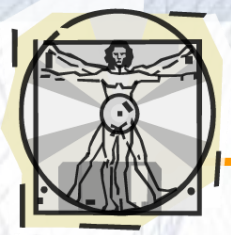


IDE: 1. Create Script

- Install Selenium IDE add-on for FireFox

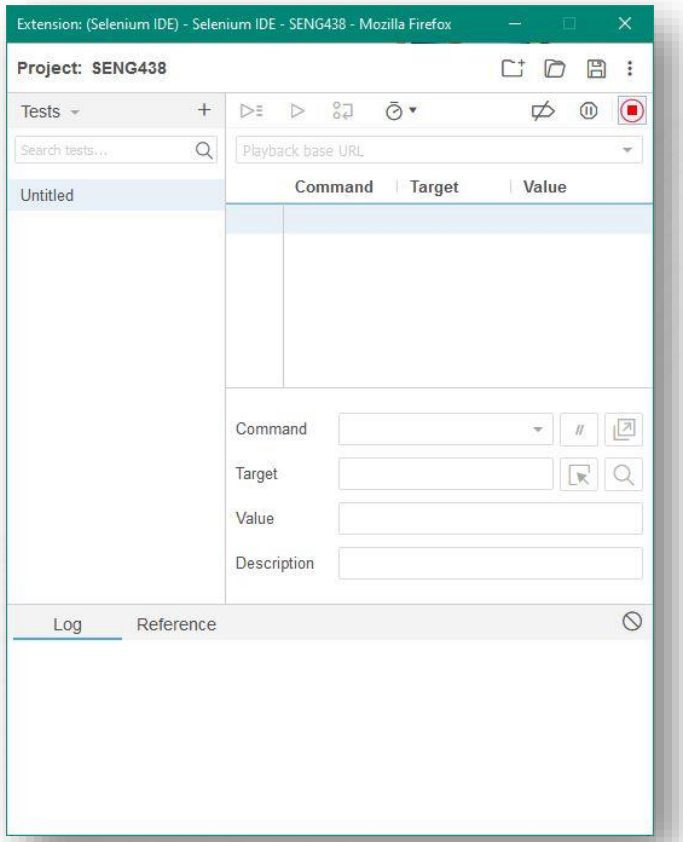
- **Step 1 –** Launch the Firefox and open Selenium IDE from the menu bar
- **Step 2 –** Enter the address of web application under test, e.g. (“https:// d2l.ucalgary.ca”) inside the Base URL textbox





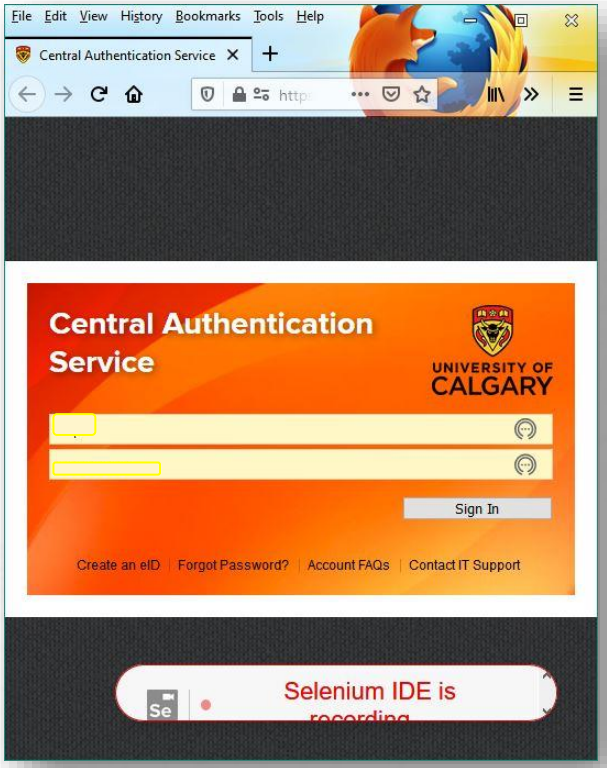
IDE: 1. Create Script

- **Step 3 –** By default, the Record button is in ON (remember to tune it ON if it is OFF) so as to enable the recording mode
- **Step 4 –** Open the application under test (https://d2l.ucalgary.ca) in the Firefox

