

CURS 11-12.

TESTARE AUTOMATĂ

Verificare, validare și testare automată
[09 Noiembrie 2019]

Lector dr. Camelia Chisăliță-Crețu
Universitatea Babeș-Bolyai, NTT Data
Programul Postuniversitar de Pregătire și Formare Profesională în Informatică

Conținut

- **Testare automată**
 - Testare manuală. Definiție. Caracteristici
 - Testare automată. Definiție. Caracteristici. Motivație. Avantaje
 - Testare manuală vs Testare automată
 - De ce ? Când ? Ce ? Cum ?
 - Framework de testare automată. Definiție. Clasificare
 - Procesul de automatizare. Etape
 - Principii generale de automatizare
 - Mituri
 - Concluzii
- **Tool-uri de testare automată**
 - Prezentare și clasificare
 - Criterii de alegere a unui tool de testare automată
 - Tool-uri de testare automată
- **Test Automation Demo**
 - Serenity BDD. Tipuri de proiecte
 - Proiect Demo
 - Crearea proiectului
 - Structura proiectului
 - Componenta Pages. Componenta Steps. Componenta Tests
 - Cerință. Caz de testare
- **Întrebări pentru examen**

TESTARE AUTOMATĂ

Testare manuală. Definiție. Caracteristici

Testare automată. Definiție. Caracteristici. Motivație. Avantaje

Testare manuală vs Testare automată. De ce ? Când ? Ce ? Cum ?

Framework de testare automată. Definiție. Clasificare

Procesul de automatizare. Etape

Principii generale de automatizare

Mituri. Concluzii

Testare manuală. Definiție. Caracteristici

- **testare manuală** (*engl. manual testing*):
 - tip de testare în care testerul execută manual și cu atenție cazurile de testare;
- **caracteristici:**
 - primul tip de testare aplicat unui produs soft;
 - tip de testare primitiv, în care testerul joacă rolul utilizatorului final;
 - nu necesită cunoștințe legate de testare, i.e., instrumente de testare;
 - tip de testare folosit înainte de a automatiza procesul de testare, i.e., testarea automată;
 - necesită un efort de execuție mai mare în comparație cu testarea automată, dar este necesară pentru a stabili dacă anumite teste se vor automatiza sau nu;
 - aplicabilitate:
 - toate nivelurile de testare, i.e., testare unitară, testare de integrare, testate de sistem (testare funcțională, **anumite tipuri de testare non-funcțională**), testare de acceptare.

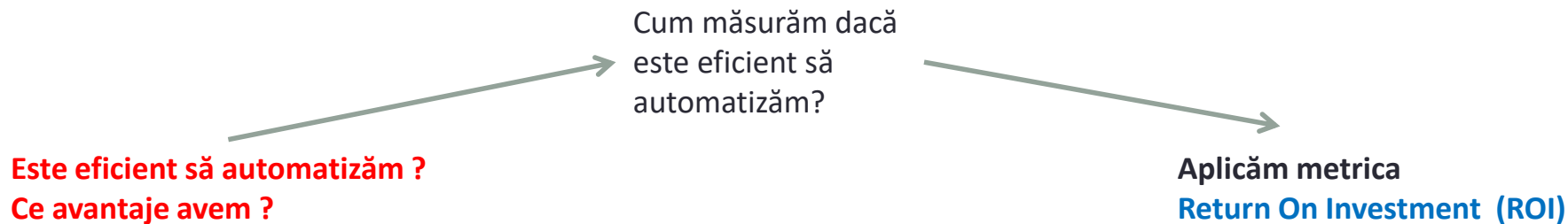
Testare automată. Definiție. Caracteristici

- **testare automată** (*engl. automated testing*):
 - tip de testare în care se utilizează un instrument (*engl. tool*) pentru execuția testelor;
- **caracteristici:**
 - **nu necesită intervenția factorului uman la execuție;**
 - necesită cunoștințe legate de testare, e.g., instrumente de testare, limbaje de programare, pentru proiectarea cazurilor de testare;
 - **necesită resurse suplimentare pe lângă factorul uman;**
- **caracteristicile tool-urilor care realizează testare automată:**
 - permit transmiterea **datelor de intrare** către produsul testat;
 - determină **rezultatul** testării, i.e, **passed** sau **failed**;
 - **generează rapoarte** detaliate referitoare la teste și pașii executați;
 - pot dispune de facilități capture/re-play pentru analiza ulterioară rulării unui test.

Testare automată. Motivație

- există teste care nu se pot rula manual;
- timp redus de dezvoltare a produsului soft (*engl.* **Time-To-Market, TTM**);
- comportamentul uman poate genera erori de utilizare;
- permite economisirea de timp și bani, i.e., **Return On Investment**.

Testare automată. ROI (1)



$$\text{ROI} = \frac{\text{Return}}{\text{Investment}} = \frac{\text{Beneficiu}}{\text{Cost sau Efort}} = \frac{\text{Numărul de teste executate}}{\text{Timp necesar pentru automatizare}}$$

Beneficiu – numărul de teste executate prin automatizare;

Efort – timpul necesar pentru realizarea automatizării (proiectarea testelor, rularea testelor, întreținerea testelor)

Testare automată. ROI (2)

- E.g.: Rulăm 5 teste folosind testare manuală; timpul necesar execuției tuturor testelor este 5 minute;
 - Timpul necesar automatizării suitei de teste (TS) este 50 minute;
 - Întrebare: *Este eficient să automatizăm suita TS?*
 - Este eficient (are rost) să automatizăm testarea pentru suita TS, dacă:
 - avem nevoie să executăm TS o singură dată ?
 - $\Rightarrow \text{ROI} = 5 / 50 = 0,1$;
 - avem nevoie să executăm TS de 10 ori ?
 - $\Rightarrow \text{ROI} = 5 * 10 / (50 + \alpha)$;
 - α – mic \Rightarrow ROI are o valoare mare, supraunitară \Rightarrow automatizarea este eficientă;
 - α – mare, i.e., întreținerea este costisitoare \Rightarrow ROI are o valoare mică, subunitară \Rightarrow automatizarea nu este eficientă;

Posibilitatea de a maximiza metrica ROI reprezintă un indicator al eficienței automatizării.

Testare automată. ROI (3)

- E.g.: Efortul de automatizare este de 10 ori mai mare decât testarea manuală a suitei de teste.
 - Întrebare: *Voi automatiza testarea?*
 - *Depinde...*
 - Întrebare: *Voi rula testele de cel puțin 10 ori?*
 - Dacă da (min. 50 ori) ==> automatizarea testelor este utilă (necesară, eficientă);
 - Dacă nu (mai puțin de 50 ori) ==> automatizarea testelor este inutilă, nu își atinge obiectivul;

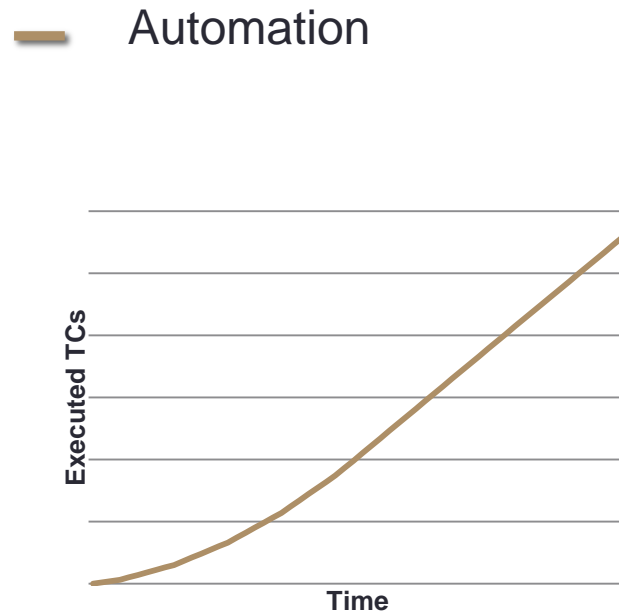
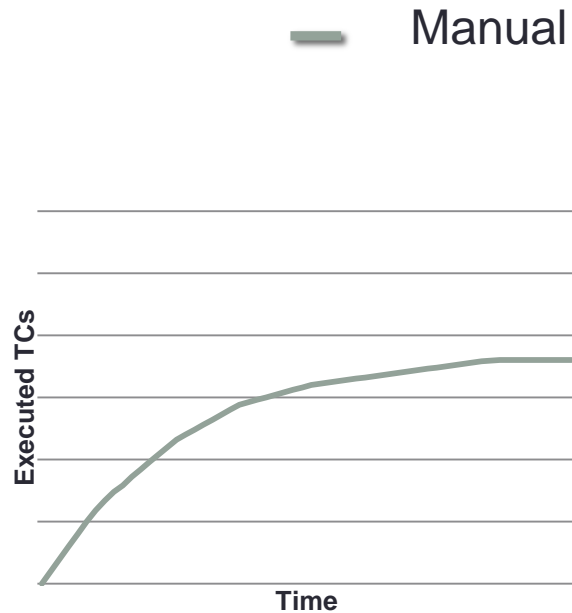
Posibilitatea de a maximiza metrica ROI reprezintă un indicator al eficienței automatizării.

