



Test Automation

Vincenzo Cuomo & Felice Cerullo



Overview

- Software Test Automation: foundations
 - GUI Testing & *TAF* – Test Automation Framework
 - Working with ANT & Jenkins
 - YouTestManager – Test Framework for Embedded SW
 - Demo
-



PART 1

Software Test Automation: foundations

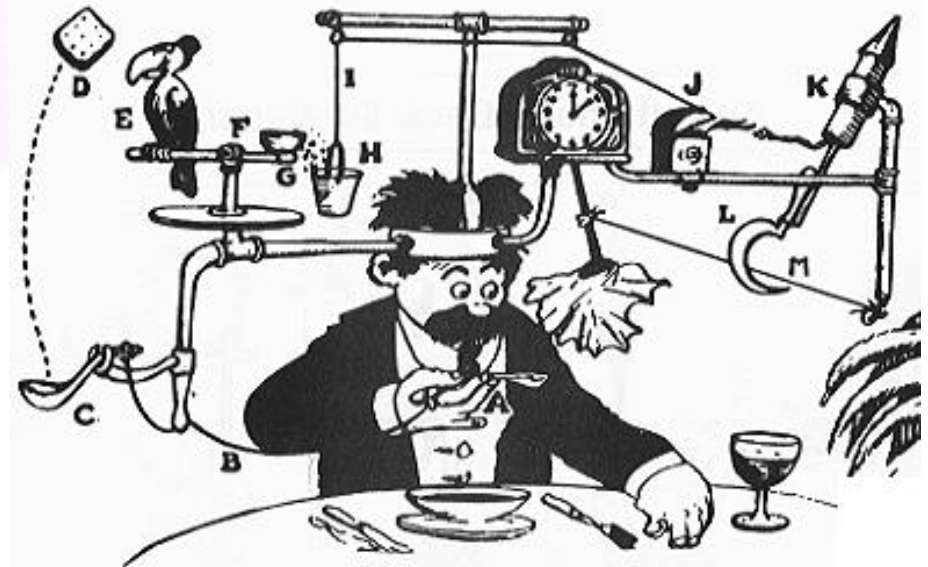
Software Test Automation

- What is Test Automation
 - Test Automation Techniques
 - Test Automation Tools
-

Test Automation

■ *Test Automation* means that you have a tool to perform or support in:

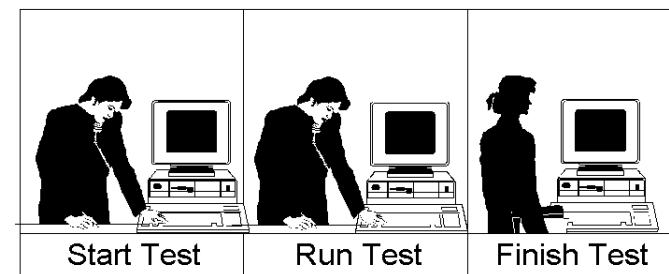
- generating test data
- setting up of test preconditions
- test execution
- comparisons of actual to expected results
- test management
- ...



Why Test Automation

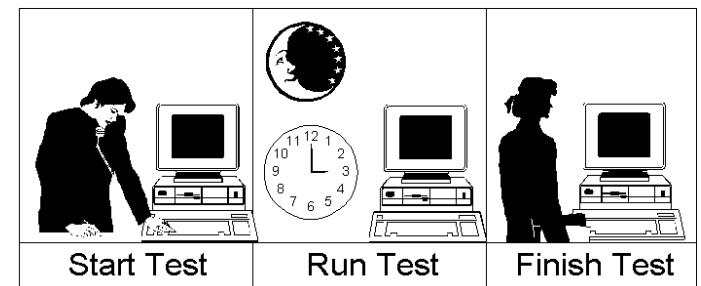
■ *Manual Testing* limitations:

- very laborious and time consuming
- it is not possible to execute more tests more often
- test effort cannot be re-used
- complete verification of results not always possible
- error prone (people make errors!)
- sometimes very monotonous (e.g. regression testing)
- ...



Why Test Automation

- Run more tests more often
- Perform tests that would be difficult or impossible to do manually (e.g. load, performance, ...)
- Repeatability of tests
- Test tools reduce the time for running tests: testers have more time for planning and designing new tests
- Automated scripts give 24 test execution hrs in a day!
- Reduced error
- ...