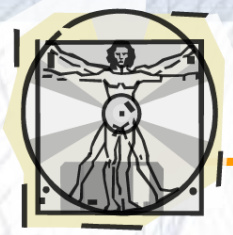


IDE: 1. Create Script

- **Step 5 –** Enter a valid username in the "User ID" textbox
- **Step 6 –** Enter a valid password in the "Password" Textbox
- **Step 7 –** Click on the "Sign in" button to complete the login process
- **Step 8 –** Stop the record

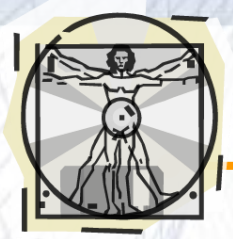


The simulation of the same user actions can be seen in the Selenium IDE test editor



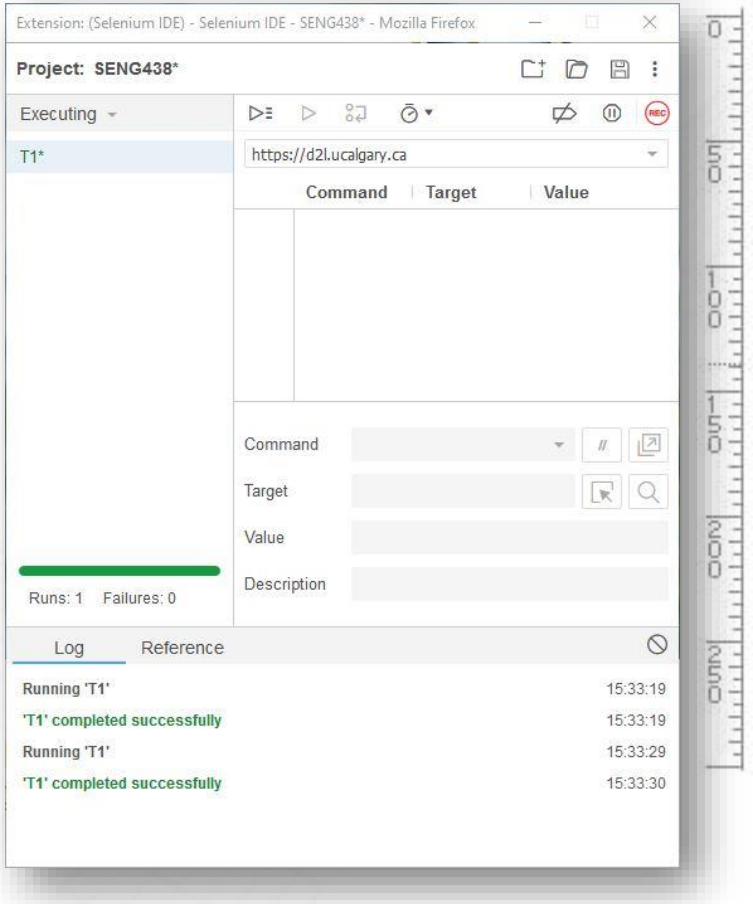
IDE: 2. Save a Test Script

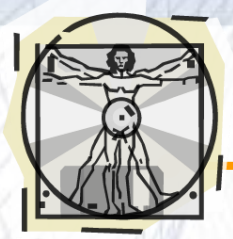
- **Step 1 –** To save the test script, Click on the File menu and select "Save Test Case" option
- **Step 2 –** The system will prompt to browse or enter the desired location to save our test case and to provide the test script name
  - Furnish the test name as "T1" and click on the "Save" button



IDE: 3. Execution

- Select the saved test and click on the playback button to execute the script
- Post-execution, all the test steps would be color coded in **green** for the successful run
- For unsuccessful execution or test case failure, the failed test step would be highlighted in **red**





