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

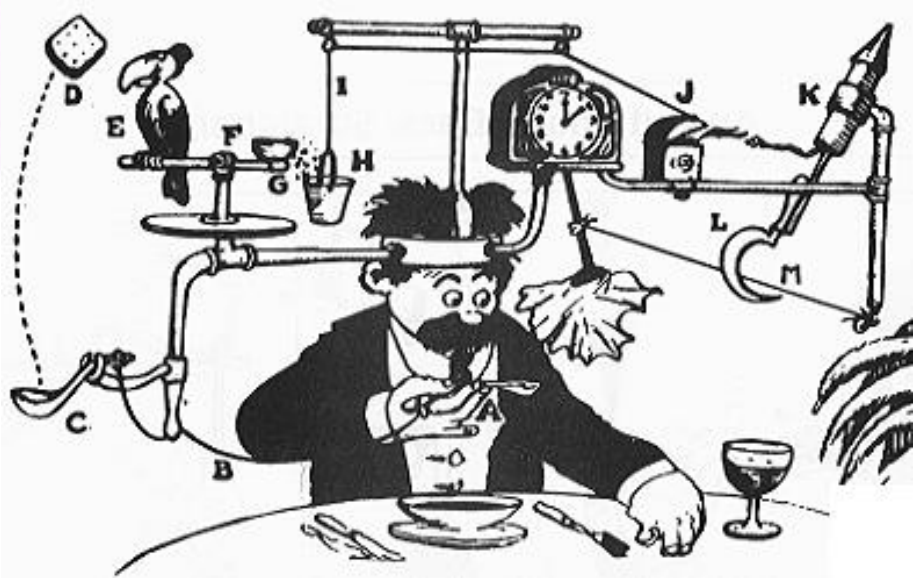
Software Test Automation: foundations

Software Test Automation

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

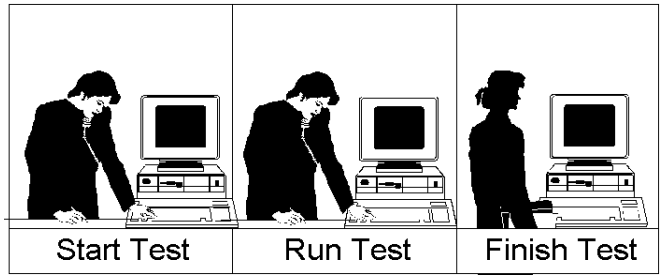
Test Automation

- ***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
  - …



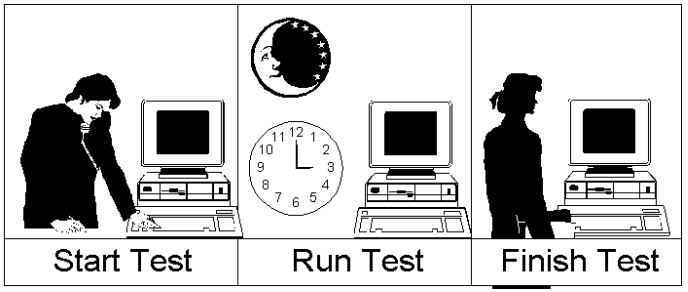
Why Test Automation

- ***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

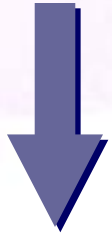
- Many bugs are found because testers have intuition, while machines can't see suspicious behavior and investigate it
- Manual tests find more defects than automated tests: 85% vs. 15%
- Tests are more complex at the *User Interface* (UI)
  - UIs very often change frequently, which increases maintenance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

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
  - …

PART 2

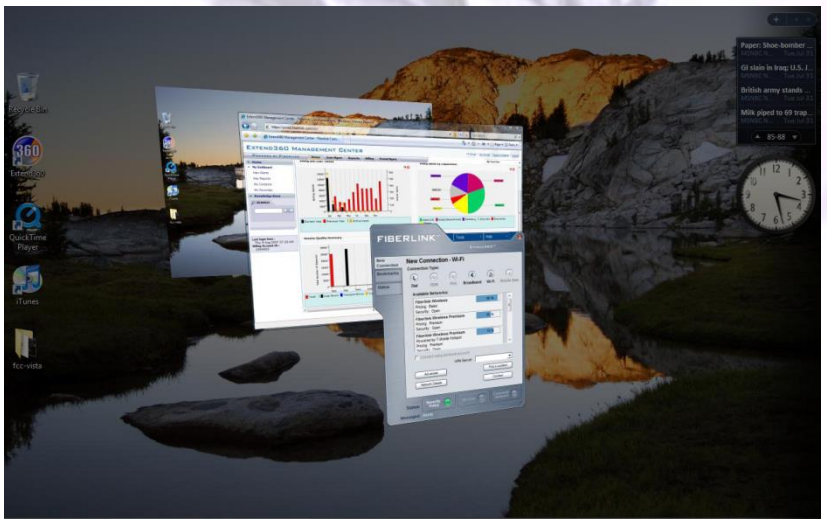
GUI Testing & *TAF* – Test Automation Framework

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!