Challenges in Test Automation

- Not all tests can be automated
 - Pays off late
 - Automation ROI typically takes longer than one project
 - From 3 to 10+ times the effort to create an automated test than to just manually do the test
 - There are some testing tasks a human is better suited to than a computer
 - Visual aspects are difficult to test by a machine
 - Humans are better at pattern matching
 - Machines can't investigate strange behaviors
-

Challenges in Test Automation

- Many bugs are found because testers have intuition, while machines can't see suspicious behavior and investigate it
- Manual tests find more defects than automated tests: 85% vs. 15%
- Tests are more complex at the *User Interface* (UI)
 - UIs very often change frequently, which increases maintenance
- ...



Challenges in Test Automation

Test Automation

SHALL BE :

1. *Application independent*
2. *Highly reusable*
3. *Easily maintainable*
4. *Flexible and extensible*



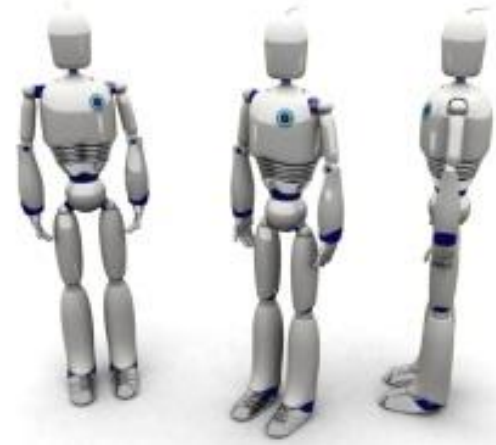
Test Automation **is** Software Development

Test Automation Tasks

- Test automation involves more tasks than just running of tool:
 - Test Strategy
 - Planning what to automate
 - Which tool to use
 - How to use tools properly (“scripting” technologies)
 - Training
 - Support
 - ...
-

Test Automation Tools

- Major Classes of Testing Tools
 - Source Code Test Tools (Static analysis, Coverage, ...)
 - Functional / GUI Test Tools
 - Performance Test Tools
 - Load / Stress Test Tools
 - (Web) Security Test Tools
 - Unit Test Tools
 - Test Management Tools
 - ...



Types of Automation

- Major types of automation:
 - Record & Playback
 - Data-driven
 - Keyword-driven
 - ...



Record and Playback

- Generating scripts by recording the user actions (e.g. keyboard and mouse actions) on the initial version of the application-under-testing (AUT)
- Playing back the scripts on the next versions of the AUT



The tester interacts with the AUT

```
//Select the "File" menu item in the menubar  
barradeimenu().click(atPath("File"));  
  
//Select the "Open" menu item in the menubar  
barradeimenu().click(atPath("File->Open..."));  
  
//Write the file name in the Open dialog  
VFSFileChooserDialog().inputChars("DocumentFile.doc");  
  
//Click the "Open" button  
Open().click();
```

A 'macro-like' script is produced

Record and Playback

■ Advantages(?):

- Less effort for automation
- Quick returns
- Does not require expertise on tools



■ Limitations(!):

- The AUT must be available for testing
- High dependency on the AUT
 - Scripts contain hard-coded values which must be changed if the AUT changes
 - Scripts contain no error handling and recovery scenarios
- Difficult to maintain the scripts
 - If the AUT changes, very often the scripts must be re-recorded!

Test Automation IS NOT “Record & Playback”

Data-driven testing

- Scripts are freed from hard-coded data
 - Data are separated from the test scripts (*captured* or *manually coded*)
 - Testing multiple data with same script
- Scripts are coded to accept variable data from an external source (e.g. file, data-base, spreadsheet, ...)

hard-coded data

```
//Select "File" menu in the application menubar  
Menubar().click(atPath("File"));  
//Select "Open" menu item in the menu "File"  
Menubar().click(atPath("File->Open"));  
//Insert the name of the file to open in the Open dialog window  
VFSFileChooserDialog().inputChars("Test.doc");  
//Click the "Open" button in the Open dialog window  
Open().click;
```

data-driven

```
// MENU, MENU_ITEM, DOCUMENT_NAME and DOCUMENT_NAME  
// are defined in file Test.dat  
//Select menu MENU in the application menubar  
Menubar().click(atPath(MENU));  
//Select MENU_ITEM menu item  
Menubar().click(atPath("MENU_ITEM"));  
//Insert the DOCUMENT_NAME to open  
VFSFileChooserDialog().inputChars(DOCUMENT_NAME);  
//Click the "Open" button in the Open dialog window  
Open().click;
```

Data-driven testing

■ Advantages:

- Reduces the overall number of scripts needed to cover the test cases
- Easy to maintain test scripts
- The volume of test data can be increased (useful if testing is data-intensive)

■ Limitations:

- Application must be available for testing
 - High dependency on the AUT
 - Scripts are still “monolithic”
 - When the AUT changes the scripts (and data) require change
 - Maintenance of the test script and data files is expensive
-

