PROIECT FINAL

TIMOFTE ALEXANDRU BOGDAN

DATA EXAMEN 10.04.2025

INTRODUCERE

Există două discipline diferite implicate în testarea software: **testarea manuală și testarea automată**. În ciuda faptului că ambele au efectiv aceeași funcție, ele sunt discipline distincte pe care companiile le folosesc pentru a-și examina pachetele software.

- Ce este **testarea automata** si ce este **testarea manuala?**

- **Testarea automată** este procesul prin care un tester folosește un instrument terț pentru a automatiza o bucată de software, examinând software-ul în timp ce acesta efectuează în mod repetat același proces pentru a se asigura că funcționează la un standard suficient de ridicat pentru o organizație. Principalul beneficiu al automatizării testelor este că este un proces mult mai rapid, în special atunci când se realizează sarcini minore, cum ar fi introducerea datelor.

- **Testarea manuală** este un tip de testare a software-ului în care un caz de testare este executat manual de către tester, fără ajutorul unor instrumente automate.

- **Rolul unui QA Tester:**
- - Identifică defecte
- - Testează funcționalitățile
- - Raportează bug-uri

TIPURI DE TESTARE SOFTWARE

IN PRACTICA TESTAREA SOFTWARE ESTE IMPARTITA IN FOARTE MULTE TIPURI. IN FUNCTIE DE NATURA SI SCOPUL APLICATIEI TESTATE DECIDEM CE FEL DE TESTE TREBUIE SA EXECUTAM.ACESTE TIPURI DE TESTE POT FI IMPARTITE IN DOUA MARI CATEGORII: TESTE FUNCTIONALE SI TESTE NEFUNCTIONALE.

Cateva exemple din categoria testelor functionale sunt:

- Unit testing
- Integration testing
- System testing
- Sanity testing
- Smoke testing
- Interface testing
- Regression testing
- Acceptance testing
- Black box testing
- White box testing

Cateva exemple din categoria testelor nefunctionale sunt:

- Performance testing
- Load testing
- Volume testing
- Stress testing
- Security testing
- Compatibility testing
- Penetration testing

Cerințe privind software-ul

- Un tester trebuie să aibă acces la cerințele software-ului. Aceasta nu se referă la hardware-ul sau sistemul de operare de care are nevoie pachetul, ci mai degrabă la briefingul pentru software-ul la care lucrează dezvoltatorul.

- Având cerințe software mai detaliate în etapa de testare, personalul de asigurare a calității caută toate caracteristicile importante încă de la început, notând unde există probleme în software și recomandând ajustări.

- Fără aceasta, un tester lucrează fără nicio îndrumare și nu știe dacă informațiile pe care le furnizează sunt utile pentru echipa de dezvoltare.

Procesul de testare contine mai multe etape:

- Test planning
- Test control
- Requirements analysis and test design
- Test creation
- Test execution
- Evaluating exit criteria and reporting
- Test closure activities.

1.Test planning (Planul de testare)

În perioada de test planning ne asigurăm că înțelegem obiectivele clienților/proiectului și riscurile asociate testării.Vom urma pasii:

- **Scopul și obiective de testare** (ex: dorim ca prin activitățile de testare să ne asigurăm că software-ul îndeplinește cerințele tehnice, funcționale și de business)
- **Ce se testează**: descrierea software-ului sau a componentelor unui software care trebuie să fie testate
- **Test approach**: se determină tehnicile de testare, cum se va desfășura testarea, cât de mult trebuie testat (ce acoperire să aibă testele)
- **Resurse**: persoanele care vor testa, test environment, echipamente hardware și identificarea oricăror altor nevoi care pot să apară pe parcursul testării
- În funcție de ce Software Development Life Cycle este folosit, se încadrează activitățile de testare într-un **orizont de timp** sau se stabilește contextul în care începe testarea
- **Riscuri**
- **Exit criteria** – se stabilesc criterii care trebuie îndeplinite pentru a putea considera testarea completă (exemplu: test coverage de 70%)

