

UNIVERSITATEA „ALEXANDRU IOAN CUZA” IAȘI

**FACULTATEA DE INFORMATICĂ**



LUCRARE DE LICENȚĂ

**Testarea automată în completarea testării manuale pentru  
aplicațiile WEB**

Propusă de

**Cosmin Jeroaea**

**Sesiunea: Iulie, 2019**

Coordonator științific

**Conf. Dr. Anca Vitcu**

**UNIVERSITATEA „ALEXANDRU IOAN CUZA” IAȘI**

**FACULTATEA DE INFORMATICĂ**

**Testarea automată în completarea testării  
manuale pentru aplicațiile WEB**

**Cosmin Jeroaea**

**Sesiunea: Iulie, 2019**

Coordonator științific

**Conf. dr. Anca Vitcu**

Avizat,

Îndrumător Lucrare de Licență

Titlul, Numele și prenumele \_\_\_\_\_

Data \_\_\_\_\_ Semnătura \_\_\_\_\_

### **DECLARAȚIE privind originalitatea conținutului lucrării de licență**

Subsemnatul Jeroaea Cosmin Liviu, domiciliul în sat Bahluiu, com. Cotnari, jud. Iași născut la data de 04/06/1996, identificat prin CNP 1960604226794, absolvent al Universității „Alexandru Ioan Cuza” din Iași, Facultatea de Informatică specializarea Informatică, promoția 2017 declar pe propria răspundere, cunoscând consecințele falsului în declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr. 1/2011 art.143 al. 4 și 5 referitoare la plagiat, că lucrarea de licență cu titlul: Testarea automată în completarea testării manuale a aplicațiilor WEB , elaborată sub îndrumarea d-na Conf. Dr. Anca Vitcu, pe care urmează să o susțină în fața comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la introducerea conținutului său într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări științifice în vederea facilitării falsificării de către cumpărător a calității de autor al unei lucrări de licență, de diploma sau de disertație și în acest sens, declar pe proprie răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Data, .....

Semnătură student .....

## DECLARAȚIE DE CONSIMȚĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul „*Testarea automată în completarea testării manuale a aplicațiilor WEB*”, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iași, \_\_\_\_\_

*Absolvent Cosmin Jeroaea*

\_\_\_\_\_  
(semnătura în original)

## Cuprins

<b>1</b>	<b>Introducere .....</b>	<b>1</b>
1.1	Motivație.....	1
1.2	Context.....	1
1.3	Contribuții .....	2
1.4	Structura lucrării.....	2
<b>2</b>	<b>Testarea manuală.....</b>	<b>3</b>
2.1	Evoluția calității.....	3
2.2	Erori ce au rămas în istorie .....	5
2.3	Definiție.Descriere .....	6
2.4	Rolul si scopul testării software.....	6
2.5	Ciclul de viață a dezvoltării software .....	7
2.6	Modele de dezvoltare a unui sistem și aplicarea testării .....	10
2.6.1	Modelul în cascadă .....	10
2.6.2	Modelul V.....	11
2.6.3	Modelul Agile .....	13
2.7	Tipuri de testare software .....	14
<b>3</b>	<b>Testarea automată .....</b>	<b>16</b>
3.1	Definiție.....	16
3.2	Benficiile testării automate .....	16
3.2.1	Livrarea rapidă.....	16
3.2.2	Reducerea resurselor umane .....	17
3.2.3	Îmbunătățirea eficienței testării .....	17
3.3	Avantajele testării automate a aplicațiilor web .....	17
3.4	Dezavantajele testării automate a aplicațiilor web .....	18
<b>4</b>	<b>Prezentarea proiectului .....</b>	<b>19</b>
4.1	Tehnologii folosite .....	19
4.1.1	.NET Framework .....	19

4.1.2	C#.....	19
4.1.3	Selenium .....	19
<b>4.2</b>	<b>Structură/Arhitectură .....</b>	<b>20</b>
<b>4.3</b>	<b>Module ale proiectului .....</b>	<b>21</b>
4.3.1	PageObjects.....	21
4.3.2	Business .....	23
4.3.3	AutomationTesting .....	23
<b>4.4</b>	<b>Desfășurarea testelor .....</b>	<b>24</b>
<b>4.5</b>	<b>Exemplificarea structurii pe o aplicație WEB .....</b>	<b>24</b>
<b>4.6</b>	<b>Complexități și impedimente.....</b>	<b>25</b>
<b>5</b>	<b>Concluzii.....</b>	<b>26</b>
<b>6</b>	<b>Bibliografie .....</b>	<b>27</b>

# 1 Introducere

În primă instanță voi vorbi despre motivul pentru care am ales această temă, diferența față de produsele existente pe piață și contribuțiile aduse de mine astfel încât să poată fi ușor folosit de către oricine, aflat la un nivel începător.

## 1.1 Motivație

Motivul principal pentru care am ales acest domeniu este acela de a încerca să aduc la cunoștință, persoanelor ce sunt interesate și își doresc să-și înceapă drumul unei cariere pe în domeniul testării, o schiță sau, mai bine zis, bazele unui proiect de lungă durată pentru testarea automată.

În al doilea rând, consider că testarea are un rol important în dezvoltarea de produse software, astfel încât, lipsa testării unui produs poate duce la o greșeală ce poate bloca sau elimina folosirea în continuare a produsului respectiv. Astfel, prin descoperirea posibilelor erori testarea poate reduce acest risc, deși niciodată nu poate demonstra lipsa unui asemenea risc.

Totodată, cu puține cunoștințe de programare, aceștia pot învăța începuturile către caile testării software automate folosindu-se de structura soluției prezente în acest proiect.

## 1.2 Context

Testarea reprezintă un proces important în dezvoltarea un produs nou, de orice natură, fizic sau software. Un plan de testare bine organizat și o descriere amănunțită a fiecărui test tinde către o calitate superioară a acelui produs care, în lipsa testării, ar putea cauza daune la nivel fizic sau cibernetic.

De asemenea, un produs software poate lucra cu date sensibile ceea ce crește riscul de a permite, involuntar, accesul către acestea pentru a putea fi alterat. În acest scop, testarea, pe lângă rolul de asigurare al calității, joacă și rolul împiedicării producerii unei catastrofe. În prezent, criminalitatea cibernetică este tratată cu multă seriozitate de către producătorii de software intrucât datele sensibile și funcționalitățile ce lucrează cu acestea sunt foarte bine testate astfel încât produsul să tindă spre infailibil.

### **1.3 Contribuții**

În urma cercetării făcute pe acest domeniu și specific pe această ramură, am concluzionat că ar fi nevoie de multe resurse și documentație, adunate, pentru ca un proiect de teste automate să poată fi construit și folosit pe lungă durată, din punct de vedere al complexității.

Soluția creată vizează doar aplicațiile WEB și este construită și modelată în așa fel încât aceasta poate fi utilizată ca bază pentru dezvoltarea testelor automate pentru o mare majoritate din aplicațiile WEB.

Pornind de la această bază, utilizatorul poate elabora acest proiect astfel încât îi vor fi satisfăcute nevoile, totodată păstrând și o organizare a părților vizate din aplicație.

### **1.4 Structura lucrării**

Întreaga lucrare se bazează pe domeniul testării software, astfel încât capitolele acesteia vor cuprinde informații, exemple, indicații și sugestii legate de testarea manuală, cum se realizează, modalități, tehnici de testare manuală și câteva dintre cele mai bune practici în procesul de testare manuală.

În a doua parte a lucrării, atenția este îndreptată către modalitatea ușurării testării manuale și anume automatizarea testelor manuale. Este descris procesul de automatizare, cum este produsă transformarea, tehnologiile folosite pe proiect, structura proiectului și modalitatea în care un test automat este rulat.



