Laborator 12

1. Obiectivele lucrării de laborator

- Elaborarea unui plan de testare
- Efectuarea unui test experimental de penetrare a securității (engl., penetration test)
- Execuția planului de testare

2. Procesul de management al testării

Pentru asigurarea calității unui produs software, pe lângă activitățile uzuale de proiectare și dezvoltare are loc și un proces de testare și validare al sistemului rezultat. Deși în general s-ar crede că dezvoltarea unui sistem este mai complexă și mai greu de planificat, testarea acestuia nu este cu nimic mai simplă.

Procesul de management al testării software este alcătuit din mai multe faze sau etape care au loc de-a lungul ciclului de dezvoltare și testare al unei aplicații software. Aceste faze urmează un proces bine definit pentru a asigura testarea și validarea eficientă a aplicației.

Cele mai comune faze ale unui proces de management al testării software sunt:

1. **Planificarea testării:**
   - Definirea obiectivelor și scopului testării.
   - Elaborarea planului de testare, care include strategia, resursele necesare, programul și planurile de testare specifice.
   - Identificarea scenariilor de testare și definirea criteriilor de acceptare.
2. **Proiectarea testelor:**
   - Elaborarea scenariilor de testare detaliate, care includ inclusiv pașii, datele de intrare și rezultatele așteptate.
   - Specificarea tipurilor de teste care vor fi efectuate (testare funcțională, testare de performanță, testare de securitate etc.).
   - Dezvoltarea datelor de testare necesare.
3. **Configurarea și pregătirea testului:**
   - Configurarea mediului de testare, inclusiv hardware-ul și software-ul necesare.
   - Implementarea automatizării testelor, dacă este necesară.
   - Pregătirea datelor de testare și a seturilor de testare.
4. **Execuția testelor:**
   - Efectuarea testelor conform planului de testare.
   - Înregistrarea rezultatelor testelor și a defectelor identificate.
   - Raportarea și documentarea erorilor într-un sistem de gestionare a defectelor.
5. **Analiza și raportarea rezultatelor:**
   - Evaluarea rezultatelor testelor și analiza respectării criteriilor de acceptare.
   - Crearea rapoartelor de testare care includ informații despre progresul testării și defectele identificate.
   - Comunicarea cu echipa de dezvoltare pentru remedierea defectelor.

6. **Retestare și validare:**
    - După remedierea defectelor, efectuarea retestării pentru a verifica dacă acestea au fost corectate.
    - Validarea că toate cerințele și funcționalitățile au fost testate și funcționează conform așteptărilor.
7. **Încheierea testării:**
    - Evaluarea finală a rezultatelor testelor și a acoperirii testării.
    - Încheierea procesului de testare și pregătirea pentru lansarea sau livrarea aplicației.
8. **Arhivarea documentației de testare:**
    - Arhivarea tuturor documentelor de testare, a rapoartelor și a rezultatelor pentru referință viitoare și audit.

2.1. Proiectarea testelor, categorii de testare

Așa cum s-a menționat anterior, proiectarea testelor, sau mai bine spus a scenariilor de testare, reprezintă al doilea pas din procesul de management al testării software. Pentru a înțelege mai bine acest pas, se precizează faptul că testele unei aplicații software pot fi împărțite în mai multe categorii principale, fiecare având un scop specific în asigurarea calității și a funcționalității aplicației.

Principalele categorii de teste ale unei aplicații software sunt:

- Teste funcționale
- Teste de performanță
- Teste de securitate
- Teste de penetrare
- Teste de uzabilitate
- Teste de compatibilitate cu platforma
- Teste de regresie
- Teste de acceptare a utilizatorilor (engl., User Acceptance Testing - UAT)
- Teste de recuperare în caz de dezastre (engl., Disaster Recovery Testing - DR)

În lucrarea curentă de laborator ne vom concentra atenția asupra următoarelor categorii de teste:

2.1.1. Teste funcționale

Scop: Verificarea dacă aplicația îndeplinește corect funcțiile și cerințele sale specifice.
Descriere: Testarea funcțională se concentrează pe funcționalitatea aplicației și pe modul în care aceasta se comportă în raport cu intrările utilizatorilor sau datele de intrare. Include atât teste manuale, cât și automate (teste de unitate, teste de integrare și teste de sistem) pentru a verifica funcționalitățile individuale și interoperabilitatea acestora.
Exemplu:

- **Scenariu de test**: "Test de funcționalitate pentru inițierea unei cereri de reparație"

- **Scop**: Verificarea funcționalității de inițiere a unei cereri de reparație.
- **Descriere**: Acest test se concentrează pe modul în care funcționează inițierea unei cereri de reparație.
- **Pași de testare**:
    - Căutarea unei oferte disponibile
    - Completarea formularului de cerere nouă
    - Inițierea cererii
    - Verificarea notificării către meșter
    - Verificarea corectitudinii cererii inițiate

2.1.2. Teste de performanță

Scop: Evaluarea performanței, scalabilității și stabilității aplicației în diferite condiții de sarcină.
Descriere: Testarea de performanță se concentrează pe evaluarea vitezei, timpului de răspuns, utilizarea resurselor și a altor aspecte legate de performanța aplicației. Acesta poate include teste de încărcare, teste de stres și teste de scalabilitate.
Exemplu:

- **Scenariu de test**: "Test de încărcare a paginii principale"
- **Descriere**: Măsurați timpul de încărcare al paginii principale a aplicației sub sarcină simulată.
- **Pași de testare:**
    - Deschideți pagina principală a aplicației.
    - Utilizați un instrument de testare a performanței pentru a simula 100 de utilizatori simultani care accesează pagina.
    - Măsurați timpul de încărcare pentru fiecare solicitare a utilizatorilor simultani.
    - Generați un raport cu rezultatele testului și analizați performanța.

Pentru testarea performanței există diverse instrumente disponibile. Iată câteva dintre acestea:

- **Apache JMeter**, puteți accesa resurse de învățare pentru începători la [1]
- **GTmetrix**
- **Google PageSpeed Insights**
- **WebPageTest**
- etc.

2.1.3. Teste de securitate

Scop: Identificarea și remedierea vulnerabilităților de securitate ale aplicației pentru a proteja confidențialitatea, integritatea și disponibilitatea datelor și a preveni accesul neautorizat și atacurile.
Descriere: Testarea de securitate implică evaluarea aplicației pentru a identifica vulnerabilități cunoscute sau potențiale, cum ar fi injecții SQL (engl., SQL injection), cross-site scripting (XSS) sau cross-site request forgery (CSRF). Scopul este de a asigura că aplicația este rezistentă la atacuri.
Exemplu:

- **Scenariu de Test**: "Injecție SQL"
- **Descriere**: Verificați dacă aplicația este protejată împotriva injecțiilor SQL.
- **Pași de Testare**:
  - Deschideți o pagină care acceptă date de intrare (de exemplu, un formular de căutare).
  - Încercați să introduceți un șir de caractere cu potențial periculos, cum ar fi ' OR 1=1-.
  - Verificați dacă aplicația filtrează și validează corect datele introduse și nu permite injecția SQL.

2.1.4. Teste de penetrare

Testele de penetrare sunt o metodă de evaluare a securității sistemelor informatice, rețelelor, aplicațiilor sau infrastructurilor pentru a identifica și remedia eventualele vulnerabilități sau puncte slabe. Scopul principal al acestor teste este de a simula atacurile cibernetice pentru a evalua eficacitatea sistemelor de securitate și pentru a asigura protecția datelor și a resurselor organizației.

Instrumente comune folosite în testele de penetrare includ:

1. **Kali Linux:** Un sistem de operare care facilitează testarea de penetrare. Este o distribuție de Linux open source, bazată pe Debian și menținută de organizația Offensive Security.
2. **Nmap (Network Mapper):** Un instrument de scanare de rețea care poate ajuta la identificarea porturilor deschise și la cartografierea infrastructurii rețelei.
3. **Metasploit:** O platformă de testare a penetrării care include o colecție de module de exploatare pentru a testa vulnerabilități.
4. **Burp Suite:** Un instrument pentru testarea aplicațiilor web, care ajută la identificarea vulnerabilităților de securitate web, cum ar fi, injecții SQL sau atacuri la sesiunea utilizatorului.
5. **Teenable Nessus:** Un instrument pentru scanarea vulnerabilităților, dezvoltat de compania Teenable Inc.
6. **Wireshark:** Un analizator de pachete care permite monitorizarea și analiza traficului de rețea.
7. **OWASP ZAP (Zed Attack Proxy):** O unealtă pentru testarea securității aplicațiilor web, dezvoltată de comunitatea OWASP (Open Web Application Security Project).