2. Test control (Control asupra testarii)

Test control-ul este o activitate care se desfășoară pe toată perioada testării și presupune că mereu se compară progresul actual cu cel planificat (Test Plan) și se iau măsurile necesare

- **Se analizează rezultatele testelor** (câte teste sunt passed/failed, considerându-se și prioritatea lor)
- **Se monitorizează test coverage-ul**
- **Se documentează progresul** și se transmit rapoarte către manager sau/și echipă
- **Se iau măsuri când este necesar** (exemple: se prioritizează defectele care au severitate ridicată, se testează mai mult unde s-au descoperit/corectat defecte, etc)

3. Requirements analysis and test design (Analiza)

În această etapă se analizează requirement-urile, se indentifică elementele care pot fi testate și se schițează Test Case-urile.

•**Se analizează requirement-urile** cu scopul de a:

- Identifica funcționalități care trebuie (și pot fi) testate
- Identifica lipsurile și ambiguitățile și pentru a cere clarificări (o eroare nedescoperită în requirements poate duce la implementarea greșită a software-ului)

•**Se crează *high-level Test Cases*** – este mai degrabă o listă cu Test Case-uri care urmează a fi dezvoltate mai târziu. Prima dată ne creăm o imagine de ansamblu și planificăm ce avem de făcut, abia apoi trecem la crearea lor (exemplu de high-level Test Case: "User-ul se poate loga pe site folosind credențialele corecte")

•Test Case-urile care acopera o anumită funcționalitate se grupează în ***Test Suites*** (spre exemplu, testele pentru Login pot fi grupate împreună, testele pentru adăugarea unui produs în coșul de cumpărături pot reprezenta un alt Test Suite)

•**Se identifică alte nevoi**, spre exemplu **tool-uri** (și se modifică Test Plan-ul dacă este necesar)

4. Test creation (Crearea testelor)

Pe baza Test Case-urilor denivel inalt dezvoltate anterior se începe crearea *Test Case-urilor denivel scazut* (descriu detaliat cazul de test, conțin rezultate asteptate etc).

Exemple de activități:

•**Crearea testelor manuale** (conțin descriere, pași de test, expected result și actual result)

•**Crearea testelor automatizate** (dacă e cazul)

•**Prioritizarea testelor**

•**Pregătirea/verificarea test environment-ului**

5. Test execution (Executia de teste)

- **Executarea Test Case-urilor**, fie ele manuale sau automate
- **Compararea rezultatelor obținute** (actual results) cu cele așteptate (expected results) – în cazul testelor manuale
- **Verificarea rezultatelor obținute** în urma executării testelor automate – dacă sunt teste care au picat atunci trebuie investigată cauza (testul e de vină sau chiar e un bug?)
- **Examinarea anomaliilor și a potențialelor defecte/bug-uri**
- **Logarea rezultatelor** în Test Case-uri. Exemplu: Passed (testul a trecut) / Failed (testul a picat)
- **Raportarea bug-urilor** – este recomandat ca de îndată ce suntem siguri că avem de-a face cu un bug, să îl și raportăm chiar dacă mai avem de testat