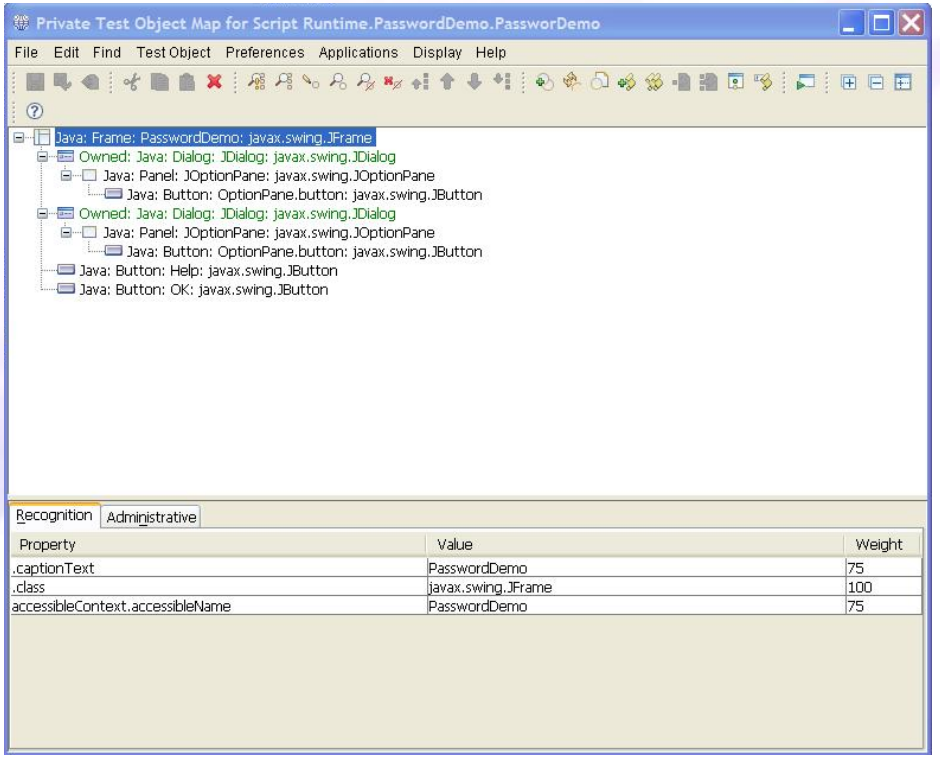


GUI Testing challenges

- 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

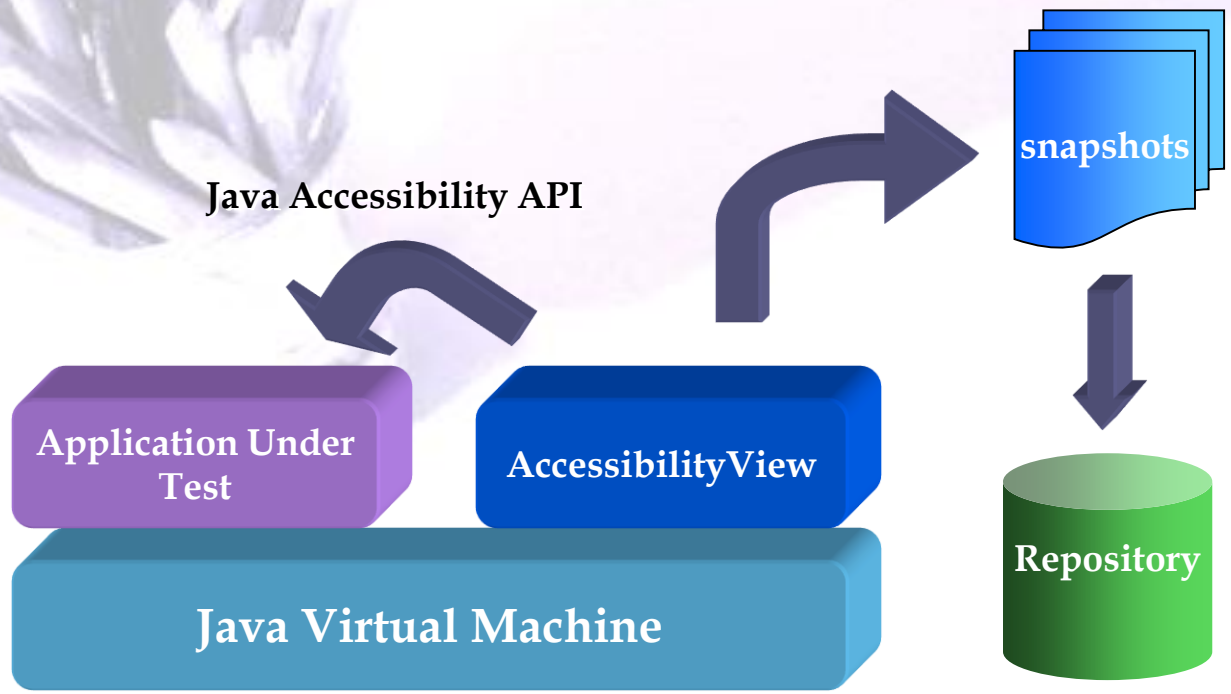
- Design and develop of a new test automation framework that manages the GUI objects to test:
  - ***dynamically at runtime***
  - ***minimizing maintenance*** in case of changes