Keyword-driven testing

- Scripts consist of sequences of (calls to) library function: each function is uniquely identified by a 'keyword'
 - Scripts consist in sequences of keywords
 - Keywords are commonly used actions or tasks that are to be executed in a test
 - In Keyword-driven testing libraries of Keywords are developed to be re-used creating test cases
 - In Keyword-driven testing, anything is "Data-driven", including functions
-

Keyword-driven testing



Test	1	
Login	mfayad	xyz
VerifyScreen	MainAccount	
Logoff		
Test	2	
Login	dschmidt	123
VerifyError	"Your user account is expired."	
Test	3	
Login	rjohnson	abc
VerifyError	"Unknown user"	

- Keyword-driven testing very often requires an *interpreter* that:
 - Reads the test file
 - Looks up the library function (*systemLogin*) associated with the keyword (*Login*)
 - Executes the function using the data on the line ("*mfayad*," "*xyz*") as arguments
-

Keyword-driven testing

■ Advantages:

- Development of Keywords can begin before the application is ready
- Test cases are easy to create and maintain
- Maintenance is easy: changes in AUT functionality very often require changes in Keywords only
- Keywords can be reused for testing other applications

■ Limitations:

- Require the development of an interpreter which is not an easy task
 - Poor logic control in test cases
-

Test Automation Tools

- Common features of the major test automation tools:
 - Record & Playback capability
 - Data-driven and Keyword-driven testing support
 - Scripts editor (using *standard* or *vendor-like* scripting language)
 - Application GUI Map Objects (recognition and) editor
 - Data Pool editor
 - GUI, DB and file checkpoints
 - Reporting
 - ...
-

Test Automation Tools

- Functional / GUI testing tools:

- *QuickTest Professional* from HP

- *Rational Functional Tester* from IBM

- *SilkTest* from Borland

- *TestPartner* from Compuware

- *Squish* from Froglogic

- *GUIDancer* from Bredex

- *Ranorex Studio* from Ranorex

- ...

Rational. software

COMPUWARE 



Borland

 **Ranorex**

 **froglogic.**



PART 2

GUI Testing & *TAF* – Test Automation Framework

GUI Testing

- Test Automation of GUI-based applications
- The *TAF* framework
 - Viewer
 - Finder

GUI Testing

- GUIs make software easy to use, understand and learn
- GUIs can constitute as much as **60%** of an application's total code
 - Difficult to develop and to test!

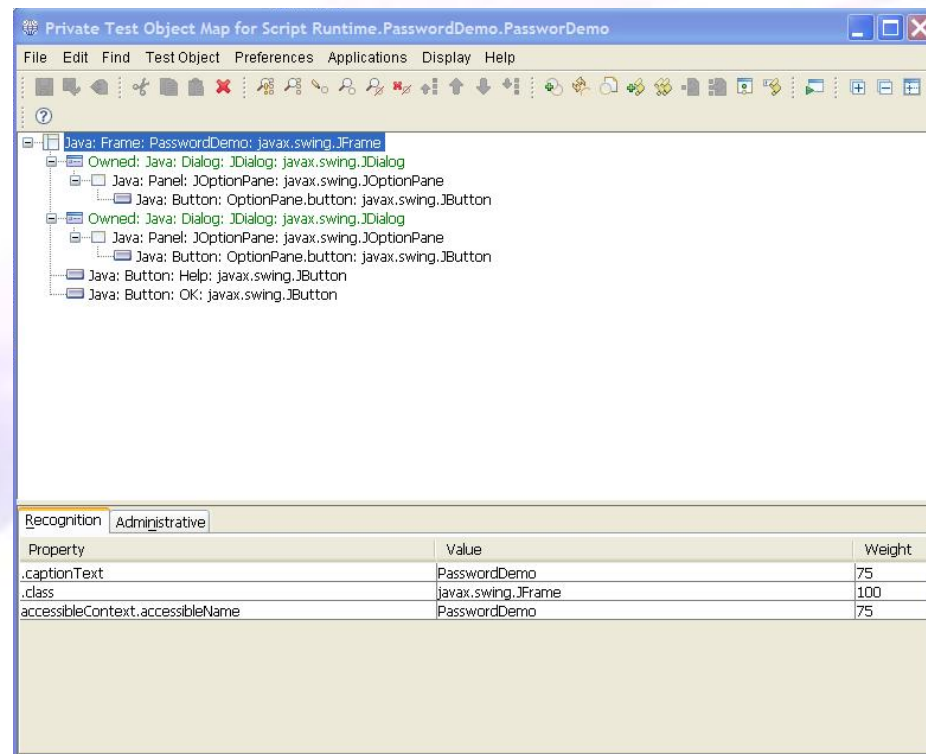


GUI Testing challenges

- GUIs make systems testing more difficult for many reasons:
 - GUI changes break the tests
 - Most early test failures are due to GUI changes
 - May need to wait for GUI 'stabilization'
 - State space and test case explosion
 - Many ways in/out: multiple ways (mouse, keyboard,...) to achieve the same goal
 - Observing visible (and invisible) GUI states/properties
 - Test Objects Map
 - Event-driven nature of GUIs
 - Unsolicited events
-