Testare automată. Avantaje

- viteză de execuție mai mare:
 - cu până la 70% mai rapidă față de testarea manuală;
 - acoperire mai mare a codului testat;
- factorul uman nu este necesar pe durata execuției testelor;
- crește productivitatea în cadrul echipei de testare, i.e., testele pot fi rulate noaptea;
- crește nivelul de acuratețe a procesului de testare;
- scripturile de testare se pot reutiliza;
- scade monotonia în desfășurarea activității de testare;
- crește moralul echipei de dezvoltare;



Testare manuală vs Testare automată



Testare manuală vs Testare automată. De ce? Când?

Testare manuală

- feedback imediat;
- proiecte de mici dimensiuni;
- interfața aplicației (UI) este instabilă (poate suferi modificări frecvente);

Testare automată

- execuție rapidă;
- proiecte de mari dimensiuni;
- acoperire cu teste mai bună:
 - echipa de testare are mai mult timp pentru testare exploratorie sau testare bazată pe risc (*engl. risk-based testing*);
- teste reutilizabile:
 - testele se pot rula pe diferite platforme sau medii (environments);
- timp de dezvoltare mai mic:
 - ciclul de testare este redus ==> TTM redus;

Testare manuală vs Testare automată. Ce testăm?

Testare manuală

- teste care trebuie rulate rar;
- teste care corespund unor părți ale aplicației care se schimbă frecvent, e.g., interfața este instabilă ==> efortul de automatizare ar fi ridicat;
- teste pentru evaluarea utilizabilității;

Testare automată

- teste care se execută frecvent, e.g., **testare de regresie**;
- **testare non-funcțională**, e.g., performance, volume, load, stress;
- **data driven testing**, i.e., rularea aceluiași scenariu de testare cu date de test diferite;
- teste care se rulează rar, dar care:
 - *nu pot fi rulate manual*;
 - *automatizarea nu necesită un efort mai mare decât testarea manuală*;

Testare manuală vs Testare automată. Cum testăm?

Testare manuală

- Testare exploratorie;
- Proiectarea testelor;
- Execuția testelor;
- Se continuă testarea exploratorie;
-
-
-

Testare automată

- Alegerea testelor care se automatizează;
- Alegerea tool-urilor de automatizare;
- Automatizarea testelor folosind tool-uri;
- Execuția testelor;
- Analiza rezultatelor;
- Raportarea rezultatelor testării;
- Raportarea bug-urilor;
- Întreținerea testelor;

Framework de testare automată. Definiție. Clasificare

- **framework de testare automată:**
 - definește cadrul general în care se va efectua/realiza testarea automată;
 - cuprinde principiile de automatizare aplicate, conceptele de testare și tool-urile care vor fi folosite la automatizare;
- **clasificare:**
 - Data Driven Automation Framework;
 - Keyword Driven Automation Framework;
 - Modular Automation Framework;
 - Hybrid Automation Framework.

Testare automată. Procesul de automatizare (1)

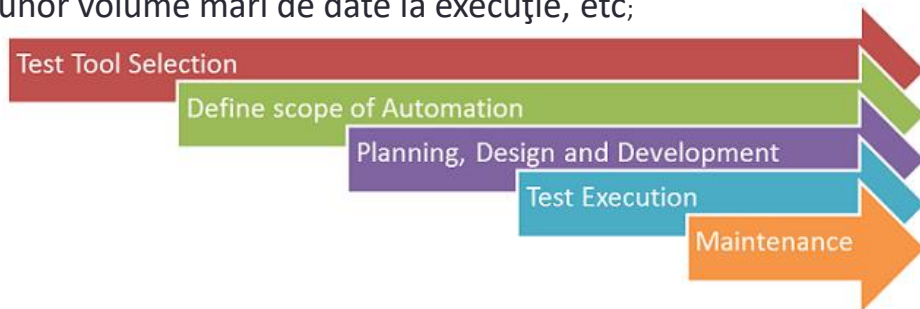
- **etapele procesului de automatizare a testării:**

1. **Alegerea instrumentului de testare automată:**

- depinde de *tehnologia folosită la dezvoltarea produsului soft* ==> trebuie să se verifice mai întâi dacă un anumit tool poate fi folosit pentru automatizarea testării unui produs soft particular;
- depinde de *echipa de testare* ==> trebuie ca echipa să cunoască tool-ul ales, i.e., să aibă experiență în utilizarea tool-ului;

2. **Identificarea caracteristicilor sau funcționalităților produsului soft testat pentru care se va aplica automatizarea:**

- se aplică diferite criterii pentru a motiva automatizarea, care pot face referire la: funcționalități importante, testarea de regresie, utilizarea unor volume mari de date la execuție, etc;

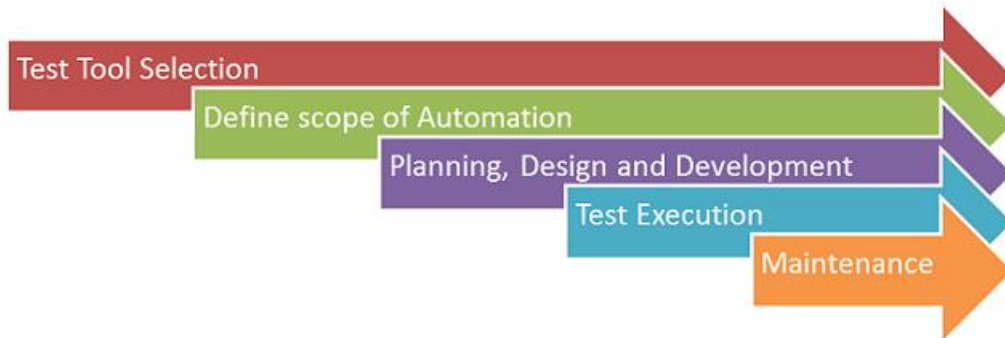


Testare automată. Procesul de automatizare (2)

- **etapele procesului de automatizare a testării:**

- 3. **Planificare, proiectare și implementare:**

- se stabilește strategia de automatizare și planul de automatizare din care fac parte:
 - tool-urile folosite;
 - framework-uri de testare și modul în care caracteristicile permit atingerea obiectivelor de automatizare;
 - pregătirea testelor;
 - prioritizarea execuției testelor;
 - documentele care vor fi obținute.

