Introducere

Primul modul al cursului de testare automatizată a prezentat succint aspectele esențiale ale teoriei testării prin prisma nivelelor de testare, tipurilor și nemijlocit tehnicilor aplicate pentru scrierea de scenarii de testare. Odată, cunoscute aceste nuanțe importante sa constat important de a eficientiza lucrul executat manual prin intermediul automatizării acestora. Pentru a putea atinge acest obiectiv, în modulul precedent au fost abordate aspecte pur tehnice ale limbajului de programare orientat pe obiecte Java, cu ajutorul căruia o parte din procedurile manuale repetitive vor fi executate în mod automatizat. Punerea în aplicare a cunoștințelor tehnice a fost efectuată prin simularea unor situații reale din viață și mai puțin prin operații concrete care au loc în cadrul unui sistem de testare automatizată. Asta deoarece, modulul curent este dedicat în totalitatea testării automatizate, ceea ce înseamnă că se urmărește îmbinarea cunoștințelor din modulele anterioare și crearea unui sistem de testare automatizată independent. Pentru o mai bună înțelegere a scopului și obiectivelor unui sistem de testare automatizată, lecția curentă pune accentul pe importanța testării automatizate, tipurile de testare automatizată și într-un final va prezenta structura de bază a unui sistem de testare automatizată.

Testarea automatizată

După cum se cunoaște deja din primul modul, prin testare automatizată se subînțelege mulțime de proceduri executate cu ajutorul unor instrumente digitale pentru a înlocui efortul manual de verificare și validare a unui produs. Știind că esența testării o reprezintă crearea și executarea scenariilor cu scopul de identifica defecte sau erori, atunci putem afirma că studiul principal al domeniul de testare automatizată îl constituie automatizarea scenariilor de testare. Pentru a putea executa scenariile în regim automatizat este nevoie de a construi un sistem independent menit să simuleze activitatea umană în raport cu subiectul testării (aplicația care este testată). Crearea unor astfel de sisteme necesită eforturi de timp și cunoștințe în domeniul programării astfel încât spre final acestea să aducă un set de avantaje în anumite condiții, cum ar fi:

- Când se vorbește despre testarea unui sistem pentru mai multe platforme cu scopul de-a verifica și valida compatibilitatea sistemului dezvoltat în raport cu diferite medii. În aceste cazuri, executarea acelorași scenarii de testare de un număr mare de ori în regim manual pentru fiecare platformă necesită mult timp și efort, iar calitatea rezultatelor poate fi vulnerabilă datorită factorului uman, care admite abateri. Utilizarea unui sistem de testare automatizată destinat testării de compatibilitate prevede scrierea o singură dată a unui scenariu și rularea lui pe toate platformele. Altfel spus, având un set de scenarii de testare care trebuie rulate 2 sau 3 browsere v-a fi mai eficient de a le automatiza o singură dată iar mai apoi rula pe totalitatea browsere-lor disponibile.
- Prezența unor acțiuni repetitive. Atunci când se vorbește despre aplicarea tehnicilor de testare black-box se are în vedere executarea unui set de scenarii similare la care diferă în cele mai multe cazuri doar datele de testare. Astfel de scenarii mai sunt numite și Data Driven Test (Teste derivate din date) iar ele sunt potrivite pentru automatizare deoarece necesită un efort repetitiv pentru executarea lor. Pentru a înțelege mai bine acest factor, se analizează execuția scenariilor de testare pentru un câmp care acceptă numere de la 16 până la 64. Pentru testarea cazurilor pozitive prin calea manuală este necesar de executat două test identice cu valori diferite ale câmpului – 16 și 64. Același caz la nivel automatizat este implementat o singură dată iar singura diferență o reprezintă datele utilizate.

