TIETO- JA SÄHKÖTEKNIIKAN TIEDEKUNTA

Marja Pellikka

PANKIN MOBIILISOVELLUKSEN GRAAFISEN KÄYTTÖLIITTYMÄN TESTAAMINEN

Diplomityö
Tietotekniikan tutkinto-ohjelma
Syyskuu 2020

Pellikka M. (2020) Pankin mobiilisovelluksen graafisen käyttöliittymän testaaminen. Oulun yliopisto, tietotekniikan tutkinto-ohjelma. Diplomityö, 74 s.

TIIVISTELMÄ

Digitalisaation myötä merkittävä osa pankkipalveluista on siirtynyt mobiilisovelluksiin. Sovellusten ominaisuudet monipuolistuvat ja niiden käyttäjämäärät kasvavat koko ajan.

Mobiilisovellusten kehittämisen ja testaamisen eräs ongelma on laitesukupolvien nopea kehitys. Eri käyttöjärjestelmät, ohjelmistoversiot, laitevalmistajat, laitteiden mallit, näyttökoot, näytön kuvasuhteet ja resoluutiot tuottavat lukemattoman määrän erilaisia laiteprofiileja, joiden kanssa sovelluksen tulisi toimia virheettömästi. Testaus on toistettava jokaisen uuden sovellus-, laite- ja käyttöjärjestelmäversion kohdalla. Laajasta laitekirjosta johtuen fyysisistä mobiililaitteista koostuvan testilaboratorion ylläpitäminen ei välttämättä ole realistista.

Tässä diplomityössä on tarkasteltu pankkipalveluiden mobiilisovellusten testausta vertaamalla laitepohjaista ratkaisua, simulaattoreita ja emulaattoreita, virtuaalikoneita sekä pilvipalveluissa tapahtuvaa testausta. Pankkisovelluksen tapauksessa tietoturvalla on keskeinen merkitys, sillä kyseessä on asiakkaiden henkilötietojen sekä raha-asioiden käsittely.

Tarkkojen ja yhtenäisten kriteereiden määrittely pankin mobiilisovelluksen graafisen käyttöliittymän testauksen automatisointiin on haasteellista. Testausratkaisun valinta riippuu kontekstista, tiimin asettamista vaatimuksista ja työyhteisön kulttuurista. Tässä tutkielmassa tehdyt analyysit antavat pohjan valinnalle.

Avainsanat: automaatiotestaus, testaustyökalukehys, natiivisovellus, pilvipohjainen monialustatestausympäristö, responsiivisuuden testaaminen, Android, iOS

Pellikka M. (2020) Testing the graphical user interface of the mobile banking application. University of Oulu, Degree Programme in Computer Science and Engineering. Master's Thesis, 74 p.

ABSTRACT

With digitalization, a significant portion of banking services have shifted to mobile applications. The features of the applications are diversifying, and the number of their users is constantly growing.

One problem with the development and testing of mobile applications is the rapid development of device generations. Different operating systems, software versions, device manufacturers, device models, screen sizes, screen aspect ratios, and resolutions produce a myriad of different device profiles with which the application should work flawlessly. Testing must be repeated for each new version of application, device, and operating system. Due to the large variety of hardware, maintaining a test lab made up of physical mobile devices may not be realistic.

This thesis examines the testing of mobile applications in banking services by comparing a device-based solution, simulators and emulators, virtual machines and testing in cloud services. In the case of a banking application, data security is of key importance, as it involves the processing of customers' personal data as well as financial matters.

Defining exact and unified criteria for automating the graphical user interface testing of a mobile banking application is challenging. The selection of the testing solution depends on the context, the requirements set by the team, and the culture of the work community. The analyses in this thesis provide a foundation for such a choice.

Key words: automation testing, testing framework, native application, cloud-based multi-platform testing environment, responsiveness testing, Android, iOS

SISÄLLYSLUETTELO

ALKULAUSE

Tämä diplomityö on tehty OP-Palvelut Oy:lle. Haluan kiittää OP-Palvelut Oy:tä työni aiheesta ja FT Antti Aueria asiantuntevasta ohjauksesta. Suuret kiitokset professori Olli Silvénille erinomaisesta ohjauksesta, kärsivällisyydestä ja kannustuksesta. Kiitokset professori Juha Röningille työni tarkastamisesta.

Haluan kiittää OP-mobiilin testaajia ja muita kollegoita vastauksista työtäni koskevissa kysymyksissä ja ongelmissa. Lopuksi haluan osoittaa kiitokseni perheelleni, ystävilleni ja erityisesti puolisolleni kaikesta saamastani tuesta ja rohkaisusta sekä opintojen suorittamisen että diplomityön kirjoittamisen aikana.

Oulu, elokuu 2020

Marja Pellikka

LYHENTEIDEN SELITYKSET

iOS	iPhone Operating System
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
PWA	Progressive Web Apps
HTTPS	HyperText Transfer Protocol Secure
HTTP	HyperText Transfer Protocol
UI	User Interface
GUI	Graphical User Interface
API	Application Programming Interface
OSI	Open Systems Interconnection
SDK	Software Development Kit
MVC	Model-View-Controller
JAR	Java ARchive
APK	Android Application Package
RAM	Random Access Memory
BDD	Behavior Driven Development
SUT	System Under Test
REST	Representational State Transfer
VNC	Virtual Network Computing
DDT	Data-Driven Testing
AUT	Application Under Test
GPS	Global Positioning System
SSL	Secure Sockets Layer
VPN	Virtual Private Network
DOM	Document Object Model
TLS	Transport Layer Security
LDAP	Lightweight Directory Access Protocol

1. JOHDANTO

Mobiililaitteiden määrä on kasvanut viime vuosina merkittävästi, ja todennäköisesti laitemäärä jatkaa kasvuaan myös tulevaisuudessa[1]. Vuoden 2018 kolmannen kvartaalin lopussa maailmanlaajuisesti käytössä oli 7.9 miljardia mobiililiittymää, ja uusia liittymiä avattiin samalla kvartaalilla 120 miljoonaa kappaletta[2].

Mobiililaitteiden yleistymisen myötä myös niille suunnattujen finanssisovellusten määrä on voimakkaassa kasvussa. Asiakkaat kaikissa ikäryhmissä käyttävät mobiilipalveluita yhä useammin ja monipuolisemmin raha-asioiden hoitamiseen. Viimeisen viiden vuoden aikana finanssisovellusten käyttö on kasvanut 354%, ja ne ovat ohittamassa perinteistä verkkopankkia ensisijaisena asiointikanavana. Kilpailu asiakkaista kovenee, sillä ala houkuttelee koko ajan uudenaikaisia ja kiinnostavia palveluita tarjoavia uusia toimijoita.[3]