WebDriver

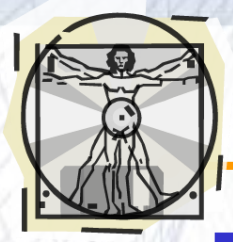
- Download Selenium JAR for Java

  https://www.seleniumhq.org/download/

- Download Selenium For Python

  https://pypi.org/project/selenium/

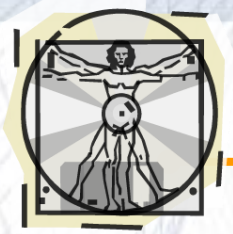
- Configure Eclipse to use Selenium



WebDriver Example 1

- Scripting with WebDriver: Verify Page Title

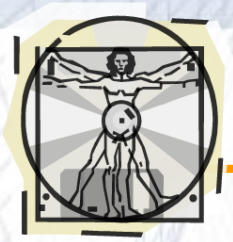
```
public static void main( String[] args )
{
    // Create a new instance of the Firefox driver
    WebDriver driver = new FirefoxDriver();
    // (1) Go to a page
    driver.get("http://www.ucalgary.ca");
    // (2) Locate an element
    WebElement element = driver.findElement(By.name("q"));
    // (3-1) Enter something to search for
    element.sendKeys("University of Calgary");
    // (3-2) Now submit the form. WebDriver will find the form for us from the element
    element.submit();
    // (3-3) Wait up to 10 seconds for a condition
    WebDriverWait waiting = new WebDriverWait(driver, 10);
    waiting.until( ExpectedConditions.presenceOfElementLocated( By.id("pnnext") ) );
    // (4) Check the title of the page
    if( driver.getTitle().equals("University of Calgary") )
       System.out.println("PASS");
    else
       System.err.println("FAIL");
    //Close the browser
    driver.quit();
}
```



WebDriver Example 2

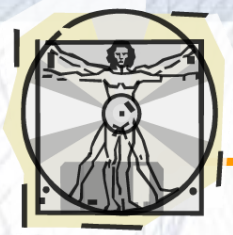
- Create a new java class named as “Gmail_Login” under the “Learning_Selenium” project
- Paste this code into it
- Under the Eclipse’s menu bar, push execute button to run the test script

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;