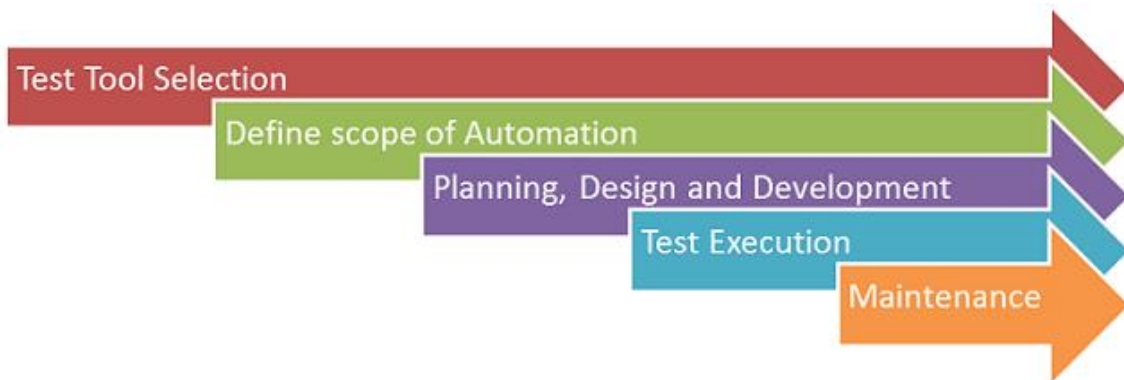


Testare automată. Procesul de automatizare (3)

- **etapele procesului de automatizare a testării:**

- 4. **Execuția testelor:**

- execuția propriu-zisă a testelor și ulterior generarea rapoartelor detaliate referitoare la testele rulate;
 - datele de intrare sunt preluate din fișiere auxiliare, e.g., .cvs, .xls;
 - tipuri de execuție:
 - **directă** – se folosește direct tool-ul de automatizare;
 - **indirectă** – se folosește tool-ul de automatizare prin intermediul tool-ului de management al testelor;

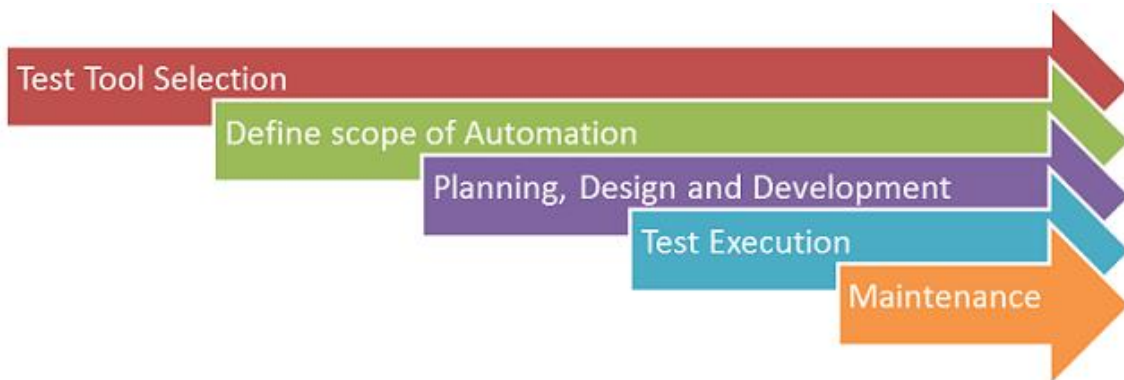


Testare automată. Procesul de automatizare (4)

- **etapele procesului de automatizare a testării:**

- 5. **Întreținerea testelor:**

- permite adăugarea de teste noi atunci când se adaugă funcționalități noi;
 - modificarea/adaptarea testelor existente;
 - etapa este necesară pentru creșterea eficienței automatizării.



Testare automată. Principii generale de aplicare (1)

1. **Stabilirea elementelor pentru care se aplică automatizarea testării trebuie să fie cunoscute înainte de automatizarea propriu-zisă:**
 - așteptările referitoare la automatizare trebuie să fie realiste;
2. **Alegerea tool-ului de automatizare adecvat:**
 - instrumentul de automatizare se alege pe baza elementelor ce urmează a fi testate și a obiectivelor de automatizare și nu pe baza popularității tool-ului;
 - se recomandă realizarea unui model (*engl.* **proof of concept, POC**) prin care se determină măsura în care tool-ul are caracteristicile necesare astfel încât obiectivele automatizării să fie atinse sau nu;
3. **Se alege un framework adecvat:**
 - e.g., data driven, keyword driven, modular, hybrid automation framework.

Testare automată. Principii generale de aplicare (2)

4. Scripturile de testare sunt standardizate:

- toate scripturile să fie uniform elaborate, folosind comentarii și indentare de cod;
- tratarea excepțiilor trebuie să fie adecvată, indicând modul în care produsul soft gestionează comportamentul neprevăzut sau situațiile excepționale;
- mesajele pentru utilizatori trebuie standarizate pentru a putea fi interpretate corect la testare.

Testare automată. Principii generale de aplicare (3)

5. Se folosesc metrice:

- **Succesul aplicării automatizării nu se determină exclusiv prin compararea efortului de automatizare cu timpul necesar realizării testării manuale.**
- se aplică metrice pentru determinarea:
 - procentului de bug-uri identificate;
 - timpului necesar pentru automatizare pentru fiecare versiune a produsului soft;
 - timpului minimal pentru emiterea unei versiuni noi;
 - îmbunătățirilor la nivel de productivitate;
 - indicelui de satisfacție al clientului.

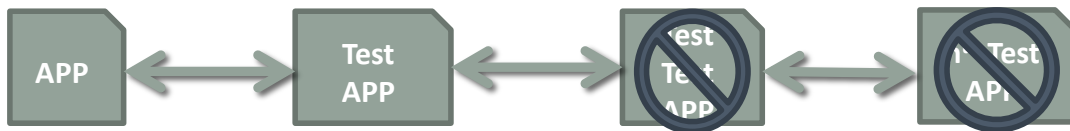
Testare automată. Principii generale de aplicare (4)

6. **Automatizarea testării începe devreme în cadrul testării produsului soft:**
 - unul din marile avantaje ale automatizării este posibilitatea de a rula același test de mai multe ori pe durata de dezvoltare a produsului soft;
7. **Automatizarea nu înseamnă doar înregistrarea comportamentului aplicației la execuție și vizualizarea ulterioară, i.e., capture/re-play, sau doar execuția unui test:**
 - automatizarea presupune existența unui proces complet și a unui mediu pentru **crearea, documentarea, gestionarea, întreținerea și execuția testelor**, precum și **raportarea rezultatelor**;
 - **gestionarea mediului de testare**;
 - la alegerea unui tool se ia în considerare și capacitatea de a realiza o întreținere eficientă a testelor, un aspect care la tool-urile care permit doar capture/re-play ar putea fi insuficient;

Testare automată. Principii generale de aplicare (5)

8. Nu se dezvoltă un program care testează produsul soft livrat:

- rezultatul va fi în acest caz un program asemănător cu produsul testat;
- acesta trebuie să fie testat, ceea ce ar însemna să se dezvolte alt program de test care, la rândul lui va trebui să fie testat;
- **De ce ar fi necesar să se dezvolte două produse soft complete, când doar unul dintre ele va fi livrat la client?**
- dezvoltatorul software nu are resurse (timp, bani) să dezvolte două produse soft distincte dar similare;



Testare automată. Principii generale de aplicare (6)

9. Codul de testare nu se clonează:

- se folosesc bibliotecile de testare existente și codul de testare deja dezvoltat;

10. Codul de testare automată trebuie să fie simplu:

- se reduce complexitatea dacă este cazul;

11. Datele de test nu se păstrează în cadrul testului:

- se folosesc fișiere cu date de intrare, e.g., .csv, .xls;
- nu se fac presupuneri referitoare la datele existente;

12. Nu se proiectează cazuri de testare care:

- nu se pot executa independent;
- nu se pot rula de mai multe ori;
- nu se pot rula în paralel.

Testare automată. Mituri

1. Testarea automată este mai bună decât testarea manuală.
2. Testarea automată poate să înlocuiască complet testarea manuală.
3. Testarea automată are un procent de identificare a bug-urilor mai mare în comparație cu testarea manuală.
4. Dacă se automatizează întreaga activitate de testare, testerii vor deveni inutili.
5. Clientul nu apreciază automatizarea și nu este dispus să plătească pentru realizarea ei.
6. Testarea automată determină creșterea calității produsului soft.

Testare automată. Concluzii



Există cazuri de testare care nu pot fi automatizate.
Există cazuri de testare care nu pot fi rulate manual.



Dacă un test se poate automatiza nu este obligatoriu să se automatizeze.