  - allowing developing the automated tests ***early*** in the lifecycle

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

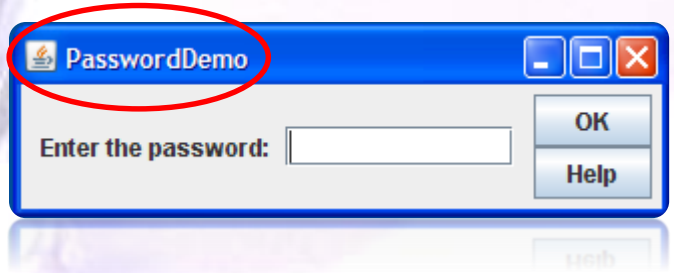


Finder

- 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***

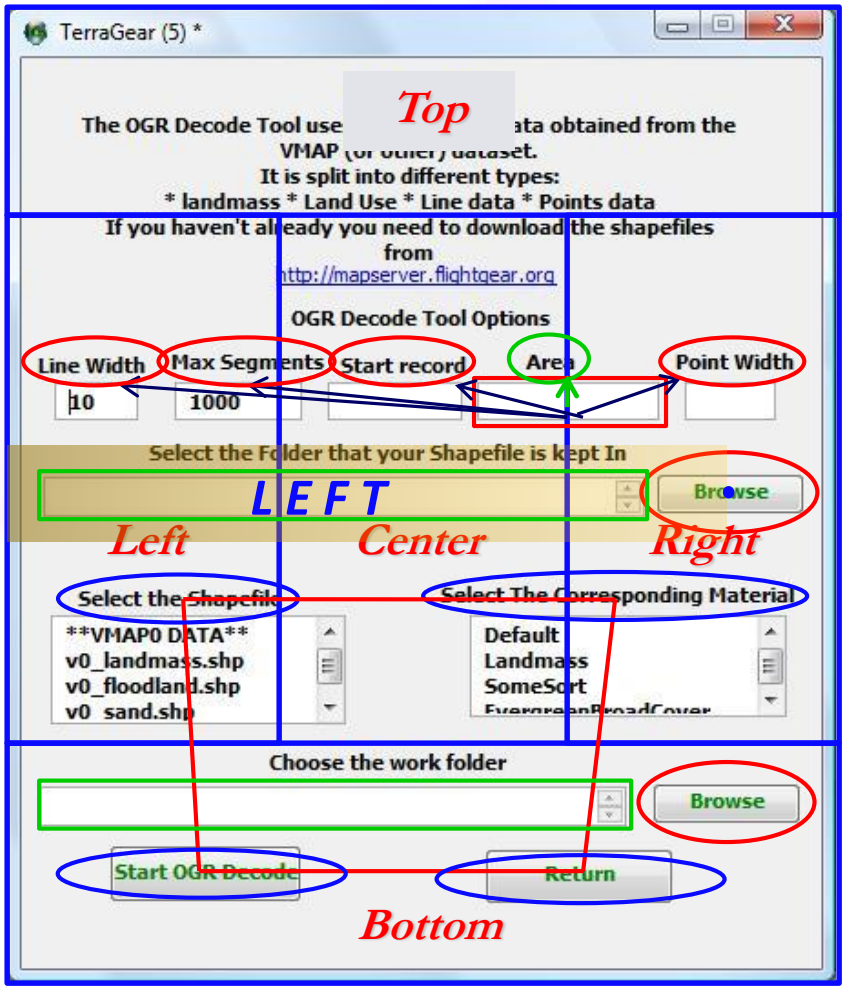
Finder

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”



