VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ KATEDRA

Mobiliųjų aplikacijų testavimas
Mobile application testing

Bakalaurinis darbas

Atliko: Mantas Stasytis (parašas)

Darbo vadovas: lekt. dr. Vytautas Valaitis (parašas)

Darbo recezentas: asist. Julija Vysockytė-Bilevičienė (parašas)

Vilnius – 2017

SANTRAUKA

Šiame darbe yra sprendžiamos mobiliųjų aplikacijų testavimo problemos su kuriomis susiduriama testuojant kiekvieną dieną. Pateikiami šių problemų sprendimai, jų priežastys. Analizuojamas DEVOPS programų kūrimo proceso modelis. Atliekamas testavimas naudojant Firebase, Xamarn test cloud ir TestDroid debesijos. Testuojamos paprastos Android aplikacijos naudojant testavimo įrankius: Instrumental, Xamarin UI, Xamarin test recorder, Espresso recorder. Pateikiami testavimo rezultatai ir įvertinami testavimo įrankiai.

Raktiniai žodžiai: debesijos. DEVOPS, mobiliųjų aplikacijų testavimas, Android, Firebase, Xamarin test cloud, TestDroid, testavimo tipai, aplikacijos, mobilios programėlės.

SUMMARY

This paper discusses the problems about mobile application testing which we deal while testing every day. Provides solutions for mentioned problems and their causes. Analyses DEVOPS software development process model. The testing was carried out using Firebase, Xamarin test cloud, TestDroid cloud services. Sample Android applications was used for testing and testing was performed using test tools: Instrumental, Xamarin UI, Xamarin test recorder, Espresso test recorder. Obtained test results and provided test tools evaluations.

Keywords: cloud servises. DEVOPS, mobile application testing, Android, Firebase, Xamarin test cloud, TestDroid, testing types, mobile applications.

Turinys

Įvadas

Mobiliųjų aplikacijų rinka yra pakankamai didelė ir toliau didės, nes dabar kiekvienas turintis savo verslą turi arba planuoja kurti mobiliąją aplikaciją. 2015 vienas milijardas išmaniųjų telefonų buvo parduota ir tai dvigubai daugiau nei kompiuterių parduota tais metais. Dabar mobiliųjų aplikacijų vartotojai vidutiniškai praleidžia 30 valandų per savaitę ir naudoja daugiau nei 25 mobilias aplikacijas reguliariai. Apie 46% mobilių aplikacijų vartotojų yra mokėję už savo aplikacijas ir šis skaičius vis didėja. Iki 2017 yra tikėtina, kad mobiliųjų aplikacijų parsisiuntimų skaičius viršys 268 milijardus parsisiuntimų ir tai sugeneruos apie 70 milijonų eurų pajamų. [RASA17]

Mobiliosioms aplikacijoms pasiekus tokia didelę rinką vartotojai turi didelį pasirinkimą tarp mobilių aplikacijų, kurios atlieka panašų ar net tokį patį funkcionalumą. Todėl mobilių aplikacijų kokybė tampa viena iš svarbesnių savybių renkantis mobiliąją aplikaciją. Dabar vis daugiau vartotojų išleidžia pinigus už mobilias aplikacijas, kurios yra kokybiškos, atlieka jiems reikalingas funkcijas ir yra palaikomos. Todėl kokybiškos mobilios aplikacijos yra daug sėkmingesnės aplikacijų parduotuvėje už kitas panašias aplikacijas, kurios nepasižymi kokybę, kurios nėra teisingai ištestuotos.

Tobulėjant išmaniesiems telefonams ir mobiliosioms aplikacijos jų testavimas tampa vis didesnis iššūkis norint pateikti greitai kokybišką ir funkcionalią aplikaciją, kuri atitiktų vartotojų poreikius. Todėl automatizavimo įrankiai tampa vis populiaresni testuojant mobilias aplikacijas, nes jie sutaupo daug laiko ir mobilios aplikacijos rinka gali pasiekti anksčiau, bet kadangi visko negalima automatizuoti tenka vis tiek atliktį rankinį testavimą, norint užtikrinti gerą mobiliųjų aplikacijų kokybę.

Darbo tikslas – išanalizuoti mobiliųjų aplikacijų testavimą, su kokiomis problemomis susiduriama ir kaip galima problemas spręsti laikantis naujausių programų kūrimo procesų metodikos. Šiam tikslui pasiekti reikia įgyvendinti šituos uždavinius:

1. Atlikti mobiliųjų aplikacijų testavimo tipų analizę
2. Išanalizuoti mobilių aplikacijų testavimo problemas, jų priežasties ir pateikti rekomendacijas.
3. Išanalizuoti mobiliųjų aplikacijų testavimą naudojantis DEVOPS programų kūrimo proceso modeliu.

1. Mobiliųjų aplikacijų tipai

Mobilios aplikacijos skirstomos:

- Vietines aplikacijas (angl. vertimas *Native apps*): tai aplikacijos, kurios sukurtos naudoti mobiliems telefonams ir planšetėms.
- Mobilias internetines aplikacijas (angl. vertimas *Mobile web apps*): tai aplikacijos kurios yra sujungtos su serveriu ir naudoja interneto naršykles chrome, safari ir kitas. Jos veikia prisijungdamos prie interneto ar bevielio interneto
- Hibridines aplikacijas (angl. vertimas *hibrid apps*) tai yra kombinacija vietinių ir internetinių aplikacijų. Jos veikia ant mobiliajame įrenginyje ir yra parašytos naudojant HTML5 and CSS ar kitas panašias technologijas.

1.1. Vietinės aplikacijos

Dauguma žmonių mobilias aplikacijas tapatina tik su vietinėmis aplikacijomis arba hibridinėmis aplikacijomis, kurios yra mūsų telefone. Šias aplikacijas galima parsisiųsti iš mobiliųjų aplikacijų parduotuvės, kurios suteikia veiklos ar patobulina jų įrenginius. Dažnai parsiustą aplikaciją yra kontroliuojama aplikacijų parduotuvės iš kur jie ją parsisiuntė ir kur jie sumoka už ją. Šis gan paprastas mechanizmas yra mobiliųjų aplikacijų kūrėjų būdas gauti pelno už savo sukurtą produktą. Vietinės aplikacijos gali prarasti savo potencialius vartotojus, jeigu reklamuojasi tik aplikacijų parduotuvėje, nes internetinės aplikacijos ir hibridinės aplikacijos pranašesnės tuo, kad paieškos varikliai juos įtraukia į paiešką ir jas lengviau surasti, negu ieškant tinkamos vietinės aplikacijos mobiliųjų aplikacijų parduotuvėje.

Vietinės aplikacijos suteikia vartotojams patirties dirbti su jomis ir jų kūrėjams leidžia pasijausti, kaip vartotojams, kad galėtų pateikti savo produktą, kad jis būtų paprastai naudojamas vartotojų ir kad jų sukurtas produktas gebėtų padaryti viską ko iš jo tikisi vartotojas, todėl sunkiausias darbas tenka testuojam, kurie turi užtikrinti, kad padarytas produktas atitiks visus reikalavimus ir veiks paprastai neskaitant didelės įvairovės vartotojų naudojamų mobiliųjų įrenginių ir jų operacinių sistemų įvairovės.

Testuotojai turi užtikrinti visą mobilios aplikacijos gyvavimo laikotarpis veiks tai kaip numatyta: nuo parsisiuntimo, naudojimo iki pat ištrynimo. Ir kaip aplikacija yra atnaujinama reikia užtikrinti ar pateikti atnaujinimai bus tinkami vartotojui ar tai nesugadins jų jau priimtų

darbo eigų. Ir manymas, kad užtenka mobiliąją aplikaciją patestuoti tik ant vienos platformos yra labai klaidingas.

Vietinės aplikacijos yra susietos vienaip ar kitaip su įrenginio techninė įranga ir operacine sistema. Sėkmingo testavimo iššūkį padidina tas, kad vietines mobilias aplikacijas reikia patestuoti ir ant fizinių įrenginių, kuriuos turi palaikyti kuriama aplikacija. Tai pat reikėtų užtikrinti veikimą ir su senesnėmis versijomis šitų operacinių sistemų, nes ne visi įrenginiai naudoja tokias pat operacinių sistemų versijas. Testuojant yra būtina atsižvelgti į atnaujinimus tų operacinių sistemų: ar tai kas pasikeitė gali įtakoti aplikacijos veikimą ypač į tuos atnaujinimus, kurie yra kritiniai operacinei sistemai ir kuriuos vartotojai susidiegs pakankamai greitai.

1.2. Internetinės aplikacijos

Internetinės aplikacijos nėra tikros aplikacijos jos labiau panašios į internetinius puslapius. Bet daugumą sukurtų tokių internetinių aplikacijų įgyvendina funkcionalumą, kad naršyklės visi mygtukai išnyksta ir galima juose naudoti įvairias įrenginio funkcijas brūkštelėjimą, dvigubą paspaudimą ir kitas.

Nors mobilios aplikacijos yra naudojamos ant mobilių įrenginių vietinės ir internetinės aplikacijos labai skiriasi. Vietinės aplikacijos yra parsisiunčiamos ir laikomos mobiliame įrenginyje. Mobilios internetinės aplikacijos yra tiesiog internetinis puslapis, kuris pritaikytas naudoti išmaniesiems telefonams ir planšetėms. Internetinės aplikacijos yra pasiekiamos per interneto naršyklę telefone, tai reiškia, kad nereikia nieko parsiųsti į telefoną, bet reikia interneto ryšio, kad būtų galima prisijungti. Internetinės aplikacijos tuo pranašesnės už vietines aplikacijas, kad jas galima atnaujinti vartotojui net nežinant.

Internetinės aplikacijos visada būtina atsižvelgti į tai, kad dauguma vartotojų jas naudos be bevielio interneto ryšio, dauguma naudos savo mobiliojo plano tiekėjų tiekiamu internetu, kuris bus ganėtinai lėtas dauguma atveju, tai reikia atsižvelgti į tai, kad internetinės aplikacijos turi būti greitos.

Testavimas mobiliųjų internetinių aplikacijų turi vieną aiškų privalumą prieš vietines aplikacijas, tas, kad jas testuojant galima naudotis standartinius naršyklių programuotojų įrankius ir tai labai padeda programuojant, derinant internetinę aplikaciją. Testavimui internetinių aplikacijų sukurtą jau daug naudingų įrankių internete, kur galima peržiūrėti savo

internetinės aplikacijos atvaizdavimą ant daug skirtingų modelių su skirtingomis operacinėmis sistemomis. [THPE15]

1.3. Hibridinės aplikacijos

Hibridinės aplikacijos yra pusiau vietinės, pusiau internetinės. Dažnai dauguma žmonių jas maišo su internetinėmis aplikacijomis. Kaip ir vietinės aplikacijos jos yra prieinamos aplikacijų parduotuvėje ir kaip internetinės aplikacijos jos yra įjungiamos naršyklėje.

Jos populiarios tuo, kad įmonės norėdamos pritraukti vartotojus gali tiesiog sukurti hibridinę aplikaciją, kuri bus paremta jau sukurta internetine aplikacija, bet jie gaus papildomų lankytojų, nes žmonės jas pamatys aplikacijų parduotuvėje.

Hibridinės aplikacijos gali veikti be interneto ryšio ir turi daugiau priėjimo prie įrenginio prietaisų tokių kaip kamera, GPS ir kiti sensoriai.

2. Mobiliųjų aplikacijų testavimo tipai

Kuriant mobilias aplikacijas užtikrinti kodo kokybę nėra paprasta, todėl kokybės specialistai turi gebėti surasti problemas ir jų priežastis, kas jas sukelia. Bereikalingas maigymas mobilios aplikacijos pradžiai gali duoti rezultatų, bet galiausiai reikia išsikelti mobiliai aplikacijai reikalingus testavimo tipus. Dėl šitos priežasties kokybės specialistai sudaro testavimo strategijas. Testavimo strategijos sudarytos iš įvairių testavimo tipų priklausant nuo testavimo iteracijos ir regresijos. Mobilių aplikacijų testavimo tipai skirstomi: funkcinį testavimą, laboratorinį testavimą, našumo testavimą, atminties praradimo testavimą, pertraukimo testavimą, panaudojimo testavimą, įrašymo testavimą, saugumo testavimą, vietos testavimą, pasenusios operacinės sistemos testavimą, apkrovos testavimą.

