TEMA – Testimi i automatizuar i aplikacioneve web

ABSTRAKT

Ne ditet e sotme aplikacionet web luajne nje rol shume te rendesishem ne perditshmerine tone. Ato ndihmoje ne shume operacione biznesi dhe per kete arsye siguria dhe funksionaliteti i tyre eshte bere shume i rendesishem.

Kjo teme do te beje një analizë krahasuese të thelluar të tre teknologjive të automatizimit të uebit të njohura: Selenium, Cypress dhe TestComplete.

Selenium, një platformë me burim të hapur, ka qenë gjatë kohës një mjet i rëndësishëm në fushën e automatizimit të uebit. Cypress, një framework më i ri, ka fituar popullaritet për thjeshtësinë dhe qasjen miqësore të zhvilluesit. Në anën tjetër, TestComplete, një mjet automatizimi komercial, ofron një zgjidhje të përgjithshme me karakteristika të përparuara dhe mbështetje.

Ky studim hulumton shumë aspekte të këtyre teknologjive të automatizimit për të ndihmuar organizatat të marrin vendime të informuara për strategjinë e tyre të testimi automatizimit. Krahasimi përfshin faktorë si lehtësia e përdorimit, ndryshueshmëria, aftësitë e skriptimit, kompatibiliteti me shfletues të ndryshëm dhe mbështetja për teknologjitë moderne të uebit. Po ashtu, studimi merr në konsideratë kostot e licencës, mbështetjen e komunitetit dhe integrimin në ekosistemin e tyre.

Përmes testimet dhe eksperimentet e kujdesshme, kjo analize synon të vlerësojë pikët e forta dhe dobësitë e çdo teknologjie dhe të ofrojë shqyrtime në përputhshmërinë e tyre për skenarë të ndryshëm të testimi. Gjate kesaj analize do të prezantohen shembuj praktikë dhe raste përdorimi për të ilustruar aplikueshmërinë në botën reale të çdo mjeti.

Gjetjet e kërkimit te kesaj analize synojnë të shërbejnë si një burim i vlefshëm për ekipet e zhvillimit të softuerit dhe profesionistët e sigurisë së cilësisë që kërkojnë të përmirësojnë praktikat e tyre të testimi automatizimit të uebit. Në fund të kësaj analize, lexuesit do të kenë një kuptim më të qartë të aftësive dhe kufizimeve të Selenium, Cypress dhe TestComplete, duke u mundësuar të marrin vendime të informuara kur zgjedhin teknologjinë më të përshtatshme të automatizimit të uebit për nevojat e tyre specifike.

**Tabela e permbajtjes**

ABSTRAKTI   
LISTA E TABELAVE  
Lista e figurave

Lista e shkurtimeve

* ISTQB

// tbd – update kapitujt

KAPITULLI I – disa shpjegime per qellimin e ketij studimi

Kapitulli 2 – Cfare eshte testimi – (ISTQB infos)

Kapitulli 3 – Testimi i web apps (selenium, cypress) – FREE

Kapitulli 4 Testimi I web apps (paid tools) Testcomplete ?

Kapitulli 5 Krahasimi i free charge dhe paid tools (PROS CONS)

QELLIMI I STUDIMIT

Framework-et e automatizimit te testeve po perdoren gjithmone dhe me teper ne procesin e zhvillimit te softuerve, pasi ato permiresojne procesin e testimit te produktit. Kto frameworks lejojne zhvilluesit te automatiozojne testet qe perseriten shpesh dhe shpesh jane TIME-CONSUMING. Ne vitet e fundit ne treg mund te gjejme shume frameworks te ndryshem falas ose me pagese qe ndihmojne ne automatizimin e procesit te testimit.

Pavarsisht avantazheve qe cdo framework testmi ka, secila nga ato ka specifikat e veta dhe ne varesi te natyres se projektit nje framework mund te jete me i mire se tjetri. Ne kete studim ne do te perpiqemi te krahasojme dy framework shume te njohura qe jane falas si Selenium dhe Cypress dhe te vleresojme gjithashtu si qendrojne ato ne raport me njera tjetren dhe ne raport me Test Complete, nje framework komercial per automatizin e testeve qe eshte me pagese.

OBJEKTIVAT E KETIJ STUDIMI

1. Te rishikojme literaturën mbi kornizat e automatizimit dhe krahasoni kornizat e ndryshme bazuar në veçoritë dhe aftësitë e tyre
2. Te vleresojme pikat e forta dhe të dobëta të Selenium,Cypress dhe Test Complete bazuar në kritere të tilla si lehtësia e përdorimit, pajtueshmëria, besueshmëria, shpejtësia dhe mbështetja për gjuhë të ndryshme programimi
3. Te krahasojme kornizat e automatizimit falas dhe me pagesë bazuar në veçoritë, modelet e çmimeve, mbështetjen e klientit dhe miratimin e përdoruesit
4. Te japim rekomandime për organizatat që po shqyrtojnë përdorimin e kornizave të automatizimit bazuar në analizën dhe vlerësimin e kornizave të ndryshme

Qellimi i ketij studimi eshte qe te bejme nje krahasim midis te githave **framework**-eve dhe **tools** te siperpermendura dhe te ndihmojme ne zgjedhjen e framework/tool per projektin tuaj.

KAPITULLI 2 Cfare eshte testimi? (ISTQB)

Testimi i software eshte procesi I vleresimit te nje software ose sistemi per te percaktuar nese permbush kerkesat e specifikuar dhe funksionon sic duhet. Procesi perfshin verfikimin dhe validimin e aspekteve te ndryshme te nje softueri si, funksionaliteti, perdorshmeria, besueshmeria, performanca dhe siguria.

Testimi i softureit eshte nje proces shume i rendesishem ne ciklin e zhvillimit te softuerit sepse ndihmon ne identifikimin e erroreve ne softuer dhe sigurohet qe pritshmerite e perdoruesve ne lidhje me cilesine e produktit permbushen. Testimi si proces mund te jete manual, qe do te thote testerat ndjekin nje set me hapa te parapercaktuar ose automatic duke bere ekzukitimin e testeve ne nje kohe me te shpejt dhe rezultate te sakta.