Tool-urile de automatizare nu sunt instrumente magice, DAR utilizate corespunzător pot face minuni !

“Automation should not be used as a replacement of manual testing, but rather an enhancement of it.” **[James Bach]**

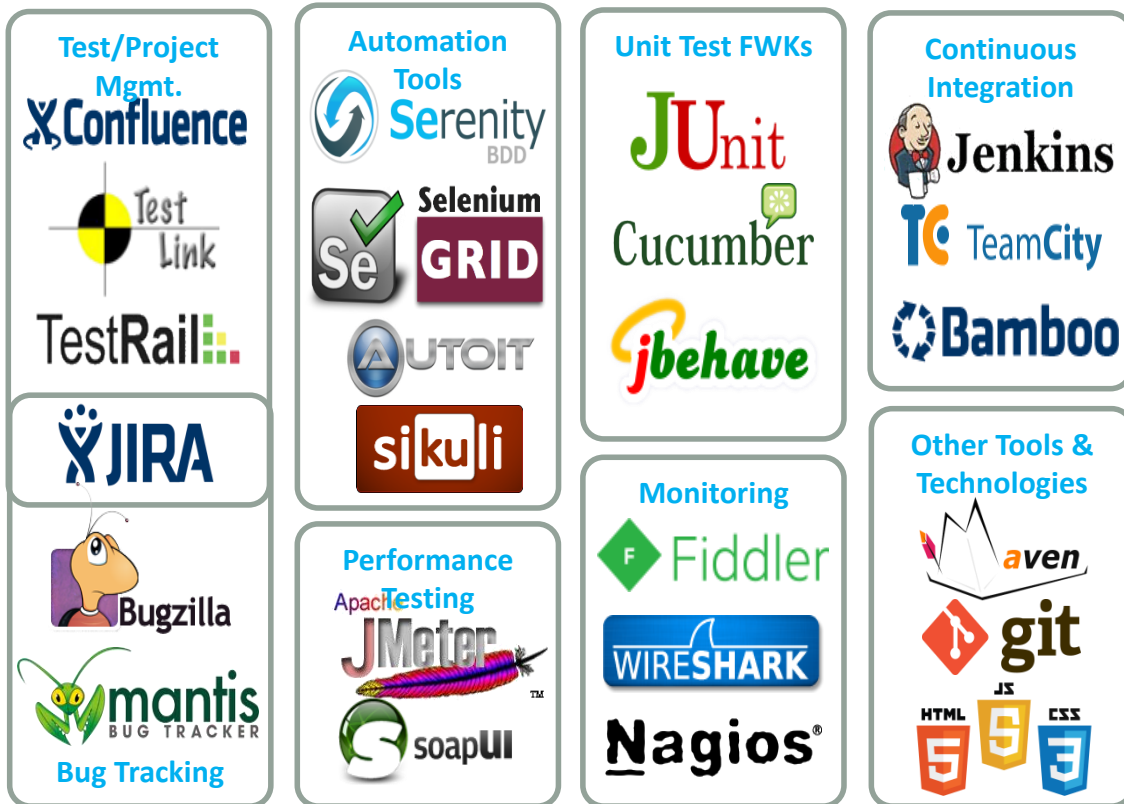
TOOL-URI DE TESTARE AUTOMATĂ

Prezentare și clasificare

Criterii de alegere a unui tool de testare automată

Tool-uri de testare automată

Tool-uri de testare automată. Prezentare și clasificare



Criterii de alegere a unui tool de automatizare

- accesul la materiale suport/documentații;
- ușurința în aplicare;
- limbajul pentru descrierea scripturilor;
- testarea imaginilor;
- diverse activități legate de testare și tipuri de testare, e.g., testare funcțională, managementul testelor, etc;
- testarea bazelor de date;
- paradigme de programare, e.g., programare orientată obiect;
- utilizarea de framework-uri diverse;
- ușurința în întreținerea testelor;
- posibilitatea de identifica și folosi obiectele din mediul în care se lucrează;
- posibilitatea de a genera rapoarte de testare;
- minimizarea costurilor legate de asimilarea cunoștințelor de utilizare a tool-urilor.

Tool-uri de automatizare (1)

- **Selenium IDE**

- tool de testare folosit în special pentru testarea de regresie;
- **tool open source care permite înregistrarea și vizualizarea**, i.e., record/re-play la testare;
- browserul implicit folosit este Mozilla Firefox;
- facilități de export a scripturilor în alte limbaje de programare, e.g., Java, Python, C#, etc;

- **Selenium WebDriver**

- se utilizează împreună cu JUnit și TestNG;
- permite execuția simultană a mai multor teste;
- **permite identificarea elementelor din pagina web** pe baza id-ului, numelui, xpath, etc.

Tool-uri de automatizare (2)

- **QTP (HP UFT)**

- **tool folosit pentru testarea funcțională și de regresie;**
- folosește conceptul de keyword la crearea și întreținerea testelor (*engl.* **keyword driven testing**);
- permite crearea testelor direct din aplicație;
- ușor de folosit pentru testerii non-technici care doresc să adapteze și să creeze teste noi pe baza celor existente;
- permite documentarea bug-urilor identificate, permițând o depanare facilă din partea programatorului;
- crearea și documentarea testelor se regăsește în același loc;
- permite parametrizarea testelor;
- dispune de un mecanism eficient pentru identificarea obiectelor.

Tool-uri de automatizare (3)

- **Rational Functional Tester**

- **tool orientat obiect pentru testarea funcțională, testare de regresie, data-driven testing și testare GUI;**
- permite utilizarea unui număr mare de protocoale și limbaje de programare;
- **facilități pentru record/re-play la cerere;**
- facilități de integrare cu alte tool-uri de managementul codului sursă;
- permite crearea unui keyword asociat unui script de testare pentru a fi reutilizat.

Tool-uri de automatizare (4)

- **Watir**

- **tool open source pentru testarea de regresie;**
- permite crearea de teste care sunt ușor de parcurs și de întreținut;
- operează doar cu Internet Explorer;

- **Watir WebDriver**

- permite utilizarea și altor browsere web, e.g., Chrome, Firefox, Opera, etc;
- folosește limbajul Ruby;
- **permite testarea aplicațiilor web indiferent de tehnologia folosită pentru dezvoltarea acestora.**

Tool-uri de automatizare (5)

- **SilkTest**

- **tool pentru testarea funcțională și testarea de regresie;**
- este dedicat aplicațiilor e-buniness;
- dezvoltarea este preluată în 2006 de **Borland**;
- folosește un limbaj de programare similar cu C++, dispune de conceptele de programare orientată obiect;
- permite transformarea scripturilor în comenzi GUI;
- facilitează pentru identificarea mișcărilor mouse-ului;
- identifică toate ferestrele și componentele grafice ale aplicației ca și obiecte, determinând și utilizând apoi attributele acestora.

TEST AUTOMATION DEMO

Istoric. Tipuri de proiecte

Proiect Demo

Crearea proiectului

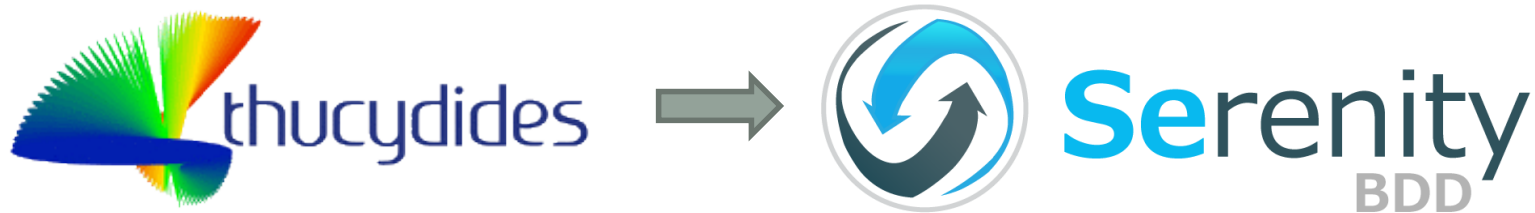
Structura proiectului

Componenta Pages. Componenta Steps. Componenta Tests

Cerință. Caz de testare

Serenity BDD. Istoric

- **2014: Serenity BDD** framework aka **Thucydides**;



Serenity BDD. Tipuri de proiecte

- **net.serenity-bdd:serenity-cucumber-archetype**
 - proiect Serenity pentru testare de acceptare care folosește Selenium 2/3, JUnit și Cucumber (+Gherkin: descrierea scenariilor folosind construcții de forma: given/when/then);
- **net.serenity-bdd:serenity-jbehave-archetype**
 - proiect Serenity pentru testare de acceptare care folosește Selenium 2/3, JUnit și JBehave;
- **net.serenity-bdd:serenity-junit-archetype**
 - proiect Serenity pentru testare de acceptare care folosește Selenium 2/3 și JUnit;
- **net.serenity-bdd:serenity-junit-screenplay-archetype**
 - proiect Serenity pentru testare de acceptare care folosește Screenplay, Selenium 2/3 și JUnit.