2.1.5. Teste de utilizabilitate

Scop: Evaluarea experienței utilizatorului și a intuitivității aplicației pentru a asigura o interfață prietenoasă.
Descriere: Testarea de utilizabilitate constă adesea în implicarea unor grupuri de utilizatori sau a unor testeri pentru a evalua experiența de utilizare a aplicației. Scopul este de a identifica probleme legate de navigare, fluxul de lucru sau alte aspecte care pot afecta experiența utilizatorului.
Exemplu:

- **Scenariu de test**: "Căutarea ofertelor"
- **Descriere**: Evaluați experiența utilizatorului în timp ce efectuează o căutare pentru ofertele de reparație disponibile.
- **Pași de testare**:
    - Solicitați utilizatorului să găsească o anumită ofertă folosind bara de căutare.
    - Observați modul în care utilizatorul interacționează cu bara de căutare și rezultatele căutării.
    - Colectați feedback cu privire la ușurința utilizării, eficiența căutării și orice probleme întâmpinate.

2.1.6. Teste de compatibilitate cu platforma

Scop: Verificarea funcționării corecte a aplicației pe diferite platforme hardware și software.
Descriere: Testarea de compatibilitate verifică modul în care aplicația se comportă pe diferite sisteme de operare, browser-e web, dispozitive mobile și configurații hardware. Scopul este de a asigura o experiență uniformă pentru utilizatori.
Exemplu:

- **Scenariu de test**: "Compatibilitate cu diverse browser-e"
- **Descriere**: Verificați dacă aplicația funcționează corect pe mai multe browser-e web.
- **Pași de testare**:
    - Accesați aplicația folosind diferite browser-e, cum ar fi Chrome, Firefox, și Internet Explorer.
    - Testați funcționalitățile cheie, cum ar fi autentificarea și navigarea.
    - Verificați dacă interfața aplicației se afișează corect și funcționează fără erori pe toate browser-ele testate.

2.1.7. Teste de regresie

Scop: Verificarea impactului modificărilor recente ale aplicației asupra funcționalităților existente.
Descriere: Testarea de regresie se concentrează pe retestarea funcționalităților existente după ce au fost efectuate modificări sau actualizări în codul aplicației. Scopul este de a identifica defecte noi introduse accidental în timpul dezvoltării.
Exemplu:

- **Scenariu de test**: "Regresie după actualizarea interfeței utilizator"
- **Descriere**: Verificați dacă o actualizare a interfeței utilizatorului (UI) nu a afectat funcționalitățile existente ale aplicației.
- **Pași de testare**:
    - Efectuați un test funcțional pentru funcționalitățile UI existente (de exemplu, autentificarea și navigarea).
    - Actualizați interfața utilizatorului cu o nouă versiune.
    - Refaceți testul funcțional pentru funcționalitățile UI existente.
    - Verificați dacă funcționalitățile UI vechi funcționează la fel ca înainte de actualizare.

2.2. Instrumente de management al testării

Având în vedere creșterea complexității aplicațiilor moderne, a apărut inevitabil nevoia de a utiliza sisteme și instrumente specializate pentru managementul testării. În cele ce urmează vom prezenta câteva dintre cele mai utilizate instrumente, fără a avea pretenția de a genera o listă exhaustivă:

Instrument	Descriere	Opțiune Free	Link site oficial
TestRail	TestRail este o platformă de management a testării care oferă gestionarea cazurilor de testare, planificarea testelor, raportarea rezultatelor și integrarea cu alte instrumente de dezvoltare.	NU	https://www.testrail.com/
Testiny	Testiny este o platformă avansată de management al testelor, cu o interfață ușor de utilizat, rapoarte multiple, integrare cu alte instrumente și multe alte funcționalități. Permite managementul testelor manuale și automate	DA	https://www.testiny.io/
Xray	Xray for Jira este o extensie pentru Jira care adaugă funcționalități avansate de management al testării, inclusiv testare exploratorie și gestionarea testelor automate.	NU	https://marketplace.atlassian.com/apps/1211769/xray-test-management-for-jira
qTest	qTest este cunoscut pentru capacitatea sa de a gestiona testele într-un mod cuprinzător și pentru ușurința sa de utilizare.	NU	https://www.tricentis.com/products/unified-test-management-qtest
Testlink	TestLink, ca instrument open-source, are o bază de utilizatori devotați și este preferat în special de organizațiile care caută o soluție gratuită.	DA	https://testlink.org/
TestLodge	TestLodge este cunoscut pentru simplitatea sa și este adesea alegerea pentru organizațiile mici și mijlocii care doresc o soluție ușor de folosit.	NU	https://www.testlodge.com/

Alte instrumente: Zephyr, Zephyr Scale, PractiTest, TestCollab, Prueba, Qmetry

3. Execuția planului de testare

După cum s-a putut observa, un plan de testare conceput folosind platforma Testiny cuprinde un set de teste care descriu pas cu pas câte un scenariu de testare. Așadar, în etapa de execuție a acestui plan, inginerul de testare urmează pașii pentru fiecare test și notează într-un raport rezultatul acestuia.

În general, execuția unui plan de testare poate fi realizată prin următorii pași:

1. **Pregătirea mediului de testare**:
    - **Pregătirea aplicației**: Inginerul de testare va actualiza și construi/ instala versiunea aplicației ce urmează a fi testată și va realiza configurările necesare. Dacă aplicația cuprinde mai multe componente, de exemplu backend și frontend, acesta se va asigura că toate componentele sunt actualizate corespunzător.
    - **Pregătirea datelor**: Dacă testele necesită existența unor date (înregistrări) inițiale, inginerul de testare se va asigura că acestea există în baza de date.
2. **Pregătirea unei execuții (engl., test run) în platforma de management al testării**: Anumite platforme, precum Testiny, oferă posibilitatea de a crea multiple execuții pentru un plan de testare. În continuare sunt prezentate câteva observații importante cu privire la aceste execuții:
    - o execuție poate fi percepută ca echivalentul unei sesiuni de testare;
    - este important de știut că la crearea unei execuții pot fi selectate toate testele din plan, dar există si posibilitatea de a selecta numai un subset;
    - aceste execuții cuprind în general teste manuale, pe care un tester le execută și apoi actualizează rezultatul în cadrul acestei execuții;
    - există și integrări cu platforme de testare automată, astfel încât o execuție să fie actualizată în urma execuției testelor automate, dar acest lucru necesită un efort considerabil de integrare și configurare ale acestor platforme;
3. **Execuția planului de testare**: În această etapă inginerul de testare va parcurge fiecare test din planul de execuție, operând aplicația conform instrucțiunilor acestuia. De asemenea, va marca rezultatul în planul de execuție.
4. **Raportarea execuției**: În general, platformele de management al testării, inclusiv platforma Testiny, includ și instrumente de generare a unui raport. Un astfel de raport poate conține atât informații generale, cum ar fi numărul de teste eșuate din numărul total de teste executate, dar și anumite detalii despre testele care au eșuat.
5. **Raportarea problemelor (bug-urilor)**: Problemele identificate în timpul testării sunt raportate de obicei în sistemul de management al proiectului. În Jira, de exemplu, sunt raportate sub formă de tichete de tip “Bug” sau “Task”, în funcție de cum este configurat proiectul. Raportarea acestor probleme în sistemul de management se poate face imediat ce au fost descoperite, sau la finalul execuției întregului plan de test, în funcție de varianta care este mai eficientă.