Qellimi i testimit te softuerit eshte te sigurohet qe produkti funksioni sic parashikohet, eshte i lehte per tu perdorur dhe permbush nevojat e perdoruesit. Testimi mund te ndihmoje gjithashtu per te identifikuar permiresime te mundshme ne produkt dhe te sugjeroje modifikime qe mund te permiresojne cilesine e produktit ne pergjithesi.

Shkurtimisht, testimi i softuerit eshte nje proces shume kritik qe ndihmon ne sigurimin e cilesise, funksionalitetit, performances te softuerit dhe sistemeve te ndryshme. Ai ndihmon ne identifikin e defekteve dhe sigurohet qe software permbush nevojat dhe pritshmerite e perdoruesve te tij, duke e bere ate nje aspekt esencial/shume te rendesishem ne ciklin e zhvillimit te nje softueri.

TESTIMI Sipas ISTQB permbledhje 5-10 faqe

Cfare eshte testimi (permbledhur me SIPER)

Gjate 5 dekadeve te fundit nje numer i madh sugjerimesh per principet e testimit jane bere per te ofruar disa udhezimie te pergjitheshme per te gjitha llojet e testimit.

**7 Principet e testimit**

1. Testimi tregon pranine e defekteve, por jo mungesen e tyre. -> Testimi redukton mundesine e defekteve ne aplikacion, por edhe nese asnje defekt nuk eshte gjetur, testimi nuk mund të provojë saktësinë e objektit të testit.
2. Te testosh gjithcka eshte e pamundur -> Te testosh gjithcka eshte e pamundur pervec ne disa raste specifike. Fokusi kryesor duhet te jene tekninat e testimit, prioritizimi i test cases, dhe testimi I bazuar ne rrezik sesa testimi i gjithckaje.
3. Testimi i hershem kursen kohe dhe para -> Testimi i hershem qe ne fillim te procesit te zhvillimit te aplikacioneve nuk do te shkaktoje defekte te mevonshme ne produktet e punes qe pasojne. Kostoja do te reduktohet pasi do te kete me pak defekte.
4. Grumbullimi i defekteve -> Shumica e defekteve gjenden në një numër të vogël komponentesh. Sipas parimit Paereto, i njohur gjithashtu si rregulli 80/20, 20% e komponenteve përmbajnë 80% të defekteve të gjetura.
5. Kujdes nga paradoksi i pesticideve -> Nese te njejtet teste perseriten shume here, ato nuk do te jene me efektive per te zbuluar defekte te reja. Prandaj eshte e rendesishme qe tested te rishikohen dhe modifikohen vazhdimisht.
6. Testimi varet nga konteksti -> Lloje të ndryshme të aplikacioneve testohen në mënyra të ndryshme
7. Mungesa e erroreve eshte ide e gabuar -> Nëse aplikacioni ose sistemi nuk është funksional dhe nuk i përmbush kërkesat e klientit dhe nevojat e përdoruesve, atëherë gjetja dhe rregullimi e defekteve është e padobishme.

**Aktivitet dhe detyrat e testimit 1.4.2**

Nje proces testimi konsiston ne nje sere aktivitetesh:

1. Plani i testimit – Konsiston ne definimin e objektivave te testimit dhe me pas ne zgjedhjen e nje metodologjie qe na ndihmon me se miri ne arritjen e ketyre objektivave.
2. Monitorimi dhe kontrolli I testimit -> Monitorimi I testimit perfshin kontrollin e vazhdueshem te te gjitha aktiviteve te testimit dhe krahasimin e progresin aktual perkundrejt planit. Kontrolli i testimit konsisiton ne marrjen e veprimeve te nevojshme per te arritur objektivat e testimit.
3. Analiza e testimit -> Identifikimi i **features** qe jane te testueshme dhe prioritizimi i tyre. Gjithashtu dhe vleresimi I rreziqeve.
4. Dizanji I testimit -> Perpunimi i kushteve te testimit ne raste testimi. Identifikimi I mbulimit te testimit, teknikat e testimit, definimi I datave qe do te perdoren per testim, mjediset e testimit, infrastruktura dhe mjetet qe do te perdoren. Pyetja qe ngrihet ne kete aktivitet: Si te testojme?
5. Implementimi I testimit -> konsiston ne krijimin e nje software per ekzekutimin e testeve.
6. Ekzekutimi I testit -> Konsiston ne ekzekutimin manual ose te automatizuar te testeve. Rezultate aktuale te testimit krahason me rezultatet e pritura. Identifikohen anomalite dhe analizohen per te gjetur shkakun e tyre.
7. Perfundimi I testimit -> Ky aktivitet ndodh zakonisht pas cdo release ose iteracioni. Konsiston ne Krijimin e nje raporti I cili u dorezohet paleve te interesuara.

**Nivelet e testimit**

1. Testimi i Komponentit (ndryshe Testimi I njesise): Përqendrohet në testimin e komponenteve të një sistemi ose aplikacioni në izolim dhe zakonisht kryhet nga zhvilluesit në mjediset e tyre të zhvillimit.
2. Testimi i Integritetit të Komponenteve (Testimi i Integritetit të Njësive): Përqendrohet në testimin e ndërfaqeve dhe ndërveprimeve midis komponenteve të një sistemi.
3. Testimi i Sistemit: Përqendrohet në sjelljen dhe aftësitë e përgjithshme të një sistemi ose produkti të tërë. Përfshin testimin funksional të detyrave prej fillimi në fund dhe testimin jo-funksional të cilësive të cilësisë. Mund të kryhet nga një ekip i pavarur të testimi dhe është i lidhur me specifikimet për sistemin.
4. Testimi i Integritetit të Sistemit: Përqendrohet në testimin e ndërfaqeve të sistemit nën test dhe sistemeve të tjera dhe shërbimeve të jashtme. Kërkon mjedise të përshtatshme të testimit, të preferueshme të ngjashme me mjedisin operacional.
5. Testimi i Pranimit: Përqendrohet në validim dhe demonstrimin e gatishmërisë së sistemit për implementim, duke u siguruar që të plotësojë nevojat e biznesit të përdoruesit. Mund të kryhet idealisht nga përdoruesit e synuar dhe përfshin formën e testimi të pranimit të përdoruesit, testimin operacional, testimin kontraktual dhe ligjor.