Mobiililaitteille on useita käyttöjärjestelmiä, joista suosituimmat ovat tällä hetkellä Applen iOS ja Googlen Android[4]. Käyttöjärjestelmistä on kuluttajien käytössä useita eri ohjelmistoversioita, ja lisäksi markkinoilla on esimerkiksi Androidille monta eri laitevalmistajaa. Laitteiden malli, koko, näytön resoluutio ja kuvasuhde vaihtelee, ja tämän takia käytössä olevien mobiililaitteiden kirjo on valtava. Uusia malleja ja ohjelmistoversioita tulee myös markkinoille koko ajan lisää. Ilmiötä kutsutaan fragmentaatioksi, ja vuonna 2019 mobiililaitteille löytyi 63 000 erilaista laiteprofiilia, vuosittaisen kasvuvauhdin ollessa lähes 20%.[5]

Yhä useammalla yrityksellä on nykyään www-sivujen lisäksi mobiilisovellus. Älypuhelinten määrän lisääntymisen myötä kuluttajille tarjolla olevien sovellusten luku kasvaa koko ajan, esimerkiksi kahdessa johtavassa sovelluskaupassa Google Playssa ja Applen App Storessa oli vuonna 2017 yhteensä viisi miljoonaa sovellusta[6]. Finanssisovellukset ovat vuonna 2019 kummassakin sovelluskaupassa viidentoista suosituimman kategorian joukossa jättäen taakseen mm. uutiset, urheilun, lääketieteen, kartat sekä sosiaalisen median ja muut verkkoyhteisöpalvelut[7] [8].

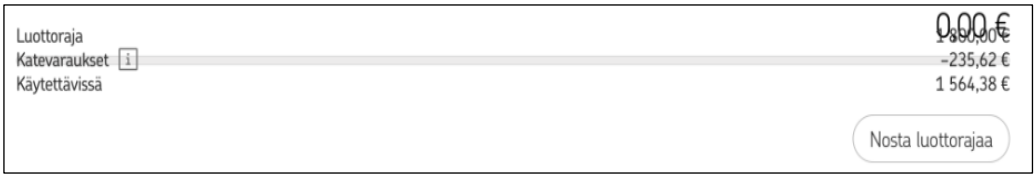
Sovellukset, kuten esimerkiksi pankkisovellus, on suunniteltava responsiiviseksi, eli sen sisällön ja käyttöliittymän täytyy mukautua näyttölaitteille, joiden koko ja resoluutio vaihtelee. Finanssisovellukset ovat käyttäjälle kriittisiä, sillä niiden kohdalla kyse on asiakkaan omasta taloudesta ja sen suunnittelusta. Lisäksi niillä pystytään hoitamaan helposti jokapäiväisiä raha-asioita, mikä helpottaa arjen sujumista. Tietoturva ja käytön vaivattomuus ovat siten ensiluokkaisen tärkeitä ominaisuuksia näille sovelluksille.

Yritykset käyttävät runsaasti aikaa ja resursseja mobiilisovellusten suunnitteluun ja kehitykseen, jotta ne näyttäisivät samalta ja toimisivat samalla tavalla kaikissa mobiililaitteissa. Mikäli ulkoasu tai toiminta on epäjohdonmukaista, se vaikuttaa negatiivisesti kuluttajien mielipiteeseen sovelluksen laadukkuudesta. Kuluttajien odotukset ovat korkealla, ja käyttäjä poistaa sovelluksen nopeasti, mikäli se on huonosti suunniteltu, hankalasti käytettävä, latautuu tai toimii hitaasti, tai siinä on

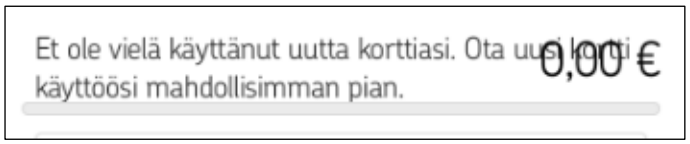
[1] https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/
[2] https://www.ericsson.com/en/mobility-report/reports/november-2018/mobile-subscriptions-worldwide-q3-2018
[3] https://liftoff.io/resources/report/mobile-finance-apps-2019/
[4] https://gs.statcounter.com/os-market-share/mobile/worldwide
[5] https://www.scientiamobile.com/device-fragmentation-growing-20-per-year/
[6] https://www.statista.com/chart/12455/number-of-apps-available-in-leading-app-stores/
[7] https://www.statista.com/statistics/270291/popular-categories-in-the-app-store/
[8] https://www.statista.com/statistics/279286/google-play-android-app-categories/

teknisiä ongelmia [1, p. 3]. Asiakkaiden käytössä olevat eri selainversiot, käyttöjärjestelmät ja eri kokoiset laitteet saattavat johtaa kriittisten seikkojen virheelliseen tai harhaanjohtavaan esittämiseen. Esimerkiksi kuittauspainike tai jokin muu käyttöliittymän tärkeä elementti voi jäädä piiloon virtuaalisen näppäimistön tai ylä- tai sivupalkin alle.

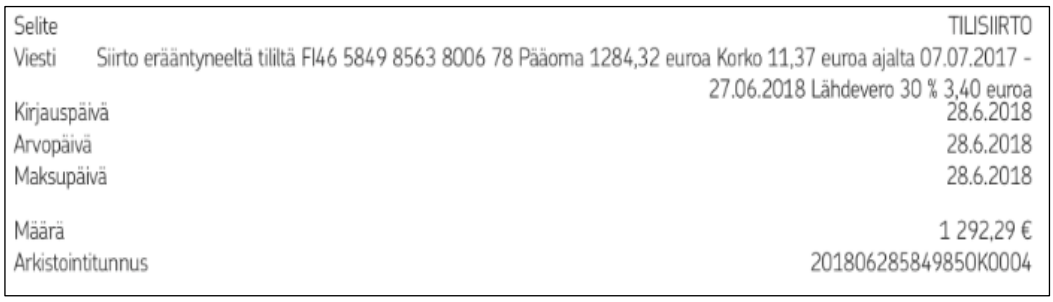
Kuvissa 1, 2, 3, 4, 5 ja 6 on esitetty joitakin esimerkkejä pankkisovelluksen testauksessa löydetyistä graafisen käyttöliittymän responsiivisuuden ongelmista.



Kuva 1. Luottokortin saldonäkymän rikkinäinen sivupohja.

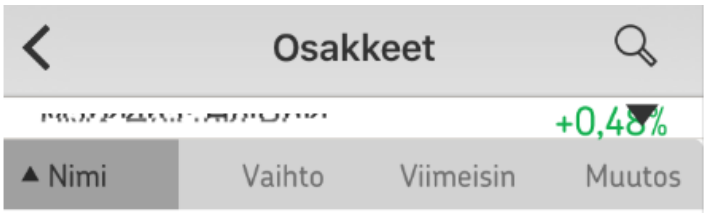


Kuva 2. Käyttämättömän luottokortin rikkinäinen sivupohja.



Kuva 3. Skaalautumisvirhe tilitapahtuman vaakanäkymässä.

Kuva 4. Otsikko leikkautuu vahvistusnäkymässä.



Kuva 5. Virhe osakelistaus-näkymässä.

Kuva 6. Virhe hälytysten asetusnäkymässä.