Serenity BDD. Proiect Demo



+



Serenity
BDD

- Selenium WebDriver
 - tool pentru testarea funcțională;
 - **permite automatizarea interacțiunii cu browserul în testarea aplicațiilor care folosesc browser web;**
- Serenity BDD
 - framework de testare care folosește Selenium WebDriver;
 - **permite rularea testelor și generarea rapoartelor de testare detaliate.**

Proiect Demo. Crearea proiectului

- **1. se creează un proiect Maven**
 - **Maven** – tool pentru managementul dependențelor la nivelul unui proiect;
 - gestionează toate dependențele, i.e., JUnit, Selenium WebDriver, Serenity;
- **2. se stabilește tipul de archetype, i.e., organizare a proiectului:**
 - se folosește **net.serenity-bdd:serenity-junit-archetype**, versiunea **1.8.4**;
 - adresa web: <https://mvnrepository.com/artifact/net.serenity-bdd/serenity-junit-archetype> ;
- **3. se precizează următorii parametri:**
 - **GroupId**: numele pachetului root;
 - **ArtifactId**: numele proiectului;

Proiect Demo. Crearea proiectului

- 4. după crearea proiectului, in fişierul pom.xml se adaugă dependența

```
<dependency>
  <groupId>javax.xml.bind</groupId>
  <artifactId>jaxb-api</artifactId>
  <version>2.3.0</version>
</dependency>

<dependency>
  <groupId>com.sun.xml.bind</groupId>
  <artifactId>jaxb-impl</artifactId>
  <version>2.3.0</version>
</dependency>

<dependency>
  <groupId>javax.activation</groupId>
  <artifactId>activation</artifactId>
  <version>1.1.1</version>
</dependency>

<dependency>
  <groupId>com.sun.xml.bind</groupId>
  <artifactId>jaxb-core</artifactId>
  <version>2.3.0</version>
</dependency>
```

Proiect Demo. Structura proiectului (1)

- **driver pentru browser-ele web:**

- **Firefox:**

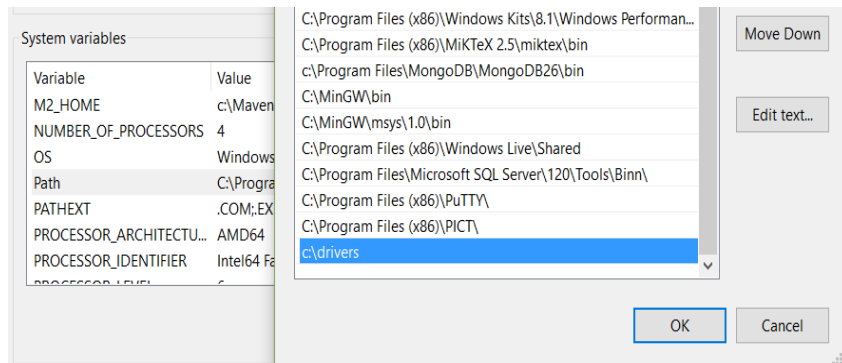
- adresa web: <https://github.com/mozilla/geckodriver/releases>;
 - e.g.: **geckodriver-v0.23.0-win64.zip**;

- **Chrome:**

- adresa web: <https://sites.google.com/a/chromium.org/chromedriver/downloads>;
 - e.g.: **chromedriver_win32.zip**;

- driverele se descarcă, se dezarchivează și se salvează într-un folder, e.g., **c:\drivers**, de unde pot fi folosite ulterior din orice proiect de testare;

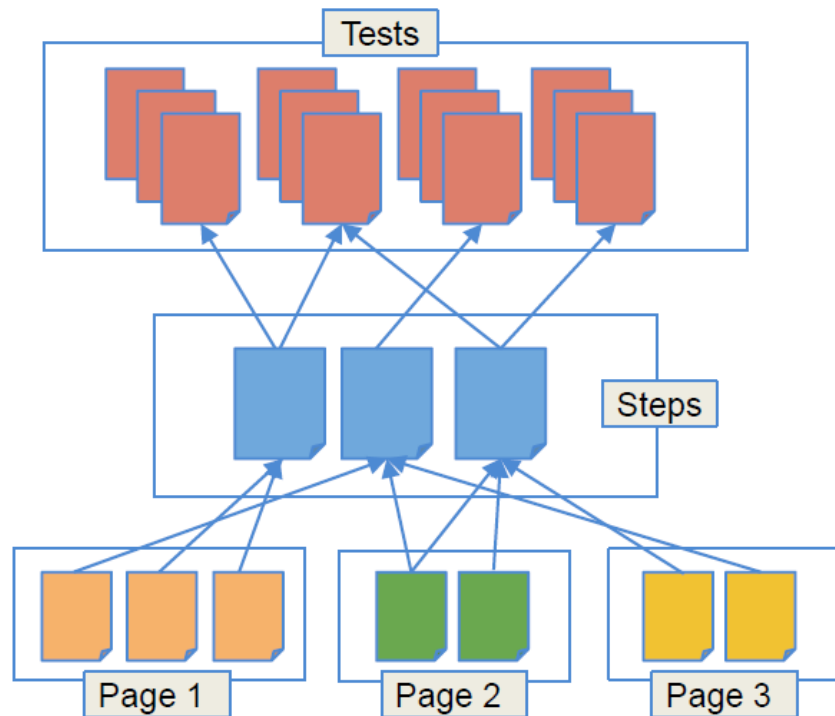
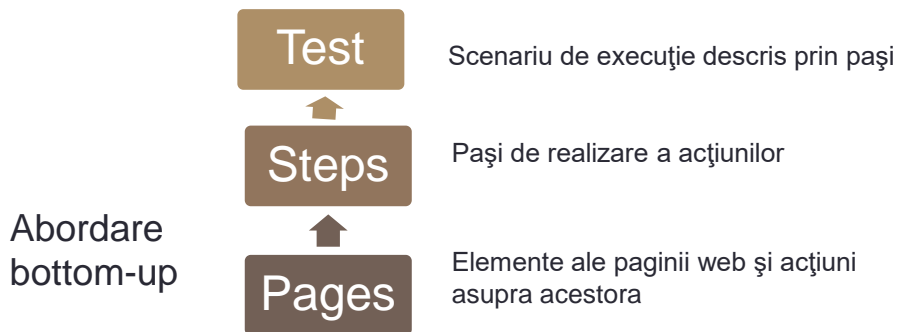
- în variabila de mediu **Path** se adaugă calea către folderul care conține driverele pentru browser-ele web, i.e., **c:\drivers**;



Proiect Demo. Structura proiectului (2)

- Se utilizează șablonul **Page Objec**, i.e., **page object model**;