**Llojet e testimit**

1. Testimi Funksional: Testimi funksional vlerëson funksionet që një komponent ose sistem duhet të kryejë dhe ka për qëllim të verifikojë kompletësinë funksionale, saktësinë funksionale dhe përputhshmërinë funksionale. Kjo përqendrohet në "çfarë" duhet të bëjë objekti i testimi.
2. Testimi Jo-funksional: Testimi jo-funksional vlerëson atributet që janë të tjera përveç karakteristikave funksionale të një komponenti ose sistemi dhe ka për qëllim të kontrollojë cilësinë jo-funksionale të softuerit, duke përfshirë efikasitetin e performancës, kompatibilitetin, përdorshmërinë, besueshmërinë, sigurinë, mirëmbajtshmërinë, dhe përtëritshmërinë. Kjo përqendrohet në "sa mirë sjell sistemi".
3. Testimi i Kutise se zeze: Testimi i kutise se zeze është bazuar në specifikime dhe përdor dokumentacion të jashtëm për të krijuar testet. Qëllimi kryesor i testimit të kuq është të kontrollojë sjelljen e sistemit në krahasim me specifikimet e tij.
4. Testimi i kutise se Bardhë: Testimi i kutise se bardhë është bazuar në strukturë dhe përdor strukturën dhe implementimin e sistemit për të krijuar testet. Qëllimi kryesor i testimit të bardhë është të sigurojë që struktura e pasosë testohet në nivelin e duhur.
5. Testimi i Konfirmimit: Testimi i konfirmimit verifikon se një mangësi origjinale është ndrequr me sukses. Sipas riskut, versionin e ndrequr të softuerit mund ta testoni në disa mënyra, përfshirë:
   1. Ekzekutimin e të gjitha rasteve të testimit që më parë kanë dështuar për shkak të mangësisë, ose
   2. Shtimin e testeve të reja për të mbuluar ndonjë ndryshim që është bërë për të ndrequr mangësinë.
   3. Megjithatë, kur koha ose paratë janë të kufizuara për ndreqjen e mangësive, testimi i konfirmimit mund të kufizohet në thjeshtë ekzekutimin e hapat që duhet të riprodhojnë gabimin e shkaktuar nga mangësia dhe verifikimin që gabimi nuk ndodh.
6. Testimi i Regresionit: Testimi i regresionit konfirmon se nuk ka pasur pasojë të dëmshme nga një ndryshim, përfshirë një ndreqje që është tashmë testuar për konfirmim. Këto pasojë të dëmshme mund të prekin të njëjtin komponent ku u bë ndryshimi, komponentë të tjerë në të njëjtin sistem, ose madje sisteme të tjera të lidhura. Testimi i rregullimit mund të mos kufizohet vetëm në objektin e testimit, por mund të ketë lidhje edhe me mjedisin. Është e këshillueshme të bëhet një analizë e ndikimit për të optimizuar shtrirjen e testimit të rregullimit. Analiza e ndikimit tregon cilat pjesë të softuerit mund të ndikohen.

Seritë e testimit të rregullimit ekzekutohen shpesh dhe në përgjithësi numri i rasteve të testimit të rregullimit rritet me çdo iteracion ose version të ri, kështu që testimi i rregullimit është një kandidat i fortë për automatizim. Automatizimi i këtyre testeve duhet të fillojë herët në projekt.

1. Testimi i mirembajtjes: Mirëmbajtja mund të përfshijë korrigjime, përmirësime të planifikuara, ose zgjidhje të menjëhershme të problemeve (hot fixes). Testimi i mirëmbajtjes përfshin vlerësimin e suksesit të zbatimit të ndryshimeve dhe verifikimin e mundësive të rregullimeve në pjesët e sistemit që nuk janë ndryshuar. Shtrirja e testimi të mirëmbajtjes përcaktohet nga rreziku i ndryshimit, madhësia e sistemit ekzistues dhe ndryshimi i kryer.

**Differences between Static and Dynamic Testing – Diferencat midis testimit statik dhe atij dinamik**

Testimi Dinamik: Përfshin ekzekutimin e softuerit për të verifikuar sjelljen dhe funksionimin e tij. Rastet e testimi janë të projektuara dhe ekzekutohen për të vlerësuar performancën e softuerit.

Testimi Statik: Përqendrohet në shqyrtimin dhe ekzaminimin e produkteve të punës, si kodi dhe dokumentacioni, pa ekzekutimin e softuerit. Qëllimi është gjetja e mangësive në fazat e hershme të zhvillimit dhe sigurimi i respektimit të standardeve.

* Natyra e Testimit:
  + Testimi Statik: Është një lloj testimi ku kodi ose softueri nuk ekzekutohet. Përfshin shqyrtimin dhe ekzaminimin e produkteve të punës si kodi, specifikimet dhe dokumentacioni.
  + Testimi Dinamik: Në këtë lloj testimi, softueri ekzekutohet për të verifikuar funksionimin e tij. Rastet e testimi janë të projektuara, dhe kodi testohet gjatë ekzekutimit të tij.
* Qëllimi:
  + Testimi Statik: Qëllimi kryesor është gjetja e mangësive në fazat e hershme të zhvillimit, përmirësimi i cilësisë dhe sigurimi i respektimit të standardeve.
  + Testimi Dinamik: Synon të verifikojë nëse softueri përmbush kërkesat e tij funksionale dhe të performancës dhe zbulon mangësi nëpërmjet ekzekutimit të tij të vërtetë.
* Kur Kryhet:
  + Testimi Statik: Mund të kryhet gjatë të gjithë ciklit të jetës së zhvillimit të softuerit (SDLC), nga analiza e kërkesave deri te shqyrtimi i kodit.
  + Testimi Dinamik: Zakonisht kryhet gjatë fazave më të vona të SDLC, si testimi i sistemit, testimi i integritetit dhe testimi i pranimit nga përdoruesit.
* Teknikat:
  + Testimi Statik: Teknikat përfshijnë shqyrtimet, inspektimet dhe shqyrtimet e dokumentacionit, ku kodi dhe dokumentet ekzaminojnë manualisht.
  + Testimi Dinamik: Teknikat përfshijnë projektimin e rasteve të testimi, krijimin e të dhënave të testimi dhe ekzekutimin e testimi.
* Automatizimi:
  + Testimi Statik: Automatizimi është më i rrallë, megjithëse ekzistojnë mjete për analizën statike.
  + Testimi Dinamik: Automatizimi përdoret gjerësisht, dhe ka shumë mjete testimi për testimin dinamik.