2.1. Funkcinis testavimas

Funkcinis testavimas užtikrina, kad aplikacija atitinka reikalavimus ir taip veikia. Dauguma testų atliekamą per vartotojo sąsają ir pasirinkimų kvietimą. Funkcinio testavimo metu pildomos formos siunčiama informacija ir tikrinami atsakymai. Tikrinamas funkcionalumas įvairių komponentų vaizdo, garso, grafikos. Šis testavimas yra vienas iš svarbiausių ir atliekamas ankstyvoje programavimo fazėje. Funkciniai testai priklauso nuo to įrenginio modelio, operacinės sistemos, naršyklės, vietos, kalbos ir mobiliojo ryšio tiekėjo.

2.2. Laboratorinis testavimas

Laboratorinis testavimas dažniausiai atliekamas mobiliojo plano tiekėjų simuliuojant bevielį ryšį. Šis testas atliekamas tam, kad būtų aptinkama ar yra sutrikimų, kai mobili aplikacija naudoja įvairias garso, interneto prieigas.

2.3. Našumo testavimas

Našumo testavimas tai testavimo procesas, kuris tikrina mobilios aplikacijos veikimą įvairiomis sąlygomis kai telefono baterija mažai pakrauta, yra prastas ryšys, mažai atminties lieko telefone, keletas mobilios aplikacijos prisijungimų su skirtingais vartotojais į aplikacijos serverį ir kitos panašios sąlygos. Šį testavimą būtina atlikti naudojant skirtingus įrenginius, nes

aplikacijos veikimas priklausys nuo įrenginio greičio ir sąlygų. Nes dabartiniai vartotojai nori greitų ir atsakančių mobiliųjų aplikacijų. Nes susidūrę su problemomis vartotojai nori iškart sprendžiamų problemų arba jie ieškosi kitų panašių aplikacijų. Mobilios aplikacijos, kurios susiduria su mažomis našumo problemomis dažniausiai yra gerai įvertinamos vartotojų.

2.4. Atminties praradimo testavimas

Atminties praradimas nutinka, kai mobili aplikacijai ar serveriui nebepakanka atminties, tai pakankamai sulėtina mobiliosios aplikacijos veikimą ir sulėtina visą mobilų telefoną. Dauguma mobiliųjų telefonų turi nedaug vidinės atminties. Todėl šitas testavimas yra svarbus aplikacijų testavimo scenarijams, kad mobili aplikacija būtų pilnai ištestuota. [BIPA14]

2.5. Pertraukimo testavimas

Pertraukimo testavimas tai kai aplikacijos veikimo metu įvyksta įvairūs pertraukimai, trukdžiai kurie pertraukia mobilios aplikacijos veikla: žinutės, skambučiai, pranešimai, baterijos išėmimas arba išsikrovimas, mobiliojo tinklo praradimas ar atsiradimas, vaizdo ar garso grotuvo veikla fone, GPS signalo praradimas ar atsiradimas ir kiti trukdžiai. Mobili aplikaciją turėtų būti patikrinta šitose situacijose ir atitinkamai reaguoti sustabdyti savo veiklą jei reikia ir pratęsti savo veikla. Aplikacija veikia skirtingai nuo to kokia jos veikla. [SWSE14]

2.6. Panaudojimo testavimas

Panaudojimo testavimas, tai testavimas kai patikrinama ar mobili aplikacija pasiekia jai iškeltus reikalavimus ir vartotojai yra patenkinti aplikacija. Atlikus šitą testavimą galima nuspėti aplikacijos pasisekimą rinkoje. Šiame testavime tai pat peržiūrima ar mobili aplikacija veikia tai pat ar pakankamai panašiai ir ant kitų modelių ar operacinių sistemų Šituo testavimu reikia užtikrinti, kad kitose mobiliuose įrenginiuose negausime kritinių klaidų. Tai vienas iš svarbiausių testavimų prieš paleidžiant aplikaciją į rinką. Tai pat atliekant šitą testavimą patariama naudotis išoriniais žmonėmis, kurie neprisidėjo visiškai prie šitos aplikacijos ir nežino jos funkcionalumo. Tada galima išgirsti pastebėjimus iš vartotojų ir pastebėti problemas su kuriomis jie susiduria. Gavus atsakymus galima suprasti vartotojų nuomonę apie mobilią

aplikaciją. Atlikus tai ankstyvose fazėse galima apsisaugoti nuo nesėkmingų produktų [AVSH15].

2.7. Įrašymo testavimas

Įrašymo testavimas apima aplikacijos įrašymą, naujinimą ir ištrynimą. Mobilios aplikacijos įrašomos skirtingai iš aplikacijų parduotuvės ar kitur, tai pat šio testavimo metu reikia užtikrinti, kad aplikacijos įrašymo metu nekiltų sudėtingų klausimų vartotojui. Šis testavimas svarbus, nes kad ir kokia gera jūsų aplikaciją, bet jei jos nepavyksta įsirašyti į kokios nors tipo telefonus vartotojų skaičius mažėja. Testuojant nebūtina naudoti visus įmanomus įrenginius, bet reiktų pasirinkti populiariausius modelius ir jų populiariausias versijas. Nors dabar jau galima visai paprastai naudojant įrenginių fermas (angl. device farms) pabandyti mobilios aplikacijos įdiegimą ant tūkstančių įrenginių. [XBOS]

2.8. Saugumo testavimas

Saugumo testavimo metu reikia patikrinti ar aplikacija neturi saugumo spragų, autentifikavimo problemų ir nebūna prarandama sesija su vartotoju ar perjungiama į kito vartotojo sąsają, tai pat apsisaugota nuo virtualių įsilaužimų į aplikaciją. Tai atliekant šitą testavimą reiktų atsižvelgti sertifikavimo lygį su mobilia aplikacija kiek prieigos gali turėti aplikacija. Prie kokios vartotojo saugomos informacijos ji gali prieiti. Tai pat aplikacija turi atitikti įrenginių platintojų nustatytas normas. Testuotojai testuodami turėtų įtraukti į savo testavimo scenarijus, testus kurie tikrintų aplikacijos veikimą ir duomenų priėjimą, kai į telefoną prisijungęs žmogus yra tik svečias, kuris negali prieiti prie svarbios informacijos.

2.9. Vietos testavimas

Vietos testavimas susijęs su tuo, kokiai rinkai pateikiama aplikacija, į kokias kultūras reikia atsižvelgti. Norint atlikti šitą testavimą reikia išbandyti aplikaciją naudojant įvairias abėcėlės, kalbų rinkinius. Tai pat šis testavimas reiškia vertimą aplikacijos vartotojo sąsajos, meniu ir paruošimą jos norimai rinkai, kultūrai, kad atitiktų jų standartus.

2.10. Pasenusios operacinės sistemos testavimas

Pasenusios operacinės sistemos testavimas svarbus tuo, kad vartotojai dažnai turi mobiliuosius telefonus senesnių modelių ir nesivargina juose atnaujinti operacinę sistemą ir nevisuose modeliuose net įmanoma atsinaujinti į naujesnę operacinę sistemą, nes telefonas nepalaiko tos versijos. Todėl testuojant aplikaciją reikia apsibrėžti nuo kokios versijos jis veiks ir patestuoti šitas mobiliojo telefono operacinės sistemos versijas.

2.11. Apkrovos testavimas

Apkrovos testavimas, tai kai vienu metu daug vartotojų bando naudoji jūsų aplikaciją, siunčia užklausas, parsisiunčia ar atnaujiną ją. Didelio apkrovos metu aplikaciją gali pradėti veikti lėčiau, nebeveikti ar visiškai sutrikti ir tai privers vartotojus ieškoti alternatyvų ir atsisakyti jūsų aplikacijos. Nors mobiliųjų įrenginių atmintis ir našumas padidėjo per kelis metus, jie vis tiek smarkiai atsilieka nuo kompiuterių. Atliekant funkcinį ar našumo testavimą gali nesimatyti tikrojo aplikacijos veikimo, kai ji veiks piko metu. Norint tiksliai patestuoti mobilią aplikaciją reiktų atlikti tuos pačius testus įvairiomis sąlygomis ir įvairiais greičiais.

3. Mobiliųjų aplikacijų testavimas

Mobilių aplikacijų testavimas yra daug laiko užimantis, brangus, bet būtinas dalykas norint išlaikyti vartotojus. Būtina užtikrinti, kad vartotojas įsijungęs mobilią aplikaciją gerai praleis laiką ir nesusidurs su klaidomis nuo pat pirmo jo prisijungimo. Neatlikus gero testavimo vartotojas tampa testuoju ir ne kaip mobilios aplikacijos testuotojų komanda vartotojas neturi laiko ir noro testuoti jūsų aplikacija ir pranešti apie klaidas. Tai pat elgiantis su vartotoju, kaip su bandomuoju triušiu mobilios aplikacijos kūrėjai praranda prestižo.

Sakykime programuotojai gerai suprogramavo aplikaciją be klaidų, bet vis tiek reikia atlikti testavimas, nes testavimo tikslas, ne tik surasti klaidas aplikacijoje, bet nustatyti jūsų aplikacijos kokybę, ar ji teisingai veikia, ar atitinka visus reikalavimus, ar ji atitiks vartotojų norus.

Mobilios aplikacijos testavimas yra ganėtinai sudėtingas ir sukelia daug iššūkių. Yra daug pasirinkimų ir galimybių iš kurių reikia pasirinkti reikalingas technikas ir metodologijas norint sėkmingai ištestuoti mobilią aplikaciją. Kiekvienas pasirinktas testavimo metodas turės pliusų ir minusų todėl tampa aišku, kad nėra vieno testavimo metodo, kuris visiškai tenkina. Todėl tenka apgalvoti testavimo strategijas, kurios sujungia kelias testavimo metodikas, kad suteiktų visišką mobilios aplikacijos ištestavimą atsižvelgiant į kainą, kaštus ir laiką[VVHE15].

4. Pagrindiniai mobiliųjų aplikacijų testavimo įšūkiai

Nuolatinis testavimo tobulinimas parodo, kad susiduriama su įvairiai iššūkiais. Testuojant iškyla daug įvairiausių problemų, kurias sukelia pagrinde dvi priežastys: mobiliųjų telefonų įvairovė ir jų operacinių sistemų įvairovė.

4.1. Mobiliųjų telefonų įvairovė

Kadangi mobiliųjų telefonų ir operacinių sistemų įvairovė yra labai plati, kad pavyktų įveikti šitą iššūkį yra trys alternatyvos. Galima testuoti aplikacijas naudojant realius prietaisus, galima testuoti naudojant emuliatorius arba galiausiai pasitelkti kombinacija abiejų kas tikriausiai yra geriausias sprendimas, nes naudojant kitus būdus susidursite su nepatogumais, kad nepavyks visko ištestuoti pakankamai gerai, kad užtikrintų kokybę arba testuojant tik su

realiais jums teks atsisakyti pakankamai daug platformų, norint pateikti produktą kokybišką ir naudojant kuo mažiau laiko.

4.2. Mobiliųjų įrenginių operacinių sistemų įvairovė

Realūs įrenginiai turi privalumą, kad testuojant juos galima pamatyti visus jų trūkumus ir keistenybes kurios pasireiškia tik naudojant tokią techninę įrangą, operacinę sistemą, turint tam tikrų mikroprogramų rinkinį, kurias turės jūsų potencialūs vartotojai.

Tačiau testavimas su realiais įrenginiais yra pakankamai brangus ne visos kompanijos gali sau tai leisti, o ypač nedidelės kompanijos ar programuotojai, kurie kuria mobilias aplikacijas vieni patys. Galbūt pavyktų ir pasiskolinti iš įrenginių gamintojų prietaisus, bet dažniausiai jums teks prisijungti į laukiančiųjų sąrašą, kur jus prietaisus gausite, tik gal po kelių mėnesių ar net metų ir jums tikriausiai neužteks vienos įmonės prietaisų, todėl teks prašyti pasiskolinti prietaisus ir iš kitos kompanijos, kuri galbūt net neskolina mažoms komandoms arba išvis neskolina. Aišku pasiskolinus vis tiek jums teks mokėti kažkokius mokesčius, taigi išlaidų bus daugiau. Ir galiausias testavimas pagaliau gavus reikalingus prietaisus tikriausiai nevyks sklandžiai ir reiks daug laiko sėkmingai ištestuoti aplikaciją ant vieno prietaiso, jeigu dar testavimo procesai nėra išdirbti tai užims pakankamai laiko, kol galėsite sėkmingai vykdyti testavimo procesą įvykdyti testus, surinkti rezultatus, registruoti klaidas ir jas atkartoti vykdant viską tvarkingai paeiliui neperšokant kokių nors žingsnių [DAKN12].