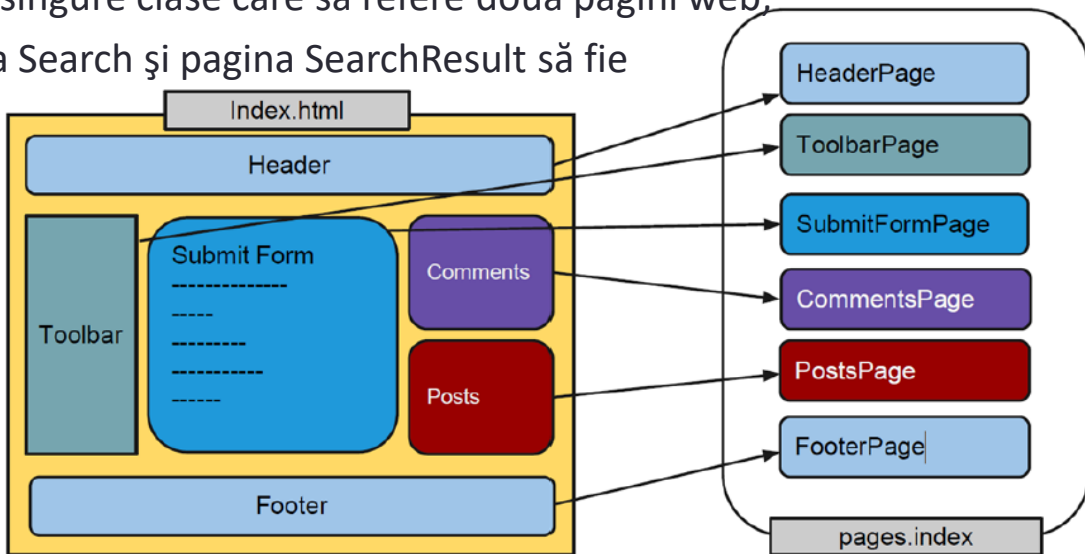
- **Pages** – se folosește Selenium WebDriver;
- **Steps** – se folosește Selenium WebDriver;
- **Tests** – se folosește JUnit, Serenity;



Proiect Demo. Componenta Pages (1)

- **Pages:**


- asociază elementelor dintr-o pagină web clase Java derivate din clasa `PageObject`;
- clasa obținută conține attribute de tipul `WebElement` care corespund tag-urilor HTML;
- **nu** este recomandată crearea unei singure clase care să refere două pagini web;
 - e.g., nu se recomandă ca pagina `Search` și pagina `SearchResult` să fie descrise prin aceeași clasă Java;



Proiect Demo. Componenta Pages (2)


stabilirea corespondenței între tag-ul HTML și un obiect WebElement

```
public class SearchPage extends PageObject {  
    @FindBy(name="gbqfq")  
    private WebElementFacade searchInput;  
  
    @FindBy(name="btnG")  
    private WebElementFacade searchButton;  
}
```

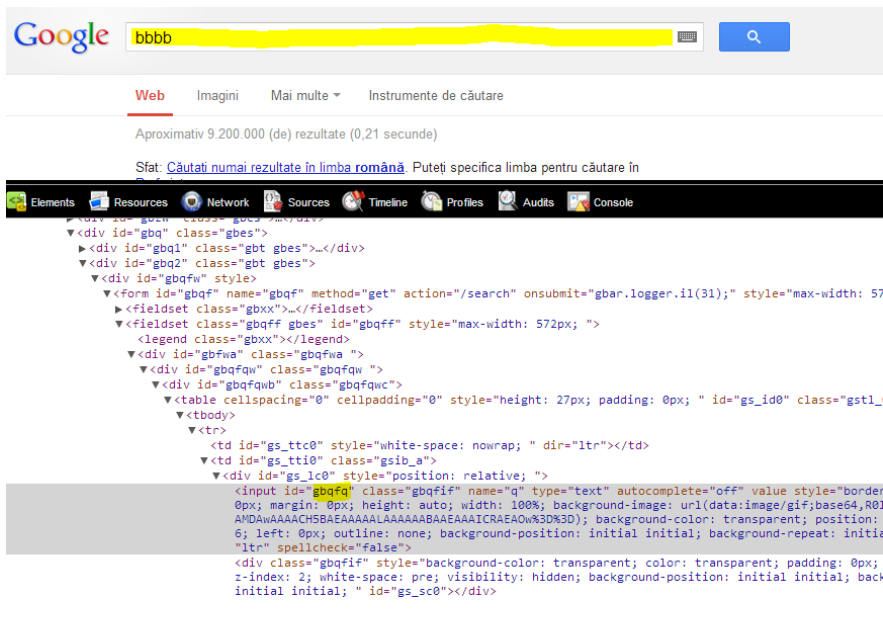


definirea **acțiunilor** asupra **elementelor**

```
public void inputSearchTerm(String searchTerm){  
    element(searchInput).waitUntilVisible();  
    searchInput.sendKeys(searchTerm);  
}  
  
public void clickOnSearch(){  
    element(searchButton).waitUntilVisible();  
    searchButton.click();  
}
```



Proiect Demo. Corespondența tag HTML – WebElement



```
public class SearchPage extends PageObject {
```

```
@FindBy(name="gbqfq")
private WebElementFacade searchInput;
```

```
@FindBy(name="btnG")
private WebElementFacade searchButton;
```

```
public void inputSearchTerm(String searchTerm){
    element(searchInput).waitUntilVisible();
    searchInput.sendKeys(searchTerm);
}
```

```
public void clickOnSearch(){
    element(searchButton).waitUntilVisible();
    searchButton.click();
}
```

Proiect Demo. Componenta Steps (1)

- **Steps:**
 - una sau mai multe acțiuni/măsurii aplicate cu scopul de a atinge un obiectiv;

```
public class SearchSteps extends ScenarioSteps {  
  
    private static final long serialVersionUID = 1433700707950737137L;  
    SearchPage searchPage;  
    SearchResultsPage searchResultsPage;  
  
    // general navigation action  
    @Step  
    public void navigateTo(String url) {  
        getDriver().get(url);  
    }  
  
    // search in google page for a keyword  
    @Step  
    public void searchForKeyword(String keyword) {  
        searchPage.inputSearchTerm(keyword);  
    }  
  
    // retrieve results  
    @Step  
    public void clickOnSearch() {  
        searchPage.clickOnSearch();  
    }  
  
    // stepGroup for search and retrieve results  
    @StepGroup  
    public void lookFor(String term) {  
        searchForKeyword(term);  
        clickOnSearch();  
    }  
  
    // navigation to a selected link  
    @Step  
    public void goToLink(String linkTitle) {  
        searchResultsPage.clickOnNamedLink(linkTitle);  
    }  
}
```

Proiect Demo. Componenta Steps (2)

Declararea paginii care
sunt utilizate de pașii
descriși în clasă

```
public class SearchSteps extends ScenarioSteps {  
  
    private static final long serialVersionUID = 1433700707950737137L;  
    SearchPage searchPage;  
    SearchResultsPage searchResultsPage;  
  
    // general navigation action  
    @Step  
    public void navigateTo(String url) {  
        getDriver().get(url);  
    }  
  
    // search in google page for a keyword  
    @Step  
    public void searchForKeyword(String keyword) {  
        searchPage.inputSearchTerm(keyword);  
    }  
  
    // retrieve results  
    @Step  
    public void clickOnSearch() {  
        searchPage.clickOnSearch();  
    }  
  
    // stepGroup for search and retrieve results  
    @StepGroup  
    public void lookFor(String term) {  
        searchForKeyword(term);  
        clickOnSearch();  
    }  
  
    // navigation to a selected link  
    @Step  
    public void goToLink(String linkTitle) {  
        searchResultsPage.clickOnNamedLink(linkTitle);  
    }  
}
```

Adnotarea @StepGroup

Adnotarea @Step

Proiect Demo. Componenta Steps (3)