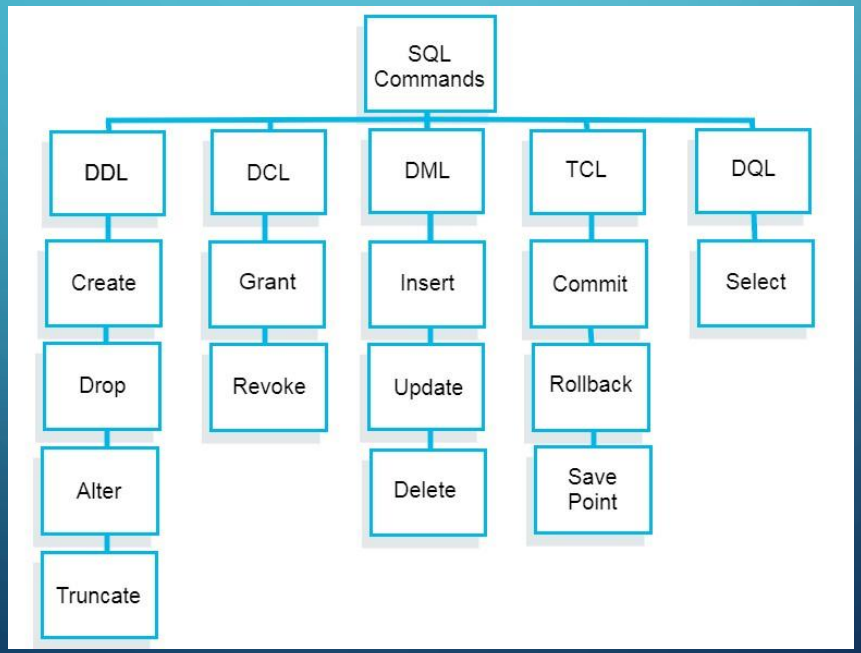
6. Test closure activities (Finalizarea procesului de testare)

- **Raportarea rezultatelor** prin crearea unui document numit *Test Report* (periodic sau la finalul perioadei de testare) – acest document va fi folosit de echipă, manager, client, etc
- **Ne asigurăm că toate rapoartele bug-urilor au fost procesate** (fie corectate și retestate, fie amânate pentru un release următor sau respinse din motive obiective)
- **Se face o retrospectivă pentru a identifica aspecte care ar putea fi îmbunătățite în viitor.**

Un alt tool in programare utilizat de aproape toate bazele de date relaționale, pentru interogarea, gestionarea și definirea datelor, precum și pentru controlul oferirii accesului este MySQL

- MySQL este un sistem open source de gestionare a bazelor de date relaționale bazat pe SQL. A fost proiectat și optimizat pentru aplicațiile web și poate rula pe orice platformă. Odată cu apariția noilor cerințe de pe internet, MySQL a devenit platforma preferată a dezvoltatorilor web și pentru dezvoltarea aplicațiilor bazate pe web. Deoarece este conceput pentru a procesa milioane de interogări și mii de tranzacții, MySQL reprezintă o opțiune populară pentru companiile de comerț electronic, care trebuie să gestioneze numeroase transferuri de bani. Caracteristica principală a MySQL este flexibilitatea on-demand.

Comenzile esențiale necesare pentru crearea, gestionarea și manipularea bazelor de date sunt:



1. SELECT

Instrucțiunea SELECT este utilizată pentru a extrage date dintr-o bază de date. Puteți specifica ce coloane să selectați și ce tabel să interogați:

```
SELECT column1, column2 FROM nume_tabel;
```

Exemplu:

```
SELECT first_name, last_name FROM employees;
```

2. INSERT INTO

Instrucțiunea INSERT INTO este utilizată pentru a introduce înregistrări noi într-un tabel:

```
INSERT INTO table_name (column1, column2) VALUES (value1, value2);
```

Exemplu:

```
INSERT INTO employees (first_name, last_name) VALUES ("John", "Doe");
```

3. UPDATE

Instrucțiunea UPDATE este utilizată pentru a modifica înregistrările existente într-un tabel:

```
UPDATE table_name SET column1 = value1 WHERE condition;
```

Exemplu:

```
UPDATE employees SET last_name = 'Smith' WHERE id = 1;
```

4. DELETE

Instrucțiunea DELETE este utilizată pentru a elimina înregistrări dintr-un tabel:

```
DELETE FROM nume_tabel WHERE condiție;
```

Exemplu:

```
DELETE FROM employees WHERE id = 1;
```

5. CREATE TABLE

Instrucțiunea CREATE TABLE este utilizată pentru a crea un tabel nou în baza de date:

```
CREATE TABLE table_name ( column1 datatype, column2 datatype, ... );
```

Exemplu:

```
CREATE TABLE employees ( id INT PRIMARY KEY, first_name VARCHAR(50), last_name

VARCHAR(50) );
```