5. Mobiliųjų aplikacijų testavimo įrankiai

Mobiliųjų aplikacijų testavimo įrankių pasirinkimas yra labai platus. Testavimo įrankiai vis keičiasi, tobulėja. Įrankių įvairovė sukelia sunkumą renkantis tinkamą įrankį testavimui. Testavimą ir testavimo įrankius galima skirstyti į du tipus rankinį ir automatinį testavimą.

5.1. Rankinis testavimas

Rankinis testavimas yra svarbi dalis mobiliųjų aplikacijų testavimo. Atliekant rankinį testavimą galima atlikti ne tik testus bet ir analizuoti aplikacijos veikimą, defektų priežastis. Dažniausiai rankinis testavimas pasižymi neefektyvumu ir lėtu darbu, bet šis testavimas leidžia atlikti sudėtingus testus, kurie neišeitų atkartoti automatiškai.

5.1.1. Realūs įrenginiai

Testuojant mobilią aplikaciją būtiną juos išbandyti ant realių įrenginių, kad galėtumėte susidaryti vaizdą, kaip jie atrodys ir veiks skirtinguose įrenginiuose. Dažnai prieiga turėsite tik prie kelių pagrindinių įrenginių. Todėl norint ištestuoti ant daugybės įrenginių verta pasinaudoti interaktyvių debesų servisu, kuris turės priėjimą prie daugelio įrenginių. Testavimas ant realių įrenginių dažnai ilgai dėl visų prisijungimų perdavimų aplikacijos atnaujinimų ir viso kito.

5.1.2. Emuliatoriai

Testuojant mobilią aplikaciją galima naudoti emuliatorius. Kita vertus emuliuojamus įrenginius labai paprasta valdyti. Jus galite laisvai pasirinkti norimą prietaisą ir jo nustatymus ir užkrauti jo duomenis ir jūs iškart turite naują įrenginį, kuris ganėtinai panašiai atspindi kaip jūsų vietinė ar internetinė aplikacija atrodys realiame prietaise. Kadangi emuliatoriai veikia ant ganėtinai galingų personalinių kompiuterių ir serverių jie tai pat gali surinkti įvairius duomenis apie veikimą įvairių procesų jūsų emuliuojame įrenginyje, kliento ir serverio bendravimą. Didžiausias emuliatorių trūkumas yra tai, kad jie neturi trūkumų kokius turi realūs įrenginiai, jie neturės visų realių įrenginių trūkumų Tai pat emuliuojamas įrenginys negali atvaizduoti vaizdo tai kaip matysite realiame įrenginyje visiškai taip pat. Emuliatorius kol vykdys įvairias jūsų aplikacijų užklaus jis naudos jūsų kompiuterio ar serverio apdorojimo galia, nors tai gali pasirodyti kaip privalumas, kad jis veikia greitai, bet iš tikrųjų jis labai lengvai gali paslėpti silpnas jūsų aplikacijų vietas. Emuliuojamas įrenginys tai pat neatsižvelgs į įvairias sąlygas, kurios gali trukdyti jūsų telefonui, tai dažniausiai geras dalykas, nes jūs nepertraukiamai galėsite dirbti, bet norint įsitikinti kaip veiks jūsų aplikaciją realiomis sąlygomis tam tikrose situacijose jums reiktų realaus įrenginio [SOTH16].

5.1.3. Realūs telefonai ir emuliatoriai

Tačiau nėra apribojimo galima pasirinktį abu būdus realius prietaisus ir emuliatorius testuojant mobiliąją aplikaciją. Pradžiai verta testuoti ant emuliuojamų įrenginių, kad galėtumėt pasinaudoti emuliatorių greičiu ir įrenginių pasirinkimu. Testuojant mobiliąją aplikaciją, kuo

anksčiau gali padėti jums pasiekti jūsų tikslus greičiau ir už pakankamai mažą kainą. Pradžioje produkto gyvavimo ciklo jums nereikia visiškai teisingo atvaizdavimo, jums svarbu, kad veiktų pagrindinis funkcionalumas. Pradžioje testų kiekis ir testuojamų įrenginių kiekis yra svarbesnis nei žinojimas, kad jūsų aplikaciją tikrai teisingai veikia ant vieno modelio įrenginio ir tokios operacinės sistemos. Realius įrenginius geriausia pridėti jau vėlesniame programavimo cikle tai padės jums patikrinti ar realūs įrenginiai veikia taip kaip numatytą ir galėsi atlikti testus, kurie neįmanomi ant emuliatoriaus, taip įsitikinti ar jūsų mobili aplikacija atitinka visus reikalavimus ir įvykdo visus užsibrėžtus tikslus [SOTH16].

5.2. Automatinis testavimas

Automatinai testai padidina testavimo efektyvumą, greitį ir pagreitiną aplikacijos išleidimą visuomenei naudoti. Mobiliųjų aplikacijų testavimas labai skiriasi nuo kompiuterio programų testavimo, todėl sukurta daug įvairiausių įrankių ir praktikų mobiliųjų aplikacijų testavimui. Bet darydamas to rankiniu būdų nėra efektyvu ir neatitinka aplikacijų kūrimo metodikos. Todėl testų automatizavimas suteikė daug pranašumo aplikacijų kūrėjams, kurie tuo užsiima, prieš kūrėjus, kurie naudoja tik rankinį testavimą: pagreitino jų darbą ir jų darbo kokybę.

Testų automatizavimas siūlo galimybę ištestuoti mobilias aplikacijas greitai ir efektyviai. Kai testai yra automatizuoti jie gali būti paleidžiami daug kartų. Ir tai labai palengvina regresinį testavimą, mobilių aplikacijų, kurios jau gyvuoja ilgai. Tai pat leidžia atlikti testus, kurie negali būti atliekami rankiniu būdu arba truktų labai ilgai.

Sukūrus automatinius testus juos reikia kartais kažkiek atnaujinti, bet jiems jau nebereiks skirti resursų, jų paleidimas nieko nekainuos ir sutrumpins pakartotinio testavimo laiką. Automatizuotus testus galima atlikti kiekvieną kartą atnaujinus aplikaciją, prieš kiekvieno atnaujinimo išleidimą ir jeigu reiktų visus šituos testus atlikti rankiniu būdu ant visų platformų tai būtų labai neefektyvu.

Testavimo automatizavimo įrankiai turi būti kurie gali testuoti ant visų įrenginių tipų ir operacinių sistemų versijų, kad padidėtų aplikacijos ištestavimas. Automatiniai metodai leidžia patikrinti reikalavimų išpildymą ir sukurti automatiškai testus. Pilnas automatizavimas yra didelė prabanga už kurią nevisos kompanijos yra pasiruošusios mokėti.

Automatinis testavimas padidina aplikacijos ištestavimą, nes programuotojams, testuotojams nebereikia atlikti begalės rankinių testų, kurie užima daug laiko. Jie šitą laiką gali investuoti geriau kuriant naujus testus ar naują funkcionalumą.

Automatiniai testai yra sprendimas, problemos kaip pateikti kokybišką produktą. Automatiniai testai padidina pardavimo rezultatus sutrumpindami testavimo laiką, kainą ir produkto kūrimo laiką. Yra trys būdai, kaip galima automatizuoti testus mobilioms aplikacijomis: ranka rašyti testavimo scenarijai, atkartojimo testai, automatiniai testavimo pratimai [VVHE15].

5.2.1. Ranka rašyti testavimo scenarijai

Ranka rašyti testavimo scenarijai tai dažniausiai tai geriausias pasirinkimas, kai žmonės rašantis šituos testus turi programavimo žinių. Yra daug įrankių kurie pritaikyti rašyti testavimo scenarijams. Bet parašyti testavimo scenarijus programuotojams, testuotojams užtruks laiko, bet galutinis produktas bus tai ko jus norite: gerai apgalvoti scenarijai, kurie testuoja tai ką jus norite ištestuoti jūsų mobilioje aplikacijoje.

5.2.2. Atkartojimo testai

Šitas pasirinkimas turi mažesnę galimybę būti klaidingas, nes nereikia nieko rašyti, bet šiuo atveju funkcionalumas kurį norite ištestuoti, bus ribotas. Testai gali būti lengvai įrašomi ir atkartojami ir ant skirtingų įrenginių ir operacinių sistemų. Šitie testai susitelkia ties vartotojo veiksmais ir veikla. Keletas dalykų apriboja šitą testavimą, nes bus sunku ištestuoti jei reikia integracijos su kitos technologijomis ir palaikymu su kitomis programomis.

5.2.3. Automatiniai testavimo pratimai

Automatiniai testavimo pratimai suteikia galimybę greitai patikrinti ar aplikacijoje veikia pagrindinės funkcijos. Nereikia jokių atskirų testų tiesiog šis testavimo būdas susitelkia ties pagrindinėmis funkcijomis: meniu atidarymo, mygtuko spaudimo, brūkštelėjimo ir kitomis bendravimo funkcijomis. Automatiniai testavimo pratimai suteikia mažiausiai naudingų rezultatų, bet suteikia greitai aplikacijos patikra prieš kiekvieną iteraciją.

Ranka rašyti testavimo scenarijai yra tikslūs, sukurti specialiai kažkokiam funkcionalumui testuoti, suteikia daug galimybių pasirinkimui įvairių įrankių ir karkasų, bet užima daug laiko ir gali būti parašyti klaidingai, nes yra žmogaus klaidos faktorius.

Atkartojimo testai greitai sukuriami, tikslūs, maža galimybė žmogaus klaidoms, įrankių pasirinkimas gan platus, bet tik keletas karkasų palaiko tokius testus. Automatiniai testavimo pratimai yra greičiausias ir daugiausiai automatizuotas procesas, kuris puikiai tinka pagrindinių funkcijų testavimui, bet nėra tikslus kaip realūs testai.

5.2.4. Ties kuo reikia susitelkti testuojant mobilias aplikacijas

Vartotojo sąsaja ir funkcionalumas yra pagrindiniai dalykai, kurie įtakoja mobilios aplikacijos pasisekimą rinkoje ir reikia užtikrinti, kad šitie aspektai mobilios aplikacijos veiks kaip veikę po mobilių įrenginių operacinių sistemų atsinaujinimų. Todėl reikia ištestuoti įvairius aspektus vartotojo sąsajos [MAHE15] [VVHE15]:

- Elementų ir maketo išdėstymą, kai kurioms aplikacijoms rezoliucija yra ypatingai svarbi todėl reiktų atlikti regresinį testavimą po kiekvieno pasikeitimo susijusio su maketu, kad būtų galima užtikrinti aplikacijos kokybę, testuoti šituos aspektus pakankamai lengva, nes yra įvairių automatizavimo įrankių tereikia automatizuoti testus ir bus galima juos pernaudoti.
- Ekrano pozicija. Gan daug mobilių aplikacijų neatsižvelgia į tai, kad jų aplikacijos gali būti paleistos virtualiai ar horizontaliai ir nežino, kaip turėtų tiksliai veikti, todėl būtina kūrimo apsispręsti kaip bus galima naudoti aplikaciją ir kaip visas funkcionalumas turėtų veikti ir ar išviso turėtų veikti.
- Ekrano rezoliucija. Android operacinė sistema pasižymi tuo, kad turi labai daug įvairiausių rezoliucijų, todėl naudojamas automatinis mastelio nustatymas, bet dėl to yra būtinas testavimas ant įvairių rezoliucijų.
- Grafikos našumas. Našumas turėtų būti toks pat nepriklausant nuo įrenginio. Todėl verta ištestuoti ant kiek galimą daugiau realių įrenginių, kad būtų galima žinoti kaip veikia aplikacija reiktų atlikti testus kurie trunka valandas, kad būtų galima įsitikinti, kaip veikia aplikacija esant didelei apkrovai telefone ir ar aplikacija gali efektyviai susidoroti, kai reikia veikti ilgą laiką.
- Saugumas ir patikra. Dauguma programuotojų naudoja atviro kodo komponentus savo aplikacijose. Šita praktika yra visuotinai pripažinta ir rekomenduojama, nes tai palengvina

aplikacijos kūrimą. Bet reiktų vis tiek atsižvelgti į kodo silpnas vietas ir peržiūrėti ar nėra licencijavimo problemų , ko daugumą aplikacijų kūrėjų neatlieka.