Pankkipalveluiden kohdalla on kaikkia sovelluksia koskevien ongelmien lisäksi myös muita erityispiirteitä. Pankkitoiminta on säännelty tarkasti lainsäädännöillä ja regulaatioilla, joita digitaalisten pankkipalveluidenkin on noudatettava[9]. Oman talouden hoito koskee kaikkia, joten asiakkaiden ikäjakauma on lapsista vanhuksiin, ja he edustavat kaikkia ihmisryhmiä. Palvelun helppokäyttöisyys ja selkeys on siksi erityisen tärkeää.

Vuonna 2018 astui voimaan EU:n digitaalisten verkkopalveluiden saavutettavuusdirektiivi. Direktiivi koskee kaikkia julkisia verkkopalveluita, joihin myös pankkipalvelut kuuluvat. Teknisestä näkökulmasta saavutettavuus tarkoittaa sitä, että digitaalisen palvelun sisältöä on voitava tarkastella eri tavoin ja erilaisilla teknisillä laitteilla, kuten sokeain lukulaitteilla. Saavutettavuusvaatimus onkin otettava huomioon käyttöliittymän suunnittelun, toteutuksen ja testauksen jokaisessa vaiheessa.[10]

Asiakaskunnan monimuotoisuus asettaa pankkisovellukselle monenlaisia haasteita. Toiset käyttävät vain yhtä palvelukanavaa tai laitetta, kun taas toisilla on käytössään useita erilaisia laitteita pöytäkoneesta tablettiin, ja he asioivat eri kanavissa, kuten verkkopankissa ja mobiilipankissa. Palvelun täytyy näyttää ja tuntua yhdenmukaiselta laitteesta, alustasta tai kanavasta riippumatta. Perheenjäsenillä voi olla käyttöoikeus samalle tilille, tai lainalla voi olla enemmän kuin yksi velallinen, jolloin useampi asiakas saattaa käyttää samaa tuotetta eri selaimella, sovelluksella tai laitteella. Sovellus ja selaimessa toimiva verkkopankki ovat eri asioita, mutta toisinaan asiakas saattaa sekoittaa ne keskenään ja lähettää vikailmoituksen väärään kanavaan.

Mobiililaitteiden suuri kirjo aiheuttaa haasteita sovellusten tarjoajille. Heidän täytyy varmistaa, että julkaistu sovellus tukee mahdollisimman suurta potentiaalisten

[9] https://www.finlex.fi/fi/laki/ajantasa/2008/20080878
[10] https://www.humak.fi/blogit/eun-saavutettavuusdirektiivi-mita-siita-pitaisi-tietaa/

loppukäyttäjien joukkoa, mutta kehitykseen on kuitenkin käytettävissä rajallinen määrä resursseja. Tämä tarkoittaa, että jonkinlaista rajausta tuettavista ohjelmistoversioista, näyttökoosta, resoluutiosta ja kuvasuhteesta on tehtävä.

Sovellusten kohdalla adaptoitumista erilaisiin näyttöihin pystytään kontrolloimaan paremmin kuin selaimissa, vaikka näyttöjen vaihtelevat koot ja käyttöjärjestelmäpäivitykset saattavat aiheuttaa ongelmia. Selainten tapauksessa eri käyttöjärjestelmien ja niiden versioiden lisäksi muuttujina ovat eri selainversiot ja niiden sisältämät selainmoottorit, jotka käsittelevät standardeja, kuten HTML:ää (HyperText Markup Language), CSS:ää (Cascading Style Sheets) ja JavaScriptiä, eri tavoilla, mikä voi vaikuttaa sivuston tai web-sovelluksen ulkonäköön tai toimintaan merkittävästi. [1, p. 75]

Mobiilisovelluksen testauksen automatisointiin on saatavilla lukuisia työkalukehyksiä, joista useimmat soveltuvat myös graafisen käyttöliittymän automaatiotestaukseen. Sekä Android- että iOS-käyttöjärjestelmille on tarjolla omia alustaspesifisiä työkalukehyksiä, tai niiden sijaan voi vaihtoehtoisesti käyttää monialustayhteensopivia työkalukehyksiä.

1.1. Tutkimusongelma

Testaus on elintärkeä osa pankkisovelluksen kehitystä, ja sen tehtävänä on löytää ja identifioida sovelluksessa piilevät viat kehitysprosessin aikana. Paras lopputulos saavutetaan, kun testausprosessi aloitetaan jo ohjelmistokehityksen varhaisessa vaiheessa. [2]

Pankkisovelluksen testaaminen on erityisen kriittistä, sillä käyttäjämäärät voivat olla suuria, järjestelmän on kestettävä kuormituspiikkejä, ja sovelluksen taustajärjestelmineen on käsiteltävä rahatapahtumat virheettömästi, luotettavasti ja turvallisesti. Pankkisovelluksen turvallisuusvaatimukset ovat erittäin korkeat sovelluksen käsittelemän tiedon luonteesta johtuen.

Pankkisovelluksen käyttöliittymän täytyy olla selkeä ja helppokäyttöinen, ja sen pitää skaalautua oikein mobiililaitteen näytön koosta ja resoluutiosta riippumatta. Käytettävyyden ja turvallisuuden vuoksi on tärkeää, että kaikki tarpeelliset käyttöliittymän elementit näkyvät näytöllä, eivätkä esimerkiksi jää toisen elementin alle tai näyttöalueen ulkopuolelle.

Pankkisovellus koostuu useista kerroksista. Palvelurakenne on syvällä arkkitehtuurissa, ja kerrokset ja niitä erottavat rajapinnat on testattava paloittain. Kattava testausautomaatio tarvitaan sovelluksen jokaisen osakokonaisuuden testausta varten.

Ohjelmiston testaus on erittäin aikaavievää, ja siihen tarvitaan paljon resursseja. On esitetty, että ohjelmistokehityksessä jopa 40%-50% resursseista, 30% työajasta sekä 50%-60% kustannuksista käytetään testaukseen. Vaikka testaus tuottaa ohjelmistoprojekteissa haasteita ja kustannuksia, sen ansiosta on mahdollista saavuttaa säästöä ohjelmiston hinnassa, parantaa sen laatua sekä saada tuote nopeammin markkinoille. [3]

Ohjelmistotestaus jaetaan perinteisesti kahteen peruskategoriaan, manuaalitestaukseen ja automaatiotestaukseen. Manuaalitestauksessa testaaja käyttää ohjelmistoa, suorittaa testitapaukset käsin ja laatii sen jälkeen testausraportin, josta ohjelmiston laadun tilaa voidaan seurata. Automaatiotestauksessa ihmisen sijaan testitapaukset suorittaa ohjelmisto. Myös testausautomaation rakentaminen vie aikaa ja resursseja, sillä projektissa täytyy esimerkiksi määritellä mitkä osat testauksesta

kannattaa automatisoida, mahdollisesti hankkia uusi testausohjelmisto, luoda ja ylläpitää tarvittavat testit, sekä kouluttaa testaajat automaation käyttöön. Automatisoiduilla testeillä saavutetaan kuitenkin huomattavaa etua, sillä niiden avulla saadaan selville testauskattavuus, vältetään inhimillisiä virheitä sekä nopeutetaan testien suoritusta, mikä auttaa pysymään paremmin projektin aikataulussa [3]. Automatisoidut testit myös vapauttavat testaajat toistuvan manuaalitestauksen sijaan tutkimaan haastavampia ongelmia.