public class D2L_Login {

    public static void main(String[] args) {
// objects and variables instantiation
        WebDriver driver = new FirefoxDriver();
        String appUrl = "https://D2L.ucalgary.ca";
// launch the firefox browser and open the application url
        driver.get(appUrl);
// maximize the browser window
        driver.manage().window().maximize();
// enter a valid username in the email textbox
        WebElement username = driver.findElement(By.id("Email"));
        username.clear();
        username.sendKeys("TestSelenium");
// enter a valid password in the password textbox
        WebElement password = driver.findElement(By.id("Passwd"));
        password.clear();
        password.sendKeys("password123");
// click on the Sign in button
        WebElement SignInButton = driver.findElement(By.id("signIn"));
        SignInButton.click();
// close the web browser
        driver.close();
        System.out.println("Test script executed successfully.");
// terminate the program
        System.exit(0);
    }
}
```



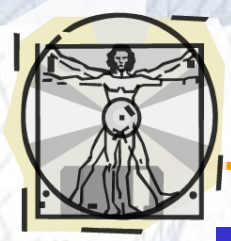
Timing issue

- There are delays between submitting a request and receiving the response
- We can wait until the response page is loaded
- Robot doesn't know!
- In WebDriver, sometimes it doesn't work if
  - Submit a request
  - Verify the response immediately
- Solution:
  - Simulate the wait. Wait until some HTML object appears



WebDriver: Evaluation

- Advantages:
  - Direct interaction with browser
  - Faster than any other selenium component
  - Easy and stable API commands
- Disadvantages:
  - Cannot readily support new browsers
  - No built-in mechanism to generate test reports
  - Separate drivers for different browsers



WebDriver Summary