**Teknikat e dizenjemit te testeve 4.1**

1. Teknikat e Testimit të Kutise se Zeze (Black-box): Njihen gjithashtu si teknika të bazuar në specifikacion dhe janë të bazuar në një analizë të sjelljes të specifikuar të objektit të testimi pa referuar në strukturën e tij të brendshme. Prandaj, rastet e testimi janë të pavarura nga mënyra se si është implementuar softueri. Si rrjedhojë, nëse ndryshon implementimi, por sjellja e kërkuar mbetet e njëjtë, atëherë rastet e testimi vazhdojnë të jenë të dobishme.
2. Teknikat e Testimit të Kutise se Bardhë (White-box): Njihen gjithashtu si teknika të bazuar në strukturë dhe janë të bazuar në një analizë të strukturës dhe procesimit të brendshëm të objektit të testimi. Pasqyrohen në mënyrë të ngushtë me mënyrën se si është projektuar softueri, dhe ato mund të krijohen vetëm pas projektimit ose implementimit të objektit të testimi.
3. Teknikat e Testimit të Bazuar në Përvojë: Përdorin efektivisht njohuritë dhe përvojën e testuesve për projektimin dhe implementimin e rasteve të testimi. Efektiviteti i këtyre teknikave varet shumë nga aftësitë e testuesit. Teknikat e bazuar në përvojë mund të zbulojnë mangësi që mund të humbasin duke përdorur teknikat e testimit të kuq dhe të bardhë.

Test management – Menaxhimi i procesit te testimit

Kapitulli "Menaxhimi i Veprimeve të Testimit" është një pjesë e rëndësishme e programit të nivelit bazë të ISTQB, sepse mbulon shkathtësitë dhe njohuritë themelore të nevojshme për të planifikuar, ekzekutuar dhe menaxhuar një projekt testimi të suksesshëm.

Këtu janë disa detaje shtesë për secilën nga temat e mbuluara në këtë kapitull:

Planifikimi i Testimit

Planifikimi i testim-it është një dokument që përshkruan shtrirjen, qasjen dhe burimet e nevojshme për testim. Është e rëndësishme të keni një plan të mirëdefinuar të testim-it para se testimi të fillojë, sepse kjo do të ndihmojë në sigurimin që testimi të kryhet me efikasitet dhe efektivitet.

Monitorimi dhe Kontrolli i Testimit

Monitorimi dhe kontrolli i testim-it është procesi i ndjekjes së përparimit të testim-it, identifikimit dhe menaxhimit të rreziqeve, dhe bërjes së ndryshimeve në planin e testim-it sipas nevojës. Ky është një proces i rëndësishëm, sepse ndihmon në sigurimin që testimi është në rrugë të duhur dhe që çdo problem potencial identifikohet dhe adresohet në fillim.

Analiza e Testimit

Analiza e testim-it është procesi i rishikimit të kërkesave, dizajnit dhe kodit për të identifikuar mangësitë potenciale dhe për të zhvilluar rastet e testim-it. Ky është një proces kyç, sepse siguron që rastet e testim-it janë të plotësueshme dhe efektive.

Dizajni i Testimit

Dizajni i testim-it është procesi i krijoj të rasteve të testim-it që do të verifikojnë që softueri përmbush kërkesat dhe objektivat e testim-it. Ka një shumëllojshmëri teknikash të dizajnit të testim-it që mund të përdoren, si testimi i kutisë së zi, testimi i kutisë së bardhë, dhe testimi i bazuar në rreziqe.

Implementimi i Testimit

Implementimi i testim-it përfshin zhvillimin dhe ekzekutimin e rasteve të testim-it. Ky proces mund të jetë manual ose i automatizuar. Testimi manual është qasja tradicionale e testim-it, ku testuesi ekzekuton rastet e testim-it manualisht. Testimi i automatizuar përdor mjete për të ekzekutuar automatikisht rastet e testim-it.

Ekzekutimi i Testimit

Ekzekutimi i testim-it përfshin ekzekutimin e rasteve të testim-it dhe regjistrimin e rezultateve. Është e rëndësishme të regjistroni me saktësi rezultatet e çdo rasti të testim-it, që ato të mund të analizohen dhe raportohen.

Përfundimi i Testimit

Përfundimi i testim-it përfshin vlerësimin e rezultateve të testim-it, raportimin e gjetjeve dhe mbylljen e projektit të testim-it. Rezultatet e testim-it duhet të vlerësohen për të përcaktuar nëse softueri përmbush kërkesat dhe objektivat e testim-it. Gjetjet e testim-it duhet të raportohen te palët e interesuara relevante. Projekti i testim-it duhet të mbyllet pasi testimi të përfundojë dhe gjetjet të raportohen.

**Testimi I automatizuar INTRO dhe disa nga frameworks dhe tools me te perdorur**

Testimi i automatizuar i softuerit është procesi i përdorimit të mjeteve të specializuara softuerike për të ekzekutuar teste të parashkruara në një aplikacion ose sistem softueri, në vend të testimit manual. Ai përfshin shkrimin e skripteve ose rastet e testimit që mund të ekzekutohen automatikisht për të testuar softuerin, në vend që testuesit të kryejnë çdo rast testimi manualisht. Qëllimi i testimit të automatizuar është të përmirësojë efikasitetin, shpejtësinë dhe saktësinë e procesit të testimit, duke reduktuar përpjekjen e kërkuar për testimin manual.

Mjetet e automatizuara të testimit mund të simulojnë ndërveprimet e përdoruesve me softuerin, si futja e të dhënave, klikimi i butonave dhe lundrimi nëpër ekrane. Mjetet mund të verifikojnë më pas rezultatet kundrejt rezultateve të pritshme, të tilla si kontrolli nëse shfaqen të dhënat e sakta ose nëse mesazhet e gabimit shfaqen kur është e përshtatshme. Testimi i automatizuar mund të përdoret për një sërë llojesh testimi, duke përfshirë testimin funksional, testimin e regresionit, testimin e performancës dhe testimin e ngarkesës.

Përfitimet e testimit të automatizuar përfshijnë përmirësimin e efikasitetit të testimit, rritjen e mbulimit të testeve, reagime më të shpejta dhe reduktimin e rrezikut të gabimit njerëzor. Megjithatë, testimi i automatizuar kërkon ekspertizë në mjetet dhe kornizat e automatizimit të testimit, si dhe aftësi programimi dhe skriptimi. Është gjithashtu e rëndësishme që të dizajnohen dhe mirëmbahen me kujdes skriptet e testimit për t'u siguruar që ato të mbeten efektive dhe të sakta ndërsa softueri evoluon.