Kun ohjelmistoon tehdään jokin muutos, täytyy varmistaa, ettei se aiheuta mitään ongelmia aikaisemmin toteutetun toiminallisuuden kanssa [4]. Tämä tapahtuu suorittamalla uudelleen jo aikaisemmin suoritettuja testejä. Menetelmää kutsutaan regressiotestaukseksi, ja sen voi toteuttaa joko manuaalisesti, automaattisesti tai näiden yhdistelmänä. Toinen tärkeä testauksen osa-alue on yhteensopivuustestaus, millä pyritään varmistamaan, että aikaisemmin toteutettu toiminnallisuus on yhteensopiva sekä päivitettyjen että päivittämättömien laitteiden kanssa.

Pankkisovellusta testattaessa tuettavien laitteiden määrä aiheuttaa suuren työkuorman sovelluksen testaajille. Manuaalinen käyttöliittymän regressiotestaus ennen jokaista ohjelmistojulkaisua täytyy tehdä laite kerrallaan, mikä on hidasta ja työlästä. Regressiotestauksen toistuvasta luonteesta johtuen testitapausten toteuttaminen manuaalisesti ei ole kustannustehokasta eikä järkevää.

Tämän diplomityön tutkimusongelmat ovat seuraavat:

Mitä vaatimuksia graafisen käyttöliittymän automaatiotestaamiselle on pankkitoiminnan kaltaisten monimutkaisten järjestelmien kehittämisessä?

Millaisia työkaluja on käytettävissä mobiilitestauksen automatisointiin?

Kuinka responsiivisen mobiilisovelluksen automaattinen testaaminen on toteutettavissa?

1.2. Tutkielman tavoitteet

Tämän tutkielman tavoitteena on tutkia, löytyykö ratkaisumalli ja työkalut, joita käyttämällä testaaja pystyy automatisoimaan pankkisovelluksen käyttöliittymän elementtien, sisällön ja ulkoasun testaamisen mahdollisimman tehokkaasti ja kattavasti useilla eri mobiililaitteilla.

Tutkielman tavoitteena on selvittää mitä vaatimuksia mobiilisovelluksen testaamiseen liittyy, ja miten mobiilialustat, niiden arkkitehtuuri sekä pankkikonteksti vaikuttaa graafisen käyttöliittymän testaukseen ja testausratkaisun valintaan. Tämän tutkielman tuottama artefakti on kriteeristö, joka auttaa tiimiä tai organisaatiota valitsemaan sopivan testausratkaisun ja työkalut pankkisovelluksen graafisen käyttöliittymän automaatiotestaukseen.

Tässä tutkielmassa pankkikonteksti pohjautuu OP:n mobiilisovellukseen. Mobiilisovellusten testauksen asettamat vaatimukset ovat kuitenkin monilta osin yhteisiä sovelluksen kontekstista riippumatta, joten tutkielma tuottaa yleisiä vaatimuksia, jotka on pyritty löytämään yhden pankin mobiilisovelluksen kautta.

1.3. Tutkimusmenetelmä ja tutkielman rajaukset

Tässä tutkielmassa käytetään suunnittelutieteen tutkimusmenetelmää. Suunnittelutieteen yhtenä päämääränä on soveltaa olemassa olevaa tietoa, jotta kiinnostaviin ja ajankohtaisiin liike-elämän ongelmiin löytyisi ratkaisu [5]. Menetelmä pyrkii luomaan innovaatioita, joiden määrittelemien ideoiden, käytäntöjen, teknisten resurssien ja tuotteiden avulla pystytään suorittamaan analyysi, suunnittelu, toteutus, projektin hallinto, sekä tietojärjestelmän käyttö mahdollisimman tehokkaasti. Suunnittelutieteen perimmäinen ajatus on iteratiivisesti suunnitella, toteuttaa ja arvioida innovatiivinen artefakti, joka voi olla esimerkiksi malli, menetelmä, suunnitelma tai tuote. Arvioinnista saatua palautetta voidaan käyttää uuden artefaktin suunnittelun pohjana. [6]

Hevner, March, Park ja Ram ovat määritelleet tietojärjestelmien kehityksessä käytettävälle suunnittelutieteen viitekehykselle seitsemän ohjetta (taulukko 1). Suunnittelutieteen keskeinen toimintaperiaate, johon nämä ohjeet pohjautuvat, on se, että tietämys ja ymmärrys suunnitteluongelmasta ja sen ratkaisusta on hankittu artefaktin luomisen ja käytön yhteydessä. [6]

Taulukko 1. Suunnittelutieteen tutkimuksen ohjesäännöt

Ohjesääntö	**Kuvaus**
1. Suunnittele artefakti	Suunnittelutieteen tutkimusmenetelmän täytyy tuottaa toimintakelpoinen artefakti, joka voi olla konstruktio, malli, menetelmä tai järjestelmä.
2. Ongelman merkitys	Tutkimusmenetelmän tavoitteena on kehittää teknologiaan perustuva ratkaisu tärkeään ja olennaiseen liiketoiminnan ongelmaan.
3. Suunnittelun arviointi	Artefaktin hyödyllisyys, laatu ja tehokkuus täytyy todistaa tieteellisiä arviointimenetelmiä käyttämällä.
4. Tutkimuksen kontribuutio	Tuottavan suunnittelutieteen tutkimuksen täytyy tarjota selkeä ja todistettava kontribuutio suunnittelun eri osa-alueille.
5. Tutkimuksen täsmällisyys	Artefaktin rakentaminen ja arviointi tehdään käyttämällä täsmällistä tieteellistä menetelmää.
6. Suunnittelu tutkimusprosessina	Toimivan artefaktin suunnitteleminen on iteratiivinen prosessi, jonka tarkoituksena on tuottaa tehokas ratkaisu ongelmaan.
7. Tutkimuksen viestintä	Tutkimus täytyy esittää tehokkaasti sekä tekniselle yleisölle että johtoryhmälle.

Tämä tutkielma käsittelee pankkijärjestelmän asettamia erityisvaatimuksia graafisen käyttöliittymän testausautomaatiolle, sovelluksen käyttöliittymän testauksen automatisoimiseen soveltuvia työkaluja sekä mobiililaitteiden fragmentaatiosta aiheutuvaa pankkisovelluksen graafisen käyttöliittymän responsiivisuustestauksen ongelmaa. Se ei ota kantaa sovellusten suunnittelun tai kehityksen haasteisiin, eikä käsittele sovelluksen muiden kerrosten tai rajapintojen testausta.

Tutkielman motivaattorina toimii OP:n mobiilisovellus, mutta sen toteutuksen ja testauksen yksityiskohtia ei tässä käsitellä yrityssalaisuudesta johtuen. Tutkielmassa perehdytään mobiilisovelluksen graafisen käyttöliittymän testaamiseen soveltuviin automaatiotestaustyökalukehyksiin sekä pilvipohjaisiin mobiilitestausalustoihin. Pankin mobiilisovelluksen automaatiotestien toteuttaminen vertailussa mukana olevilla työkaluilla ei sisälly tutkielman rajaukseen. Muiden pankkien mobiilisovellukset ja niiden testaus on myös rajattu tutkielman ulkopuolelle.