- Interaktyvus debesų servisas, norint pritaikyti aplikaciją plačiai rinkai reikėtų naudoti interaktyvų debesų servisą, kur galima gauti šimtus įrenginių iškart testavimui. Paleisti automatinius testus tokiems servisams nėra sunku, tai pat šie servisai suteikia visą reikalingą informaciją apie aplikacijos veiklą.

5.2.5. Testavimo įrankiai

Automatinių testavimo įrankių pasirinkimas yra gan platus ir gan sudėtinga išsirinkti įrankį, kuris atitiktų mobilios aplikacijos standartus ir aplikacijos kūrimo komandos standartus. Nes kuriant mobilią aplikaciją ne visada prireiks atkartojimo testų tai jau priklausys nuo komandos pajėgumo rašant automatinius testus, tai pat nuo aplikacijos tipo. Todėl renkantis testavimo įrankį reikia atsižvelgti kokiai operacinei sistemai pritaikyta aplikacija, kokie komandos sugebėjimai rašant testus, kokie testai bus vykdomi testuojant mobilią aplikaciją ir kita. Todėl verta paminėti testavimo įrankius, kurie yra pakankamai skirtingi ir turi pranašumą skirtingose testavimo situacijose [VVHE16].

Robotium yra automatinis Android operacinės sistemos testavimo įrankis, kuris pilnai palaiko vietines ir hibridines aplikacijas. Šis įrankis leidžia lengvai rašyti automatinius juodos dėžės testus Android aplikacijoms. Testuotojai naudodami šitą įrankį gali lengvai rašyti testavimo scenarijus testavimui.

Ulautomator yra paprastas įrankis leidžiantis lengvai testuoti vartotojo sąsają. Šis įrankis geba kurti automatiškai funkcinius testavimo scenarijus, kurie gali vykdomi aplikacijoms realiuose įrenginiuose ir emuliatoriuose. Tai pat šis įrenginys siūlo grafinį įrankį, kuris leidžia analizuoti vartotojo sąsajos komponentus.

Espresso yra gan naujas automatinio testavimo įrankis, kuris yra atviro kodo ir leidžia programuotojams ir testuojamas geriau testuoti vartotojo sąsają. Šitas įrankis turi paprasta aplikacijų programavimo sąsają, dėl kurios lengva pritaikyti įvairiems aplikacijų tipams. Todėl ja naudojantis paprasta parašyti patikimus Android aplikacijos vartotojo sąsajos testus.

Calabash palaiko Android ir IOS operacines sistemas vietinėms ir hibridinėms aplikacijoms. Šito įrankio privalumas, kad net ne techniniai žmonės gali perprasti ir įvykdyti automatinius patvirtinimo testus.

Appium yra aplikacijų automatinių testų rašymo įrankis, kuris tinka visiems aplikacijų tipas ir palaiko Android ir IOS operacines sistemas. Šis įrankis yra geras pasirinkimas jeigu kuriate aplikaciją ar žaidimą, nes aplikacijos abiem atvejais yra ganėtinai panašios ant abiejų operacinių sistemų. Parašyti testai gali būti leidžiami ir ant Android ir ant IOS operacinių sistemų. Kitas gan reikšmingas šito įrankio privalumas yra tai, kad programuotojai gali pasirinkti jiems patinkančią kalbą ir ją naudoti. Kalbą pasirinkimas labai platus galima rinktis iš Java, Objective-C, JavaScript, PHP, Ruby, Python, C# ir dar kitų. Leidžia vartotojams leisti testus ant mobilų renginių nepriklausant nuo jų operacinės sistemos. Šio įrankio architektūrinė sistema yra gan paprasta ir programuotojams suteikia pagalba nepriklausant nuo jų pasirinktos programavimo kalbos. Tai pat gali vykdyti paprastus atkartojimo testus.

6. Automatinių įrankių rekomendacijos

6.1. Programų kūrimo proceso modelis DEVOPS

Automatinių įrankių pasirinkimas yra labai platus todėl įrankius bandžiau rinkti pagal programų kūrimą procesą DEVOPS. Šis programų kūrimo procesas yra visai naujas programų kūrimo procesas, kuriame taikomos praktikos lanksčiųjų (angl. agile) ir lieknųjų (angl. lean) metodikų darbo praktikos. DEVOPS tai praktika operacijų, kai programuotojai, inžinieriai kartu dalyvauja visame kūrimo gyvavimo cikle nuo pateitkties šablonų sudarymo per visą vystymo procesą iki pat programos palaikymo. Mobiliųjų DEVOPS programų kūrimo procesas susitelkia į komunikacija, integracija, diegimą, automatizaciją ir bendradarbiavimą tarp programuotojų, testuotojų, kokybės specialistų. Visas šitas programų kūrimo procesas remiasi naudojimų tinkamų įrankių, metodų ir infrastruktūros, kad būtų galima sukurti, pateikti ir ištestuoti mobilią programą greitai ir efektyviai [ERMU16].

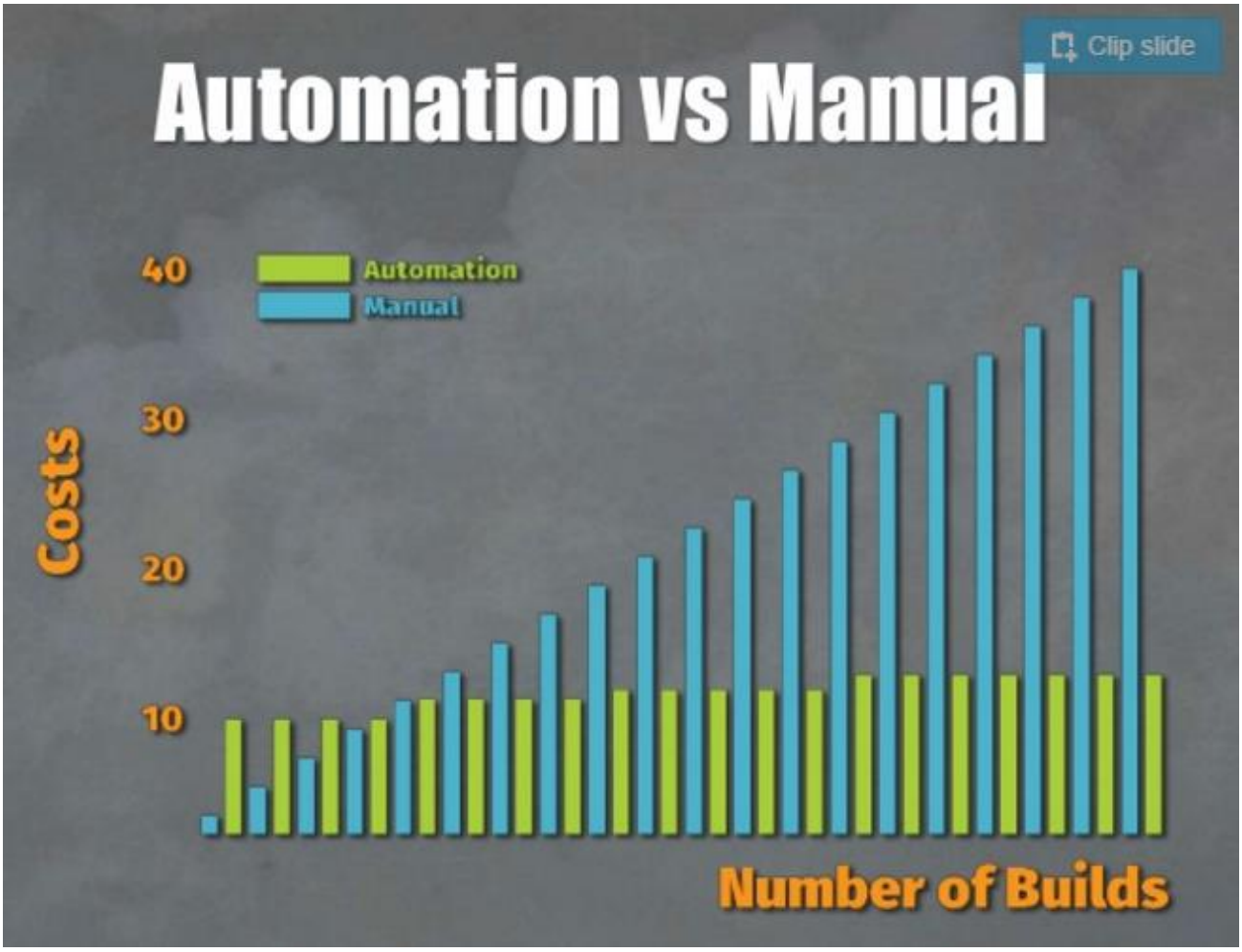
6.2. Mobiliųjų aplikacijų testavimas pagal DEVOPS

DEVOPS programų kūrimo procesas skatina taikomas programas testuoti naudojant realius įrenginius, nes vartotojai mobiliąsias aplikacijas naudos ant jau savo realių įrenginių jie nesinaudos emuliatoriais. Emuliatorių ir modeliavimo įtaisų naudojimas testavime galimas tik labai ankstyvose kūrimo fazėse, kol dar nėra daug funkcionalumo ir procesų, kuriuos galės naudoti vartotojas. Atsiradus funkcionalumui ,kuri jau vartotojas turės vienintelis priimtinas jo testavimas yra naudojant realius įrenginius. Dabar įmonėms nėra būtina išleisti daug pinigų investuojant į realius įrenginius, nes tą patį testavimą galima pasiekti naudoja debesų servisus, kurie pasiūlys galimybę testuoti su realiais mobiliais įrenginiais jų neperkant ir tai yra priimtina naudojant DEVOPS programų kūrimo procesą [VVHE16].

Mobiliųjų aplikacijų testavimas rankiniu būdu neatitinka lanksčiųjų (angl. agile) ir DEVOPS metodologijos praktikų. Šis gan naujas programų kūrimo procesas skatina viska ir kuo anksčiau automatizuoti. Nors pradžioje bus gan dideli kaštai skiriami testavimui, bet tai smarkiai sutrumpins testavimo laiką vėlesnėse fazėse.

DEVOPS efektyvumas norima pasiekti galutinį rezultatą naudojant minimalius resursus. Siekiant pagerinti efektyvumą ir našumą jau pradinėse fazėse gali pagerinti viso programos kūrimo proceso produktyvumą. Todėl automatinis testavimas yra būtinas kiekvienoje

aplikacijos kūrimo fazėse. Testų automatizavimas suteikia daug privalumų, nes galima tikrinti iš anksto ar produktas atitinka iškeltus palaikymo reikalavimus, lengviau surandami defektai ir galima iškart juos taisyti. Šitaip pastebėti defektai ištaisomi anksčiau ir nenusikelia į vėlesnes produkto versijas. Kadangi programos klaidos būna pastebimos vos tik atnaujinant taikomąją programą todėl galutinio produkto kokybė iškart gerėja, nes testavimas atliekamas naudojant realius įrenginius [MDOM].



1 pav. Automatinio ir rankinio testavimo palyginimas gausėjant mobilios aplikacijos funkcionalumui [VVHE16]

6.3. DEVOPS testavimo įrankiai ir testų automatizavimo karkasai

DEVOPS siūlo testuojant naudoti tik realius įrenginius, o neturint daug realių įrenginių reikia naudoti debesijos paslaugos, kurios suteikia priėjimą prie daugelio įrenginių ir palaiko jų kokybę. Todėl norint išanalizuoti ir pateikti rekomendacijas pasirinkau kelis debesijos paslaugų tiekėjus: Firebase, Xamarin test cloud, TestDroid. Nes šie servisai siūlė bandomąjį laikotarpį, kurio metu buvo galima panaudoti jų įrankius, tai pat šie servisai buvo pakankamai skirtingi turėjo skirtingų testavimo įrankių ir tai suteikė galimybę palyginti testavimo įrankių galimybes. Tai pat buvo galima rinktis iš daugiau debesijos servisų kaip pavyzdžiui Amazon Web Services device farm, Kobitron, Perfecto, bet jų bandomasis laikotarpis arba būdavo per trumpas arba negalimas registruojantis be mokėjimo kortelės. Todėl šie servisai buvo atmesti [DARA17].