* + 1. Perfitimet dhe Rreziqet e testimit te automatizuar - ISTQB

Testimi I automatizuar i softuereve eshte i rendesishem per shume arsye, si:

1. Kursimi i kohës: Testimi i automatizuar mund të ekzekutojë testet shumë më shpejt se testimi manual, duke i lejuar testuesit të ekzekutojnë më shumë teste në më pak kohë. Kjo mundëson reagime më të shpejta dhe mund të zvogëlojë kohën në treg për produktet softuerike.

Konsistenca: Testet e automatizuara kryejnë të njëjtat hapa dhe kontrolle sa herë që ekzekutohen, duke siguruar qëndrueshmëri në procesin e testimit. Kjo zvogëlon gjasat e gabimit njerëzor dhe përmirëson besueshmërinë e rezultateve të testit.

1. Mbulimi: Testimi i automatizuar mund të mbulojë një numër të madh rastesh testimi, duke përfshirë ato komplekse që mund të jenë të vështira për t'u ekzekutuar me dorë. Kjo siguron që të gjitha aspektet e softuerit të testohen tërësisht dhe redukton rrezikun e defekteve të pazbuluara.
2. Ripërdorshmëria: Testet e automatizuara mund të ripërdoren nëpër versione dhe ndërtime të ndryshme të softuerit, duke kursyer kohë dhe përpjekje në krijimin e testeve të reja për çdo version.
3. Me kosto efektive: Ndërsa investimi fillestar në testimin e automatizuar mund të jetë më i lartë se testimi manual, testimi i automatizuar përfundimisht mund të kursejë kostot në afat të gjatë duke reduktuar nevojën për testues manualë dhe duke zvogëluar kohën dhe përpjekjen e kërkuar për testim.
4. Sigurimi i cilësisë: Testimi i automatizuar mund të ndihmojë në përmirësimin e cilësisë së softuerit duke identifikuar defektet në fillim të ciklit të zhvillimit, duke reduktuar gjasat që klientët të zbulojnë defekte pas lëshimit.

Në përgjithësi, testimi i automatizuar i softuerit mund të përmirësojë efikasitetin, efektivitetin dhe cilësinë e procesit të testimit, duke e bërë atë një mjet thelbësor për ekipet moderne të zhvillimit të softuerit.

Ekzistojne shume framework-e dhe mjete per testimin e automatizuar, falas si dhe me pagese. Me poshte permendim disa prej tyre.

**Teknologji falas per automatizimin e softuereve**

1. Selenium WebDriver: Selenium WebDriver është një framework i hapur i përdorur për automatizimin e shfletuesve të internetit. Ai mbështet shumë gjuhë programimi dhe ka një komunitet të madh të përdoruesve dhe kontribuuesve.
2. Appium: Appium është një framework i hapur i përdorur për automatizimin e aplikacioneve mobile. Ai mbështet platformat iOS, Android dhe Windows.
3. Robot Framework: Robot Framework është një framework i automatizimit të testim-it që mund të përdoret për një larmi të llojeve të testim-it. Ai ka një sintaksë të lehtë për t'u përdorur dhe mbështet shumë gjuhë programimi.
4. Cypress: Cypress është një framework i hapur i përdorur për testimin e aplikacioneve web. Ai ofron aftësi testimi të shpejta, të besueshme dhe të lehta për t'u përdorur, me një fokus në testimin end-to-end.

**Teknologji komerciale per automatizimin e softuereve**

1. TestComplete: TestComplete është një mjet komercial i përdorur për testimin funksional automatik të aplikacioneve web, desktop dhe mobile. Ai mbështet shumë gjuhë scriptimi dhe ka aftësi të integruara për menaxhimin e testeve.
2. Ranorex: Ranorex është një mjet komercial i përdorur për testimin automatik të aplikacioneve desktop, web dhe mobile. Ai ka një ndërfaqe të lehtë për t'u përdorur dhe mbështet shumë gjuhë programimi.
3. Telerik Test Studio: Telerik Test Studio është një mjet komercial i përdorur për testimin funksional automatik të aplikacioneve web dhe desktop. Ai ka një ndërfaqe të lehtë për t'u përdorur dhe mbështet shumë gjuhë programimi.
4. Katalon Studio: Katalon Studio është një mjet komercial i përdorur për testimin automatik të aplikacioneve web, mobile dhe API. Ai ka një sistem të integruar për menaxhimin e testeve dhe mbështet shumë gjuhë scriptimi.

**SELENIUM dhe CYPRESS**

**-Selenium**

Selenium is an open-source test automation framework that is widely used for automating web browsers. It was originally created by Jason Huggins in 2004 as an internal tool at ThoughtWorks, a software consultancy firm. The tool quickly gained popularity and was released as an open-source project in 2008. Since then, it has become one of the most popular test automation frameworks in the industry, with a large community of users and contributors.

Selenium provides a suite of tools for automated testing of web applications, including Selenium IDE, Selenium WebDriver, and Selenium Grid. Selenium IDE is a record-and-playback tool that allows users to create automated tests in a user-friendly interface. Selenium WebDriver is a programmatic interface for interacting with web browsers, allowing users to create more sophisticated tests using programming languages such as Java, Python, and C#. Selenium Grid is a tool for distributed testing, allowing tests to be run across multiple machines and browsers simultaneously.

One of the key benefits of Selenium is its flexibility. It supports a wide range of programming languages and can be integrated with a variety of other testing tools and frameworks. It also supports multiple browsers and operating systems, allowing tests to be run across different platforms. Additionally, Selenium is open-source and free to use, making it accessible to a wide range of organizations and individuals.

Overall, Selenium is a powerful and versatile tool for automated testing of web applications. Its popularity and community support make it a valuable asset for any testing team.

**Selenium IDE** (Integrated Development Environment) is a record-and-playback tool for creating automated tests in Selenium. It is a simple and easy-to-use interface that allows testers to record their interactions with a web application and then play back those interactions as automated tests. Selenium IDE also allows testers to manually edit and customize the recorded tests, making it a flexible and powerful tool for automated testing.