1.4. Tutkielman rakenne

Toisessa luvussa käsitellään mobiilisovellusten kehittämisen yleisiä periaatteita erityisesti responsiivisuuden näkökulmasta. Kolmannessa luvussa kuvataan mobiilisovellusten alustojen yleistä rakennetta, OP:n pankkisovelluksen kerrosmallia ja pankkijärjestelmän testaukseen liittyviä erityispiirteitä.

Neljännessä luvussa kuvataan mobiilisovelluksen graafisen käyttöliittymän testaamista, sen haasteita ja vaatimuksia, eli vastataan ensimmäiseen tutkimuskysymykseen. Viides luku käsittelee mobiilisovelluksen graafisen käyttöliittymän testaustyökalukehyksiä sekä pilvipohjaisia mobiilitestausalustoja, eli se vastaa toiseen ja osittain kolmanteen tutkimuskysymykseen. Kuudennessa luvussa täydennetään vastausta kolmanteen tutkimuskysymykseen, ja seitsemäs luku on tutkielman yhteenveto.

2. MOBIILISOVELLUSTEN KEHITTÄMINEN

Mobiilisovellusten kehityksessä ja jakelussa on muutamia erityispiirteitä perinteiseen ohjelmistoon verrattuna. Sovellus tarjoaa yleensä tarkasti kohdennetun ja suppean määrän toimintoja ja ominaisuuksia, ja ohjelmiston koko on pieni. Sovelluksesta riippuen kehittäjien määrä vaihtelee. Isoilla organisaatioilla voi olla iso joukko kehittäjiä, tai toisaalta sovellus voi olla vain yhden henkilön suunnittelema ja kehittämä. [7] [8]

Mobiililaiteympäristö asettaa omat vaatimuksensa sovelluksen kehitykselle. Tekniset ominaisuudet, kuten akunkesto, verkkoyhteydet ja rajapinnat laitteen muihin sovelluksiin ja komponentteihin erottavat sovelluksen kehityksen perinteisestä ohjelmistokehityksestä. Uusien versioiden julkaisusykli on yleensä nopea, ja paine kehittää uusia ja myyviä ominaisuuksia on korkea. Mobiililaitteiden ripeä evoluutio vaikeuttaa stabiilin ratkaisun kehittämistä monialustaympäristön kehitysongelmiin. Kehitystyö on usein ad hoc -tyyppistä, ja dokumentaatio puutteellista. Sovelluksen testauksen haasteina on automaatiotestaustyökalujen huono testauskattavuus ja mahdollisuus ajaa testejä vain rajallisella laitemäärällä. [7] [8]

Mobiilisovellus toimitetaan asiakkaille käyttöjärjestelmäkohtaisen sovelluskaupan kautta. Sovelluskaupan etuna on, että se tuo sovelluksen ison asiakasjoukon saataville, ja se on tasa-arvoinen jakelukanava kaiken kokoisille toimittajille. Hyvät käyttäjäarviot ja medianäkyvyyden saaneen sovelluksen latausmäärät voivat kasvaa huomattavan suureksi. Uusin versio sovelluksesta on helppo toimittaa loppukäyttäjälle sovelluskaupan kautta. Sovelluskauppa tuo mukanaan myös haasteita, sillä kilpailu on kovaa, markkinat muuttuvat nopeasti, käyttäjien laatuvaatimukset ovat korkealla, ja kynnys vaihtaa toiseen vastaavaan sovellukseen on matala. Asiakkaiden kiinnostuksen herättämiseksi sovelluksen kuvauksen, kuvakkeen, hakusanojen ja kuvankaappausten täytyy olla houkuttelevia, lisäksi eri maissa loppukäyttäjien mieltymykset ja tarpeet vaihtelevat suuresti. Sovelluksen hinnoittelu on kriittinen valintaperuste, yli puolet asiakkaista ei ole valmiita maksamaan sovelluksesta, ja noin viidennes lataa maksullisen sovelluksen vain, jos vastaavaa ilmaista sovellusta ei ole tarjolla. [7] [8]

Sovelluskaupat ovat säätäneet sovelluksia koskevan ohjeistuksen, joka päivittyy säännöllisesti, ja sen sisältö eroaa eri sovelluskauppojen välillä. Mikäli sovellus ei täytä kaupan sille asettamia ehtoja, se poistetaan kyseisen sovelluskaupan valikoimasta. Sovelluksen saatavuus, menestys tai epäonnistuminen voi siis merkittävästi riippua myös sovelluskaupasta. [8]

Sovelluskaupat tarjoavat tutkijoille paljon tietoa. Release notet, binäärikoodit, lokit ja käyttäjätutkimukset ovat pohjana asiakkaiden kuluttajakäyttäytymisen analysoinnille. Tutkimusdataa ja jaossa olevia binäärejä käytetään myös apuna uusia kehitystyökaluja rakennettaessa. [7] [8]

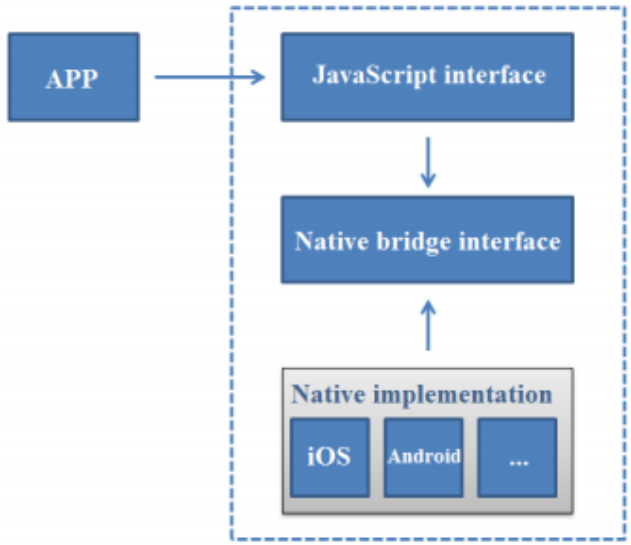
2.1. Mobiilisovellustyypit

Mobiilisovellusten päätyypit ovat natiivi, web ja hybridi, lisäksi uutena tulokkaana on progressiivinen web-sovellus. Natiivisovelluksesta kehitetään oma versio jokaiselle käyttöjärjestelmälle, ja eri versiot toteutetaan käyttöjärjestelmän omalla ohjelmointikielellä ja työkaluilla eri kehitysympäristössä, mikä vaatii paljon resursseja ja nostaa kehityskustannuksia. Lisäksi ylläpito on työläämpää, sillä jokainen uusi ominaisuus ja päivitys täytyy tehdä erikseen jokaista käyttöjärjestelmää varten.