- o clasă `AAASSteps` folosește acțiunile definite de clase `Page`;
 - clasele `Steps` extind de obicei clasa `ScenarioSteps`, care implementează interfața `Serializable`;
 - clasa `AAASSteps` conține un atribut de tipul clasei `Page` pe care dorește să o folosească;
 - toate metodele care sunt considerate `Step` trebuie să folosească adnotarea `@Step` sau `@StepGroup`;
 - metodele adnotate vor fi incluse în raportul generat după testare;
 - metodele se numesc astfel încât Serenity să poată identifica ușor acțiunea realizată de un anumit pas;
 - E.g.: metoda **clickOnSearch** va apărea în raport *Click on Search*;
- gruparea mai multor pași într-un *step group* permite gestionarea structurii raportului.

```
public class SearchSteps extends ScenarioSteps {  
  
    private static final long serialVersionUID = 1433700707950737137L;  
    SearchPage searchPage;  
    SearchResultsPage searchResultsPage;  
  
    // general navigation action  
    @Step  
    public void navigateTo(String url) {  
        getDriver().get(url);  
    }  
  
    // search in google page for a keyword  
    @Step  
    public void searchForKeyword(String keyword) {  
        searchPage.inputSearchTerm(keyword);  
    }  
  
    // retrieve results  
    @Step  
    public void clickOnSearch() {  
        searchPage.clickOnSearch();  
    }  
  
    // stepGroup for search and retrieve results  
    @StepGroup  
    public void lookFor(String term) {  
        searchForKeyword(term);  
        clickOnSearch();  
    }  
  
    // navigation to a selected link  
    @Step  
    public void goToLink(String linkTitle) {  
        searchResultsPage.clickOnNamedLink(linkTitle);  
    }  
}
```

Proiect Demo. Organizarea Pages și Steps (1)

- **Pages**

- conține corespondența/ maparea între tag-urile HTML și obiectele `WebElement`;
- definește acțiunile realizate asupra elementelor paginii;

- **Steps:**

- grupează acțiunile definite de **Pages**;
- acțiunile pot fi realizate/aplicate asupra oricărui număr de pagini;
 - E.g.: într-o clasă `Steps` putem defini un scenariu care să navigheze `HeaderPage` și `ToolBarPage`;
 - E.g.: într-o clasă `Steps` putem avea acțiunea *click* pe **checkBox** dacă o anumită condiție este îndeplinită;

```
public class SearchSteps extends ScenarioSteps {  
  
    private static final long serialVersionUID = 1433700707950737137L;  
    SearchPage searchPage;  
    SearchResultsPage searchResultsPage;  
  
    // general navigation action  
    @Step  
    public void navigateTo(String url) {  
        getDriver().get(url);  
    }  
  
    // search in google page for a keyword  
    @Step  
    public void searchForKeyword(String keyword) {  
        searchPage.inputSearchTerm(keyword);  
    }  
  
    // retrieve results  
    @Step  
    public void clickOnSearch() {  
        searchPage.clickOnSearch();  
    }  
  
    // stepGroup for search and retrieve results  
    @StepGroup  
    public void lookFor(String term) {  
        searchForKeyword(term);  
        clickOnSearch();  
    }  
  
    // navigation to a selected link  
    @Step  
    public void goToLink(String linkTitle) {  
        searchResultsPage.clickOnNamedLink(linkTitle);  
    }  
}
```

Proiect Demo. Organizarea Pages și Steps (2)

```
public class SearchSteps extends ScenarioSteps {

    private static final long serialVersionUID = 1433700707950737137L;
    SearchPage searchPage;
    SearchResultsPage searchResultsPage;

    // general navigation action
    @Step
    public void navigateTo(String url) {
        getDriver().get(url);
    }

    // search in google page for a keyword
    @Step
    public void searchForKeyword(String keyword) {
        searchPage.inputSearchTerm(keyword);
    }

    // retrieve results
    @Step
    public void clickOnSearch() {
        searchPage.clickOnSearch();
    }

    // stepGroup for search and retrieve results
    @StepGroup
    public void lookFor(String term) {
        searchForKeyword(term);
        clickOnSearch();
    }

    // navigation to a selected link
    @Step
    public void goToLink(String linkTitle) {
        searchResultsPage.clickOnNamedLink(linkTitle);
    }
}
```

```
public class SearchPage extends PageObject {

    @FindBy(css="input[title='Search']")
    private WebElementFacade searchInput;

    @FindBy(name="btnG")
    private WebElementFacade searchButton;

    public void inputSearchTerm(String searchTerm) {
        element(searchInput).waitUntilVisible();
        searchInput.sendKeys(searchTerm);
    }

    public void clickOnSearch() {
        element(searchButton).waitUntilVisible();
        searchButton.click();
    }
}
```

Project Demo. Componenta Tests (1)



Proiect Demo. Componenta Tests (2)

- un test conectează elementele necesare realizării scenariului de execuție:
 - **TestRunner:**
 - precizează programul Runner care va executa testul;
 - testele care folosesc date preluate dintr-un fișier folosesc `ParameterizedSerenityRunner`;
 - **WebDriver:**
 - precizează modul în care vor fi executate testele din clasa dată;
 - **uniqueSession** indică faptul ca dacă în clasa curentă sunt definite mai multe teste atunci se va folosi aceeași instanță a browser-ului web, i.e. browser-ul web nu se va închide după fiecare test;
 - **Pages:**
 - toate testele trebuie să declare o pagină web inițială (*engl.* **default URL**), care se setează în adnotarea `@ManagedPages`;
 - **Steps:**
 - sunt declarați pentru fiecare test și descriu ordinea de realizare a pașilor incluși în scenariul de execuție;
 - un test poate conține mai multe obiecte `Steps`;
 - fiecare referință `Steps` este adnotată cu `@Step`;
 - **Tests:**
 - descriu scenariul testului, i.e. acțiunile și ordinea de execuție a acestora.

```
@RunWith(SerenityRunner.class)
public class GoogleSearchTest {

    @ManagedSession(uniqueSession = true)
    public WebDriver webDriver;

    @ManagedPages(defaultUrl = "http://www.google.com")
    public Page page;

    Steps
    public SearchSteps googleSearchSteps;

    @Test
    public void googleSearchTest() {
        googleSearchSteps.navigateToPage(getDefaultBaseUrl());
        googleSearchSteps.lookFor("paua");
        googleSearchSteps.goToLink("paua - Wikipedia, the free encyclopedia");
    }
}
```

Proiect Demo. Componenta Tests (3)

- Folosirea fișierelor cu date de intrare

Runner parametrizat

Test parametrizat

Metode getter/setter pentru câmpurile
din fișierul cu date de intrare

```
@RunWith(SerenityParameterizedRunner.class)
@UseTestDataFrom("src/test/test-data/GoogleTestData.csv")
public class GoogleSearchTestDdt {

    @Managed(uniqueSession = true)
    public WebDriver webdriver;

    @ManagedPages(defaultUrl = "http://www.google.com")
    public Pages pages;

    @Steps
    public SearchSteps googleSearchSteps;

    @Test
    public void googleSearchTestDDT() {
        googleSearchSteps.navigateTo(pages.getDefaultBaseUrl());
        googleSearchSteps.lookFor(getName());
        googleSearchSteps.verifyDefinition(getDefinition());
    }

    public String name;
    public String definition;

    @Qualifier
    public String getQualifier() {
        return name;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getDefinition() {
        return definition;
    }

    public void setDefinition(String definition) {
        this.definition = definition;
    }
}
```

Proiect Demo. Componenta Tests (4)

- testele care utilizează date de intrare din fișiere au câteva caracteristici:
 - folosesc un runner specializat, i.e., `SerenityParameterizedRunner`;
 - testul va permite iterarea fișierului cu date de intrare pentru preluarea de date;
 - fiecare linie din fișier conține datele de intrare pentru un test;
 - fișierul care conține datele de intrare se precizează prin intermediul adnotării `@UseTestDataFrom`;
 - fișierul cu date de intrare este un fișier `.csv`, i.e., datele sunt separate prin virgulă;
 - numele variabilelor din clasa de test care referă datele de intrare preluate din fișier trebuie să corespundă celor precizate în fișierul `.csv`, acestea fiind indicate pe prima linie;
 - metodele `getter` și `setter` vor fi folosite de runner-ul parametrizat;
 - fișierul `.csv` poate conține și rezultatele așteptate, ceea ce va permite validarea, i.e., stabilirea dacă testul este **passed** sau **failed**;
 - adnotarea `@Qualifier` se folosește la raportare și leagă un câmp de un anumit test.

```
@RunWith(SerenityParameterizedRunner.class)
@UseTestDataFrom("src/test/test-data/GoogleTestData.csv")
public class GoogleSearchTestDdt {

    @Managed(uniqueSession = true)
    public WebDriver webDriver;

    @ManagedPages(defaultUrl = "http://www.google.com")
    public Pages pages;

    @Steps
    public SearchSteps googleSearchSteps;

    @Test
    public void googleSearchTestDDT() {
        googleSearchSteps.navigateTo(pages.getDefaultBaseUrl());
        googleSearchSteps.lookFor(getName());
        googleSearchSteps.verifyDefinition(getDefinition());
    }

    public String name;
    public String definition;

    @Qualifier
    public String getQualifier() {
        return name;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getDefinition() {
        return definition;
    }

    public void setDefinition(String definition) {
        this.definition = definition;
    }
}
```