## **2 Testarea manuală**

Acest capitol cuprinde atât o viziune de ansamblu asupra a ceea ce înseamnă testarea software, un scurt istoric, bazele acesteia, cât și o privire în detaliu legată de tipurile multiple de testare și tehnicile folosite.

### **2.1 Evoluția calității**

Gelperin și Hetzel au analizat evoluția conceptului de testare a unui sistem informatic și au împărțit evoluția acestuia în mai multe etape, în funcție de filozofia care a stat la baza conceptului:

1945 - 1956 - Orientarea spre depanare

Testarea programelor informatice este o activitate care a apărut o dată cu procesul de dezvoltare a acestora. În perioada apariției primelor sisteme informatice - 1945-1956, procesul de testare era orientat către partea hardware a unui sistem. Defectele de software nu erau atât de importante la vremea respectivă. Testarea era făcută de persoanele care se ocupau și de dezvoltare, această activitate fiind denumită verificare. Termenul de bug<sup>1</sup> apare în anii '70, când Thomas Alva Edison folosește această denumire într-o scrisoare către Theodore Puskas, descriind un defect într-un sistem. Se pare că această denumire era foarte des folosită în perioada lui Edison.

O primă lucrare ce poate fi considerată ca o primă abordare a conceptului de testare o reprezintă articolul lui Alan Turing publicat în anul 1949, articol ce abordează principiile teoretice ale verificării unui program. În anul 1950, într-un alt articol, Turing ridică problema inteligenței artificiale definind un test pe care un sistem trebuie să îl treacă și anume răspunsurile pe care sistemul trebuie să le ofere la întrebările unui tester, să nu difere de cele oferite de către un om.

1957 - 1978 - Orientarea spre demonstrație

Odată cu creșterea în număr a sistemelor, costurile de producție și complexitatea acestora, testarea a căpătat din ce în ce mai multă atenție. Din cauza impactului pe care l-ar fi avut erorile și defectele unui sistem asupra economiei, testarea se extinde la o scară

---

<sup>1</sup> Denumirea folosită pentru o eroare găsită într-un sistem informatic din trecut până astăzi

largă. Testarea este luată în serios și de persoanele implicate în dezvoltarea programelor informatice care devin conștiente de riscurile ce pot apărea la un produs deja livrat, soluția fiind prevenirea acestora prin testarea amănunțită a sistemului și fixarea erorilor apărute de pe urma acestora.

În anul 1958, Gerald M. Weinberg, manager al dezvoltării sistemelor operaționale pentru proiectul Mercury înființează prima echipă oficială de testare.

#### 1979 - 1982 - Orientare spre defectare

Conceptele de depanare și testare devin mai riguros separate o dată cu publicarea de către Glenford J. Myers a lucrării “The Art of Software Testing”, în 1979. Myers face distincția față de depanare. Aceasta este acum considerată activitatea ce ține de dezvoltarea programului iar testarea este acum activitatea de rulare a unui program cu scopul de a descoperi erori în sistemul dezvoltat. Totodată, Myers menționează faptul că testarea poate fi realizată inconștient pe un set de date care ar asigura funcționarea corectă a sistemului. Aceasta reprezintă un pericol deoarece multe defecte vor trece neobservate.

În cartea sa, “Software Engineering Economics”, Barry W. Boehm introduce noțiunea de costul fixării unui bug care crește exponențial în timp.

#### 1983 - 1987 - Orientarea spre evaluare

În 1983, Biroul Național de Standarde din Statele Unite ale Americii publică un set de practici adresate activităților de verificare, validare și testare a programelor de calculator. Metodologia prezentată este adresată instituțiilor americane federale și cuprinde metode de analiză, evaluare și testare care pot fi aplicate în procesul de testare pe durata de dezvoltare a unui sistem.

În anii '70 crește nivelul de profesionalism a persoanelor pe partea de testare. Așadar nu întârzie să apară și posturile dedicate acestora, și anume tester, manager de teste sau analist de teste. Instituțiile americane ANSI și IEEE încep elaborarea unor standarde care să formalizeze procesul de testare, efort concretizat în standarde precum ANSI IEEE STD 829, în 1983, care stabilea anumite formate care să fie utilizate pentru crearea documentației de testare.

1988 - în prezent - Orientarea spre prevenire

Testarea software ia amploare, ocupând toate fazele de dezvoltare a unui sistem în paralel, după o publicație a IEEE. Ea are următoarele activități principale: planificare, analiză, proiectare, implementare, execuție și întreținere. Respectarea acestei metodologii duce la scăderea costurilor de dezvoltare și de întreținere a unui sistem prin scăderea numărului de defecte care ajung nedetectate în produsul final.

## **2.2 Erori ce au rămas în istorie**

De-a lungul istoriei, de la începuturile testării manuale, factorul uman a jucat un rol important în asumarea calității unui produs. Acesta poate conduce un produs către o calitate superioară datorită creativității și eficienței cu care poate asigura calitatea.

Cu toate acestea, eroarea umană nu a încetat să își facă apariția și astfel, bug-urile au început să apară.

În 1996, pe data de 4 Iunie, o rachetă fără echipaj, Ariane 5, lansată de către Agenția Spațială Europeană a explodat la doar 40 de secunde din momentul lansării din Kourou, French Guiana. Procesul de dezvoltare a durat un deceniu și a costat 7 miliarde de dolari. Racheta și încărcătura acesteia valora 500 milioane de dolari.

După o investigație amănunțită, raportul dezvăluie cauza erorii în sistem. Numărul specific vitezei orizontale a rachetei, definit pe 64 biți float, este convertit într-un integer pe 16 biți. Din cauza faptului că numărul este mai mare decât 32,767, cel mai mare număr ce poate fi reprezentat într-un integer pe 16 biți, conversia eșuează, aruncând o excepție. Netratarea acelei excepții în cod duce la producerea exploziei rachetei și pierderea a tot ce a însemnat proiectul Ariane 5.



În anul 1962, pe data de 22 Iulie, la începuturile explorării spațiului cosmic, Marine 1, o navetă spațială se îndreaptă spre Venus, menită să colecteze date științifice despre planetă. La doar 293 secunde după lansare, un ofițer ordonă autodistrugerea navetei deoarece aceasta acționează o manevră de ridicare necomandată.

Raportul arată că o cratimă lipsă în instrucțiunile calculatorului în programul de editare a datelor primite de pe Pământ, duce la instrucțiuni de ghidaj incorecte către navetă, cauzând manevre de schimbare a traiectoriei.

## 2.3 Definiție.Descriere

Testarea software este procesul de investigare care oferă informații vis-a-vis de calitatea produsului sau serviciului în curs de testare, luând în considerare și contextul în care acesta va fi folosit. Testarea software pune la dispoziție o imagine detaliată în legătură cu produsul aflat în dezvoltare oferind posibilitatea de îmbunătățire a logicii aplicației, de anticipare a riscurilor legate de implementarea software.

Poate fi folosită în 2 moduri, astfel încât informația să fie cât mai consistentă:

-  Procesul de rulare a programului sau a aplicației în vederea identificării erorilor/defectelor din program
-  Procesul de validare și verificare în vederea respectării businessului stabilit, în vederea cerințelor tehnice și pentru a stabili dacă produsul se comportă conform așteptărilor

În dependență de metodologia folosită, testarea poate fi implementată la orice pas din procesul de dezvoltare. Testarea începută mai devreme poate preveni produsul sau serviciul de apariția erorilor. Cu cât aceasta este înaintată ca și stadiu de start, probabilitatea de apariție a defectelor și în special a erorilor fatale crește.

Nu există un proces de testare altfel încât să fie posibilă identificarea tuturor defectelor existente într-un produs. Dar, un proces amănunțit de acest tip poate furniza informații detaliate în legătură cu acele criterii de acceptare precum metrici, constrângeri sau specificații definite în primele etape ale dezvoltării.