Natiivisovelluksen etuja on hyvä suoritusnopeus ja käyttökokemus, ja sillä on pääsy mobiililaitteen muihin sovelluksiin, kuten kameraan, paikannusjärjestelmään, kontakteihin tai kalenteriin. Natiivisovellusta voi ajaa muiden sovellusten taustalla, ja käyttäjä saa laitteen näytölle ilmoituksen, kun sovelluksessa tapahtuu jotain huomionarvoista. Sovellus toimii myös ilman verkkoyhteyttä, sillä tietoa voidaan tallentaa esimerkiksi sovelluksen välimuistiin. Käyttäjä voi asentaa sovelluksen laitteeseen kyseisen käyttöjärjestelmän sovelluskaupasta. [9]

Web-sovellus on mobiilioptimoitu verkkosivu, joka ajetaan mobiililaitteen selaimessa, ja se on kehitetty käyttämällä web-teknologioita, kuten HTML:ää, JavaScriptiä ja CSS:ää. Koska sovellus toimii selaimella, se on käyttöjärjestelmäriippumaton, ja siten yksi versio sovelluksesta käy kaikille mobiilikäyttöjärjestelmille, jolloin kehitystyö on nopeampaa ja kustannukset pienemmät kuin natiivisovelluksen kohdalla. Web-sovellus toimii natiivisovellusta hitaammin, sillä se sijaitsee verkkopalvelimella, eikä sitä voi käyttää ilman verkkoyhteyttä. Web-sovellusta ei ole saatavilla sovelluskaupasta eikä sitä voi asentaa laitteeseen. Koska sovelluskoodi ajetaan selaimessa, tietoturvasyyt estävät sovelluksen pääsyn mobiililaitteen muihin sovelluksiin [9]. Monet web-sovellukset sallivat kolmannen osapuolen tarjoamien lisäosien asentamisen sovelluksen kustomoitavuuden parantamiseksi. Vaikka lisäosia sisältävien web-sovellusten laajennettavuus ja muokattavuus paranee, mahdollisuus lisäosien integroimiseen avaa sovellukseen uuden tietoturva-aukon muista tietoturvatoimista huolimatta. [10]

Hybridisovelluksessa on ominaisuuksia sekä natiivi- että web-sovelluksesta. Kehityksessä käytetään web-teknologioita, minkä jälkeen sovellus voidaan paketoida natiivisovellukseksi eri käyttöjärjestelmille käyttämällä paketointityökalua, esimerkiksi Apachen avoimen lähdekoodin Cordovaa. Paketoitu sovellus sisältää web-teknologioilla kehitetyn ohjelmakoodin sekä yleisen JavaScript-rajapinnan, jonka kautta palvelupyynnöt kulkevat web-sovellukselta käyttöjärjestelmälle kuvan 7 mukaisesti. JavaScript-rajapinta mahdollistaa pääsyn laitteen muihin sovelluksiin, mutta rajoitetummin kuin natiivisovelluksessa. [11]



Kuva 7. Hybridisovelluksen arkkitehtuuri (Bosnic, S. et al. "The development of hybrid mobile applications with Apache Cordova", 2016 24th Telecommunications Forum (TELFOR), © 2017 IEEE)

Hybridisovelluksen kehitys ja ylläpito on monialustayhteensopivuudesta johtuen nopeaa ja kustannustehokasta. Käyttöjärjestelmäkohtaisesti paketoidut hybridisovellukset ovat natiivisovellusten tavoin saatavilla sovelluskaupoista. Haittapuolena on hybridisovelluksen huonompi suorituskyky ja rajoitetumpi pääsy laitteen muihin sovelluksiin natiivisovellukseen verrattuna. [11]

Hitaus, edellytys verkkoyhteydestä sekä sovelluksen pääsyn estäminen laitteen muihin sovelluksiin vaikuttaa haitallisesti web-sovellusten käyttäjäkokemukseen. Google esitteli vuonna 2017 uuden teknologian, progressiiviset web-sovellukset (Progressive Web Application, PWA), paikkaamaan näitä puutteita. Teknologia sisältää joukon ohjelmointirajapintoja, joista käytetään termiä *service worker*. Teknisesti service worker on omassa säikeessään ajettava JavaScript-moduuli, jonka avulla käyttöön saadaan välimuistitus ja offline-toiminnot, taustalla tapahtuva datan säännöllinen synkronointi sekä push notifikaatio -viestit. Sovellus on palvelimella ja sitä käytetään ensimmäisellä kerralla selaimen kautta, minkä jälkeen se voidaan asentaa natiivisovelluksen tapaan laitteen aloitusnäytölle. Erona asennuksessa on se, että se tapahtuu palvelimelta, ei sovelluskaupan kautta. Asennuksen jälkeen käyttäjä saa täyden hyödyn progressiivisen web-sovelluksen tarjoamista uusista ominaisuuksista. Turvallisuussyistä PWA käyttää HTTPS-protokollaa (Hypertext Transfer Protocol Secure) salaamattoman HTTP-protokollan sijaan. [12]

Lukuisat seikat vaikuttavat mobiilisovelluksen tyypin valintaan. Tuetut käyttöjärjestelmät ja versiot, sekä niiden kehitykseen vaadittu työmäärä, selaimen kapasiteetti, laitteen ominaisuudet, joita sovellus tarvitsee käyttöönsä, käyttäjäkokemuksen tavoiteltu taso, sovelluksen suorituskyky ja markkinoilla olevien useiden yhtäaikaisten versioiden hallinta ovat teknisiä seikkoja, jotka projektin suunnittelussa täytyy huomioida. Myös jakelukanava ja sen tuoma näkyvyys, mahdollinen lisensointi tietyille puhelinmalleille tai sovelluskaupalle, sekä rahoitukseen liittyvät asiat vaikuttavat osaltaan valintaprosessiin. [13]

2.2. Mobile first -strategia

Sekä mobiililaitteiden määrän, niiden fragmentaation, että niille suunnatun sisällön määrän kasvu on saanut ohjelmistojen tarjoajat siirtymään mobile first (mobiili ensin) -strategiaan [14], jonka Luke Wroblewski esitteli vuonna 2009[11]. Nimensä mukaisesti mobile first tarkoittaa sitä, että web-sivun tai sovelluksen suunnittelu ja kehitys suunnataan ensin mobiililaitteille, ja vasta kehityksen myöhemmässä vaiheessa toiminnallisuus laajennetaan pöytäkoneille sekä kannettaville tietokoneille, tai niille suunnitellaan kokonaan oma ratkaisu. [14]

Strategian taustalla on ajatus siitä, että mobiililaitteet ovat käytössä kaikkialla ja niiden teknologia on parantunut huimasti, erityisesti verkon peittoalueen ja datan siirtonopeuden suhteen. Strategiassa oletetaan myös, että asiakas ensisijaisesti käyttää palvelua aina mobiililaitteella. [14]

Strategian heikkoutena on, että suunnittelussa voidaan keskittyä vain johonkin tiettyyn laiteryhmään käyttäjien laitteistoriippumattomat tarpeet unohtaen. Osa potentiaalisista asiakkaista ei esimerkiksi halua käyttää mobiilisovelluksia tai ei omista älypuhelinta, jolloin sovelluksen lisäksi myös moniselainympäristössä toimiva versio palvelusta olisi tarpeen. [14]