Lentelė 1. Debesijos ir jų pasirinkimo tikslas

Debesija	Kūrėjas	Pasirinkimo tikslas
Firebase	Google	Bandomasis laikotarpis: galimybė naudotis 5 realius įrenginius per dieną, Galimybė testuoti naudojant realius android įrenginius Testavimo įrankiai: Robo test, Instrumental Testavimo rezultatų peržiūra.
Xamarin test cloud	Xamarin	Bandomasis laikotarpis: galimybė naudotis 2 realius įrenginius visą bandomąjį laikotarpį 30 dienų, automatinis testų diegimas į debesijos servisą. Galimybė testuoti naudojant realius android, IOS, windows įrenginius Testavimo įrankiai: Xamarin test recorder, Xamarin UI test, Calabash Testavimo rezultatų peržiūra
TestDroid	BitBar	Bandomasis laikotarpis: galimybė naudotis 2 realius įrenginius visą bandomąjį laikotarpį 30 dienų, Galimybė testuoti realius android ir IOS įrenginius Testavimo įrankiai: AppCrawler, Intrumental, Calabash Testavimo rezultatų peržiūra

Testavimas buvo atliekamas naudojant tik Android operacinę sistemą, nes tai leido naudoti daugiau skirtingų testavimo įrankių ir norint ištestuoti kitas operacines sistemas trūko reikalingų testavimo priemonių. Testavimui buvo pasirinkti įrenginiai LG Nexus 5 versija 5.0.1 ir LG Nexus 6.0.1, nes šie įrenginiai buvo prieinami pasirinktose debesijos servisuose ir pagal android versijų pasiskirstymą šitie įrenginiai yra vis dar aktyviai naudojami (žr. priedus lentelė 14)

Lentelė 2. Testavimo įrenginių specifikacija

Modelis	Versija	API	CPU	Atmintis	Vieta	Raiška
LG Nexus 5	5.0.1	21	Quad core 2.3 GHz Krait 400	2048 MB	16 GB	Full HD 1920x1080
LG Nexus 5	6.0.1	23	Quad core 2.3 GHz Krait 400	2048 MB	16 GB	Full HD 1920x1080

6.3.1. Automatiškai sukurti testai

Kai kurie iš šių debesijos servisų siūlo savo automatišką mobilios aplikacijos vartotojo sąsajos testavimą: Firebase turi Robo test. TestDroid turi AppCrawler. Išbandžius testavimą naudojant šitas testų kūrimo programas buvo gauti šitokie rezultatai. Kad būtų galima lyginti rezultatus pasirinkta testavimui naudojant realius įrenginius ir tokius pat modelius.

Lentelė 3. Debesijos servisai gebantys kurti automatinius testus

Debesijos	Kūrėjas	Testavimas
Firebase	Google	Robo test – automatiškai sukurtų testų programa
TestDroid	BitBar	AppCrawler – automatiškai sukurtų testų programa

Pirmam testavimui buvo pasirinkta labai paprasta programa (žr. priedus 14 pav.) visi testavimo servisai pateikė labai panašius rezultatus. Testų kūrimo programoms pavyko atlikti visus scenarijus ir patestuoti aplikaciją.

Lentelė 4. BasicSample automatiškai sukurtų testų rezultatai

Testų kūrimo programa	Aplikacija	Įrenginys	API	Testų skaičius	Testų trukmė
Robo test	BasicSample	LG Nexus 5 4.4.4	19	1	72s
Robo test	BasicSample	LG Nexus 5 5.0.1	21	1	74s
Robo test	BasicSample	LG Nexus 5 5.1.1	22	1	72s
Robo test	BasicSample	LG Nexus 5 6.0.1	23	1	79s
AppCrawler	BasicSample	LG Nexus 5 5.0.1	21	1	77s
AppCrawler	BasicSample	LG Nexus 5 6.0.1	23	1	85s

Tolimesniam, testavimui buvo pasirinkta kiek kitokią mobilioji aplikacija (žr. priedus 15 pav.). AdvanceSample aplikacijoje reikėjo telefonui atlikti paveikslavimo funkciją. Abi automatinių testų kūrimo programoms pavyko įsijungti programą, bet nė vieną nesugebėjo padaryti nuotraukos (AppCrawler pavyko įsijungti kamerą) . Nors automatiškai sukurti testai parašė, kad programa veikia teisingai stebint atlikti veiksmus aplikacija nebuvo ištestuota.

Lentelė 5. AdvanceSample programos testavimo rezultatai

Testų kūrimo programa	Aplikacija	Įrenginys	API	Testų skaičius	Testų trukmė	Testų rezultatai
Robo test	AdvanceSample	LG Nexus 5 5.0.1	21	1	32s	Passed
Robo test	AdvanceSample	LG Nexus 5 6.0.1	23	1	36s	Passed
AppCrawler	AdvanceSample	LG Nexus 5 5.0.1	21	1	54s	Passed
AppCrawler	AdvanceSample	LG Nexus 5 6.0.1	23	1	58s	Passed

Testuojant panašias aplikacijas kur yra įvesties laukais, kitoks funkcionalumas kaip kamerą, autofill pasirinkimai testavimo įrankiai jų ištestuoti sėkmingai nesugebėjo. Rezultatai buvo pateikiami, kad aplikacija veikia, nors pagrindinis funkcionalumas buvo nepaliestas. Todėl tolimesniam testavimui buvo pasirinkta mobili aplikacija, kuri turi daug būsenų, mygtukų.

Lentelė 6. RetrofitSample programos testavimo rezultatai

Testų kūrimo programa	Aplikacija	Įrenginys	API	Testų skaičius	Testų trukmė
Robo test	RetrofitSample	LG Nexus 5 5.0.1	21	1	106s
Robo test	RetrofitSample	LG Nexus 5 6.0.1	23	1	174s
AppCrawler	RetrofitSample	LG Nexus 5 5.0.1	21	1	131s
AppCrawler	RetrofitSample	LG Nexus 5 6.0.1	23	1	185s

Patestavus keletą programų buvo pastebėta, kad automatiškai sukurti testai geba vaikščioti po meniu, spausti mygtukus taip išbandyti dalį aplikacijos ir patikrinti ar ji veikia ant tokio įrenginio. Tačiau sudėtingesnės logikos ar veiksmų kurie įtakoja sąveika su išoriniais servisais nepavyko atlikti ir taip negalima testuoti aplikacijos veiksmų ir veikimo. Atlikus tokį testavimą, negalima teigti, kad aplikaciją veikia, nes pastebėta, kad testai sėkmingai įvykdomi visais atvejais kol negaunama sisteminė klaida.

6.3.2. Atkartojimo testavimas

Testuojant vartotojo sąsają testų rašymas užima pakankamai daug laiko, nors tai pakankamai paprastas testavimas paspaudi mygtuką, įvedi tekstą, patvirtini. Todėl galima palengti testavimą pasitelkiant atkartojimo testus. Atkartojimo testų kūrimas yra paprastas procesas įsijungti emuliatorių arba realų įrenginį ir atlieki testą. Lyginant atkartojimo testus buvo pasirinkti tie patys debesijos tiekėjai ir naudojami jų atkartojimo testavimo programos (žr. lentele 7). Testai buvo kuriami tokie patys tik naudojant skirtingus testų atkartojimo įrankius. Testavimui buvo pasirinktas LG Nexus 5 6.0.1 realus įrenginys dėl testavimo serviso limituotų resursų ir jis vienintelis buvo prieinamas visose testavimo servisuose.

Lentelė 7. Debesijos paslaugų tiekėjų testavimo programos

Debesijos	Kūrėjas	Testavimas
Firebase	Google	Espresso recoder – atkartojimo testų programa
Xamarin test cloud	Xamarin	Xamarin test recorder – atkartojimo testų programa
TestDroid	BitBar	Espresso recoder – atkartojimo testų programa

```
[Test]
0 references
public void CoffeTest()
{
    app.Tap(x => x.Id("editText"));
    app.EnterText(x => x.Id("editText"), "coffee");
    app.Tap(x => x.Id("button"));
    app.WaitForElement(x => x.Id("inputValidationSuccess"));
}
```

2 Pav. Xamarin test recoder sukurtas atkartojimo testas

```
@Test
public void coffeeTest() {
    ViewInteraction editText = onView(
            allOf(withId(R.id.editText), isDisplayed()));
    editText.perform(replaceText("Coffee"), closeSoftKeyboard());

    ViewInteraction button = onView(
            allOf(withId(R.id.button), withText("Validate"), isDisplayed()));
    button.perform(click());

    ViewInteraction textView = onView(
            allOf(withId(R.id.inputValidationSuccess), withText("Good choice!"),
                    childAtPosition(
                            childAtPosition(
                                    withId(android.R.id.content),
                                    0),
                            3),
                    isDisplayed()));
    textView.check(matches(withText("Good choice!")));
```

3 Pav. Espresso recorder sukurtas atkartojimo testas

Lentelė 8. MatcherSample programos atkartojimo testų pirmo testavimo rezultatai

Testavimo servisas	Testų kūrimo programa	Aplikacija	Įrenginys	Testų skaičius	Sėkmingi testai	Testų trukmė	Įrenginio testavimo trukmė
FireBase	Espresso	MatcherSample	LG Nexus 5 6.0.1	4	2	14s	185s
Xamarin	Xamarin	MatcherSample	LG Nexus 5 6.0.1	4	4	7s	240s
BitBar	Espresso	MatcherSample	LG Nexus 5 6.0.1	4	2	25s	

Po pirmo leidimo buvo pastebėta, kad Espresso recorder įrašė atkartojimo testus neteisingai (Aplikacijos hierarchijos medyje buvo pasirenkamas neteisingas elementas lyginimui). Kuriant testą dar kelis, kartus pastebėta, kad testai visada sukuriami neteisingi. Todėl testus teko koreguoti ranka, kad jie butų teisingi.

Lentelė 9. MatcherSample programos atkartojimo testų antro testavimo rezultatai

Testavimo servisas	Testų kūrimo programa	Aplikacija	Įrenginys	Testų skaičius	Sėkmingi testai	Testų trukmė	Įrenginio testavimo trukmė
FireBase	Espresso	MatcherSample	LG Nexus 5 6.0.1	4	4	15s	165s
Xamarin	Xamarin	MatcherSample	LG Nexus 5 6.0.1	4	4	6s	238s
BitBar	Espresso	MatcherSample	LG Nexus 5 6.0.1	4	4	32s	

Pataisius klaidas visi testai įvykdyti teisingai. Testuojant aplikacijos, kurios turi daugiau būsenų, statusų. Buvo pastebėta, kad įrašyti atkartojimo testai neveikia teisingai. Teko koreguoti testus rankomis. kad visi testai būtų įvykdyti sėkmingai.

Lentelė 10. MatcherSample programos atkartojimo testų testavimo rezultatai

Testavimo servisas	Testų kūrimo programa	Aplikacija	Įrenginys	Testų skaičius	Sėkmingi testai	Testų trukmė	Įrenginio testavimo trukmė
FireBase	Espresso	MatcherSample	LG Nexus 5 6.0.1	4	1	18s	172s
Xamarin	Xamarin	MatcherSample	LG Nexus 5 6.0.1	4	3	7s	300s
BitBar	Espresso	MatcherSample	LG Nexus 5 6.0.1	4	1	32s	

Taisant atkartojimo testus buvo pastebėtos klaidos Espresso recorder programoje: mobilios aplikacijos vartotojo sąsajoje elementų hierarchijos medis yra neteisingai atvaizduojamas, blogai suskaičiuojami vaikiniai elementai, neteisingai nustatomi elementų būsenos. Xamarin test recorder programoje rašant testus atsirado papildomi žingsniai, pakartojus kelis kartus testus testai buvo įrašyti teisingai, bet pastebėtos klaidos, kad atkartojimo testai neteisingai nustato elementų būsenas.

Lentelė 11. MatcherSample aplikacijos testavimo duomenys

Testavimo servisas	Testų kūrimo programa	Aplikacija	Įrenginys	Testų skaičius	Sėkmingi testai	Testų trukmė	Įrenginio testavimo trukmė
FireBase	Espresso	MatcherSample	LG Nexus 5 6.0.1	4	4	19s	250s
Xamarin	Xamarin	MatcherSample	LG Nexus 5 6.0.1	4	4	9s	400s
BitBar	Espresso	MatcherSample	LG Nexus 5 6.0.1	4	4	34s	

Atliekant atkartojimo testus ant kitų tipų aplikacijų pastebimos tos pačios problemos. Tai pat įrašant atkarojimo testus, aplikacijose, kuriose sąveikauja su foto kamera bendravimas su kamera nebuvo užfiksuojamas. Atkartojimo testai gebėjo patestuoti vartotojo sąsają, bet reikia juos prižiūrėti paredaguoti. Pastebimos problemos kai programos sąveikauja su išoriniu funkcionalumu, kuris nėra pačios programos dalis. Todėl pasiekti testavimo rezultatai netenkina negalima laikyti mobilią aplikaciją sėkmingai ištestuota.