One of the key benefits of Selenium IDE is its simplicity. It is a lightweight tool that is easy to install and requires minimal configuration. It also includes a number of useful features such as element locators, assertions, and variables, making it easy to create and customize tests.

However, it is worth noting that Selenium IDE is primarily intended for creating simple and straightforward tests. For more complex tests, testers may need to use other tools in the Selenium suite, such as Selenium WebDriver.

**Selenium Grid** is a tool in the Selenium suite that allows users to run tests across multiple machines and browsers in parallel. It is designed to provide distributed testing capabilities, allowing tests to be executed across multiple machines and browsers simultaneously.

With Selenium Grid, testers can create a hub that acts as a central point for distributing tests to different nodes, which are machines or virtual machines that have browsers installed. The hub can be configured to distribute tests based on a variety of criteria, including browser type, operating system, and available resources.

One of the key benefits of Selenium Grid is its scalability. It allows testers to run tests in parallel across multiple machines and browsers, which can significantly reduce test execution times. It also provides a cost-effective solution for testing across different platforms and browsers, as it enables testers to run tests on machines with different configurations without the need for physical devices.

Selenium Grid is compatible with Selenium WebDriver, which means that tests created using WebDriver can be executed on Grid. This makes it easy for testers to scale their test automation efforts and run tests across a variety of environments.

Overall, Selenium Grid is a powerful tool for distributed testing and can significantly improve test automation efficiency and scalability. Its compatibility with Selenium WebDriver and ability to run tests in parallel across multiple machines and browsers make it a valuable asset for any testing team.

**Selenium WebDriver** is a popular open-source test automation tool that allows testers to automate web applications. It provides a simple and powerful interface for interacting with web elements such as buttons, forms, and dropdowns.

One of the key benefits of WebDriver is its ability to interact with web elements using real browsers, allowing testers to perform more realistic tests. It supports multiple programming languages such as Java, Python, and JavaScript, making it flexible and accessible to a wide range of users.

WebDriver also includes features such as element locators, waits, and assertions, which make it easy to create robust and reliable tests. It also supports headless browsers, making it possible to run tests without a user interface.

Overall, Selenium WebDriver is a powerful and versatile tool for web application testing that is widely used by testing teams around the world. Its flexibility, compatibility, and ease of use make it an ideal choice for both beginners and experienced testers alike.

**PROS/CONS SELENIUM**

Pros:

Open Source: Selenium is an open-source tool, which means it is free to use, and users have access to the source code. This makes it a cost-effective option for test automation.

Cross-Browser Compatibility: Selenium supports multiple browsers such as Chrome, Firefox, and Internet Explorer, making it easy to run tests across different platforms.

Flexibility: Selenium supports multiple programming languages such as Java, Python, and JavaScript, making it accessible to a wide range of users.

Robust Framework: Selenium provides a robust framework for web application testing, with features such as element locators, waits, and assertions.

Community Support: Selenium has a large and active community of users who provide support and share knowledge and resources.

Cons:

Limited Support for Mobile Testing: Selenium is primarily designed for testing web applications, and has limited support for mobile testing.

Requires Technical Knowledge: Using Selenium requires a certain level of technical knowledge and programming skills, which can be a barrier for non-technical testers.

Setup and Maintenance: Setting up and maintaining Selenium can be time-consuming and requires expertise in configuring the test environment.

Flakiness: Selenium tests can be prone to flakiness or inconsistency, which can be challenging to troubleshoot and fix.

Overall, while there are some challenges associated with using Selenium, its flexibility, robust framework, and community support make it a popular choice for web application testing.

Selenium komandat kryesore (Provide examples per secilin)

1. Komandat e browser
2. Locating Web Elements - Gjetja e elementeve të uebit
3. Interacting with Web Elements - Ndërveprimi me Elementet e Uebi
4. Navigation – Lundrimi/Navigimi
5. Assertions - Pohimet

get(url): Kjo komandë hap një faqe interneti në URL-në e dhënë.

back(): Kjo komandë e kthen shfletuesin një faqe prapa.

forward(): Kjo komandë e drejton shfletuesin përpara një faqe.

refresh(): Kjo komandë rifreskon faqen aktuale të internetit.

maximize\_window(): Kjo komandë maksimizon dritaren e shfletuesit.

set\_window\_size(width, height): Kjo komandë vendos madhësinë e dritares së shfletuesit në gjerësinë dhe lartësinë e specifikuar.

get\_current\_url(): Kjo komandë merr URL-në aktuale të faqes së internetit.

get\_title(): Kjo komandë merr titullin e faqes aktuale të internetit.

FirefoxDriver() -

Close() -

Quit() -

find\_element\_by\_id(): Kjo metodë gjen një element nga atributi i saj ID.

find\_element\_by\_name(): Kjo metodë gjen një element sipas atributit të emrit të tij.

find\_element\_by\_xpath(): Kjo metodë gjen një element duke përdorur një shprehje XPath.

find\_element\_by\_link\_text(): Kjo metodë gjen një element nga teksti i saktë i një lidhjeje.

find\_element\_by\_partial\_link\_text(): Kjo metodë gjen një element nga një përputhje e pjesshme e tekstit të lidhjes së saj.

find\_element\_by\_tag\_name(): Kjo metodë gjen një element me emrin e etiketës HTML.

find\_element\_by\_class\_name():Kjo metodë gjen një element me emrin e klasës së saj CSS.

find\_element\_by\_css\_selector(): Kjo metodë gjen një element duke përdorur një përzgjedhës CSS.

Këto metoda përdoren për të lokalizuar elementët e uebit në faqe bazuar në atribute të ndryshme, të tilla si ID, emri, teksti, emri i etiketës, emri i klasës ose përzgjedhësi CSS. Pasi të gjendet një element, mund të ndërveprohet me të duke përdorur komanda të tjera Selenium, të tilla si click(), send\_keys(), ose get\_attribute().

click(): Kjo metodë klikon mbi elementin.

send\_keys(): Kjo metodë dërgon tekst te elementi.

clear(): Kjo metodë pastron tekstin nga një element hyrës.

get\_attribute(): Kjo metodë kthen vlerën e atributit të specifikuar të elementit.

is\_displayed(): Kjo metodë kthen një vlerë boolean që tregon nëse elementi shfaqet aktualisht në faqe.

is\_enabled(): Kjo metodë kthen një vlerë boolean që tregon nëse elementi është aktualisht i aktivizuar.

is\_selected(): Kjo metodë kthen një vlerë boolean që tregon nëse elementi është zgjedhur aktualisht.