[11] https://www.lukew.com/ff/entry.asp?933

2.3. Graceful degradation ja progressive enhancement -suunnittelumenetelmät

Verkkosivujen suunnittelussa on perinteisesti kaksi lähestymistapaa, jotka ovat toistensa vastakohtia. Graceful degradation -suunnittelumenetelmä tarkoittaa, että uusimmille ja suorituskykyisimmille selaimille kehitetään ensin optimaalinen, täyden toiminnallisuuden ja kaikki ominaisuudet sisältävä sivusto, ja sen jälkeen kehityksen loppuvaiheessa siitä riisutaan ominaisuuksia kerros kerrokselta, jotta varmistetaan sivuston toiminta myös vanhemmissa selaimissa. Menetelmää kutsutaan hallituksi heikentämiseksi (graceful degradation). [15]

Virheensietokyky (fault tolerance) on HTML:n ja CSS:n ominaisuus, joka mahdollistaa hallitun heikentämisen. Se tarkoittaa sitä, että selain ohittaa kaiken HTML- ja CSS-koodin, jota se ei ymmärrä. Tämän ansioista sivun perusominaisuudet ovat käytettävissä myös vanhoilla selaimilla. [15]

Hallitun heikentämisen vastakohta on asteittainen parantaminen (progressive enhancement). Kehittäminen aloitetaan kaikkien selainten tukemista perustoiminnoista, ja uudempien selainten tukemaa kehittyneempää toiminnallisuutta lisätään kerroksittain, kunnes täysi käyttäjäkokemus saavutetaan. [15]

2.4. Sovellusten sisältöadaptaatio

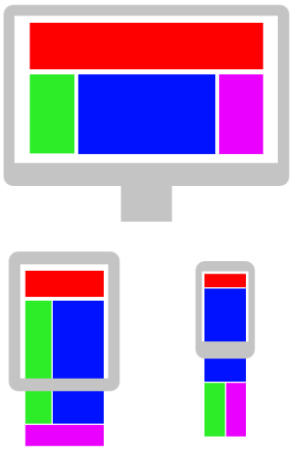
Adaptiivinen ja responsiivinen suunnittelu ovat menetelmiä, joita käytetään, kun sovelluksen tai verkkosivujen sisältö halutaan mukauttaa näyttölaitteille, joiden koko ja resoluutio vaihtelee. Toisinaan termejä käytetään toistensa synonyymeinä, vaikka niiden lähestymistapa ongelmaan eroaa toisistaan.

Adaptiivisessa verkkosuunnittelussa eri näytönleveyksille on oma staattinen sivupohja (layout), joka valitaan ja ladataan tunnistetun näyttökoon perusteella. Tämä tarkoittaa sitä, että sivustosta täytyy luoda ja ylläpitää useita versioita. Sivupohjat rakennetaan tavallisesti älypuhelinten, tablettien, kannettavien tietokoneiden sekä pöytäkoneiden näyttöjen yleisimpien näyttöresoluutioiden mukaan. Tämä suunnittelutapa on käyttökelpoinen silloin, kun olemassa olevaa, tietylle resoluutiolle suunniteltua sivustoa halutaan laajentaa tukemaan myös muita laitekokoja.[12]

Responsiivinen verkkosuunnittelu eroaa adaptiivisesta siten, että yksi versio sivustosta sopeutuu joustavasti jokaiseen näyttökokoon (kuva 8). Responsiivinen suunnittelu käyttää CSS3:n media query -komponenttia, jonka avulla sivun tyylejä voi vaihtaa esimerkiksi näytön tyypin, leveyden tai korkeuden mukaan. Responsiivisen suunnittelun etuna on, että yhden version kehittäminen ja ylläpitäminen vaatii pienemmän työkuorman adaptiiviseen suunnitteluun verrattuna. Jokaiselle näyttökoolle sopiva responsiivinen toteutus voi kuitenkin olla kompleksinen ja staattista sivupohjaa vaikeampi toteuttaa.[12]

Responsiivisuuden periaatteet soveltuvat kaiken kokoisille näytöille, mutta pöytäkoneista poiketen mobiililaitteissa on kosketusnäyttö. Perinteisesti sivua on klikattu tai vieritetty hiirellä, mutta mobiililaitteilla syötteet tapahtuvat napauttamalla näyttöä kerran tai kahdesti, painamalla pitkään, pyyhkäisemällä, liu'uttamalla tai zoomaamalla kahdella sormella. Mikäli verkkosivua ei ole optimoitu mobiiliselaimille, jokainen touch event liipaisee myös click eventin, joka täytyy suodattaa pois ennen datan lähettämistä palvelimelle. Myöskään näppäinkomentojen käyttäminen ei onnistu mobiililaitteen virtuaalinäppäimistöltä. [16]

[12] https://www.uxpin.com/studio/blog/responsive-vs-adaptive-design-whats-best-choice-designers/



Kuva 8. Responsiivisen verkkosivun periaate eri kokoisilla näytöillä (Procházka, T. CCO 1.0 Universal Public Domain Dedication).

2.4.1. Natiivisovellus

iOS-käyttöjärjestelmä tarjoaa automatiikan natiivisovellusten käyttöliittymän adaptaatioon mobiililaitteessa. Sovellusten kehitysympäristöissä on tarjolla työkaluja, joiden avulla käyttöjärjestelmän tarjoamaa responsiivisuutta voidaan vielä parantaa. UIKit on ohjelmointityökalukehys, joka tarjoaa iOS-sovellukselle infrastruktuurin, kuten näkymäarkkitehtuurin, tuen monipistenäytölle, animaatioille, dokumenteille, piirtämiselle ja tulostamiselle, hakutoiminnolle tai sovelluksen lisäosille[13]. Työkalukehys määrittelee myös käyttöliittymän komponentit, kuten tekstikentät, painonapit, animaatiot tai interaktiiviset elementit. Komponentit adaptoituvat ja päivittyvät automaattisesti ympäristön muutosten mukaan.[14]

Sovelluksen toimintaympäristöä määrittäviä karkean tason ominaisuuksia (traits) ovat mm. näytön skaalaus, laitteen tyyppi sekä vertikaalinen ja horisontaalinen kokoluokka. Näitä ominaisuuksia käyttämällä on mahdollista konfiguroida sovellukselle eri näkymä eri kokoluokan laitteille[15]. Käyttöliittymän komponenteille määritellään suhteellinen sijainti, koko sekä etäisyys toisista komponenteista, jolloin näkymän sisältö mukautuu näytön koon mukaan. Sijainnin, koon ja etäisyyksien määrittävistä rajoista käytetään termiä *constraints*.[16]

Android-käyttöjärjestelmä suorittaa perustason skaalauksen ja koon muuttamisen mukauttaakseen käyttöliittymän eri kokoisille näytöille. Kehittäjän on kuitenkin varmistettava, että sovellus adaptoituu jokaiselle näyttötyypille. Sovelluksen käytössä olevan näytön koko ei ole sama kuin fyysinen näytön koko, sillä laitteen orientaatio, valikot tai ikkunakonfiguraatiot voivat muuttua. Pienimpiinkin näytön koon eroihin