GUI Maps

- A *GUI Map* describes the *test objects* in the application-under-test
- It is used by the test automation tool to recognize the GUI objects to test
- Each script is associated with a test object map



GUI Maps



- Known limitations:
 - *Static*
 - *A priori*
 - *Need to be updated* in case of GUI changes (...and GUIs change very often!)
 - Unable to develop the automated tests before the availability of the application-under-test
 - ↳ test automation will always *start later* in the software lifecycle
 - ...
-

Beyond GUI Maps

- Design and develop of a new test automation framework that manages the GUI objects to test:
 - *dynamically at runtime*
 - *minimizing maintenance* in case of changes
 - allowing developing the automated tests *early* in the lifecycle

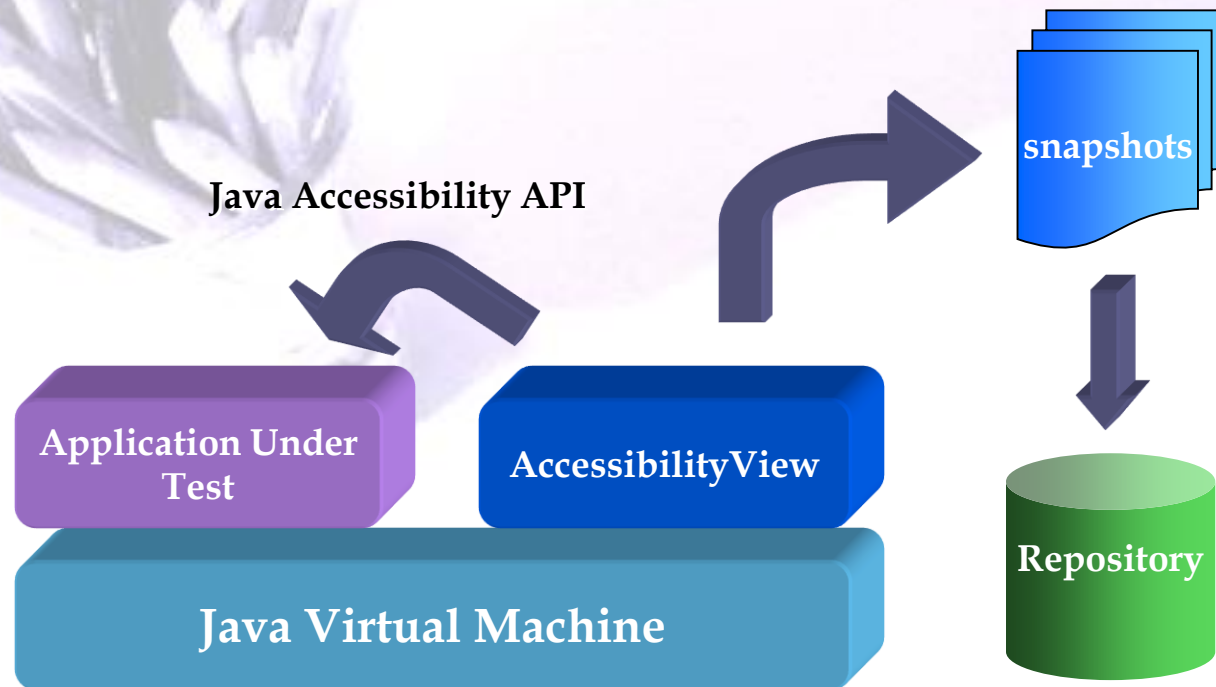


Test Automation Framework

- TAF at runtime during test execution:
 - *takes “snapshots”* of the current user interface
 - ↳ ...and turn them into a XML documents
 - *searches* within snapshots for the GUI objects to test
 - *executes* the test actions on the test objects
-

Viewer

- *Get information* about the GUI objects hierarchy by the Java Virtual Machine where the AUT is running
 - Based on Java Accessibility
- *Build a snapshot* using the given syntax and semantics