- Atunci când aplicația suferă schimbări frecvente iar scenariile de regresie sunt intens utilizate. Pentru a putea explica acest factor este important să e definim testarea de regresie, care reprezintă un set de scenarii de testare din cele anterior executate (adesea cele mai prioritare dintre ele) care sunt executate cu scopul de a se asigura că schimbările efectuate nu au impactat negative funcționalitatea sistemului construit până la momentul schimbării propriu-zise. Așadar, într-un mediu agile, executarea pachetului cu teste de regresie este una dintre cele mai frecvente acțiuni, ceea ce manual ar implica multe resurse umane, timp și efort. Automatizarea testelor de regresie poate fi considerată o soluție eficientă mai ales în cadrul sistemelor care au o perioadă lungă de viață. Odată cu creșterea numărului de funcționalități crește și importanța testelor de regresie automatizată, care scutesc echipa de efort umane fiind doar menținute (adică ajustate pentru a reflecta comportamentul așteptat al sistemului). Concluzionând, testarea de regresie în regim de automatizare este eficientă deoarece reduce consumul de resurse.
- Necesitatea rulării într-un timp restrâns a scenariilor de testare. Timpul este un dușman aprig al echipelor care dezvoltă produse software, deoarece niciodată nu este suficient – mai ales în domeniul asigurării calității unde apariția defectelor este imprevizibilă și niciodată nu poate fi estimată obiectiv. Odată automatizate un set de scenarii ele pot fi rulate în regim programa, cum ar fi în timpul nopții sau de fiecare dată când o modificare este executată de către echipa de dezvoltatori.
- Rularea în paralel a unor scenarii de testare. Deși acest lucru poate fi făcut și manual, rularea paralelă a scenariilor de testare poate fi făcută mai eficient datorită testelor automatizate. Acestea pot fi rulate de mai multe instanțe concomitent sau secvențial.
- Crearea datelor de testare reprezintă o investiție de timp importantă. Prin testele automatizate nu se urmărește doar rularea scenariilor, acestea fiind și o sursă esențială de creare a datelor de testare. Fiecare scenariu de testare poate avea un set amplu de pre-condiții, care în cele mai multe cazuri constituie proporția ce mai mare de timp consumat. Pentru a reduce din presiunea creării datelor de testare și a executării pre-condițiilor se recurge adesea la testarea automatizată, acolo unde acestea pot fi implementate o singură dată, apoi menținute și executate ori de câte ori este necesar.

Deși aparent testarea automatizată pare a fi mult mai eficientă deoarece nu implică eforturi umane colosale în identificarea defectelor, oferă o precizie înaltă și prezintă informațiile despre riscurile sistemului într-un mod simplificat - nu poate fi neglijat rolul testării manuale care are următoarele avantaje în următoarele situații:

- Scenariile de testare au o prioritate minoră și reprezintă niște cazuri marginale care necesită a fi executate o singură dată. Prin urmare, nu merită consumarea efortului de automatizare al acestora deoarece ele nu își vor recupera investiția în viitor. În schimb, rularea lor o singură dată sau de câteva ori manual nu depășesc investițiile automatizării și menținerii.
- Atunci când documentația este incertă, testarea manuală permite specialistului să vină cu abordări ne ordinare, cu scenarii ad-hoc imprevizibile.
- Când se vorbește de testarea experienței utilizatorului în raport cu sistemul. Un aport uman poate face diferența dintre o funcționalitate reușită și una care pur și simplu respectă specificațiile tehnice. Ulterior, în urma testării manuale pot fi recomandate modalități de îmbunătățire a sistemului.
- Atunci când sistemul nu are funcționalități stabile. Asta are loc în primele faze de dezvoltare a proiectului, când testarea manuală este cea care verifică sistemul încă instabil. Este ineficient să se automatizate niște scenarii care nu vor fi relevante în scurt timp după executarea lor.

Tipul de testare utilizat în procesul de verificare și validare al sistemului este ales în dependență de situațiile de mai sus sau reieșind din aspectele comune ale acestora cu alte situații ne ordinare. Cert este că pe lângă avantaje, atât testarea manuală cât și cea automatizată au un set de dezavantaje. Dacă în cazul testării manuale s-a specificat consumul colosal de timp pentru executarea procedurilor de verificare și validare, atunci când se i-a în calcul testarea automatizată trebuie să se țină cont de investițiile inițiale sporite pentru crearea sistemului de testare și de faptul că nu toate aspectele unei aplicații pot fi supuse testării automatizate. Sistemele de testare automatizate urmăresc creșterea eficienței prin automatizarea scenariilor și pot fi întâlnite în diferite structuri, menite să testeze anumite tipuri de software. În continuare se propune analiza celor mai des întâlnite sisteme de testare automatizată.