## 2.4 Rolul și scopul testării software

Am observat că eroarea umană poate cauza un defect sau o eroare ce poate fi introdusă oricând în ciclul de dezvoltare a unui produs, iar în urma acestui defect sau acestei erori, consecințele pot fi catastrofice. Testarea riguroasă este necesară în timpul dezvoltării și mentenanței pentru a identifica defecte, în scopul de a reduce defectele în mediul operațional și pentru a crește calitatea produsului livrat. Această testare implică explorarea părților din interfața aplicației unde un utilizator ar putea produce o greșeală introducând date greșite într-un input sau interpretând greșit datele de ieșire, cauze ce ar conduce la potențiale atacuri.

Testarea ne ajută în măsurarea calității prin intermediul numărului de buguri, numărul de rulări de teste și procentul de acoperire a aplicației de către teste. Acestea pot fi realizate pentru fiecare dintre testarea funcțională și non-funcțională. Testarea poate oferi încredere în calitatea produsului software dacă sunt găsite puține erori sau chiar numărul acestora este 0. Acest lucru garantează că testarea este una suficient de riguroasă. Atunci când, în urma testării, defecte sunt găsite, calitatea produsului crește odată cu fixarea acelor defecte, dovedind că rezolvarea problemelor are o atenție crescută.

## **2.5 Ciclu de viață a dezvoltării software**

De-a lungul timpului, îmbunătățirea procesului de dezvoltare a fost în continuă creștere, pentru a putea construi un produs ce satisface principiile calității. De aceea, au fost create diferite modele ale ciclului de viață în dezvoltarea software ce oferă un traseu clar de urmat în procesul de dezvoltare.

Ciclu de viață al dezvoltării este un proces sistematic pentru construirea software-ului ce asigură calitatea și corectitudinea produsului dezvoltat.

Acest proces are ca scop producerea softului de calitate ce satisface așteptările clientului. El constă într-un plan detaliat care explică cum se face planificarea, construirea și mentenanța unui sistem specific. Fiecare fază a ciclului are propriul proces iar output-ul unei faze facilitează producerea următoarei.

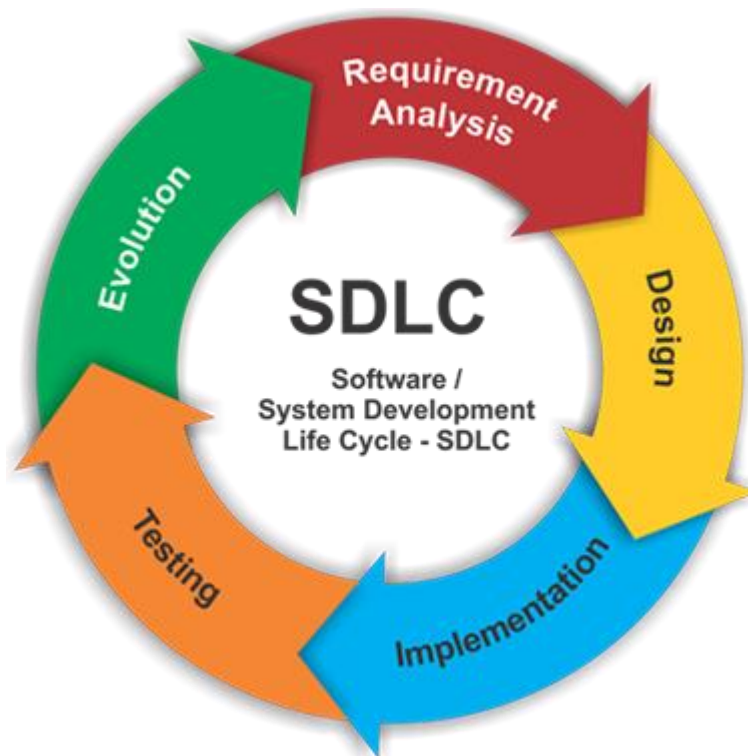


Figura 1 Ciclul de viață al dezvoltării software [pag. 27, nr. 10]

Câteva dintre avantajele folosirii unui ciclu de viață pentru dezvoltarea sistemelor software sunt următoarele:

- ✚ oferă o bază pentru planificare, programare a termenilor limită și estimare a volumului de muncă
- ✚ este un mecanism pentru controlul și urmărirea progresului
- ✚ crește vizibilitatea planificării proiectului și implică toate părțile implicate în procesul de dezvoltare
- ✚ implică șansa de creștere a vitezei de dezvoltare
- ✚ îmbunătățește relațiile cu clienții
- ✚ ajută la diminuarea riscurilor și a alterării planului de gestionare a proiectului

Fazele ciclului de viață a dezvoltării software sunt:

- a. Colecționarea cerințelor și analiza acestora
- b. Studiul de fezabilitate
- c. Designul
- d. Programarea (Codarea)
- e. Testarea
- f. Instalarea
- g. Mentenanța

**Colecționarea cerințelor și analiza acestora** este prima fază a ciclului, condusă de echipa de seniori care aduc datele necesare începerii proiectului din partea părților implicate în dezvoltarea acestuia și de la experți în domeniu. Planificarea cerințelor pentru stabilirea calității este realizată în această fază.

Odată ce faza planificării după cerințe este gata, **studiul de fezabilitate** este realizat, definind și documentând necesitățile software-ului. Include, de asemenea, și totul ce ar trebui proiectat și dezvoltat în timpul ciclului de viață a proiectului.

În faza **designului** documentele sistemului și proiectarea sistemului sunt pregătite pentru fiecare cerință și specificație a proiectului. Această fază servește ca date de intrare pentru următoarea fază.

După proiectarea specificațiilor proiectului, următoarea fază este **codarea**. În această fază, persoanele responsabile de scrierea codului vor începe să creeze întregul sistem utilizând limbajul de programare ales. În faza de codare, sarcinile sunt împărțite în unități sau module ce sunt atribuite câte unei persoane.

Odată ce sistemul este complet și este pus pe mediul de test, echipa de testare începe **testarea** tuturor funcționalităților sistemelor. Această testare este realizată pentru verificarea faptului că întreg sistemul funcționează conform cerințelor clientului. În timpul acestei faze, QA<sup>2</sup> și echipa de testare pot găsi erori sau defecte care sunt comunicate celor ce s-au ocupat de dezvoltare. Echipa de dezvoltare rezolvă respectivele erori/defecte și le trimite către QA pentru retestare. Acest proces continuă până când sistemul este lipsit de buguri, este stabil și funcționează conform logicii necesare sistemului.

---

<sup>2</sup> Quality Assurance- asigurarea calității

După ce sistemul este lipsit de buguri și nici o eroare nu este lasată în acesta, **instalarea** finală începe, fiind de asemenea verificată și aceasta de erori posibile. Odată având mediul nou cu sistemul instalat, clientul începe a-l folosi, echipa de dezvoltare se va ocupa de **mentenanță** și anume de fixarea erorilor, upgrade-uri ale sistemului și adăugarea unor funcționalități noi.

## 2.6 Modele de dezvoltare a unui sistem și aplicarea testării

Pornind de la fazele ciclului de viață a dezvoltării software, au fost create modele de dezvoltare ce au stat la baza dezvoltării sistemelor:

### 2.6.1 Modelul în cascadă

Modelul în cascadă a fost primul model creat, fiind un model liniar al ciclului de viață. Este ușor de înțeles și de folosit. Fiecare fază trebuie să fie completă înainte ca următoarea să poată fi începută. De obicei acest model este folosit pentru proiectele mici care nu au anumite cerințe sofisticate. La finalul fiecărei faze, o analiză este realizată pentru a determina dacă un proiect este pe calea cea bună și pentru a decide dacă proiectul merită să fie continuat sau închis.