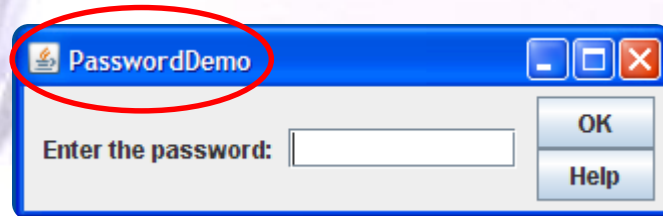
Finder

- Design and develop a test objects *Finder* that:
 - can effectively retrieve the test objects which are *relevant*
 - ↳ while retrieving as few non-relevant objects
 - *without compromising* in speed or performance
 - by supporting *partial or incomplete matching*



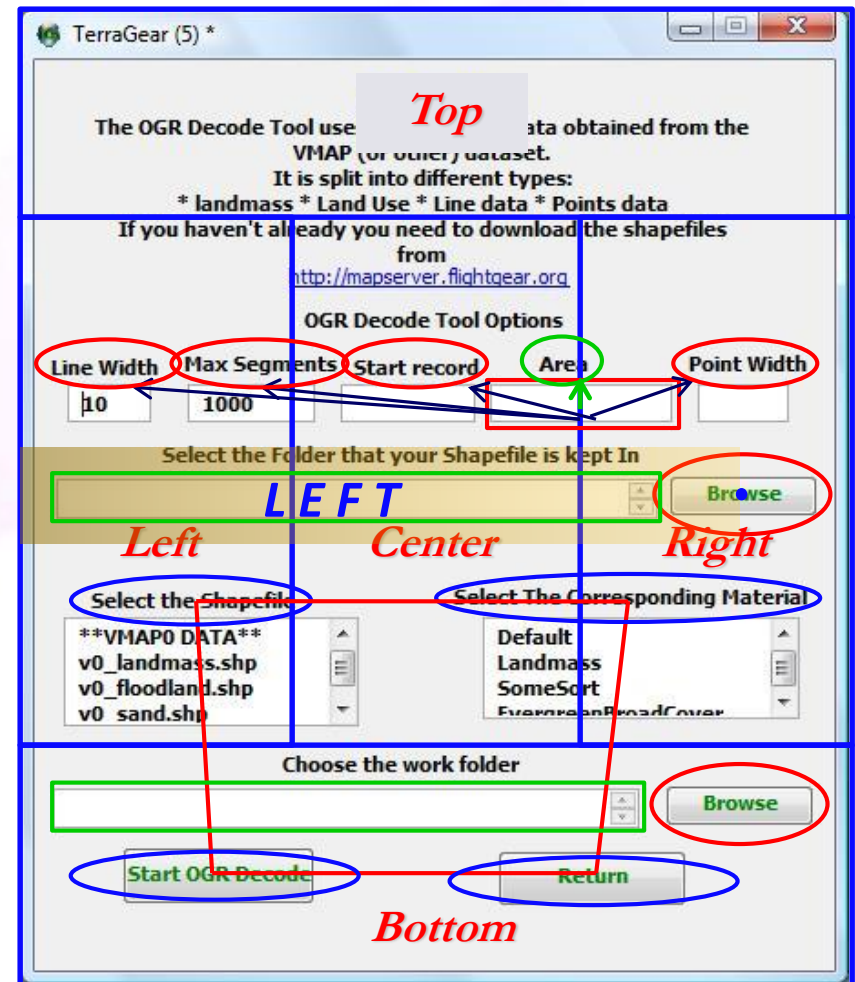
Finder

Exact Match	Allows searching for test object exactly as it is entered	Query: "PasswordDemo"
Case Insensitive Match	Allows finding an exact match, but the match is not case-sensitive	Query: "passworddemo"
Partial Match	Allows finding a match on partial data	Query: "Demo"



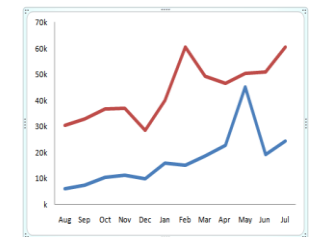
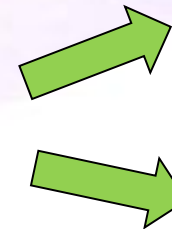
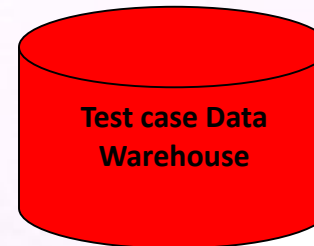
'Zone' Finder

- *Name-Role* strategy known limitations:
 - Impossible to recognize Gui Objects with same *Name* and *Role*
 - Problems linking a *text-area* to the associated *label*
- Find the GUI object to test in an 'area' of the user interface
 - Area of a *polygon*
 - Top, Bottom, Middle, Left, Right...
 - Near to, Up to, Down to, ...