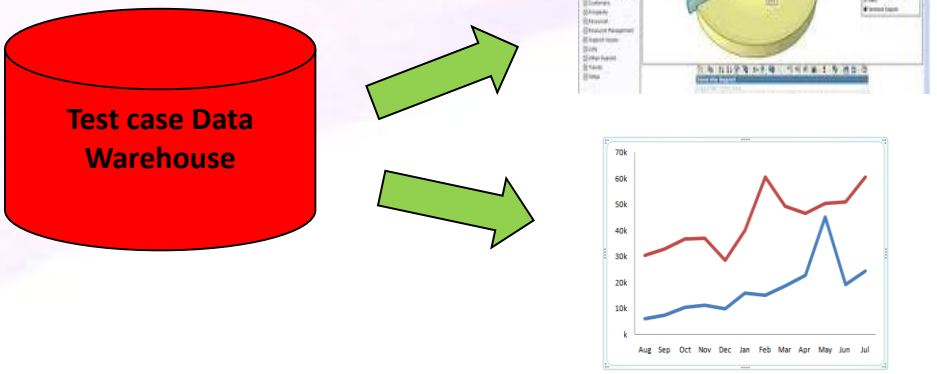
‘Zone’ Finder

- *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, …

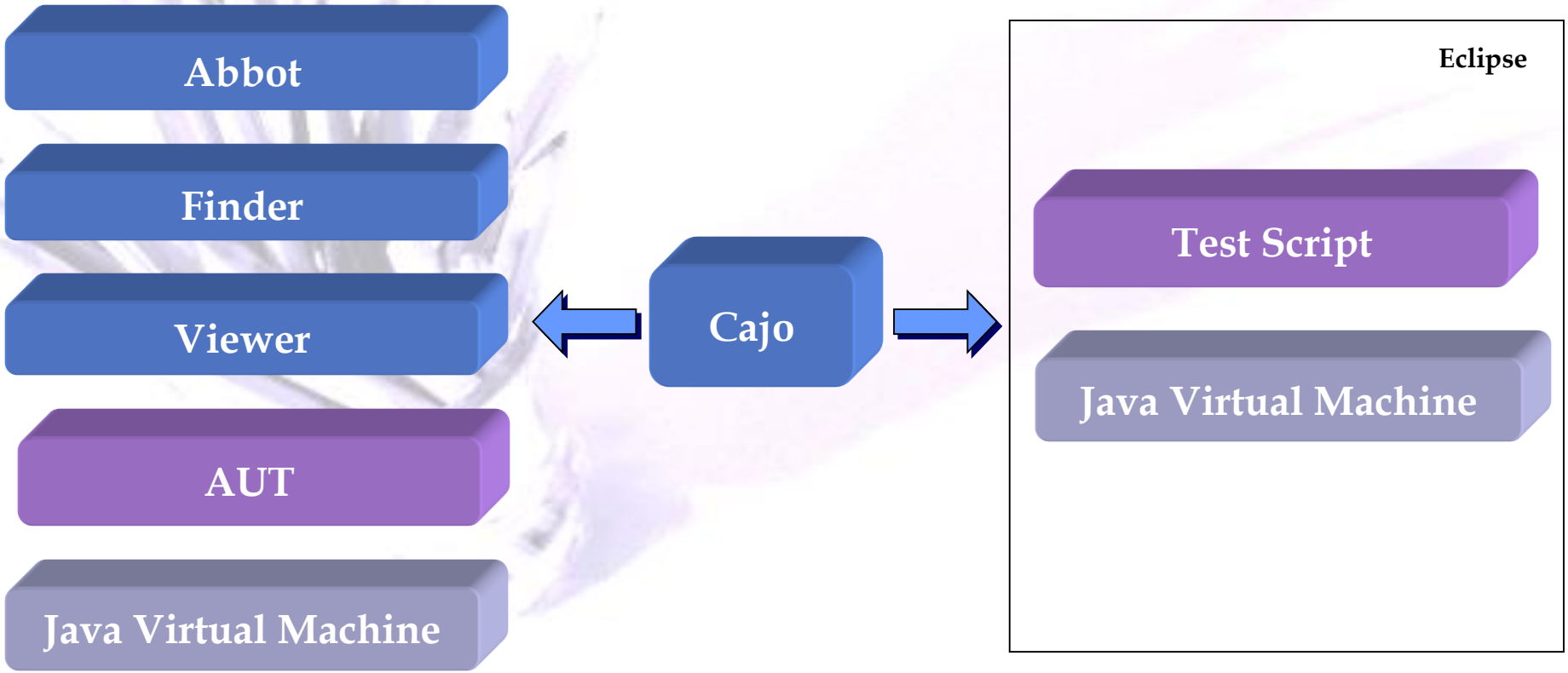