- A solution for the automated testing
  - Simulate user actions
  - Functional testing
    - One could even program in a test script
    - Create regression tests to verify functionality and user acceptance
  - Browser compatibility testing
    - The same script can run on any Selenium platform
  - Volume testing
  - Stress testing

SCHULICH School of Engineering

UNIVERSITY OF CALGARY

Section 3

GUI Testing Tools

GUI Testing - Approaches

- Generation 1: Completely manual
- Generation 2: Semi-automated generation
- Generation 3: Automated generation

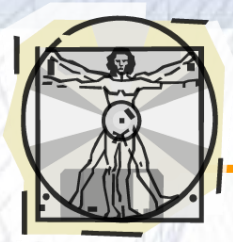
	Test-case design	Test execution	Test evaluation (pass/fail)
Generation 1: Completely manual	Human	Human	Human
Generation 2: Capture/replay	Human	Machine	Human/Machine
Generation 3: Model-based test generation	Machine	Machine	Human/Machine

GUI Testing Tools

- Some free/open source GUI automation tools

Product	Licensed Under	URL
AutoHotkey	GPL	http://www.autohotkey.com/
Selenium	Apache	http://docs.seleniumhq.org/
Sikuli	MIT	https://launchpad.net/sikuli
SikuliX	GPL	http://sikulix.com/
Robot Framework	Apache	www.robotframework.org
watir	GPL	http://www.watir.com/
Dojo Toolkit	GPL	http://dojotoolkit.org/

And many more …

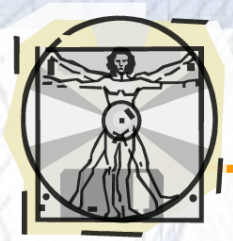


GUI Testing Tools

- Some commercial GUI automation tools

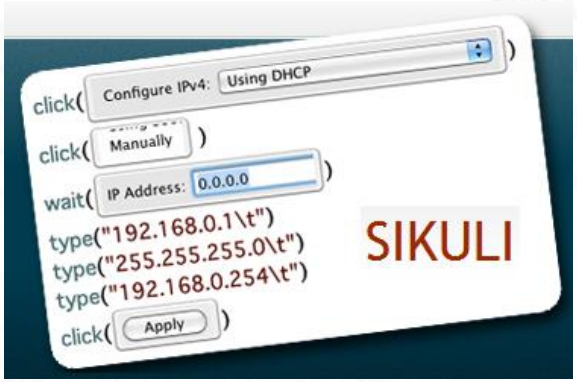
Product	Vendor	URL
AutoIT	AutoIT	http://www.autoitscript.com/site/autoit/
EggPlant	TestPlant	www.testplant.com
QTP	Hp	http://www8.hp.com/us/en/software-solutions/
Rational Functional Tester	IBM	http://www-03.ibm.com/software/products/us/en/functional
Infragistics	Infragistics	www.infragistics.com
iMacros	iOpus	http://www.iopus.com/iMacros/
CodedUI	Microsoft	http://www.microsoft.com/visualstudio/