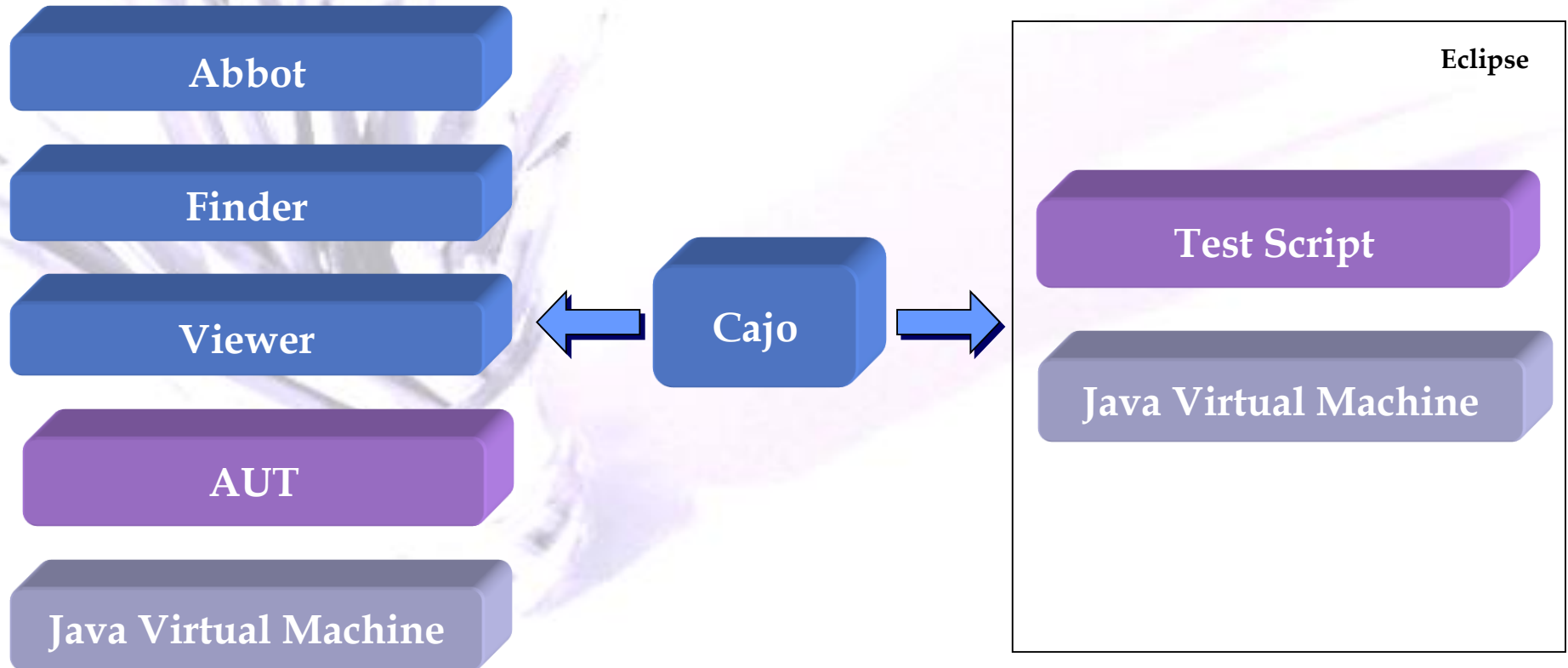


Metrics

- Test case and Test Result data warehouse
- Reporting system to monitor progress of tests and quality
- Intelligent system for:
 - automatic test planning
 - decision supporting
- And in the future...
 - advanced test scheduling
 - link to requirements & bug tracking systems
 - ...



System Architecture





PART 3

Working with ANT and Jenkins

What is Ant

- ❑ Apache **Ant** is a Java library and command-line tool whose mission is to drive processes described in XML build files as targets and extension points dependent upon each other
- ❑ Ant supplies a number of built-in **tasks** allowing to compile, assemble, test and run Java (C , C++ ...) applications
- ❑ More generally, Ant can be used to pilot any type of process which can be described in terms of targets and tasks



Ant & JUnit

Ant integrates with Junit. This allows:

- ❑ executing tests as part of the build process
- ❑ capturing their output
- ❑ generating rich colour enhanced reports on testing.

```
<pathelement location="C:/TAF_1_5/lib/poi-3.7-20101029.jar"/>
<pathelement location="C:/TAF_1_5/lib/jaxen.jar"/>
<pathelement location="C:/TAF_1_5/bin/bootstrap.jar"/>
<pathelement location="C:/TAF_1_5/bin/proxy.jar"/>
<pathelement location="C:/TAF_1_5/lib/jdom.jar"/>
<pathelement location="C:/TAF_1_5/bin/keydrivenTesting.jar"/>
<pathelement location="C:/TAF_1_5/lib/cajo.jar"/>
<pathelement location="C:/TAF_1_5/lib/freemarker.jar"/>
</path>
<target name="init">
  <mkdir dir="bin"/>
  <copy includeemptydirs="false" todir="bin">
    <fileset dir="src">
      <exclude name="**/*.launch"/>
      <exclude name="**/*.java"/>
    </fileset>
  </copy>
</target>
<target name="clean">
  <delete dir="bin"/>
</target>
<target depends="clean" name="cleanall"/>
<target depends="build-subprojects,build-project" name="build"/>
<target name="build-subprojects"/>
<target depends="init" name="build-project">
  <echo message="${ant.project.name}: ${ant.file}"/>
  <javac debug="true" debuglevel="${debuglevel}" destdir="bin" includeantruntime="false">
    <src path="src"/>
    <classpath refid="TAF_Demo.classpath"/>
  </javac>
</target>
<target description="Build all projects which reference this project. Useful to propagate c">
<target name="ButtonTest">
  <mkdir dir="${junit.output.dir}"/>
  <junit fork="yes" printsummary="withOutAndErr">
    <formatter type="xml"/>
    <test name="my.taf.demo.junit.ButtonTest" todir="${junit.output.dir}"/>
    <classpath refid="TAF_Demo.classpath"/>
  </junit>
</target>
<target name="junitreport">
  <junitreport todir="${junit.output.dir}">
    <fileset dir="${junit.output.dir}">
      <include name="TEST-*.xml"/>
    </fileset>
    <report format="frames" todir="${junit.output.dir}"/>
  </junitreport>
</target>
</project>
```

Ant & JUnit Report

App

Home

Packages

Classes

Unit Test Results.

Designed for use with [JUnit](#) and [Ant](#).

Package

Classes

Name	Tests	Errors	Failures	Time(s)	Time Stamp	Host
ButtonTest	1	0	0	6.530	2015-08-06T09:28:26	
CheckboxTest	1	0	0	7.925	2015-08-06T09:28:33	
ComboBoxTest	2	0	0	28.030	2015-08-06T09:28:42	
DialogTest	1	0	0	6.782	2015-08-06T09:29:11	
GuiObjectTest	2	0	0	12.018	2015-08-06T09:29:19	
MenuTest	1	0	0	7.013	2015-08-06T09:29:32	
PasswordTextTest	2	0	0	46.606	2015-08-06T09:29:40	
RadioButtonTest	1	0	0	8.765	2015-08-06T09:30:28	
SpinBoxTest	2	0	0	15.849	2015-08-06T09:30:38	
TableTest	2	0	0	21.301	2015-08-06T09:30:55	
TextTest	6	0	0	122.548	2015-08-06T09:31:17	
ToggleButtonTest	1	0	0	6.712	2015-08-06T09:33:21	
TreeTest	2	0	0	10.833	2015-08-06T09:33:28	

global test results



detailed test results

App

Home

Packages

Classes

Unit Test Results.

Designed for use with [JUnit](#) and [Ant](#).

Class

Name	Tests	Errors	Failures	Time(s)	Time Stamp	Host
TextTest	6	0	0	122.548	2015-08-06T09:31:17	

Tests

Name	Status	Type	Time(s)
selectCopyPaste	Success		15.146
selectCutPaste	Success		14.994
selectDelete	Success		13.739
insertText	Success		12.520
append	Success		22.455
setText	Success		40.980

[Print](#)
[JUnit](#)

What is Jenkins

- ❑ Jenkins is Java open source **continuous integration tool**
- ❑ Jenkins supports SCM tools including CVS, Subversion, Git, etc., and can execute Apache Ant and Apache Maven based projects as well as arbitrary shell scripts and Windows batch commands
- ❑ Builds can be started by various means, including being triggered by commit in a version control system, by scheduling via a cron-like mechanism, by building when other builds have completed, etc.



Jenkins

Jenkins Jobs

Dashboard [Jenkins]

localhost:8080/jenkins/

App Dashboard [Jenkins] Tool Distribution Env

Jenkins

Utenti

Cronologia Build

Configura Jenkins

Credentials

Elenco build

Nessun Build in Coda.