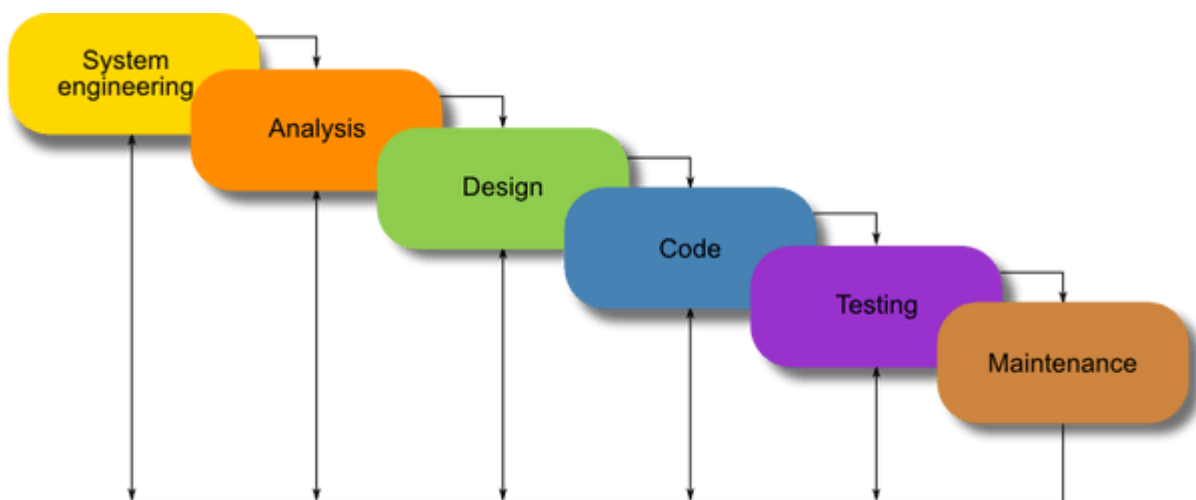


Figura 2 Modelul în cascadă[pag. 27, nr. 11]



Avantajele modelului în cascadă:

- ✚ Modelul este simplu și ușor de folosit
- ✚ Este ușor de întreținut având în vedere rigiditatea modelului – fiecare fază are rezultate și o revizuire specifică
- ✚ În acest model, fiecare fază este procesată complet câte una pe rând
- ✚ Fazele nu se suprapun
- ✚ Modelul funcționează foarte bine pentru proiectele mici atunci când cerințele sunt perfect definite și foarte bine înțelese

Dezavantajele modelului în cascadă:

- ✚ Odată ajuns în faza de testare, este dificilă întoarcerea la faza de implementare pentru a schimba o funcționalitate ce nu a fost gândită bine de la început
- ✚ Nu este produs software utilizabil decât spre finalul ciclului de dezvoltare
- ✚ Risc ridicat și incertitudine
- ✚ Nu este un exemplu bun de model pentru proiectele complexe și orientate-obiect
- ✚ Model slab pentru proiecte de lungă durată, unde cerințele au un risc ridicat de a fi schimbate

În concluzie, modelul în cascadă nu este prielnic testării întrucât acesta rezolvă doar problema descoperirii erorilor din sistem și nu fixarea acestora prin retestare după fixare.

### 2.6.2 Modelul V

Modelul V a fost dezvoltat pentru a aborda unele probleme întâlnite folosind modelul în cascadă. În acest model execuția fazelor se produce într-o manieră secvențială. Este un model extrem de disciplinat, fiecare fază începând decât după terminarea celei precedente. Este o extensie a modelului în cascadă și se bazează pe asocierea fazei de testare cu fiecare fază de dezvoltare corespondentă. Asocierea respectivă aduce un plus acestui model față de modelul în cascadă deoarece erorile apărute pe parcursul oricărei faze, încă de la început, pot fi remediate pe durata fazei respective. În acest mod, validarea calității produsului, la finalul acestuia, crește.

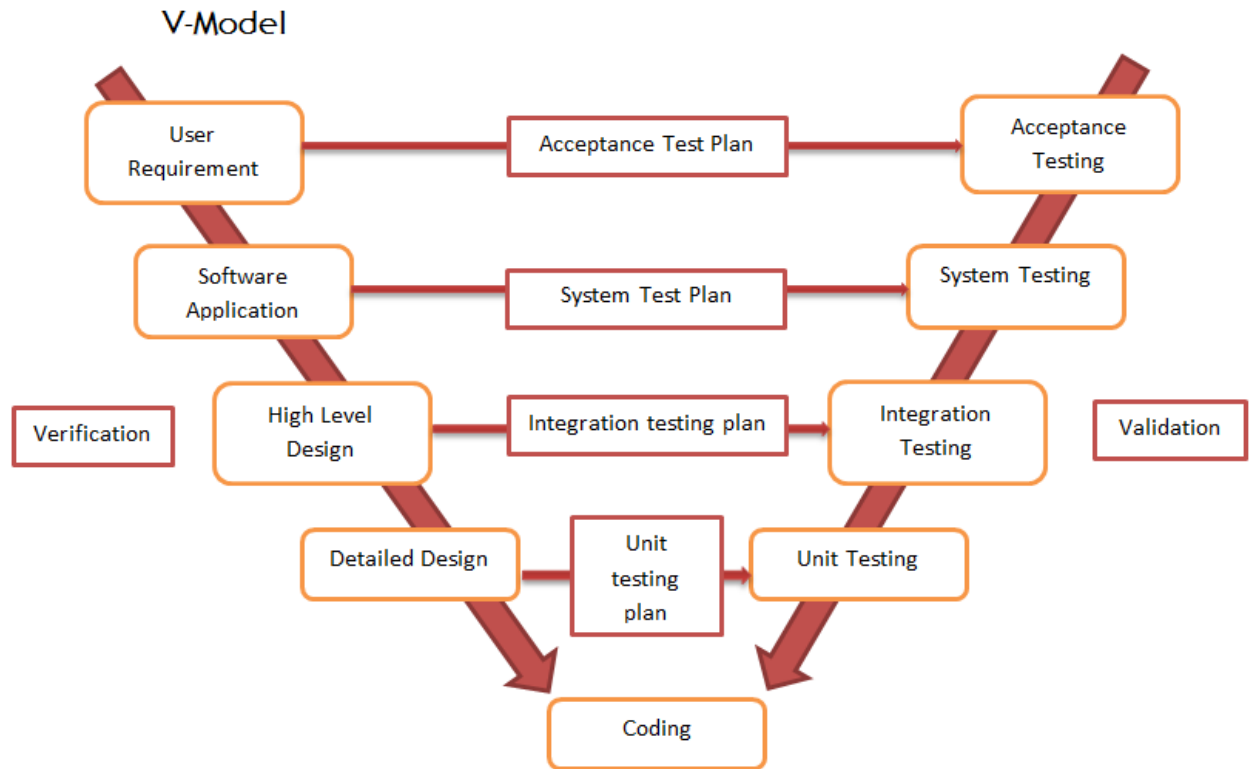


Figura 3 Modelul V [pag. 27, nr.12]

Avantajele modelului V:

- ✚ Model cu un nivel ridicat de disciplină
- ✚ Rată mai mare de succes peste modelul în cascadă datorită startului devreme a testării, încă de la începuturile ciclului de viață al dezvoltării
- ✚ Funcționează foarte bine pentru proiectele mici spre medii unde cerințele sunt ușor de înțeles
- ✚ Ideal pentru gestionarea timpului, pentru proiectele ce trebuie să respecte termene limită de-a lungul dezvoltării

Dezavantajele modelului V:

- ✚ Risc ridicat și incertitudine
- ✚ Nu este un bun model pentru proiectele complexe orientate obiect
- ✚ Model slab pentru proiectele de lungă durată
- ✚ Software-ul funcțional este produs spre finalul ciclului de dezvoltare

### 2.6.3 Modelul Agile

Acest model a fost înaintat pentru a ajuta la dezvoltarea construirii unui proiect ce este adaptabil la schimbări rapide si sofisticate. Modelul se bazează pe dezvoltarea software-ului în segmente scurte si rapide. Ca rezultat al implementării acestui model, sunt posibile lansări frecvente în producție sau fixări ale erorilor la clienții unde acestea au fost întâmpinate. Totodată, testarea se realizează încă de la începuturile ciclului de viață a dezvoltării, acesta repetându-se in fiecare din iterații.