And many more …



GUI Testing Tools

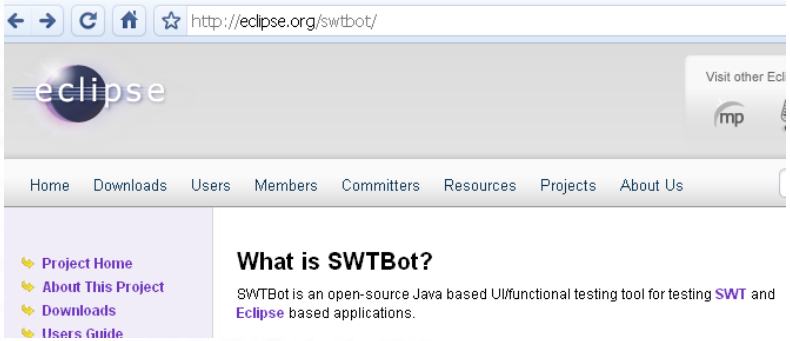
- Open source (Free)



https://www.youtube.com/watch?v=ueAQowkMbpI

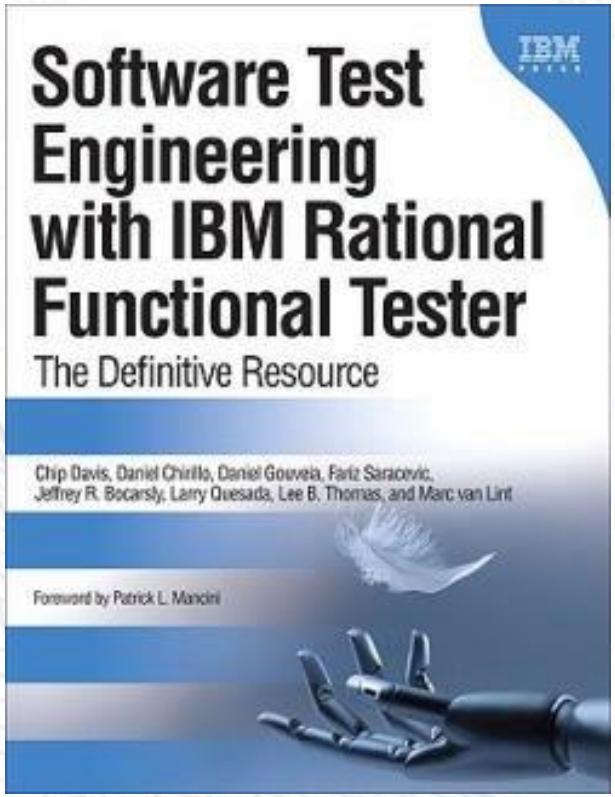
NunitForms
windows.forms unit testing

Find out more about
eggPlant
User interface testing

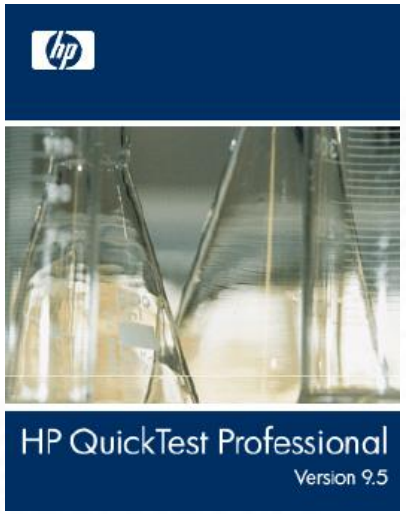


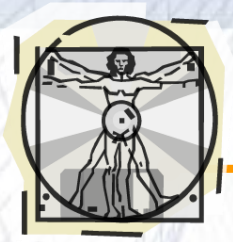
GUI Testing Tools

- Commercial



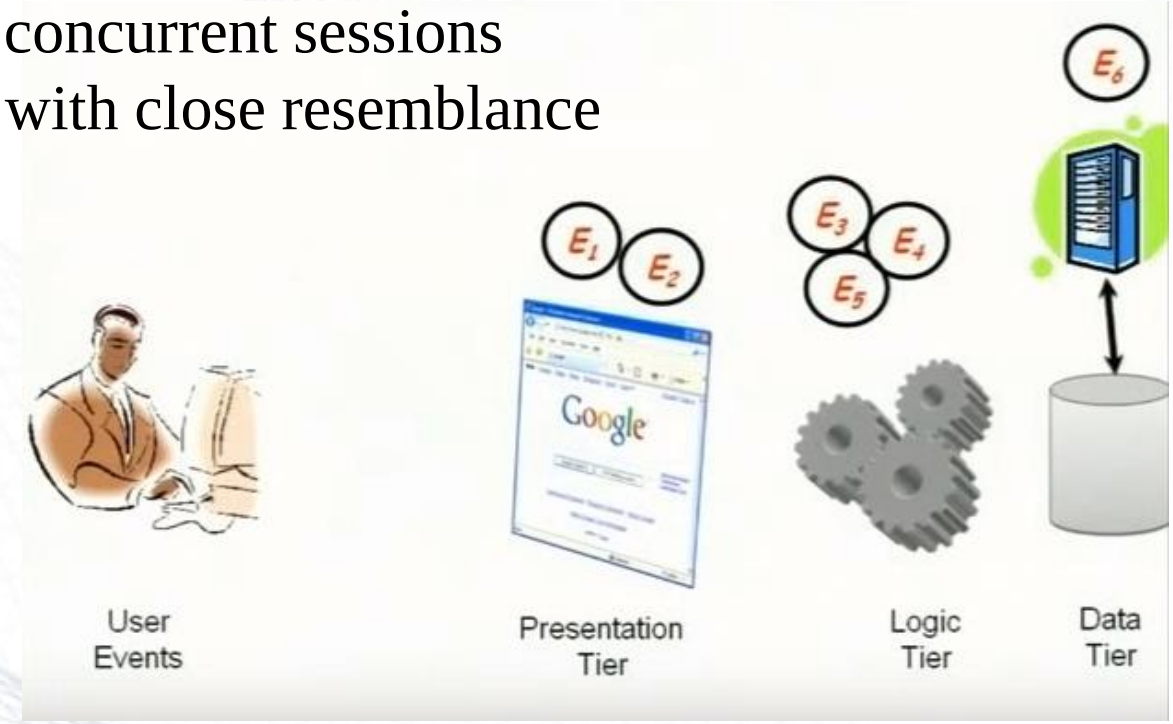
Marathon



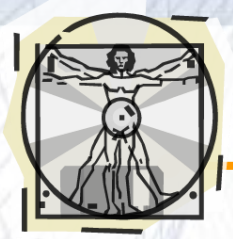


Web vs. Desktop Applications

- What is the difference between Web applications and other GUI-based apps?
  - Multiple concurrent sessions
  - Sessions with close resemblance



https://www.youtube.com/watch?v=6LdsIVvxISU



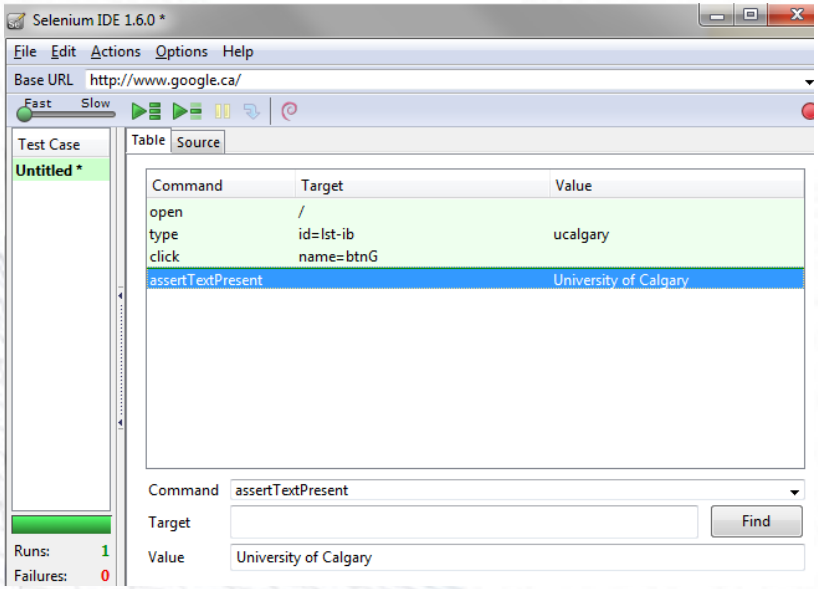
GUI Testing of Web Applications

- The example we discussed already were mostly conventional “desktop” application
- GUI testing of (rich) web applications is also very main stream now
- A large number of “record and playback” tools exist for GUI testing of web applications

GUI Testing of Web Applications

Selenium (https://github.com/SeleniumHQ)

- Test-case design and types of assertions are up to the user
- The tool automates the record and playbook
- It does not suggest test cases, or the right assertions



HtmlUnit, HttpUnit, JWebUnit

https://www.youtube.com/watch?v=BL4-_tVx2rE

GUI Testing of Web Applications

- Has a test Domain-Specific Language called Selenese (HTML-based)
- We can export the tests to a number of popular programming languages, including C#, Java, Groovy, Perl, PHP, Python and Ruby.

open	/	
type	q	selenium rc
clickAndWait	btnG	
assertTextPresent	Results * for selenium rc	

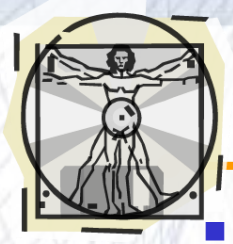
csharp java