4. Tema de laborator

În acest laborator echipele vor preda livrabilele din laboratoarele precedente, sub forma specificată la fiecare livrabil. De asemenea, livrabilele ce vor fi implementate în acest laborator vor fi transmise în laboratorul specificat în coloana termen limită.

Observații:

- Resursele ajutătoare sunt obligatorii doar la acele livrabile la care este specificat un instrument software.
- Resursele de tip tutorial/ documentație sunt orientative, prin urmare aveți posibilitatea de a utiliza și alte tutoriale/ documentații disponibile, dacă acest lucru vă ajută să implementați mai ușor și eficient respectivul livrabil.

4.1. Studiul documentației

Contribuie	Toată echipa
Responsabil	Toată echipa
Termen limită	Lab 12
Sarcini	Studiați documentația de laborator cu atenție, capitolele 1. – 3.

4.2. Plan de testare

Contribuie	Toată echipa
Responsabil	Responsabil Testare
Termen limită	Lab 13
Sarcini	Elaborați un plan de testare cu teste din cât mai multe categorii, pentru aplicația ce va fi dezvoltată, **utilizând platforma Testiny**.
Livrabile	• Document cu toate testele create, exportat din Testiny și/sau invitație în Testiny trimisă pe e-mail către coordonator. • Documentul trebuie să cuprindă și teste de securitate • Prezentare în fața coordonatorului de laborator.
Resurse	• Documentul curent, subcapitolul 2.1 • https://www.testiny.io/

4.3. Test de penetrare experimental

Contribuie	Toată echipa
Responsabil	Responsabil Testare
Termen limită	Lab 13
Sarcini	Utilizând instrumentul **ZAP Proxy**, realizați câteva teste de penetrare (scanare automată, scanare manuală etc.) pe un website dedicat testelor de acest tip (vezi [2]).
Livrabile	• Rapoarte de testare exportate din ZAP Proxy (dacă este posibil). • Prezentare în fața coordonatorului de laborator.
Resurse	• Documentul curent, subcapitolul 2.1.4 • [2]- [3]

4.4. Execuția planului de testare pentru versiunea MVP

Contribuie	Toată echipa
Responsabil	Responsabil Testare
Termen limită	Lab 13
Sarcini	Utilizând planul de testare conceput în platforma **Testiny**, creați un "**Test run**" în care să includeți toate testele aferente versiunii MVP. Utilizând aplicația integrată în laboratorul 11, pregătiți mediul de testare pe mașina locală.
Livrabil	• În platforma Testiny există un "Test run" pentru versiunea MVP. • Pe mașina locală sunt descărcate sursele de pe ramurile "mvp"
Resurse	• Documentul curent, capitolul 4

Contribuie	Toată echipa
Responsabil	Responsabil Testare
Termen limită	Lab 13
Sarcini	Executați manual testele din planul de execuție creat la pasul 5, și raportați rezultatele. Pentru problemele (bug-urile) majore identificate, creați sarcini de tip "Task" în Jira. Titlul și descrierea sarcinii trebuie să reflecte problema identificată. Aceste sarcini pot fi preluate ulterior de către dezvoltatori, pentru a rezolva problemele.
Livrabil	• Prezentarea rezultatelor în platforma Testiny, în fața coordonatorului de laborator. • Raportul execuției, exportat în format PDF. • Pentru bug-urile majore, sarcini create (documentate) în Jira.
Resurse	• Documentul curent, capitolul 4

5. Referințe bibliografice

[1] „Simplilearn - JMeter Tutorial," [Interactiv]. Available: https://www.simplilearn.com/tutorials/jmeter-tutorial.

[2] „Top 10 site-uri vulnerabile pentru teste de penetrare," [Interactiv]. Available: https://ravisarode.medium.com/hacker-has-to-good-knowledge-of-how-web-applications-work-before-he-going-to-find-the-e3366f3d9038 .

[3] „Zap Proxy, tutorial pentru începători," [Interactiv]. Available: https://www.zaproxy.org/getting-started/.

[4] „Cypress Tutorial," [Interactiv]. Available: https://www.tutorialspoint.com/cypress/index.htm.

[5] „Tutorial oficial Cypress," [Interactiv]. Available: https://docs.cypress.io/examples/tutorials.