Këto metoda mund të përdoren për të bashkëvepruar me elementët e uebit në mënyra të ndryshme, si klikimi mbi butona, plotësimi i fushave të formularit ose marrja e informacionit nga faqja. Për të përdorur këto metoda, fillimisht duhet të lokalizoni elementin duke përdorur një nga metodat për gjetjen e elementeve të ueb-it në Selenium, si find\_element\_by\_id(), find\_element\_by\_xpath(), ose find\_element\_by\_css\_selector(). Pasi të keni gjetur elementin, mund të përdorni këto metoda për të bashkëvepruar me të.

Në testimin e automatizuar, pohimet përdoren për të kontrolluar nëse rezultatet e pritura të një testi përputhen me rezultatet aktuale. Pohimet ju lejojnë të verifikoni nëse disa kushte janë përmbushur, dhe nëse jo, testi do të dështojë.

Këtu janë disa pohime të përdorura zakonisht në Selenium:

* assertEqual(): Ky pohim kontrollon që dy vlera janë të barabarta.
* assertNotEqual(): Ky pohim kontrollon që dy vlera të mos jenë të barabarta.
* assertTrue(): Ky pohim kontrollon që një vlerë boolean është e vërtetë.
* assertFalse(): Ky pohim kontrollon që një vlerë boolean është false.
* assertIn(): Ky pohim kontrollon që një vlerë është e pranishme në një listë, tuple ose varg.
* assertNotIn(): Ky pohim kontrollon që një vlerë nuk është e pranishme në një listë, tuple ose varg.
* assertIs(): Ky pohim kontrollon që dy objekte janë i njëjti objekt.
* assertIsNot(): Ky pohim kontrollon që dy objekte nuk janë i njëjti objekt.

Pohimet janë një komponent kritik i automatizimit të testit, pasi ato ju lejojnë të siguroheni që testet tuaja po prodhojnë rezultatet e pritura. Duke përdorur pohimet, ju mund të kapni gabimet dhe defektet në fillim të procesit të zhvillimit, gjë që mund t'ju kursejë kohë dhe burime në afat të gjatë.

**-Cypress**

Cypress is a modern and popular open-source test automation framework that is designed for testing modern web applications. It was first released in 2014 by Brian Mann, Drew Lanham, and Alan Chang, and has gained significant popularity in recent years. Unlike other testing frameworks, Cypress is not based on Selenium, but rather uses a unique architecture that allows it to run tests in the same context as the application being tested.

One of the key benefits of Cypress is its ease of use. It has a simple and intuitive API that allows developers and testers to write tests using JavaScript. Cypress also includes a user-friendly dashboard that provides real-time feedback on test results, making it easy to debug and troubleshoot issues.

Another benefit of Cypress is its fast execution times. Because it runs tests in the same context as the application being tested, it is able to provide quick and reliable feedback on test results. It also includes built-in support for modern front-end technologies such as React, Angular, and Vue.js, making it ideal for testing modern web applications.

Cypress also includes a number of features that make it ideal for end-to-end testing, including support for network requests and responses, visual testing, and custom timeouts and intervals. It can also be integrated with popular testing frameworks such as Mocha and Chai.

Overall, Cypress is a powerful and user-friendly test automation framework that is ideal for testing modern web applications. Its unique architecture and fast execution times make it a valuable tool for any testing team.

Here are some Cypress commands compared to their equivalents in Selenium:

cy.get(): This is the primary command for selecting DOM elements in Cypress, similar to Selenium's findElement(). However, Cypress automatically retries the selection until the element becomes available or times out.

cy.click(): This command is similar to Selenium's click(), but it automatically waits for the element to become clickable.

cy.type(): This command is similar to Selenium's sendKeys(), but it allows you to type text into an element with more natural keyboard behavior and automatically waits for the element to become available.

cy.visit(): This command is similar to Selenium's get(), but it allows you to visit a URL and automatically waits for the page to finish loading.

cy.contains(): This command allows you to select an element by its text content, similar to Selenium's findElement(By.xpath("//\*[contains(text(),'text')]")).

cy.url(): This command allows you to retrieve the current URL of the page, similar to Selenium's driver.getCurrentUrl().

cy.wait(): This command allows you to pause the test for a specified amount of time, similar to Selenium's time.sleep().

Overall, Cypress provides a more modern and streamlined API for interacting with web elements compared to Selenium. It also has built-in support for waiting for elements to become available and handling asynchronous behavior, which can be more cumbersome in Selenium.

Here are some commands that are present in Cypress but not in Selenium:

cy.intercept(): This command allows you to intercept and modify HTTP requests and responses made by your application, which can be useful for mocking APIs and testing network-related functionality.

cy.clock(): This command allows you to control the built-in JavaScript clock in the browser, which can be useful for testing time-dependent functionality.

cy.wrap(): This command allows you to wrap a non-Cypress object or value and treat it as a Cypress object, which can be useful for integrating external libraries or utilities with your Cypress tests.

cy.fixture(): This command allows you to load and use test data from external JSON or YAML files, which can be useful for parameterizing your tests and separating data from test logic.

cy.task(): This command allows you to execute custom Node.js code as a task within your Cypress test, which can be useful for integrating with external systems or performing complex setup/teardown logic.

Overall, Cypress provides a more extensive and flexible API for testing web applications compared to Selenium, allowing developers to test a wider range of functionality with less code.

Open question: Intro ne Test Complete me pas te treja projektet?

PROJEKTET both– expected 15-30 pages

* TEST CASES per te dyja projektet
* PROJEKTI SELENIUM
* PROJEKTI CYPRESS
* KRAHASIM (ex. Exe time, reports, challenges,)

**PAID AUTOMATION TOOL – Test Complete**

**Intro**

TestComplete is a commercial, comprehensive automation testing tool developed by SmartBear Software. It is renowned in the industry for its versatility, robust feature set, and extensive support for multiple application types, including web, desktop, mobile, and cross-platform applications. In this section, we will delve into the key features, capabilities, and advantages of using TestComplete as an automation testing solution.