6.3.3. Ranka rašyti testavimo scenarijai

Testavimas naudojant ranka rašytus testavimo scenarijus yra dažniausiai taikomas testuojant vartotojo sąsają. Pasirinkimas technologijos testuojant vartotojo sąsają yra labai platus.

Lentelė 12. Debesijos paslaugų tiekėjų ranka rašytų testų programos

Debesų paslaugų tiekėjai	Kūrėjas	Testavimas android mobiliųjų aplikacijų
Firebase	Google	Espresso, Robotium, UI Automator 2.0
Xamarin test cloud	Xamarin	Xamarin UI test, Calabash
TestDroid	BitBar	Calabash, Espresso, Robotium, UI Automator 2.0

Todėl renkantis technologiją vartotojo sąsajai buvo atsižvelgta į testavimo įrankių palaikymą, testavimo įrankių prieinamumą bandomajam laikotarpiui. Todėl buvo atsisakyta testavimo technologijos Calabash, nes ši technologija nebėra aktyviai tobulinama. Norint palyginti atkartojimo testus su ranką rašytais testavimo scenarijais buvo pasirinka atlikti testavimą naudojant technologijas Espresso ir Xamarin UI test.

Testuojant buvo pasirinktos tos pačios programos naudotos prieš tai buvusiuose testavimuose. Rašant Espresso testus pastebėta, kad jie skiriasi nuo atkartojimo testų nors atlieka tokį patį funkcionalumą. Ranką rašyti testuose galima palaikyti programavimo kodo geras praktikas išlaikyti tvarkingą kodą, bet atkartojimo testai tuo nepasižymi.

```
@Test
public void coffeeTest() {
    ViewInteraction editText = onView(
            allOf(withId(R.id.editText), isDisplayed()));
    editText.perform(replaceText("Coffee"), closeSoftKeyboard());

    ViewInteraction button = onView(
            allOf(withId(R.id.button), withText("Validate"), isDisplayed()));
    button.perform(click());

    ViewInteraction textView = onView(
            allOf(withId(R.id.inputValidationSuccess), withText("Good choice!"),
                    childAtPosition(
                            childAtPosition(
                                    withId(android.R.id.content),
                                    0),
                            3),
                    isDisplayed()));
    textView.check(matches(withText("Good choice!")));

}

private static Matcher<View> childAtPosition(
        final Matcher<View> parentMatcher, final int position) {

    return new TypeSafeMatcher<View>() {
        @Override
        public void describeTo(Description description) {
            description.appendText("Child at position " + position + " in parent ");
            parentMatcher.describeTo(description);
        }

        @Override
        public boolean matchesSafely(View view) {
            ViewParent parent = view.getParent();
            return parent instanceof ViewGroup && parentMatcher.matches(parent)
                    && view.equals(((ViewGroup) parent).getChildAt(position));
        }
    };
}
```

4 Pav. Ranką rašytas testavimos scenarijus (Espresso)

```
@Test
public void validateCoffee(){
    // Type a valid string and click on the button.
    onView(withId(R.id.editText)).perform(typeText("Coffee"), closeSoftKeyboard());
    onView(withId(R.id.button)).perform(click());

    // Check that the correct sign is displayed.
    onView(withId(R.id.inputValidationSuccess)).check(matches(isDisplayed()));
}
```

5 Pav. Atkartojimo testas sukurtas programos (Espresso recoder)

Rašant testus naudojant Xamarint UI programos prototipą pastebėta, kad jie skiriasi minimaliai arba nesiskiria nuo testų sukurtų programos Xamarin test recorder.

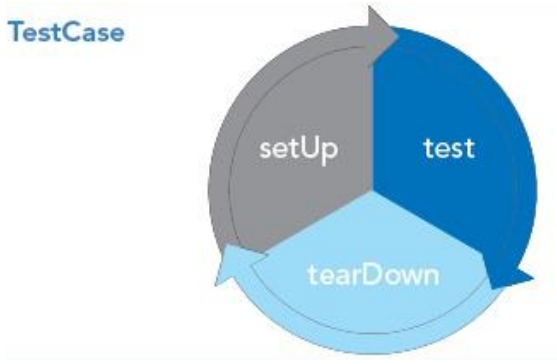
```
[Test]
0 references
public void CoffeTest()
{
    app.Tap(x => x.Id("editText"));
    app.EnterText(x => x.Id("editText"), "coffee");
    app.Tap(x => x.Id("button"));
    app.WaitForElement(x => x.Id("inputValidationSuccess"));
}
```

6 Pav. Atkartojimo testas sukurtas programos (Xamarin test recorder)

```
[Test]
0 references
public void CoffeTestManual()
{
    app.EnterText(x => x.Id("editText"), "coffee");
    app.Tap(x => x.Id("button"));
    app.WaitForElement(x => x.Id("inputValidationSuccess"));
}
```

7 Pav. Ranką rašytas testavimo scenarijus naudojant programą (Xamarin UI test)

Testuojant mobilias aplikacijas naudojant atkartojimo testus ir ranką rašytis testavimo scenarijus nustatyta, kad testavimo laikas skirtumai yra minimalūs. Pagrindiniai trūkumai tarp sukurtų atkartojimo testų ir ranką rašytų testavimo scenarijų, kad išlaikyti programavimo praktikas yra labai sunku naudojant atkartojimo testus. Testai tada jau kuriami pagal programą ir tai daroma pakankamai paprastai tai kokie buvo padaryti žingsniai todėl teksto eilutėms nekuriami atskiri kintamieji, nenaudojamos tinkamos testavimo scenarijų struktūros nekuriami pasirengimo metodai (angl. setup) ir sugriovimo metodai (angl. teardown) tai sukelia bėdų kai kartojamas tas pats funkcionalumas prisijungimas prie aplikacijos prieš kiekvieną testą ir taip testuose atsiranda identiškų kodo eilučių.



8 Pav. Testo struktūra

6.3.4. Testų leidimas ir jų peržiūra cloud services

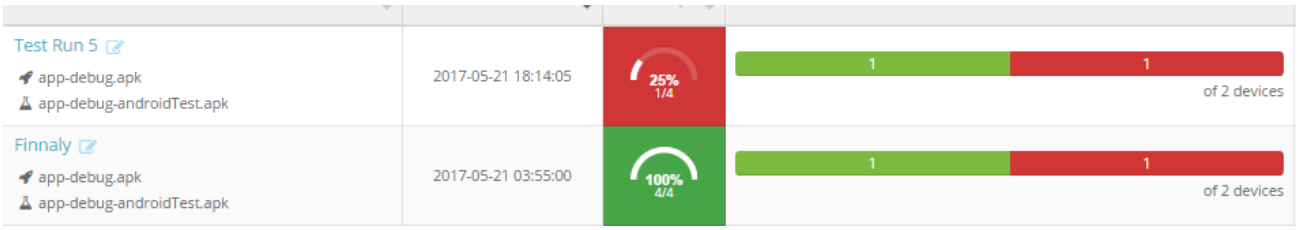
Testavimo scenarijai cloud servises leidžiami arba naudojant komandinę eilutę arba panašius servisus arba galima tiesiog įkelti mobilią aplikaciją ir jos testavimo aplikaciją. Programos įkėlimo procesas daugiausia įmonių netenkina, nes negalima automatizacija išleidžiant kitą programos versiją. Iš pasirinktų testavimui servisų visi jie turėjo galimybes testavimo automatizavimui, bet Firebase ir TestDroid bandomajai versijai tokios galimybės nebuvo suteikiamos todėl šis funkcionalumas nebuvo išbandytas.

Xamarin test cloud serviso integracija buvo paprasta. Pateikiamas servisas, kuris automatiškai eksportuoja visus testus ir norint pasirinkti įrenginių sąrašą sugeneruojamos tam tikras simbolių rinkinys kuris atitinka tą įrenginių paketą.

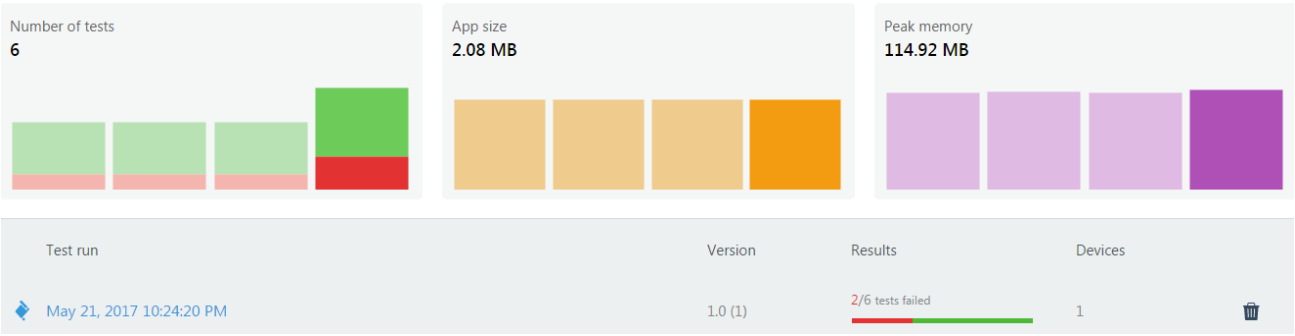
```
submit app-debug.apk 57094a4cf81319ede465f805d89ba857 --assembly-dir ./UItest2/bin/Debug
as.stasytis@mif.stud.vu.lt
Negotiating file upload to Xamarin Test Cloud.
Posting to https://testcloud.xamarin.com/ci/upload2

Uploading nunit.framework.dll ... Already uploaded.
Uploading Xamarin.UITest.dll ... Already uploaded.
Uploading app-debug_resigned.apk... 100%
Uploading UITest2.dll... 100%
Uploading AndroidTestServer.apk... 100%

Upload complete. Upload Id: 52d9e15b-78a8-44f6-a1b0-7e40f5c7936f
```

9 pav. Xamarin test cloud automatinis testų perdavimas testavimo servisui naudojant realius įrenginius

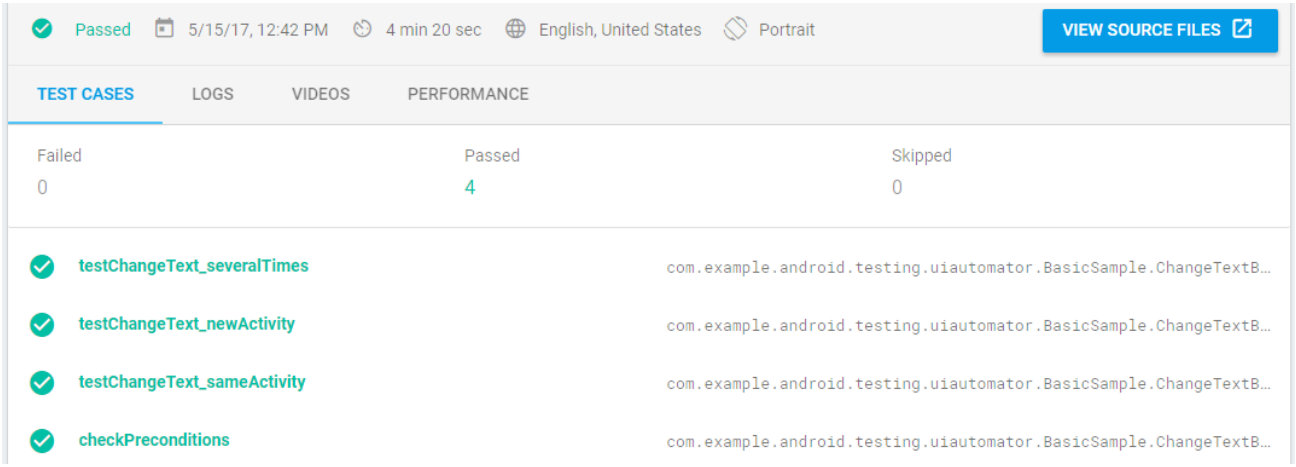
Testų rezultatus šituose servisuose galima peržiūrėti testų rezultatų faile, kuris gaunamas išsiuntus testavimo užklausas arba galima testavimo rezultatus peržiūrėti internete. Kartu su testų rezultatais ir jų trukmę šie servisai tai pat pateikia informacija apie aplikacijos veikimą: našumą, dydį, atminties naudojimą ir įrenginio resursų naudojimą įvairiais aplikacijos momentais.