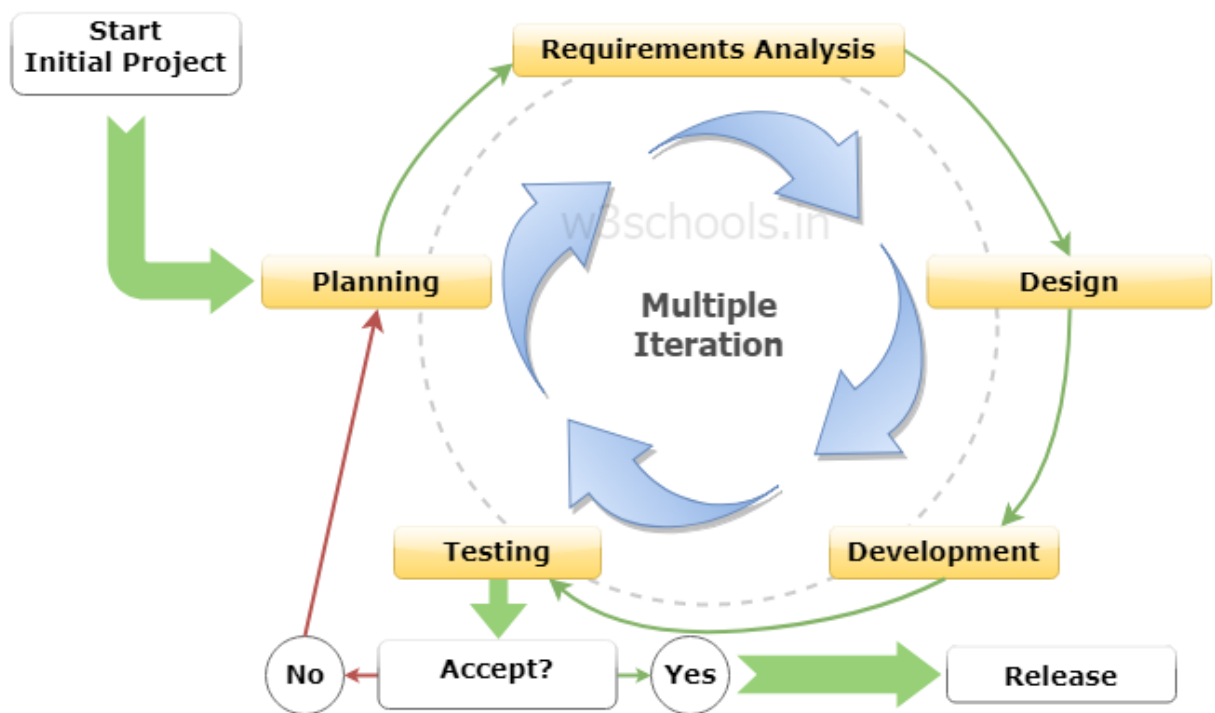


Figura 4 Modelul Agile [pag. 27, nr. 13]

Avantajele modelului Agile:

- ✚ Implică programarea în perechi din echipa de dezvoltatori ce duce la diminuarea erorilor in codul scris și aplatizarea dificultății atribuțiilor revenite unui singur dezvoltator
- ✚ Triază timpul de dezvoltare a unui proiect prin împărțirea acestuia pe iterații, unde sunt repetate fazele ciclului de viață a dezvoltării
- ✚ După fiecare iterație, noua funcționalitate este funcțională și poate fi livrată unui client. De asemenea se conturează logica proiectului, ajutând la modificarea acesteia în cazurile necesare

- ✚ Flexibilitate în ajustarea sistemului
- ✚ Comunicare mai bună cu utilizatorul sistemului
- ✚ Satisfacția clientului îmbunătățită

Dezavantajele modelului Agile:

- ✚ Dificultate în evaluarea efortului și resurselor necesare
- ✚ Accentul este pus mai puțin pe crearea de documentație

## 2.7 Tipuri de testare software

Tipurile de testare au fost introduse pentru a defini clar obiectivele unui anumit nivel de test pentru un program sau proiect. Un anumit tip de test este concentrat pe un obiectiv specific de testare ce poate fi:

- ✚ O funcție realizată de o componentă sau un sistem
- ✚ O caracteristică non-funcțională, cum ar fi fiabilitatea sau intuitivitatea
- ✚ Structura sau arhitectura unui sistem
- ✚ Modificări aduse care confirmă fixarea unor defecte și reverificarea alterării neintenționate a altor părți componente

Un sistem poate fi dezvoltat și testat în mai multe moduri, spre exemplu:

- ✚ Testarea structurală (*white-box*) - controlul fluxului în sistem
- ✚ Testarea non-funcțională - performanță, intuitivitate, securitate
- ✚ Testarea funcțională (*black-box*) - fluxul procesului, tranziția între stări
- ✚ Testarea de regresie – retestarea sistemului după fixarea erorilor rezultate din testare

Testarea structurală poate fi realizată la orice nivel de testare, deși tinde să fie mai des aplicată componentelor și integrării. Ea este des folosită ca o cale de măsurare a rigurozității testării prin acoperirea unui set de elemente structurale sau elemente de acoperire. Acoperirea este măsura pentru care o structura a fost exercitată de o suită de teste. Dacă această acoperire nu atinge 100% atunci trebuiesc construite mai multe teste pentru a acoperi și elementele lipsă netestate din acea structură.

Pentru toate nivelele, dar în special pentru testarea componentelor sau integrării componentelor pot fi folosite instrumente ajutătoare pentru a măsura cât cod este acoperit de elemente precum declarații sau decizii.

Pentru cuantificarea unor caracteristici ale unui sistem sau atributele non-funcționale ale acestuia, se realizează testarea non-funcțională. Acest tip de testare include testarea de performanță, testarea de rezistență la încărcare, intuitivitate, fiabilitate, etc. Sunt măsurați, spre exemplu, timpii de răspundere a sistemului la cererile unui utilizator.

Funcționalitățile unui sistem pot fi descrise ca specificații, cazuri de utilizare sau specificație funcțională sau pot să nu fie documentate. Funcțiile sunt exact ceea ce poate realiza sistemul.

Testarea funcțională se bazează pe funcții și *features*, descrise în documentație sau înțelese de tester și capacitatea sistemului de a face schimbul de informații cu un alt sistem specific (*3rd parties*). Tehnici bazate pe specificațiile date pot fi folosite pentru a deriva condiții și cazuri de testare pentru care sistemul să funcționeze normal, fără a produce erori.

În cazul în care erorile sunt prezente în sistem, acestea vor fi fixate, urmând ca sistemul să treacă prin retestarea integrală, verificând integritatea acestuia.

## **3 Testarea automată**

### **3.1 Definiție**

Realizată corect, testarea automată poate avea multe avantaje și poate fi extrem de benefică unui proiect.

Testarea automată este o metodă în testarea software unde instrumente speciale sunt folosite pentru a controla execuția unui test, la final comparând rezultatele actuale cu cele prezise, așteptate de tester. Toate acestea sunt realizate automat fără intervenția testerului. De asemenea, este un ajutor pentru testarea manuală dificilă și repetitivă.