6. ALTER TABLE

Instrucțiunea ALTER TABLE este utilizată pentru a modifica un tabel existent, cum ar fi adăugarea sau ștergerea de coloane:

```
ALTER TABLE table_name ADD column_name datatype;
```

Exemplu:

```
ALTER TABLE employees ADD email VARCHAR(100);
```

7. DROP TABLE

Instrucțiunea DROP TABLE este utilizată pentru a șterge un întreg tabel din baza de date:

```
DROP TABLE nume_tabel;
```

Exemplu:

```
DROP TABLE employees;
```

8. Clauza WHERE

Clauza WHERE este utilizată pentru a filtra înregistrările dintr-o interogare pe baza unei condiții:

Exemplu:

```
SELECT * FROM employees WHERE first_name = 'John';
```

- Aceste comenzi SQL de bază constituie fundamentul gestionării bazelor de date și al manipulării datelor. Prin stăpânirea acestor comenzi, puteți crea, actualiza, șterge și interoga date în baze de date relaționale. Pe măsură ce dobândiți mai multă experiență, puteți explora funcții SQL mai avansate, cum ar fi uniunile, indexurile și tranzacțiile.

- In procesul de testare se vor folosi diferite tool-uri pentru a ne face munca mai usoara si de a simplifica trecerea de informatii de la un departament la altul.

- De exemplu pentru pentru managementul proiectelor unul dintre cele mai folosite programe este **JIRA**

Caracteristicile **Jira** sunt:

•**Management centralizat al testelor**: Integrează testarea cu Jira pentru a alinia echipele de QA, dezvoltare și produse.

•**Generare de cazuri de testare bazată pe inteligență artificială**: Generează automat cazuri de testare din cerințe folosind AI.

•**Raportare în timp real**: Oferă perspective imediate cu analize detaliate despre execuția testului.

•**Gestionare flexibilă a cazurilor de testare**: Acceptă crearea, organizarea și gestionarea cazurilor de testare cu funcții precum clonarea și actualizarea în bloc.

•**Integrarea instrumentelor terțe**: Permite gestionarea testării manuale și automate în cadrul Jira prin integrarea instrumentelor externe.

În Jira, structura ierarhică a elementelor de lucru (issues) este organizată astfel:

1.Epic – O unitate mare de lucru care conține mai multe Stories, Tasks sau Bugs. Reprezintă o funcționalitate majoră sau un obiectiv mare.

2.Story – O cerință funcțională sau o caracteristică care trebuie implementată. Este o unitate de lucru mai mică dintr-un Epic.

3.Task – O sarcină independentă care trebuie realizată. Poate face parte dintr-un Epic sau poate fi un element de sine stătător.