```
    /** Add JUnit framework to your classpath if not already there
     *  for this example to work
     */
package com.example.tests;

import com.thoughtworks.selenium.*;
import java.util.regex.Pattern;

public class NewTest extends SeleneseTestCase {
    public void setUp() throws Exception {
        setUp("http://www.google.com/", "*firefox");
    }
      public void testNew() throws Exception {
          selenium.open("/");
          selenium.type("q", "selenium rc");
          selenium.click("btnG");
          selenium.waitForPageToLoad("30000");
          assertTrue(selenium.isTextPresent("Results * for selenium rc"));
    }
}
```

Resources

- GoogleTechTalks (Talk by Atiff Memon)
  - Automated Model-Based Testing of Web Applications
  - https://www.youtube.com/watch?v=6LdsIVvxISU
- From Manual Testing to Automated Testing
  - https://www.youtube.com/v/y33tcrd5ZeE?start=90&end=1020
  - Watch minute 1:30 – 17
- GUI testing of Mobile Apps
  - Mobile testing using HP QuickTest Professional
  - https://www.youtube.com/watch?v=JOyBPGYR1iI

UNIVERSITY OF CALGARY



Further to Read …

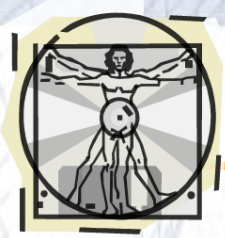
- It is not easy and straightforward to use Selenium for Angular websites (non- traditional web pages which use "single page application" concept)
- Angular unit testing with Jasmine, Karma
  - https://www.youtube.com/watch?v=BumgayeUC08
- Angular unit testing
  - https://docs.angularjs.org/guide/unit-testing
- Protractor: test automation framework for Angular apps
- Automation testing tools for Web applications:
  - https://screenster.io/automation-testing-tools-in-2018/
- What are the best UI test automation tools?
  - https://blogs.msdn.microsoft.com/testingspot/2018/11/09/what-are-the-best-ui-test-automation-tools/

SCHULICH School of Engineering

UNIVERSITY OF CALGARY

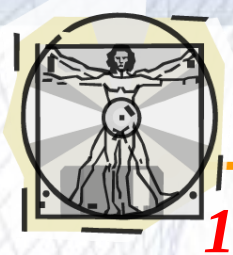
Section 3

Myths, Misses and Hints about Test Automation



Reality …

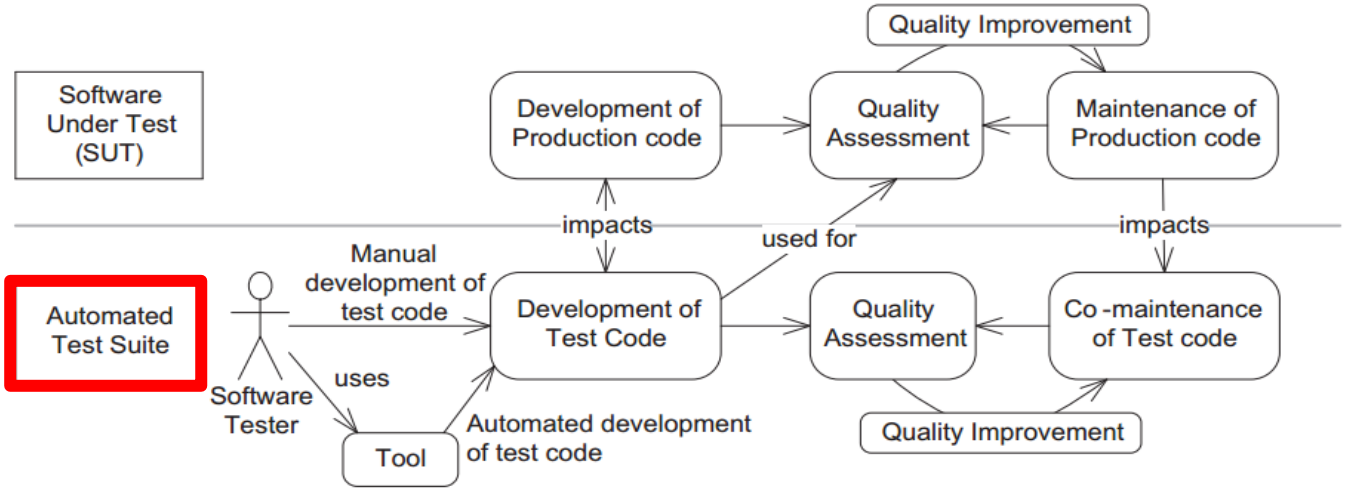
- Many attempts to use GUI level test automation tools fail
- You might have heard more failure stories than success stories

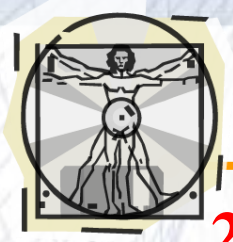


Myths – Misses – Hints

1. Test Automation is simple, every tester can do it!

- This myth is promoted by the tool sales people
- Test automation requires engineering process
- Automation should be designed, developed and tested
- Automation personnel need to have (good) programming background

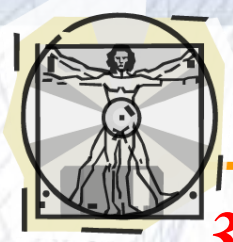




Myths – Misses – Hints

2. Don't underestimate the cost of automation

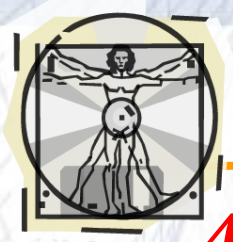
- It takes about 3-10 times as long to create, verify, minimally document, and save an individual automated test (specially for system level testing)