Proiect Demo. Componenta Tests (5)

- exemplu de fișier .csv:

```
Name,Definition,  
def pear,"a sweet yellowish- or brownish-green edible fruit which is narrow at the stalk and wider towards the base",  
def cinnamon,"an aromatic spice made from the peeled, dried, and rolled bark of a SE Asian tree.",
```

- prima linie dintr-un fișier .csv este linia **header**, care va fi ignorată de teste;
- ea indică structura fișierului cu date de intrare și este folosită pentru a lega variabilele declarate în clasa de test **cu aceleași nume** de câmpurile din fișierul .csv;
- toate câmpurile sunt separate prin virgulă;
- pentru ignorarea virgulei în datele de intrare, informația din coloană este inclusă între ghilimele;
- fișierul .csv trebuie plasat în folderul precizat în clasa de test.

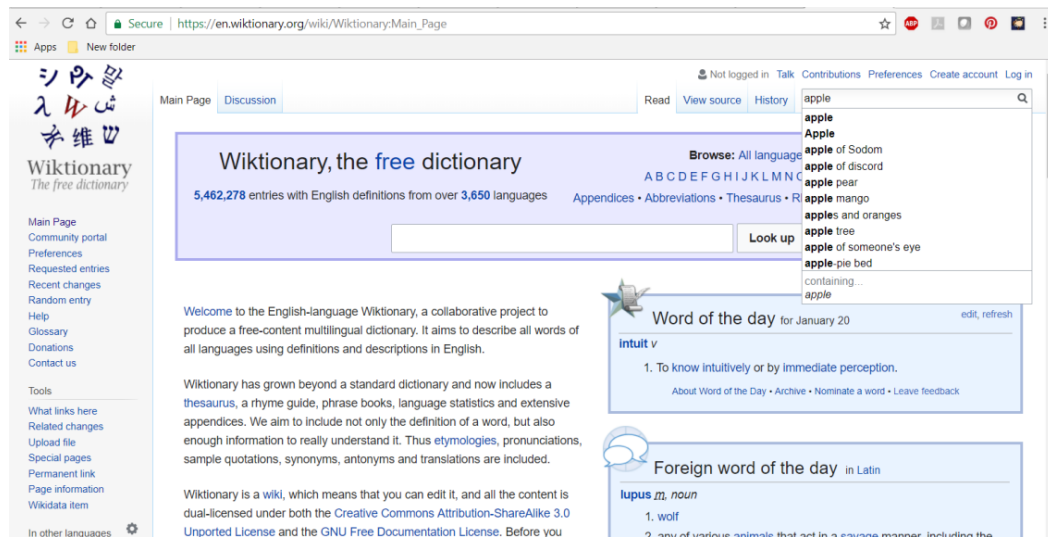
Proiect Demo. Cerință. Caz de testare

- **Cerință (Scenariu de utilizare):**

- pentru pagina web <https://en.wiktionary.org/wiki> dorim să căutăm definiția unor cuvinte, e.g., apple, pear, y#1/;

- **Caz de testare (Pași/Steps de execuție):**

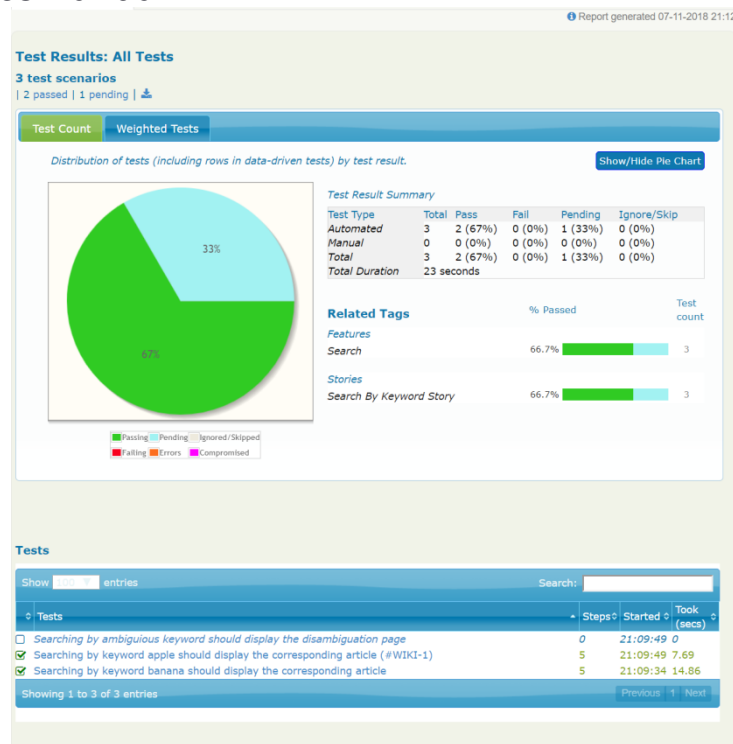
1. se accesează <https://en.wiktionary.org/wiki> ;
2. se completează termenul căutat;
3. click pe butonul search;
4. caută dacă definiția presupusă inițial se găsește pe pagina rezultată;
5. validează rezultatul, i.e., testul este **passed** sau **failed**;



Proiect Demo. Generarea raportului de testare

- în IntelliJ IDEA, în fereastra de comenzi Maven se adaugă comanda:

mvn serenity:aggregate



ÎNTREBĂRI PENTRU EXAMEN

Întrebări cu răspuns scurt

Întrebări cu răspuns lung

Întrebări cu răspuns scurt

- **Întrebări cu răspuns scurt:**

- **Curs 11:**

1. Definiți noțiunea: testare manuală. Exemplificați.
2. Enumerați trei caracteristici ale testării manuale. Exemplificați.
3. Descrieți pe scurt trei avantaje și dezavantaje ale testării manuale. Exemplificați.
4. Definiți noțiunea: testare automată. Exemplificați.
5. Enumerați trei caracteristici ale testării automate. Exemplificați.
6. Descrieți pe scurt trei avantaje și dezavantaje ale testării automate. Exemplificați.
7. Descrieți modul de aplicare a metricii ROI în testarea automată. Exemplificați.
8. Descrieți pe scurt trei situații în care testarea manuală este recomandată. Exemplificați.
9. Descrieți pe scurt trei situații în care testarea automată este recomandată. Exemplificați.

10. Descrieți pe scurt trei principii de aplicare a testării automate. Exemplificați.
11. Descrieți pe scurt trei criterii pentru alegerea unui tool de automatizare. Exemplificați.

- **Curs 12:**

- *fără întrebări din secțiunea Test Automation Demo*

Întrebări cu răspuns lung

- **Întrebări cu răspuns lung:**

- **Curs 11:**

1. Comparați noțiunile: testare manuală și testare automată. Exemplificați.
2. Enumerați trei caracteristici ale testării manuale. Descrieți avantajele acestor caracteristici. Exemplificați.
3. Enumerați trei caracteristici ale testării automate. Descrieți avantajele acestor caracteristici. Exemplificați.
4. Descrieți trei situații în care testarea manuală este recomandată. Justificați de ce testarea automată nu este utilă. Exemplificați.
5. Descrieți trei situații în care testarea automată este recomandată. Justificați de ce testarea manuală nu este utilă. Exemplificați.
6. Descrieți trei principii de aplicare a testării automate. Justificați avantajele aplicării acestora. Exemplificați.

- **Curs 12:**

- *[fără întrebări din secțiunea Test Automation Demo](#)*