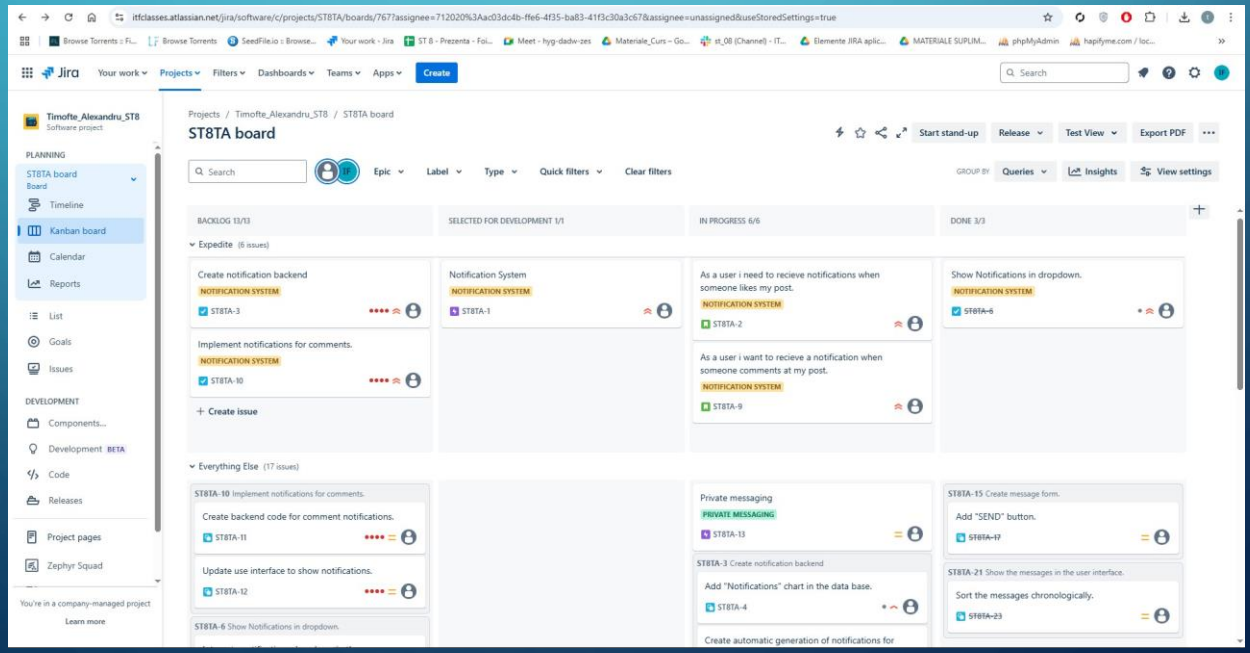
4.Sub-task – O subdiviziune a unui Task sau Story, utilizată pentru a împărți lucrul în părți mai mici.

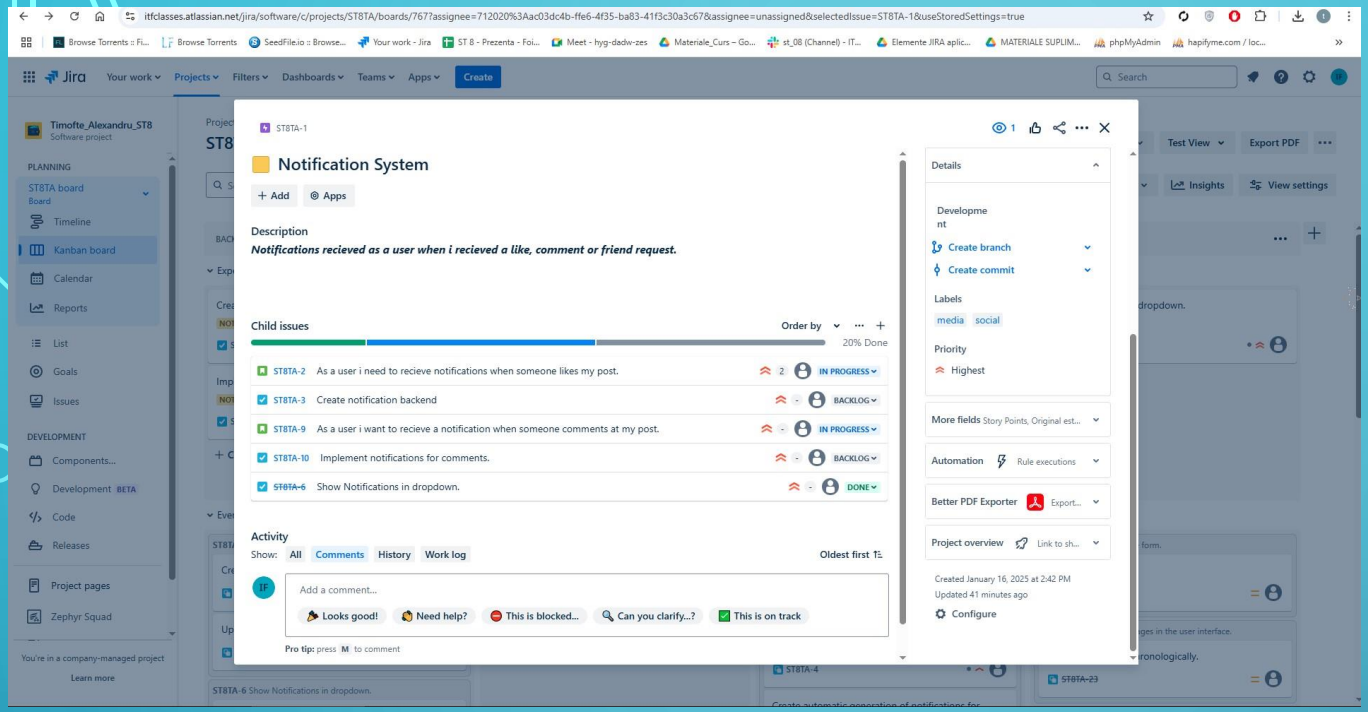
5.Bug – O eroare sau problemă care trebuie remediată. Poate fi asociat cu un Story, Task sau Epic.