- There are ways to reduce the unit cost of automated tests, but they require substantial up-front planning
- You don't gain back much in short-term
- You'll lose some time on additional training and administrative overhead



Myths – Misses – Hints

3. Don't underestimate the need for staff training

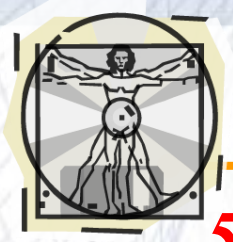
- Many test automation tools come with their own funky programming languages, their own custom debugging tools and other support tools, their own jargon, and their own vision about the best way to do product development
- You need time to learn the tool ← Plan for it



Myths – Misses – Hints

4. Commercial test tools are expensive

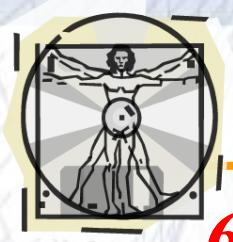
- Under the influence of this myth some companies, especially the small ones:
  - Try to develop their own test automation tools
  - Do not consider test automation at all
- Cost of tool should be compared to saving from automation
  - Commercial tools can be cheap depending saving from automation



Myths – Misses – Hints

5. Don't shoot for "100% automation"

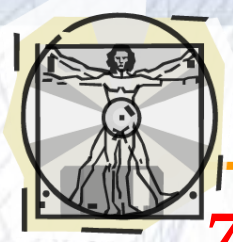
- Select test cases for automation that will be "run several times" in future
- Use automation when the SUT code is mature enough
- Use automation mainly during regression, integration and system testing



Myths – Misses – Hints

6. Don't create test scripts that won't be easy to maintain

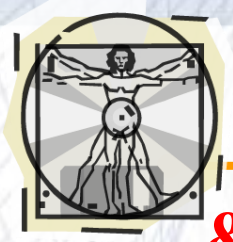
- Maintenance requirements don't go away just because your friendly automated tool vendor forgot to mention them!
- If you don't write code that is easy to maintain when the UI changes, the odds are good that you will carry very little testing code forward from Release N to N+1



Myths – Misses – Hints

7. Treat test automation as a genuine programming project

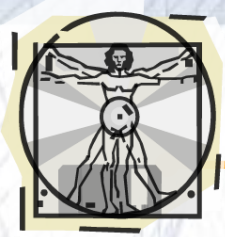
- Automation of software testing is just like all of the other automation efforts that software developers engage in—except that this time, the testers are writing the automation code
  - Understand the requirements
  - Adopt an architecture that allows to efficiently develop, integrate, and maintain features and data
  - Adopt and live with standards
  - Be disciplined



Myths – Misses – Hints

8. Don't forget to document your work

- Documentation of test cases and testing strategy is often treated as an afterthought by automated test developers
- You can't maintain code that you don't understand
- You can't effectively reuse test cases if you don't know what tests they include



Conclusion

- GUI test automation is a must in systematic testing of modern software systems, apps and web applications
- The automation technique relies heavily on the tool/plug-ins used
- There is no silver-bullet …

"Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live!" (Martin Golding)