Metrics

- 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

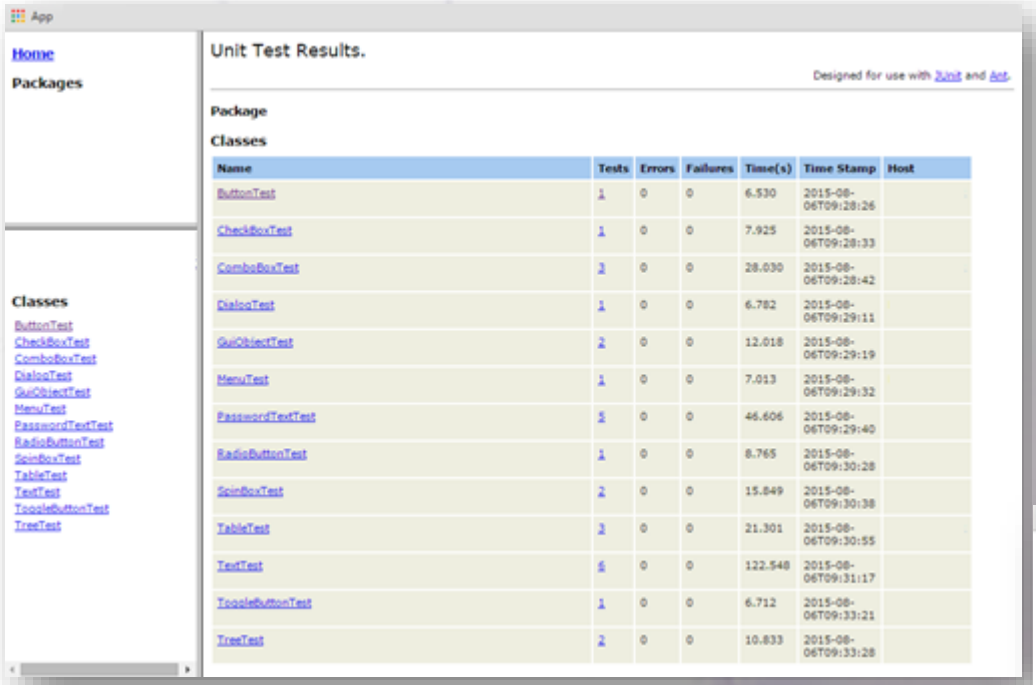
Ant & JUnit

Ant integrates with **Junit**. This allows:

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

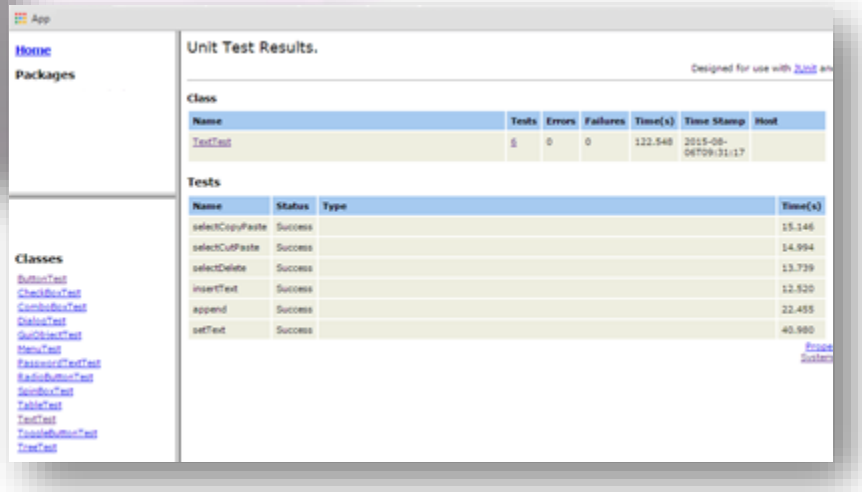
```
        <pathelement location="C:/TAF_1_5/lib/poi-3.7-20101029.jar"/>
        <pathelement location="C:/TAF_1_5/lib/jaxen.jar"/>
        <pathelement location="C:/TAF_1_5/bin/bootstrap.jar"/>
        <pathelement location="C:/TAF_1_5/bin/proxy.jar"/>
        <pathelement location="C:/TAF_1_5/lib/jdom.jar"/>
        <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



global test results

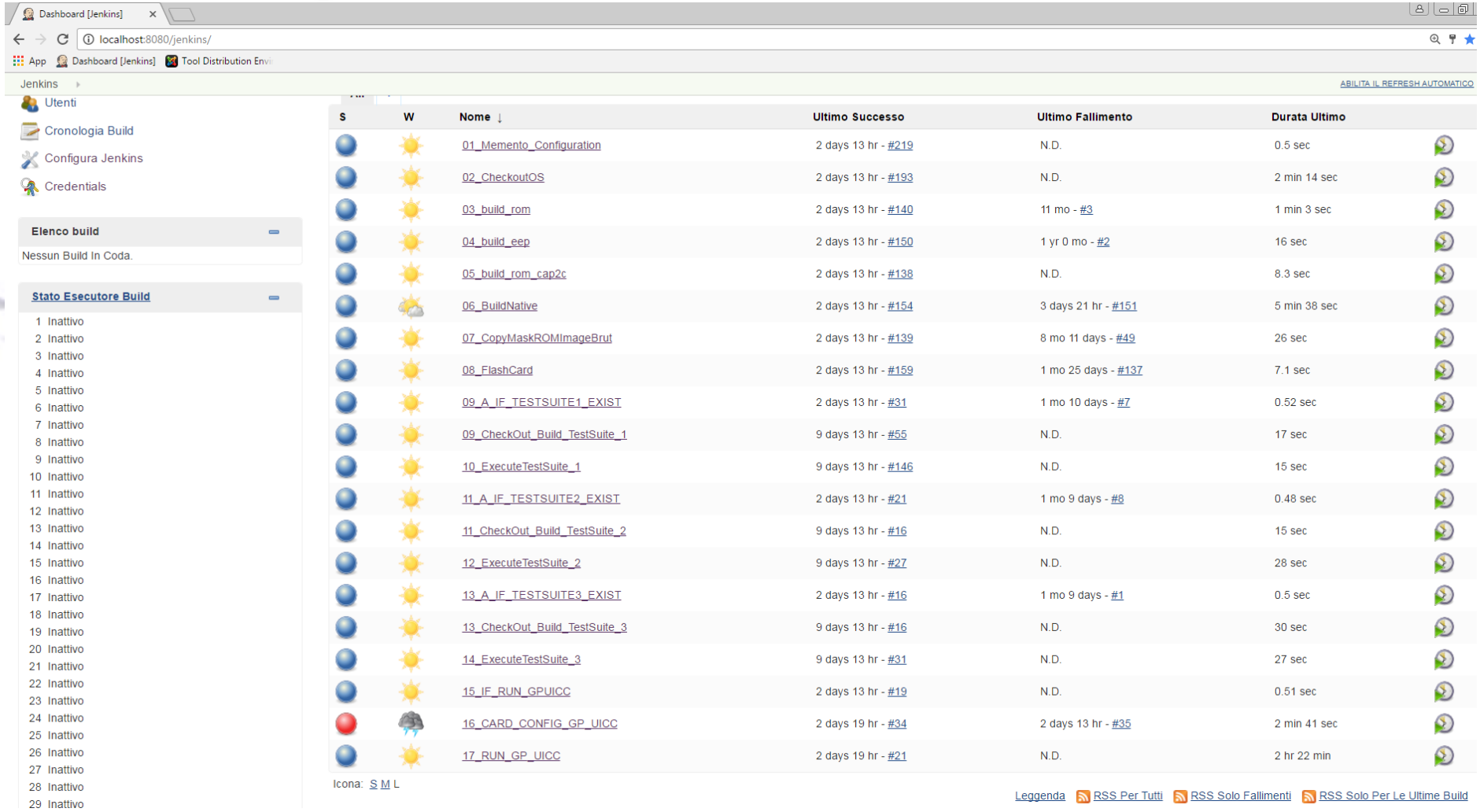
detailed test results



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 Jobs



Jenkins & JUnit