### **3.2 Beneficiile testării automate**

Setarea așteptărilor și a rezultatelor dorite în urma acțiunii unei funcționalități a aplicației contribuie și este un factor decisiv în construirea testelor automate. În acest fel, testerul va ști mereu când o funcționalitate nu va funcționa adecvat și totodată proveniența bugului va fi ușor de identificat.

Așadar, printre numeroasele beneficii, sunt de menționat:

#### **3.2.1 Livrarea rapidă**

Rezultatele investiției în crearea de teste automate reduce costul testării față de testarea manuală continuă. Distanța de timp între găsirea și fixarea unui defect poate fi mare din cauza găsirii unui bug târziu în ciclul de dezvoltare, fiind apoi nevoită întoarcerea la codul scris, mai vechi, ce trebuie din nou reînvățat. Aici, testarea automată își arată contribuția pe care o aduce.

Cheia este crearea unei legături strânse între dezvoltarea codului și testare a ceea ce ar trebui să facă codul respectiv. De aici se ajunge la reducerea costului de fixare a defectelor, fiind și realizată într-un timp mult mai scurt, ceea ce este un mare avantaj al testelor automate.

### **3.2.2 Reducerea resurselor umane**

Începuturile testelor automate pot fi costisitoare și consumatoare de resurse umane. Continuând, în timp, această investiție începe să își arate profitul, rezultatele fiind reducerea resurselor umane, costul scăzut al fixării defectelor, acestea fiind găsite într-un timp mai scurt, de asemenea și reducerea timpului de testare prin rularea testelor automate oricând este nevoie, chiar și după o implementare nouă sau schimbare a codului.

### **3.2.3 Îmbunătățirea eficienței testării**

Testarea își asumă o bună parte din timpul ciclului de viață al dezvoltării software. De aici deducem că până și cea mai mică îmbunătățire poate avea repercursiuni enorme în timpul total de dezvoltare a unui proiect. Totodată, setarea și crearea testelor vor consuma resurse importante de timp, urmând ca apoi rularea testelor automate să reducă timpul alocat testării semnificativ.

## **3.3 Avantajele testării automate a aplicațiilor web**

Avantajele testării automate a aplicațiilor web sunt:

- ✚ Îmbunătățește acuratețea testării față de testarea făcută direct de către om
- ✚ Crește acoperirea testării, fiind posibilă rularea testelor în paralel pentru diferite scenarii de testare
- ✚ Ajută la găsirea bug-urilor mult mai repede, în special a celor de regresie, atunci când zone vechi ale aplicației sunt modificate
- ✚ Eliminarea erorii umane prin repetarea strictă a acelorași pași, nefăcând posibilă ratarea unuia dintre ei
- ✚ Reducerea costului de personal, testarea de regresie neavând nevoie de interacțiune umană
- ✚ Rapiditate a execuției testelor

### 3.4 Dezavantajele testării automate a aplicațiilor web

Printre dezavantajele testării automate a aplicațiilor web se numără:

- ✚ Învățarea unui framework de testare pentru a putea scrie teste
- ✚ Mentenanța testelor în cazul unor schimbări, chiar și minore, mai ales în interfața aplicației



## 4 Prezentarea proiectului

Proiectul se prezintă ca o bază în crearea unei soluții ample de teste automate, pentru o aplicație WEB, ce dorește a simula un utilizator uman. Această simulare este realizată prin interacțiunea codului cu un browser prestabilit, paginile aplicațiilor și respectiv elementele componente prezente în fiecare pagină.

### 4.1 Tehnologii folosite

#### 4.1.1 .NET Framework

.NET este un *framework* de programare ce facilitează construcția și administrarea unei aplicații, aceasta fiind robustă și cu rezultate înalte. Suportă o multitudine de limbaje de programare, C# fiind limbajul ales pentru a-mi crea soluția. Prezintă o gamă largă de librării și totodată compatibilitate între limbajele de programare.

#### 4.1.2 C#

Este un limbaj elegant de programare, orientat obiect, ce permite dezvoltarea unor aplicații complexe, robuste, ce rulează pe .NET *framework*. Poate fi folosit pentru a crea aplicații Windows, servicii Web, aplicații client-server, proiecte pentru testare, etc.

Întreaga soluție dezvoltată se bazează pe acest limbaj de programare întrucât mi-a facilitat nevoile avute în scopul dezvoltării proiectului prin gruparea, declararea și folosirea tuturor elementelor ce îl constituie.

#### 4.1.3 Selenium

Selenium este un set de instrumente software, fiecare cu o abordare diferită față de cealaltă în scopul de a facilita automatizarea testelor. Întreaga suită de instrumente conține diferite funcții de testare special create pentru testarea aplicațiilor Web de diverse tipuri.

Aceste funcții sunt extrem de flexibile, permițând diverse opțiuni de identificare a elementelor în interfața grafică precum și compararea rezultatelor așteptate cu cele actuale produse de comportamentul aplicației. Una dintre caracteristicile cheie ale framework-ului Selenium este execuția unui singur test pe browsere diferite.

## 4.2 Structură/Arhitectură

Pentru a fi înțeles și elaborat cu ușurință, proiectul respectă o structură formată din 3 straturi (*layers*). Această structură presupune independența fiecărui layer, putând fi accesat din exterior pentru interacțiunea cu un alt layer.

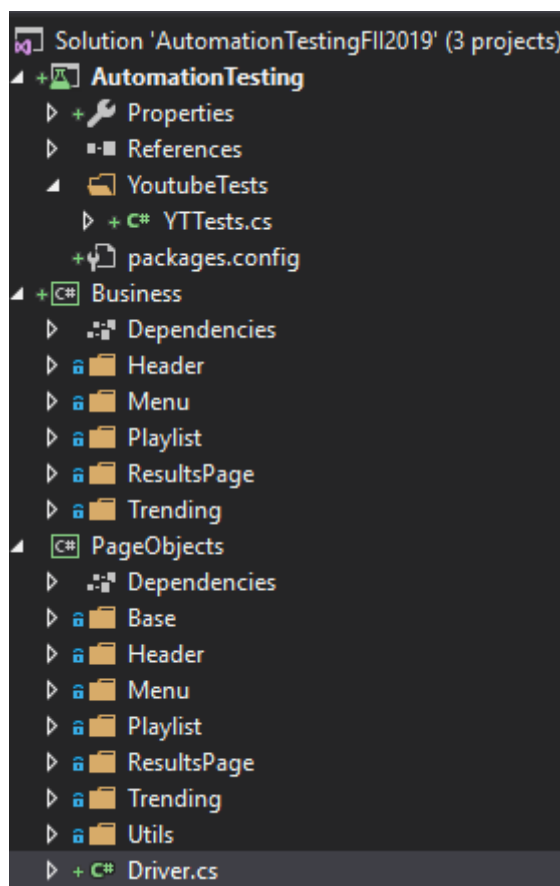


Figura 5 Structura proiectului

- ✚ PageObjects – Acest layer cuprinde baza, startul unui test. Aici vor fi stocate paginile aplicației împreună cu elementele aferente fiecărei pagini. Un conținut important în acest layer îl constituie folderul *Base*, clasa *Driver* și clasa *Framework*
- ✚ Business – Se ocupă cu logica necesară realizării acțiunilor pe care aplicația le pune la dispoziție. Layerul acest va interacționa cu PageObjects, de unde se va folosi de elementele identificate în paginile Web pentru a crea acele acțiuni
- ✚ AutomationTesting – Ultimul layer este cel de teste. Folosește layer-ul *Business*, de unde va prelua metodele care vor construi funcționalitatea vizată pentru testare.

## 4.3 Module ale proiectului

### 4.3.1 PageObjects

Modulul *PageObjects* pornește cu construirea clasei *Driver*. În această clasă se află o proprietate, de tipul *IWebDriver*, numită *Browser*. Accesorul *get* va face o verificare preliminarie pentru existența unei instanțe de *Browser*.