Tipuri de sisteme de testare automatizată

Diversitatea sistemelor de testare automatizată este dependentă de mulțimea produselor IT pe care acestea le verifică și validează. Ele sunt construite cu ajutorul limbajelor de programare și a bibliotecilor deja existente puse la dispoziție de către acestea. Nu există restricții la nivelul organizării sau creării unui sistem de testare automatizată, cu toate acestea există un set de recomandări menite să îmbunătățească funcționarea acestuia, acestea mai sunt numite și șabloane de scriere a codului. Făcând trimitere la modul de organizare al sistemelor de testare precum și a destinațiilor acestora, distingem următoarele tipuri de sisteme de testare automatizată:

- **Sisteme de testare automatizată a interfeței utilizatorului** – acestea au la bază un instrument ce simulează activitatea umană, care poartă numele de driver. Sistemele respective sunt predestinate testării funcționale din perspectiva utilizatorului, ceea ce înseamnă că sistemul urmează să navigheze anumite pagini, să apese diferite butoane și să compare dacă rezultatele actuale corespund cu cele așteptate în regim automatizat. Dacă se face o divizare a părți de front-end pe care o vede utilizatorul de Back-end unde are log prelucrarea logică a datelor, atunci poate fi afirmat că sistemele de testare automatizată a interfeței implică verificare ambelor părți arhitecturale prin intermediul front-end-ului. Majoritatea scenariilor cap-coadă (numite și end-to-end) sunt executate în modul respectiv, deoarece simultan acoperă ambele părți ale sistemului. Un avantaj important de menționat este că acestea pot rula testele pe diferite platforme și versiune ale acestora, ceea ce constituie unul dintre proprietățile principale ale testării automatizate.
- **Sistemele de testare a interfeței de programare** – la fel ca cele prezentate mai sus, acestea sunt dezvoltate cu ajutorul unui limbaj de programare și au menirea de-a verifica și valida activitatea părții arhitecturale de back-end. Aceste tipuri de sisteme fac abstracție de partea vizuală a sistemului, de ceea ce vede utilizatorul final și pun accent pe prelucrarea logică a programului. Știind că partea vizuală de front-end este integrată cu partea logică de back-end, sistemele în cauză acționează undeva la mijloc și interacționează cu partea logică doar. Decizia de-a crea un astfel de sistem este rațională din aspectul timpului destinat testării, deoarece petnru executarea unui set de câteva sute de teste se pot consuma câteva secunde sau minute. Același lucru în mod integral, adică prin intermediul interfeței utilizatorului ar putea lua inclusiv ore. Testarea automatizată a interfeței de programare (altfel numită și API) va fi abordată în cadrul următorului modul al cursului curent, unde se vor învăța instrumente necesare pentru scrierea și executarea sceanriilor automatizate, fără a scrie un sistem de testare personalizat.

- **Sisteme de testare a dispozitivelor mobile** – acestea sunt specifice prin faptul că sunt predestinate testării aplicațiilor mobile de tip nativ. Ele sunt dezvoltate în mod prioritar pentru a testa aplicațiile din aspect al compatibilității cu mai multe dispozitive. Acestea folosesc instrumente de integrare cu dispozitivele mobile sau simulatoare virtuale care le înlocuiesc. Eficiența lor crește odată cu creșterea numărului de dispoiztive care necesită a fi verificate și a sistemelor de operare pe care le posedă acestea.
- **Sisteme de testare automatizate de tip robot** – reprezintă sistemele de testare automatizate care au instrucțiuni predefinite și sunt menite reutilizării. Se optează pentru astfel de sisteme atunci când timpul și resursele financiare sunt limitate. Un dezavantaj al acestora poate fi considerat faptul că unele activități sau acțiuni nu sunt predefinite și nici prevăzute a fi implementate, ceea ce poate bloca procesul de testare al anumitor funcționalități.

Sistemele de mai sus nu reprezintă totalitatea tipurilor de sisteme de testare automatizate existente pe piață, însă acestea sunt cele mai frecvent întâlnite. Reieșind din importnața acordată ficărui dintre tipurile de sisteme, cursul curent v-a pune accent pe crearea unui sistem de testare prin utilizarea interfeței utilizatorului prin utilizarea cunoștințelor tehnice dobândite în modulul Java.