10 Pav. Testdroid serviso mobiliųjų aplikacijų testavimo statistika



11 pav. Xamarin test cloud serviso aplikacijos testavimo statistika



12 pav. Firebase testavimo statistika

6.3.5. Debesijos servisų mobilių aplikacijų testavimo rekomendacijos

Išbandžius visus debesijos servisus aptikta, kad atkartojimo testai ir automatiškai sukurti testai yra labai lengva realizuoti mobilioje aplikacijoje, bet jie gali ištestuoti dalį aplikacijos funkcionalumo. Norint ištestuoti aplikaciją reikia koreguoti testus ir rašyti juos ranka. Atsikartojimo testų įrankiuose pastebėtas privalumas Xamarin test recorder, kad testai rašomi gan paprastai ir nesukuriama dideli mastai išeities kodo, todėl šito įrankio testų kokybę palaikyti yra nesudėtinga. Debesijos servisų testų įvykdymo trukmė skirtumai yra minimalūs.

Bandomasis laikotarpis lyginant bandomuosius laikotarpius išsiskiria Firebase ir Xamarin test cloud debesijos. Firebase bandomasis laikotarpis neturi pabaigos, bet testavimas yra apribotas, kiek galima įvykdyti testų kiekvieną dieną. Xamarin test cloud banduomoju laikotarpiu suteikiamos beveiks visos funkcijos, todėl galima viską naudoti ir per 30 dienų galima nuspręsti ar šitos paslaugos atitinka reikalavimus. TestDroid nieko neišsiskiria iš šitų lyginamų servisų nei įrenginių skaičiu nei bandomojo laikotarpiu.

Lentelė 13. Bandomasis laikotarpis

Debesija	Dienų skaičius	Įrenginių skaičius	Testų skaičius
Firebase	Neribojama	5	5 testų paketai
TestDroid	30 dienų	2	Neribojam
Xamarin test cloud	30 dienų	1000	Neribojama

Testų rezultatus peržiūrint matomi visiškai skirtingos vartotojo sąsajos todėl jas yra gan sunku lyginti. Lentelėje išskirstytos pagrindinės vietos ir ypatumai servisų. Daugelis servisų panašiai pateikia duomenis tik pastebėti keli išsiskyrimai. TestDroid vartotojo sąsajoje peržiūrint testavimo rezultatus testų metu nuotraukos ne visada būdavo išsaugomos ir joje pastebėtos našumo bėdos. Firebase servisas turėjo įdomų priedą veikimo diagramas (žr. priedus 13 pav.) , kurios parodo mobilios aplikacijos kelius kaip pasiekiami kiti vartotojo sąsajos langai. Xamarin test cloud servisas turėjo patogiausią vartotojo sąsają: visos diagramos pateikiamos viename ekrane, lengva keisti testavimo projektus ir pateikiama mobilios programėlės atminties analizė su vis tolimesnėmis versijomis.

Lentelė 14. Testų rezultatų peržiūra

Debesija	Registras	Klaidos	Nuotraukos	Vaizdo įrašas	Našumas	Veikimo medis	Atminties analizė
Firebase	Taip	Taip	Taip	Taip	Taip	Taip	Ne
TestDroid	Taip	Taip	Taip / Ne	Taip	Taip	Ne	Ne
Xamarin test cloud	Taip	Taip	Taip	Taip	Taip	Ne	Taip

Testavimas mobiliųjų aplikacijų naudojant šias debesijas reikalavimo įvairių įrankių įdiegimo ir testų nustatymo keitimo norint jas paleisti atitinkamose testavimo karkasuose.

- Firebase testavimui buvo pasirinkta Android Studio ir reikalingi Espresso testavimo programos įskiepiai. Testų paleidimo procesas vykdomas paprastai sugeneruojant atskiras programėles ir jas įkeliant į Firebase servisą.
- TestDroid testavimui buvo pasirinkta Android Studio ir reikalingi Espresso testavimo programos įskiepiai. Testų paleidimo procesui reikalingi testų nustatymų keitimas, kad veiktų šiame servise.

- Xamarin test cloud testavimui buvo pasirinkta Visual Studio su Xamarin įskiepiu. Atliekant programos diegimą įvykdavo klaidos todėl teko daryti rankinį diegimą. Norint sukurti testavimo projektus reikia parsisiųsti reikalingus plėtinius ir projektų maketus. Testų paleidimas vyko per komandinę eilutę. Praėjus visą šitą procesą testų leidimą galima automatizuoti ir supaprastinti.
-

Lentelė 15. Testų kūrimo kalbos, įrankių įdiegimas sudėtingumai

Debesija	Testų rašymo kalbos	Įrankių įdiegimas ir testų paleidimas	Vartotojo sąsajos testai	Atkartojimo testai	Automatiškai sukurti testai
Firebase	Java	Labai paprastas	Taip	Taip	Taip
TestDroid	Java, Ruby	Paprastas	Taip	Taip	Taip
Xamarin test cloud	C#, Ruby	Sudėtingas	Taip	Taip	Ne

Išbandžius pasiriktas debesijas pateiktas jų vertinimas. Debesijos vertintos pagal gautus rezultatus naudojantis bandomąsias šių debesijų servisų paslaugas.

Lentelė 16. Asmeninis debesijos servisų vertinimas dešimties balų skale
(10 -puiku, 1 – labai prastai)

Debesija	Bandomasis laikotarpis	Testų rezultatų peržiūra	Įrankių diegimas ir testų paleidimas	Galutinis Balas (suapvalinta)
Firebase	8	7	9	8
TestDroid	5	6	8	6
Xamarin test cloud	9	8	6	8

Rezultatai ir išvados

Šio bakalaurinio darbo metu buvo apžvelgti mobiliųjų aplikacijų testavimo tipai. Pateikti jų apibrėžimai. Apibrėžti mobiliųjų aplikacijų tipai. Iškeltos pagrindinės mobiliųjų aplikacijų testavimo problemos ir iššūkiai. Išsamiai išanalizuoto pagrindiniai testavimo iššūkiai: mobiliųjų telefonų įvairovė ir operacinių sistemų įvairovė. Palyginti mobiliųjų aplikacijų testavimo įrankiai ir testavimas naudojant emuliatorius ir realius įrenginius. Išnagrinėtas automatinis testavimas: apibrėžti automatinių testų tipai jų privalumai ir trūkumai. Pateikti automatiniai testavimo įrankiai.

Autorius norėdamas išanalizuoti DEVOPS mobiliųjų programų kūrimo procesą atliko testavimą laikydamasis DEVOPS standartų apibrėžtų knygoje „Mobile DevOps Manifesto“. Atliktas testavimas naudojant tris skirtingas debesijas. Testai buvo atliekami vartotojo sąsajos naudojant realius įrenginius. Atlikti testai buvo trijų tipų: automatiškai sukurti, atkartojimo testai ir ranką rašyti. Testavimas atliktas debesijose naudojant realius Android įrenginius ten tai pat pateikiami gauti testavimo rezultatai. Atlikus testavimą autorius įvertino pasirinktos debesijos pagal savo testavimo patirtį ir aprašė pagrindines problemas.

Rezultatai

Darbe buvo pasiekti šie rezultatai:

1. Suformuoti ir išanalizuoti pagrindiniai mobiliųjų aplikacijų testavimo tipai.
2. Iškeltos mobiliųjų aplikacijų testavimo problemos iššūkiai ir galimi jų sprendimo būdai
3. Ištestuotos aplikacijos naudojant DEVOPS programų kūrimo proceso modelį ir skirtingus debesijos servisus.

Išvados

Testavimas yra būtiną aplikacijos pasisekimo dalis, kai aplikacijų yra labai plati ir konkurencinga rinka. Bet netgi blogai suplanuotas testavimas užima apie 20% ~ 50% aplikacijos kūrimo laiko, dėl to tai bus daugiausiai išlaidų reikalaujanti sritis Mobiliųjų aplikacijų testavimas yra plati sritis, kuri susideda iš daugybės dalykų (rankinių, automatinių testų, įvairių mobilių aplikacijų testavimo tipų ir strategijų). Mobilios aplikacijų tobulėjimas ir įrenginių skirtumai suteikė daug privalumų ir iššūkių norint ištestuoti mobiliąsias aplikacijas. Pagrindinės išryškėjusios problemos tai testavimas ant daugelio platformų, kuris rankiniu būdų truktų labai ilgai ir kainuotų daug.

Mobiliųjų aplikacijų testavimas juda prie vis didesnio automatizavimo, bet vis keičiantis mobiliesiems įrenginiams atsiranda naujų, nenumatytų atvejų ir keičiantis mobiliosioms aplikacijoms testai turi būti labai lankstūs ir vis pritaikomi. Automatizavimo įrankiai palengviną darbą. Todėl iš anksto reikia planuoti atsakingai testų automatizavimą. Daugelis aspektų aplikacijos gali būti automatizuoti ir tuo reiktų naudotis nuo pat pradžių norint pilnai ištestuoti aplikaciją ir padidinti jos kokybę. Aplikacija keisis programavimo metu ir į tai reikia atsižvelgti, nes testavimas rankiniu būdu nebus pajėgus ištestuoti viso funkcionalumo iš naujo, todėl niekaip neapsieisite be automatinių testų.

Naujesni programų kūrimo procesai juda link pilno automatizavimo ir testavimui naudojimo tik realių įrenginių. Tai seniau atlikti buvo gana sudėtinga, bet dabartinės debesijos ir įrenginių fermos (angl. device farm) suteikia galimybes dabartiniams programų kūrimo modeliams kaip DEVOPS judėti į priekį ir įgyvendinti testavimą naudojant realius įrenginius ir automatizuojant viską. Dabar galima ištestuoti mobilias aplikacijas neeikvojant didelių resursų.

Galima daryti išvadą, kad kokybiškos mobilios aplikacijos išleidimas yra labai sunkus darbas. Dabar mobilių aplikacijų testavimas juda teisinga linkme vis daugiau atsiranda testavimo įrankių, testavimo strategijų, vis daugiau automatizavimo mažiau rankinio darbo. Visa mobiliosios aplikacijos sėkmė priklauso nuo to kiek kūrėjai laiko investuos į mobiliosios aplikacijos testavimą ir kiek jie laiko įdės rinkdamiesi savo testavimo įrankius ir strategiją.

Šaltiniai

[ADMA17] Android kūrėjų dokumentacija (angl. Android developers manifest) https://developer.android.com/about/dashboards/index.html

[AVSH15] Aviram Shotten *Application level performance testing and more* https://www.infoq.com/articles/mobile-app-performance-testing

[BIPA14] Bijit Pattanaik *Memory leak testing. Why it is important? How it is done?* https://intensetesting.wordpress.com/2014/03/28/memory-leak-testing-why-it-is-important-how-is-it-done/

[DAKN12] Daniel Knott *How to choose the right mobile test devices* https://devblog.xing.com/qa/how-to-choose-the-right-mobile-test-devices/

[DARA17] David Ramel *7 TOP Device Clouds for mobile app testing* https://adtmag.com/blogs/dev-watch/2017/05/device-clouds.aspx

[ERMU16] Ernest Mueller *What is devops* https://theagileadmin.com/what-is-devops/

[MAHE15] Matthew Heusser and Justin Rohrman *How to test mobile responsive design applications* http://searchsoftwarequality.techtarget.com/tip/How-to-test-mobile-responsive-design-applications

[MDOM] BitBar *Mobile DevOps Manifesto*

[SOTH16] SoftwareTestingHelp *Beginner's guide to mobile application testing* http://www.softwaretestinghelp.com/beginners-guide-to-mobile-application-testing/

[SWSE14] Swati Seela *Interrupt testing* http://www.guru99.com/interrupt-testing.html

[RASA17] Rahis Saifi *The 2017 Mobile App Market: Statistics, Trends, and Analysis* http://www.business2community.com/mobile-apps/2017-mobile-app-market-statistics-trends-analysis-01750346#2ejvLV1uZgJiMkQ1.97

[THPE15] Thomas Peham *How to approach mobile web testing?* http://usersnap.com/blog/mobile-first-how-to-approach-mobile-website-testing/

[VVHE15] Ville-Veikko Helppi *The basics of test automation for apps, games and the mobile web* https://www.smashingmagazine.com/2015/01/basic-test-automation-for-apps-games-and-mobile-web/

[VVHE16] Ville-Veikko Helppi *Best practices for DevOps in Mobile App Testing* https://www.slideshare.net/bitbar/best-practices-for-devops-in-mobile-app-testing

[YVFA15] Yvette Francino *Master interrupt testing on mobile devices* http://searchsoftwarequality.techtarget.com/tip/Master-interrupt-testing-on-mobile-devices

[XBOS] Xbosoft *Mobile installion testing* https://xbosoft.com/software-quality-blog/mobile-installation-testing-step-1/

Santrumpos ir sąvokų apibrėžimas

API - Aplikacijų programavimo sąsaja (angl. Application Programming Interface)

UI – vartotojo sąsają (angl. user interface)

DEVOPS – programų kūrimo modelis kurio pavadimas kilo iš programų sistemų ir informacinių technologinių operacijų (angl. software **DEV**elopment and information technology **OP**eration**S**)

Android – atviro kodo operacinė sistema naudojama išmaniusiuose įrenginiuose (telefonuose, planšetėse, laikrodžiuose ir t.t.).