4.1 Key Features and Capabilities

TestComplete offers a wide array of features and capabilities that make it a preferred choice for many organizations in their automation testing endeavors. Some of its notable features include:

4.1.1 Multi-platform Support:

TestComplete supports testing across various platforms, including Windows, macOS, and Linux, making it suitable for diverse application ecosystems.

4.1.2 Multi-language Support:

It allows test script development in popular programming languages such as JavaScript, Python, and VBScript, offering flexibility to development teams with different language preferences.

4.1.3 Object Recognition:

TestComplete employs advanced object recognition techniques to identify and interact with elements in the application's user interface. This ensures stability and resilience in test scripts, even when the UI changes.

4.1.4 Test Visualizer:

The tool provides a Test Visualizer feature, allowing testers to capture screenshots and videos during test execution for effective debugging and reporting.

4.1.5 Comprehensive Test Reporting:

TestComplete generates detailed test reports with a wealth of information, including logs, metrics, and visual evidence, to aid in identifying and resolving issues.

4.2 Advantages of TestComplete

4.2.1 Rapid Test Automation Development:

TestComplete's record and playback functionality simplifies test script creation, enabling testers to quickly generate automated tests with minimal coding effort.

4.2.2 Extensive Test Coverage:

It supports testing on various browsers, operating systems, and devices, facilitating comprehensive test coverage to ensure application compatibility.

4.2.3 Integration Capabilities:

TestComplete can integrate seamlessly with various CI/CD tools, test management systems, and issue tracking platforms, promoting collaboration and automation within development pipelines.

4.2.4 Rich Community Support:

SmartBear's strong community and user base contribute to the availability of resources, tutorials, and support, making it easier for teams to adopt and master TestComplete.

4.3 Licensing and Costs

It's important to note that TestComplete is a commercial tool, and its licensing costs may vary depending on factors such as the number of users, project complexity, and required features. Organizations need to consider their budgetary constraints when evaluating TestComplete as an automation testing solution.

4.4 Real-world Application

To demonstrate the real-world applicability of TestComplete, this thesis will include case studies and practical examples of how organizations have successfully utilized TestComplete to improve their testing processes and enhance the quality of their software products.

In summary, TestComplete offers a robust, versatile, and well-supported automation testing solution for organizations looking to ensure the quality and reliability of their software applications. Its extensive feature set, multi-platform support, and integration capabilities make it a valuable asset in the toolkit of automation testers and quality assurance professionals.

**Komandat kryesore**

**Projekti me te njejtat test cases**

**KRAHASIM FREE dhe PAID Framework/tools – tabele krahasuese selenium/cypress vs paid tool tbd**

|  |  |  |  |
| --- | --- | --- | --- |
| **Criteria** | **Selenium** | **Cypress** | **TestComplete** |
| **Open Source** | Yes | Yes | No |
| **License Type** | Apache License 2.0 | MIT License | Commercial |
| **Ease of Setup** | Moderate | Easy | Easy |
| **Programming Languages** | Java, Python, C#, etc. | JavaScript | JavaScript, Python, VBScript, etc. |
| **Browser Compatibility** | Excellent | Excellent | Excellent |
| **UI Testing** | Yes | Yes | Yes |
| **Cross-browser Testing** | Yes | Yes | Yes |
| **Mobile App Testing** | Yes (with Appium or other tools) | Limited (focused on web apps) | Yes |
| **Test Recorder** | Limited (Third-party tools available) | Yes | Yes |
| **Community Support** | Strong | Growing, but smaller than Selenium | Strong |
| **Commercial Support** | Limited (through third-party vendors) | Limited (through third-party vendors) | Yes |
| **Continuous Integration** | Yes (integrates with CI/CD tools) | Limited (requires additional setup) | Yes (seamless integration with CI/CD) |
| **Object Recognition** | Varies by WebDriver implementation | Advanced object recognition | Advanced object recognition |
| **Test Reporting** | Varies by test framework | Limited (additional plugins required) | Comprehensive reporting with visual aids |
| **Licensing Costs** | Free (open-source) | Free (open-source) | Commercial (varies based on usage) |
| **Maturity** | Mature and widely adopted | Rapidly evolving and improving | Mature and feature-rich |

Albanian Version

|  |  |  |  |
| --- | --- | --- | --- |
| **Kriteret** | **Selenium** | **Cypress** | **TestComplete** |
| **Burimi i hapur** | Po | Po | Jo |
| **Lloji i Licencës** | Licenca Apache 2.0 | Licenca MIT | Komerciale |
| **Lehtësia e Ndarjes** | Mesatare | E lehtë | E lehtë |
| **Gjuhët e Programimit** | Java, Python, C#, etj. | JavaScript | JavaScript, Python, VBScript, etj. |
| **Kompatibiliteti me Shfletuesin** | Shumë i mirë | Shumë i mirë | Shumë i mirë |
| **Testimi i GUI-s** | Po | Po | Po |
| **Testimi i shfletuesit të ndryshëm** | Po | Po | Po |
| **Testimi i aplikacioneve mobile** | Po (me Appium ose mjete të tjera) | I kufizuar (i fokusuar tek aplikacionet uebi) | Po |
| **Regjistruesi i Testit** | I kufizuar (mund të përdoren mjete të palëve të treta) | Po | Po |
| **Mbështetja e Komunitetit** | E fortë | Rritet, por më e vogël se Selenium | E fortë |
| **Mbështetja komerciale** | E kufizuar (përmes furnizuesve të palëve të treta) | E kufizuar (përmes furnizuesve të palëve të treta) | Po |
| **Integrimi i Vazhdueshëm** | Po (integrohet me mjete CI/CD) | I kufizuar (kërkon konfigurim shtesë) | Po (integrim i pashëm me CI/CD) |
| **Njohja e Objekteve** | Ndryshon në varësi të zbatimit të WebDriver | Njohje të avancuara të objekteve | Njohje të avancuara të objekteve |
| **Raportimi i Testit** | Ndryshon në varësi të kuadratit të testimi | I kufizuar (kërkon shtesa të shtuara) | Raportim i gjerë me ndihmë vizuale |
| **Kostot e Licencës** | Falas (burimi i hapur) | Falas (burimi i hapur) | Komerciale (varion në varësi të përdorimit) |
| **Maturia** | E pjekur dhe e përdorur gjerësisht | Ndryshon dhe përmirësohet shpejt | E pjekur dhe me karakteristika të pasura |

REFERENCAT