Structura unui sistem de testare automatizată

Considerând drept obiectiv principal testarea unei aplicații web care are la bază un număr limitat de pagini web, structura unui sistem de testare automatizat al interfeței utilizatorului trebuie să emită activitatea umană (a specialistului QA) în cadrul acestora. Prin urmare, dacă vorbim despre implementarea unui astfel de sistem cu ajutorul unui limbaj orientat pe obiecte se vede necesar oglindirea situației reale în termeni informatici. Cu alte cuvinte, clasele din limabjul Java vor permite crearea unor seturi de obiecte care vor prezenta structura sistemului supus testării (paginile web ale aplicației testate) și vor coordona acțiunile asupra acestora. Pentru o mai bună înțelegere a acestora concepte, se propune detalizarea ficărei componente ale unui sistem de testare automatizată a interfeței UI:

1. **Instrument de definire și gestiune a dependențelor** – în cadrul modului cu Java am folosit librării predefinite în pachetele Java, care oferă oportunitatea de-a crea obiecte și de-a le utiliza metodele fără a le implementa. Cu alte cuvinte, se utilizează cod predefinit de către alte persoane pentru a satisface niște necesități necesare într-o anumită problemă. Există biblioteci în afara Java care pun la dispoziție un set de Clase ce definesc obiecte ce pot ajuta și facilita rezolvarea problemelor. Acestea pot fi gestionate cu ajutorul unui instrument de management al dependențelor cum ar fi maven, care urmează a fi studiat și utilizat un pic mai târziu. Utilizarea dependențelor este un avantaj enorm deoarece programatorul este scutit de scrierea codului, utilizând secvențe de cod scrise de către altcineva. În același timp, dependențele pot fi un punct de cotitură atunci când ele nu sunt menținute de către proprietarii lor și sunt șterse din repozitoriile cloud de unde sunt descărcate de către maven.
2. **Resursele de testare** – reprezintă un directoriu, care poate include date de intrare cum ar fi nume de utilizatori sau parole de acces. În mod obligatoriu, în resurse este definit și un fișier de configurare care definește platforma pe care vor fi rulate testele automatizate sau adresa sistemului supus testării, care poate să difere în dependență de mediul unde este livrată soluția.
3. **Folderul cu pagini, pageObjects** – adesea se numește componenta de pageObjects și constituie un model de organizare sau șablon pentru testarea automatizată a interfeței

utilizatorului. Reprezintă de fapt un repozitoriu ce include toate paginile sistemului web supus testării, definite cu ajutorul claselor. Cu alte cuvinte, în acest repozitoriu vor fi identificate clase precum HomePage, ContactPage sau LoginPage iar în interiorul claselor vor fi definite elementele ce se conțin pe aceste pagini și nemijlocit acțiunile care pot fi făcute cu aceste pagini.

4. **Funcționalități** – componenta în cauză stochează totalitatea scenariilor de testare, care sunt adesea scrise într-un limbaj business, simplu și ușor de înțeles. În spatele pașilor de testare stă o implementare de cod, cu alte cuvinte fiecare pas definit cu limbaj business este ulterior definit într-o altă componentă.
5. **Definiția pașilor din scenarii** – are scopul de-a crea legătura dintre funcționalitățile definite în limbaj business și elementele paginilor web definite în componenta pageObjects. Logica programării este concentrată în acești pași, deoarece aici are loc apelarea majorității metodelor destinate executării acțiunilor asupra sistemului supus testării.
6. **Utilități** – stochează totalitatea claselor utilitare, care permit adesea conțin metode statice ce permit generare de date pentru testare sau lucrul cu resursele din fișiere. În dependență de specificul sistemului și necesitățile programatorului, aici pot fi incluse un număr larg de utilități.
7. **Manageri** – este componenta destinată gestiunii informațiilor intermediare dintre pașii scenariilor de testare. Cu alte cuvinte, clasele de aici au menirea de-a gestiona datele și a le partaja la necesitate.
8. **Drivere** – compartimentul responsabil de păstrarea drivere-lor care permit simularea acțiunilor umane. Unul dintre acestea vor fi studiate în cadrul lecții următoare.

Componentele de mai sus sunt opționale și recomandate pentru crearea unui sistem de testare automatizată, acestea pot să difere după denumire și modul de organizare. Totodată numărul lor poate crește în funcție de funcționalitățile pe care le abordează aplicația supusă testării.

Concluzie

Fiind familiari cu tipurile de sisteme de testare automatizată și studiind soluția ce urmează a fi supusă testării puteți deduce care vor fi scenariile ce merită a fi automatizate și veți putea aprecia nivelul de eficiență al investiției respective. În cele mai frecvente cazuri se utilizează sistemele de testare automatizată a interfeței utilizatorului deoarece au o rază mai înaltă de acțiune, verificând atât partea de interfață a utilizatorului cât și partea de prelucrare logică a datelor.