IOS – operacinė sistema naudojama išmaniesiems „Apple“ įrenginiams.

Windows phone – operacinė sistema naudojama išmaniesiems įrenginiams.

Priedai

Lentelė 17. Informacija surinkta per 7 dienų laikotarpį, kuris baigėsi 2017-05-02
Versijos kurios pasiskirstymas mažesnis nei 0.1% nerodomos [ADMA17]

Versija	Pavadinimas	API	Pasiskirstymas
2.3.3 – 2.3.7	Gingerbread	10	1.0%
4.0.3 – 4.0.4	Ice Cream Sandwich	15	0.8%
4.1.x	Jelly Bean	16	3.2%
4.2.x		17	4.6%
4.3		18	1.3%
4.4	Kitkat	19	18.8%
5.0	Lolipop	21	8.7%
5.1		22	23.3%
6.0	Marshmallow	23	31.2%
7.0	Nougat	24	6.6%
7.1		25	0.5%

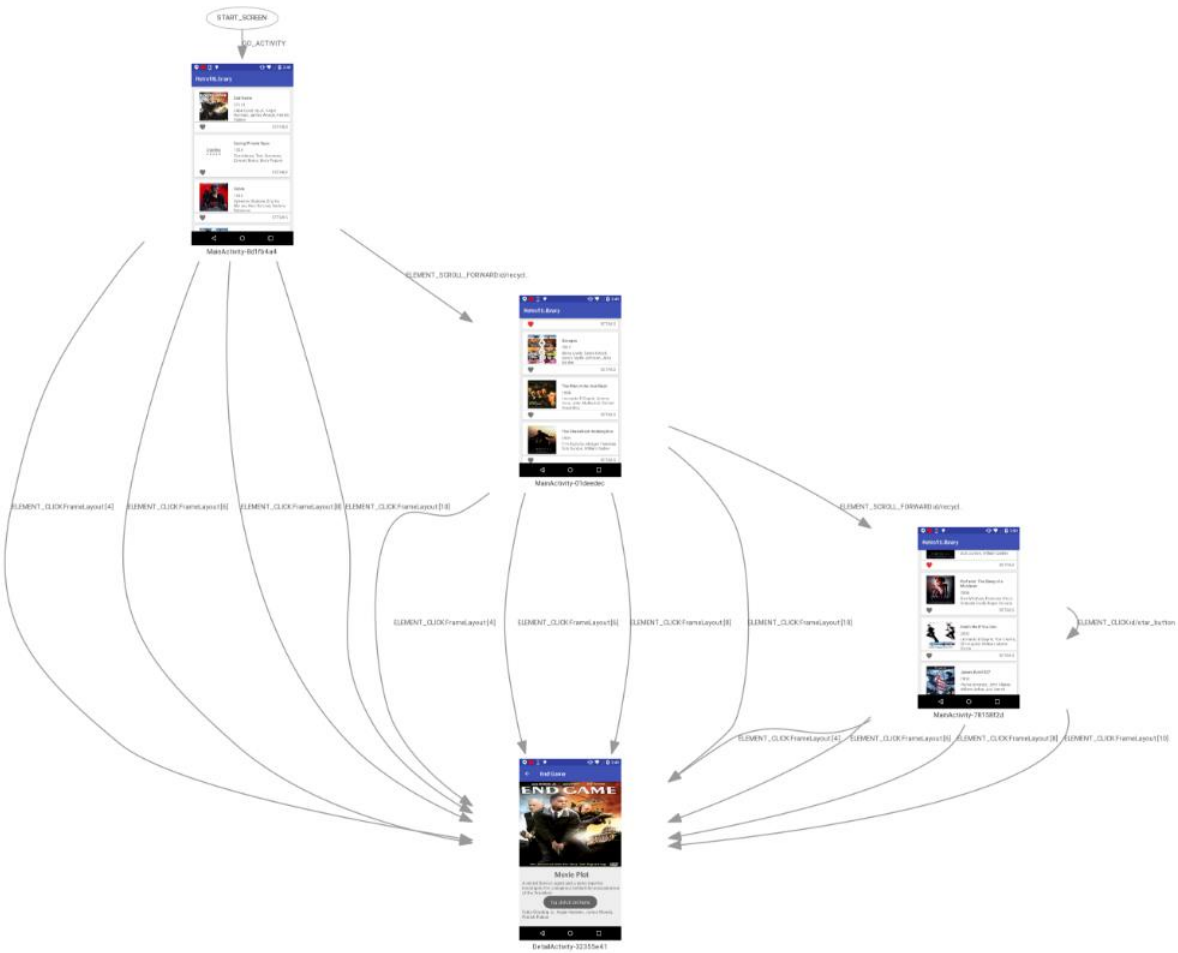
Lentelė 18. RetrofitSample programos testavimo rezultatai
naudojantis AppCrawler testavimo įrankiu

Identifikacijos numeris	Aplikacija	Testų trukmė	Rezultatai
Crawl5.01	RetrofitSample	132s	Nėra klaidų
Crawl5.02	RetrofitSample	141s	Nėra klaidų
Crawl5.03	RetrofitSample	128s	Nėra klaidų
Crawl5.05	RetrofitSample	130s	Nėra klaidų
Crawl5.06	RetrofitSample	129s	Nėra klaidų
Crawl5.07	RetrofitSample	131s	Nėra klaidų
Crawl5.08	RetrofitSample	121s	Nėra klaidų
Crawl5.09	RetrofitSample	132s	Nėra klaidų
Crawl5.10	RetrofitSample	133s	Nėra klaidų
Crawl6.01	RetrofitSample	187s	Nėra klaidų
Crawl6.02	RetrofitSample	186s	Nėra klaidų
Crawl6.03	RetrofitSample	185s	Nėra klaidų
Crawl6.04	RetrofitSample	187s	Nėra klaidų
Crawl6.05	RetrofitSample	214s	Nėra klaidų
Crawl6.06	RetrofitSample	185s	Nėra klaidų
Crawl6.07	RetrofitSample	186s	Nėra klaidų

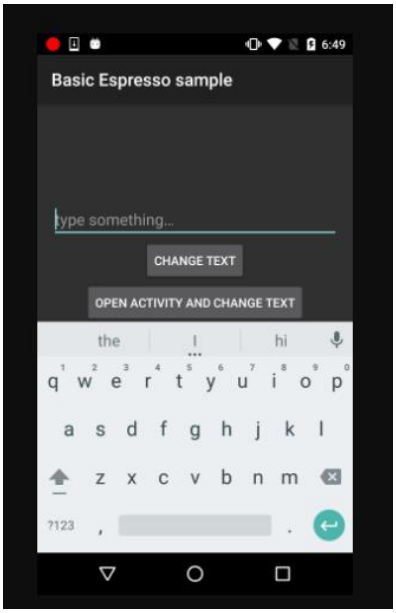
Crawl6.08	RetrofitSample	215s	Nėra klaidų
Crawl6.09	RetrofitSample	187s	Nėra klaidų
Crawl6.10	RetrofitSample	185s	Nėra klaidų

Lentelė 19. RetrofitSample programos testavimo rezultatai naudojantis Robo test testavimo programa

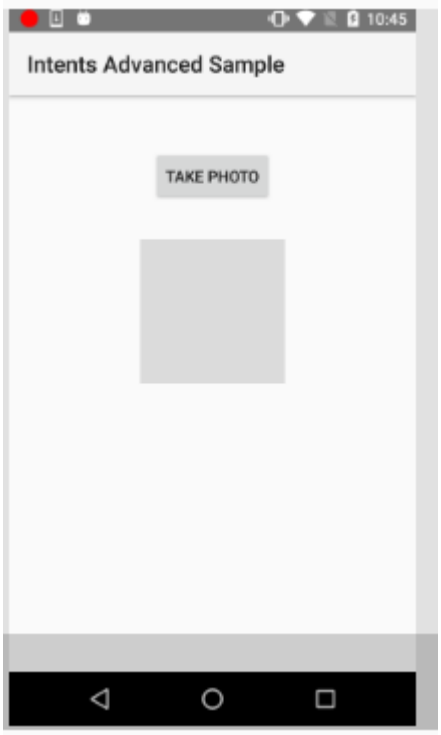
Robo5.01	RetrofitSample	101s	Nėra klaidų
Robo5.02	RetrofitSample	119s	Nėra klaidų
Robo5.03	RetrofitSample	174s	Nėra klaidų
Robo5.05	RetrofitSample	103s	Nėra klaidų
Robo5.06	RetrofitSample	146s	Nėra klaidų
Robo5.07	RetrofitSample	126s	Nėra klaidų
Robo5.08	RetrofitSample	107s	Nėra klaidų
Robo5.09	RetrofitSample	114s	Nėra klaidų
Robo5.10	RetrofitSample	116s	Nėra klaidų
Robo6.01	RetrofitSample	162s	Nėra klaidų
Robo6.02	RetrofitSample	166s	Nėra klaidų
Robo6.03	RetrofitSample	165s	Nėra klaidų
Robo6.04	RetrofitSample	214s	Nėra klaidų
Robo6.05	RetrofitSample	150s	Nėra klaidų
Robo6.06	RetrofitSample	160s	Nėra klaidų
Robo6.07	RetrofitSample	198s	Nėra klaidų
Robo6.08	RetrofitSample	169s	Nėra klaidų
Robo6.09	RetrofitSample	159s	Nėra klaidų
Robo6.10	RetrofitSample	165s	Nėra klaidų



Pav 13. RetrofitSample aplikacijos firebase sugeneruotas veiksmų medis



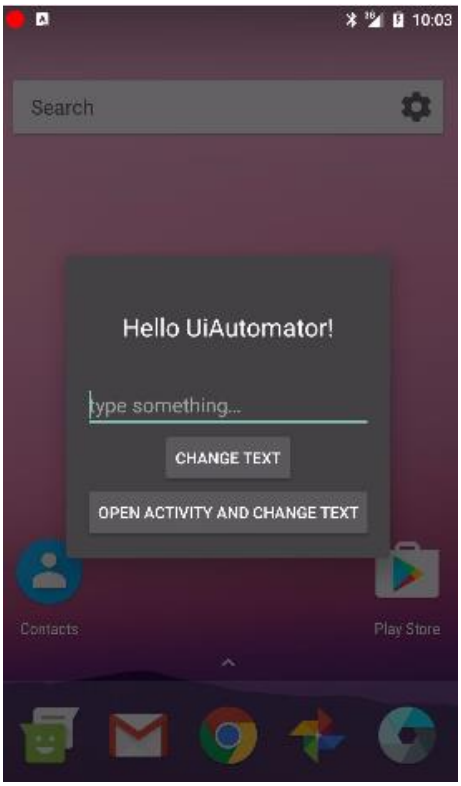
Pav 14. BasicSample aplikacijos pagrindinis vartojo sąsajos pagrindinis meniu



Pav. 15 AdvancedSample aplikacijos pagrindinis vartojo sąsajos pagrindinis meniu

Pav. 16 CustomMatcher aplikacijos pagrindinis vartojo sąsajos pagrindinis meniu

Pav. 17 MultiSample aplikacijos pagrindinis vartojo sąsajos pagrindinis meniu



Pav. 18 AutomatorSample aplikacijos pagrindinis vartojo sąsajos pagrindinis meniu