6.Spike – Un task special utilizat pentru cercetare sau investigație, folosit adesea pentru a clarifica cerințele înainte de dezvoltare.

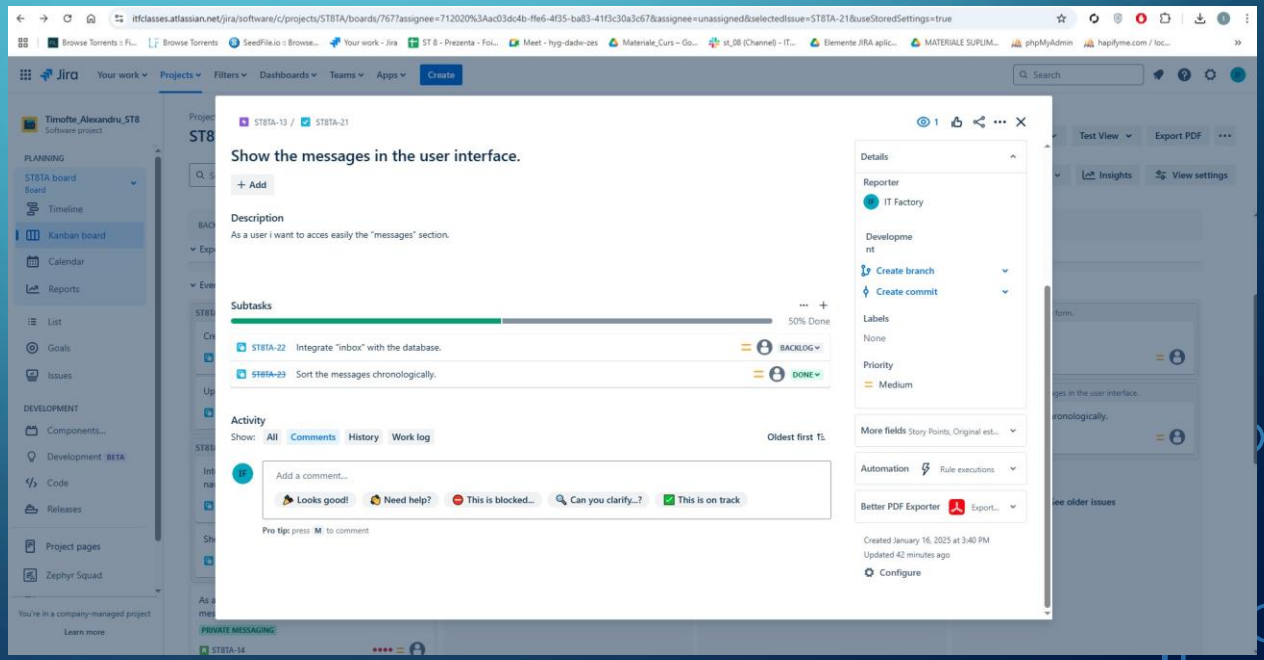
Această ierarhie ajută echipele să organizeze eficient proiectele și să urmărească progresul fiecărei componente.

Pentru urmatoarele exemple din proiect a fost folosita aplicatia https://test.hapifyme.com/index.php.





Acces catre GitHub



Mulțumesc pentru atenție!