Stato Esecutore Build

1 Inattivo

2 Inattivo

3 Inattivo

4 Inattivo

5 Inattivo

6 Inattivo

7 Inattivo

8 Inattivo

9 Inattivo

10 Inattivo

11 Inattivo

12 Inattivo

13 Inattivo

14 Inattivo

15 Inattivo

16 Inattivo

17 Inattivo

18 Inattivo

19 Inattivo

20 Inattivo

21 Inattivo

22 Inattivo

23 Inattivo

24 Inattivo

25 Inattivo

26 Inattivo

27 Inattivo

28 Inattivo

29 Inattivo

ABILITA IL REFRESH AUTOMATICO

S	W	Nome ↓	Ultimo Successo	Ultimo Fallimento	Durata Ultimo	
		01_Memento_Configuration	2 days 13 hr - #219	N.D.	0.5 sec	
		02_CheckoutOS	2 days 13 hr - #193	N.D.	2 min 14 sec	
		03_build_rom	2 days 13 hr - #140	11 mo - #3	1 min 3 sec	
		04_build_eep	2 days 13 hr - #150	1 yr 0 mo - #2	16 sec	
		05_build_rom_cap2c	2 days 13 hr - #138	N.D.	8.3 sec	
		06_BuildNative	2 days 13 hr - #154	3 days 21 hr - #151	5 min 38 sec	
		07_CopyMaskROMImageBrut	2 days 13 hr - #139	8 mo 11 days - #49	26 sec	
		08_FlashCard	2 days 13 hr - #159	1 mo 25 days - #137	7.1 sec	
		09_A_IF_TESTSUITE1_EXIST	2 days 13 hr - #31	1 mo 10 days - #7	0.52 sec	
		09_CheckOut_Build_TestSuite_1	9 days 13 hr - #55	N.D.	17 sec	
		10_ExecuteTestSuite_1	9 days 13 hr - #146	N.D.	15 sec	
		11_A_IF_TESTSUITE2_EXIST	2 days 13 hr - #21	1 mo 9 days - #8	0.48 sec	
		11_CheckOut_Build_TestSuite_2	9 days 13 hr - #16	N.D.	15 sec	
		12_ExecuteTestSuite_2	9 days 13 hr - #27	N.D.	28 sec	
		13_A_IF_TESTSUITE3_EXIST	2 days 13 hr - #16	1 mo 9 days - #1	0.5 sec	
		13_CheckOut_Build_TestSuite_3	9 days 13 hr - #16	N.D.	30 sec	
		14_ExecuteTestSuite_3	9 days 13 hr - #31	N.D.	27 sec	
		15_IF_RUN_GPUICC	2 days 13 hr - #19	N.D.	0.51 sec	
		16_CARD_CONFIG_GP_UICC	2 days 19 hr - #34	2 days 13 hr - #35	2 min 41 sec	
		17_RUN_GP_UICC	2 days 19 hr - #21	N.D.	2 hr 22 min	

Icona: [S](#) [M](#) [L](#)

Leggenda

RSS Per Tutti

RSS Solo Fallimenti

RSS Solo Per Le Ultime Build

Jenkins & JUnit

The screenshot shows the Jenkins web interface. At the top, the Jenkins logo and name are on the left, and a search bar is on the right. Below the header, the breadcrumb navigation shows 'Jenkins > MaveryxTestJob > #5 > Test Results'. On the left sidebar, there are links for 'Back to Project', 'Status', 'Changes', 'Console Output', 'Edit Build Information', 'History', and 'Test Result'. The main content area is titled 'Test Result :'. Below this, it says '0 failures' and shows a blue progress bar. To the right of the progress bar, it indicates '58 tests', 'Took 8 min 50 sec.', and a link to 'add description'. Below the progress bar, the section 'All Tests' contains a table with the following data:

Class	Duration	Fail	(diff) Skip	(diff) Pass	(diff) Total	(diff)
ButtonTest	7.8 sec	0	0	2 +2	2 +2	
CheckBoxTest	10 sec	0	0	2 +2	2 +2	
ComboBoxTest	50 sec	0	0	6 +6	6 +6	
DialogTest	7.7 sec	0	0	2 +2	2 +2	
GuiObjectTest	18 sec	0	0	4 +4	4 +4	
MenuTest	8.6 sec	0	0	2 +2	2 +2	
PasswordTextTest	1 min 27 sec	0	0	10 +10	10 +10	
RadioButtonTest	12 sec	0	0	2 +2	2 +2	
SpinBoxTest	26 sec	0	0	4 +4	4 +4	
TableTest	36 sec	0	0	6 +6	6 +6	
TextTest	3 min 58 sec	0	0	12 +12	12 +12	
ToggleButtonTest	7.9 sec	0	0	2 +2	2 +2	
TreeTest	16 sec	0	0	4 +4	4 +4	