Pe lângă această proprietate, metoda *StartBrowser* va conține un parametru de tip Enum, numit *BrowserType* și va avea rolul de a crea o nouă instanță a browserului setat în parametru. Instanța nouă va fi creată cu posibile opțiuni cum ar fi: incognito, full screen etc.

Folderul Base este format din 3 clase de bază: *BasePage*, *BasePageElementMap* și *BasePageValidator*:

- ✚ **BasePage** este declarată de două ori: prima declarare este compusă din tipul TM (tipul model), unde *TM* este clasa de baza destinată elementelor. Metoda de tipul *TM* numită *Map* va returna o nouă instanță a *TM*, facilitând accesarea elementelor *UI* identificate.

O a doua declarare a clasei este compusă din 2 tipuri, *TM* și *TV* (tipul validator), *TM* fiind clasa de baza a elementelor iar *TV* clasa de bază pentru validările în pagină. Aceasta moștenește din prima declarare a *BasePage* și conține metoda *Validate* ce va returna nouă instanță a *TV*, facilitând accesul metodelor de validare a paginilor.

- ✚ **BasePageElementMap** este clasa de bază a construirii claselor cu pagini din aplicație ce o vor moșteni, ce va primi în constructor pentru proprietatea *Browser* valoarea din clasa *Driver* ( instanța de browser creată ). În acest mod, fiecare clasă a unei pagini din aplicație va avea acces la browser, fapt ce oferă identificarea elementelor în interfață prin locatorii *HTML*.

- ✚ **BasePageValidator** permite accesul la *BasePageElementMap* prin moștenire, putând fi create metode de validare pentru funcționalitățile aplicației.

Pe lângă folderul Base, fiecare pagină sau componentă din interfață își are folderul individual ce conține:

- I. Clasa elementelor din interfața aplicației identificate prin locatorii HTML
- II. Clasa metodelor de validare în urma acțiunilor din aplicație

**Clasa elementelor din interfața aplicației** este baza interacțiunii cu paginile acesteia.

Identificarea și localizarea elementelor se face prin metoda *FindElement*. Această metodă poate fi folosită în mai multe moduri, factorul decisiv fiind codul HTML.

```
public IWebElement btnSearch => Browser.FindElement(By.Id("search-icon"));
public IWebElement SearchBox => Browser.FindElement(By.CssSelector("[name='search_query']"));
public IWebElement favplaylistChecked => Browser.FindElement(By.XPath(@"//yt-formatted-string[@title='Favourites']
//parent::div//parent::div//parent::div
//parent::div//parent::div//parent::div
//preceding-sibling::div[@id='checkboxContainer']
//child::div[1]"));
```

Figura 6 Tipuri de locatori a elementelor în HTML

Prioritatea îl are id-ul unui element. Acesta este unic și întotdeauna un element caruia îi este atribuit un id în codul HTML va fi ușor de identificat. Ca și simplitate plus performanță, CssSelector-ul este preferat. De cele mai multe ori se bazează pe numele și valoarea unor attribute din HTML.

Un ultim locator, este XPath-ul. Acesta este destul de instabil, posibile modificări ale codului HTML pot modifica decizia unui test dacă funcționalitatea în cauză este validă sau nu.

**Clasa metodelor de validare** constituie finalul testului deoarece este formată din metode ce verifică anumite elemente sau o structură din pagina respectiva. Verificarea este făcută prin *Assert*, clasă a pachetului Microsoft pentru *UnitTesting*. Această clasă are diverse metode prin care sunt comparate rezultatele așteptate cu cele returnate din simularea utilizatorului în aplicație.

```
public void CheckVideoTitle(int songNumber, string songTitleToBeChecked)
{
    Assert.AreEqual(songTitleToBeChecked, Map.SongTitle[songNumber].Text);
}
```

rezultat așteptat                      rezultat actual

Figura 7 Metodă de validare a unei acțiuni

Un plus în această soluție îl constituie și clasa **Framework**. Pentru a putea interacționa cu un element, Selenium are nevoie ca acesta să fie prezent în DOM(*Document Object Model* ). Pentru aceasta, clasa Framework conține metode ajutătoare pentru interacțiunea unui element, metode ce așteaptă până când acesta este prezent în DOM printre care se numără metoda de click pe un element, introducerea și stergerea textului dintr-un element de tip input.

### 4.3.2 Business

Modulul Business este al doilea layer din soluție. Acesta moștenește clasa de bază *BasePage*, cea compusă din cele 2 tipuri, tipul model (TM) pentru accesul către elementele identificate în pagina și tipul validator(TV) pentru accesul către metodele de validare a acțiunilor efectuate în interfață.

În acest *layer* fiecare pagină creată în PageObjects va avea corespondentul său în Business. Aici se va construi simularea acțiunilor în aplicație, prin interacțiunea cu elementele prezente în pagina respectivă.

```
public ResultsPageBL SearchSong(string songName)
{
    EnterText(Map.SearchBox, songName);
    ClickElement(Map.btnSearch);
    return new ResultsPageBL();
}
```

Figura 8 Metodă de simulare a acțiunii de cautare a unei piese

### 4.3.3 AutomationTesting

Ultimul layer al soluției îl reprezintă însuși testele automate, construcția completă a simulării utilizatorului, adăugând și validările necesare pentru a putea fi un test complet. Referințe noi către paginile de logică sunt create în clasa respectivă de teste, urmând ca utilizarea unei metode de logică implementată în Business să se realizeze prin referința respectivă.

## 4.4 Desfășurarea testelor

Pentru ca un test să fie executabil, în primul rând va avea nevoie de un browser pe care să își înceapă execuția. El va returna eroare în cazul în care nu va fi găsit instalat browserul setat din test să fie deschis. Deci, browserul favorit va fi nevoit să fie instalat în locul respectiv execuției testului.

Următorul pas este navigarea în browser către aplicația dorită. Aceasta se face simplu prin metoda *Navigate()*, conținută de *IWebBrowser*. Pașii următori sunt începerea acțiunilor pe care utilizatorul le-a descris în Business și le-a apelat în testul respectiv.

Partea de final este reprezentată de verificarea rezultatelor după ce acțiunile apelate au avut loc. Rezultatele actuale, returnate de acțiuni, sunt comparate cu rezultatele prestabilite de la începuturile proiectului, din faza de proiectare.

## 4.5 Exemplificarea structurii pe o aplicație WEB

Pentru a exemplifica soluția pe o aplicație reală și pentru a arăta motivul îndrumării către această structură, ținta aleasă a fost YouTube. Pentru aceasta am realizat cateva teste, unele de bază iar altele vizând o logică complexă ale acestei aplicații. Ele pot fi găsite în fereastra de Test Explorer al Visual Studio.

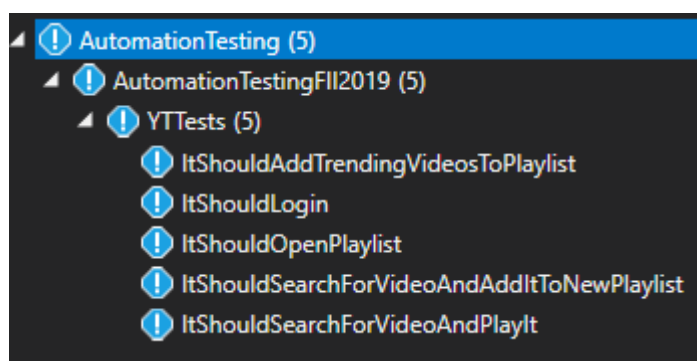


Figura 9 Teste automate în Visual Studio

Fiecare test are vizată o parte a platformei, o anumită funcționalitate. Prin aceste teste este verificată logarea cu un cont existent în aceasta platformă, funcționalitatea de căutare a unui videoclip, cu posibilitatea de redare a acestuia. După aceasta, utilizatorul se poate opri sau poate continua testul în care video-ul respectiv este adăugat într-un playlist predefinit.

Testul “ItShouldAddTrendingVideosToPlaylist” abordează mai în adâncime funcționalitatea YouTube, fiind și un test end-to-end. Etapele acestuia sunt logarea cu un cont existent pe această platformă, redirecționarea către videoclipurile ce se află în trending-ul legat de muzică ca apoi, după o anumită listă de artiști declarată în test, să facă o cautare după numele fiecărei melodii din trending și să o adauge în playlistul Favourites. Înainte de adăugare, fiecărei melodii selectată pentru a fi adăugată îi este verificată prezența în acel playlist. În funcție de această verificare, melodia este adăugată sau nu. În final, playlistul în cauză este verificat dacă melodiile ce au îndeplinit condițiile testului au fost adăugate în acest playlist.

## 4.6 Complexități și impedimente

Întrucât realizarea testelor automate se bazează pe interacțiunea cu elementele prezente într-o pagină, complexitatea identificării unui element dintr-o anumită pagină a putut fi luată în considerare. În alegerea făcută, pe platforma Youtube am întâmpinat unele dificultăți în identificarea elementelor din pagina întrucât codul HTML al YouTube este destul de generic. De aceea, locatorii elementelor sunt bazați în mare parte pe CssSelector apoi, în cazuri extreme, identificarea elementelor s-a făcut prin selectorul XPath.

De exemplu, pentru verificarea unei melodii dacă există deja într-un playlist a fost nevoie de asocierea a două elemente, checkbox-ul și numele playlisturi. Fiind, ca structură HTML, frați, a fost nevoie de selectorul XPath ce a căutat elementul cu numele playlistului dorit, ce are ca părinte elementul de tip checkbox + numele playlistului, ca apoi să poată fi verificată proprietatea de pe elementul checkbox pentru a vedea dacă melodia este deja prezentă în playlistul respectiv.

```
public WebElement favplaylistChecked => Browser.FindElement(By.XPath(@"//yt-formatted-string[@title='Favourites']  
//parent::div//parent::div//parent::div  
//parent::div//parent::paper-checkbox  
//preceding-sibling::div[@id='checkboxContainer']  
//child::div[1]"));
```

Figura 10 Identificarea unui checkbox pentru verificarea prezenței unei melodii într-un anumit playlist

## 5 Concluzii

Această lucrare pune în evidență rolul testării automate dar oferă și informații despre modul în care se realizează testarea manuală și cum se poate aplica pe un proiect. De asemenea prezintă importanța, în general, a testării software în scopul de a preveni eventualele erori ce pot deveni catastrofale.

Prin lucrarea practică, ea devine ajutorul, baza unui proiect de teste automate ce poate fi folosit pentru majoritatea aplicațiilor WEB. De asemenea poate fi extins în funcție de necesitățile fiecărui tester. Ajungându-se la o acoperire ridicată a funcționalităților dintr-o aplicație, noi modificări aduse vechiului cod sau chiar modificări ale interfeței utilizatorului vor putea fi verificate mai ușor, mai rapid, eliminând într-o oarecare măsură eroarea umană care ar putea evita un anumit scenariu de testare, astfel ajungându-se la buguri nedescoperite.

Ca îmbunătățiri, soluția pentru testele automate poate fi adaptată la performanța aplicației, prin prelungirea sau scurtarea timpilor de așteptare ca o pagină WEB să fie încărcată complet. Se poate adăuga separat și un proiect pentru utilitarele proiectului, și anume metodele de interacțiune cu un element în pagina WEB. Astfel este posibilă o organizare mai bună și a acestei părți a soluției.



## 6 Bibliografie

1. Dorothy Graham, Erik van Veenendaal, Isabel Evans, Rex Black (2006), "FOUNDATIONS OF SOFTWARE TESTING", Cengage Learning Emea
2. Elfriede Dustin, Thom Garrett, Bernie Gauf (2009), "Implementing Automated Software Testing", Addison-Wesley Professional
3. [https://www.seleniumhq.org/docs/06\\_test\\_design\\_considerations.jsp#page-object-design-pattern](https://www.seleniumhq.org/docs/06_test_design_considerations.jsp#page-object-design-pattern)
4. <https://www.techopedia.com/definition/17785/automated-testing>
5. [https://ro.wikipedia.org/wiki/Testare\\_software](https://ro.wikipedia.org/wiki/Testare_software)
6. <https://www.guru99.com/software-development-life-cycle-tutorial.html>
7. [www.testingexcellence.com](http://www.testingexcellence.com)
8. [www.techbeacon.com](http://www.techbeacon.com)
9. [www.softwaretesting.ro](http://www.softwaretesting.ro)
10. [https://www.google.com/search?q=sdlc&tbm=isch&source=iu&ictx=1&fir=lgHePlfhwZHdLM%253A%252CezcLWKtuFYHPgM%252C\\_&vet=1&usg=AI4\\_-kSwxlgpmEYE6QfgIFi5Cm5WugzGwA&sa=X&ved=2ahUKEwicueDIpY\\_jAhXDrHEKHdCfCRAQ9QEwAnoECAoQCA#imgsrc=lgHePlfhwZHdLM:](https://www.google.com/search?q=sdlc&tbm=isch&source=iu&ictx=1&fir=lgHePlfhwZHdLM%253A%252CezcLWKtuFYHPgM%252C_&vet=1&usg=AI4_-kSwxlgpmEYE6QfgIFi5Cm5WugzGwA&sa=X&ved=2ahUKEwicueDIpY_jAhXDrHEKHdCfCRAQ9QEwAnoECAoQCA#imgsrc=lgHePlfhwZHdLM:)
11. [https://www.google.com/search?biw=1366&bih=635&tbm=isch&sa=1&ei=JawXXa6OGYb4aYOBnbgF&q=sdlc+cascade&oq=sdlc+cascade&gs\\_l=img.3...2256.4135..4228...0.0..0.131.1292.0j11.....0....1..gws-wiz-img.....35i39j0j0i67j0i24j0i10i24.n9DBtUQOfMQ#imgsrc=sZgSiQXaNsyRBM:](https://www.google.com/search?biw=1366&bih=635&tbm=isch&sa=1&ei=JawXXa6OGYb4aYOBnbgF&q=sdlc+cascade&oq=sdlc+cascade&gs_l=img.3...2256.4135..4228...0.0..0.131.1292.0j11.....0....1..gws-wiz-img.....35i39j0j0i67j0i24j0i10i24.n9DBtUQOfMQ#imgsrc=sZgSiQXaNsyRBM:)
12. [https://www.google.com/search?q=v+model&source=lnms&tbm=isch&sa=X&ved=0ahUKEwiK-oTNp4\\_jAhXfQxUIHbyTBZwQ\\_AUIECgB&biw=1366&bih=635#imgsrc=3YgQQC6sPzndXM:](https://www.google.com/search?q=v+model&source=lnms&tbm=isch&sa=X&ved=0ahUKEwiK-oTNp4_jAhXfQxUIHbyTBZwQ_AUIECgB&biw=1366&bih=635#imgsrc=3YgQQC6sPzndXM:)
13. [https://www.google.com/search?q=v+model&source=lnms&tbm=isch&sa=X&ved=0ahUKEwiK-oTNp4\\_jAhXfQxUIHbyTBZwQ\\_AUIECgB&biw=1366&bih=635#imgsrc=3YgQQC6sPzndXM:](https://www.google.com/search?q=v+model&source=lnms&tbm=isch&sa=X&ved=0ahUKEwiK-oTNp4_jAhXfQxUIHbyTBZwQ_AUIECgB&biw=1366&bih=635#imgsrc=3YgQQC6sPzndXM:)