[13]https://developer.apple.com/documentation/uikit
[14]https://developer.apple.com/design/human-interface-guidelines/ios/overview/interface-essentials/
[15]https://developer.apple.com/library/archive/featuredarticles/ViewControllerPGforiPhoneOS/BuildinganAdaptiveInterface.html
[16]https://developer.apple.com/library/archive/documentation/UserExperience/Conceptual/AutolayoutPG/AnatomyofaConstraint.html

adaptoituva käyttöliittymä saadaan aikaiseksi joustavalla sivupohjalla tai vaihtoehtoisilla sivupohjilla sekä venyvillä kuvilla.[17]

Joustava sivupohja tarkoittaa sitä, että käyttöliittymäkomponenttien paikka tai koko ei ole kiinteä. Komponenteille ja näkymille määritellään niiden väliset suhteelliset sijainnit ja etäisyydet, jotka mukautuvat näytön koon muuttuessa. Joustavan sivupohjan lisäksi on hyvä luoda vaihtoehtoisia sivupohjia, jotta käyttökokemus saadaan optimoitua eri kokoluokan laitteille, kuten tableteille ja älypuhelimille. Käyttöjärjestelmä vaihtaa sivupohjaa ajon aikana sopimaan senhetkiseen näytön kokoon. Android tukee ominaisuutta nimeltä *nine-batch bitmap*, jotta sivupohjan mukana venyvät kuvat skaalautuvat oikeat mittasuhteet säilyttäen.[17]

Pikselitiheys (pixel density) tarkoittaa pikselien lukumäärää tietyllä näytön alueella, ja se eroaa resoluutiosta, joka on kaikkien näytön pikselien yhteenlaskettu määrä. Tiheysriippumattomuus (density independence) tarkoittaa sitä, että käyttöliittymän sisällön koko näyttää muuttumattomalta, vaikka näytön pikselitiheys vaihtelee. Ilman tiheysriippumattomuutta UI-komponentin koko on isompi pienen tiheyden näytöllä, jossa pikselin koko on iso, ja pienempi suuren tiheyden näytöllä, jolloin pikselin koko on pienempi. Android tukee tiheysriippumattomia pikseleitä, joita käytetään mittayksikkönä pikselien sijaan.[17]

Jotta kuvat näyttäisivät samankokoiselta jokaisella näytöllä, Android skaalaa bittikarttoja näytön tiheyden mukaan, mikä voi aiheuttaa näkyviä vääristymiä suuremman pikselitiheyden tapauksessa. Tämän välttämiseksi on hyvä luoda vaihtoehtoisia bittikarttoja suuremman resoluution näytöille. Yksinkertaiset kuvat, kuten ikonit, saadaan skaalattua eri pikselitiheyksille käyttämällä vektorigrafiikkaa, joka perustuu pikselien sijaan koordinaatteihin ja matemaattisiin funktioihin.[17]

2.4.2. Web-sovellus ja progressiivinen web-sovellus

Web-sovelluksen adaptaatio tapahtuu laitteen sijaan selaimessa. Web-sovellukset kehitetään käyttämällä web-teknologioita ja työkalukehyksiä. HTML5-dokumentti määrittelee sivuston rakenteen, CSS3 tyylit, ja ohjelmointikielellä, kuten JavaScriptillä, toteutetaan sivuston toiminnallisuus [11]. Työkalukehys yksinkertaistaa ja nopeuttaa monimutkaisten sivustojen kehitystyötä tarjoamalla mm. valmista koodia ja mallipohjia, sekä automaatiota usein toistuviin työvaiheisiin [17].

Web-sivujen ja sovellusten responsiivisuus perustuu joustaviin, ruudukkoperustaisiin sivupohjiin, venyviin kuviin, sekä CSS3:n media queryihin. Joustava sivupohja luodaan siten, että sen sarakkeiden, marginaalien ja muiden elementtien koko ja etäisyydet määritellään pikselien sijaan prosentteina, jolloin sivun mittasuhteet säilyvät samana, vaikka näytön koko tai orientaatio muuttuu. Yhden sivupohjan sijaan voidaan luoda myös vaihtoehtoisia sivupohjia eri resoluutioille. [18, pp. 13-79]

On monia tekniikoita, joilla kuvan kokoa voidaan muuttaa sen mittasuhteet säilyttäen. Mikäli kuvan koko on suurempi kuin sille määritetyn ylätason elementin koko, saadaan se skaalattua oikean kokoiseksi asettamalla sen maksimileveys (max-width) sataan prosenttiin. Mikäli selaimen koko muuttuu pienemmäksi, myös kuvan kokoa pienennetään oikeassa suhteessa. Selaimissa on sisäänrakennettuna toiminallisuus, joka huolehtii, että kuvan mittasuhteet pysyvät samana, vaikka sen koko muuttuu. Eri resoluutioille voidaan myös luoda eri kokoisia kuvia, jolloin

[17] https://developer.android.com/guide/practices/screens_support

kyseiselle resoluutiolle sopiva kuva toimitetaan sivuston palvelimelta sivun latauksen yhteydessä. [18, pp. 42-63]

Media queryt ovat CSS3:n moduuli, jonka avulla määritellään erilaisia CSS-tyylejä ja elementtien asetteluja eri kokoisille näytöille. Media queryissä määritellään breakpointit, joissa kyseiset tyylit tulevat voimaan. Breakpointit valitaan yleensä suosituimpien laitteiden resoluutioiden mukaan, tai sopiva arvo haetaan testaamalla missä koossa käyttöliittymän toiminta muuttuu. [18, pp. 64-105]

2.4.3. Hybridisovellus

Hybridisovellukset kehitetään web-sovellusten tapaan responsiivisilla web-tekniikoilla ja paketoidaan sen jälkeen natiivisovelluksen sisään kolmannen osapuolen tarjoamalla kehitystyökalukehyksellä. Hybridisovellukset eivät toimi laitteen selaimessa, vaan niiden web-sovellusosa ajetaan natiivisovelluksen sisään rakennetussa selaimessa, joka käyttää laitteen selainmoottoria HTML:n ja JavaScriptin renderöintiin. [12]

Työkalut kehittyvät koko ajan, ja nykyään on tarjolla myös työkalukehyksiä, joilla voidaan kehittää sekä käyttäjäkokemukseltaan, suorituskyvyltään, ominaisuuksiltaan että ulkonäöltään yhä enemmän natiivisovelluksen kaltaisia hybridisovelluksia. Työkalukehykset käyttävät samoja UI-komponentteja ja rajapintoja kuin natiivisovelluskin, jolloin sovelluksen renderöinti tapahtuu selaimen sijaan laitteessa. Esimerkiksi React Native on JavaScript-pohjainen työkalukehys, jolla voi kehittää alustariippumattomia sovelluksia sekä iOS:lle että Androidille.[18]

2.4.4. Saavutettavuus

Universal design on menetelmä, jolla suunniteltuja tuotteita voi käyttää mahdollisimman monessa tilanteessa ja ympäristössä mahdollisimman laaja joukko ihmisiä, joilla on erilaisia kykyjä ja rajoituksia. Saavutettavuus puolestaan tarkoittaa erityisesti vammaisille henkilöille suunniteltuja tuotteita ja palveluita. World Wide Web Consortium (W3C) on laatinut WAI-aloiteen (Web Accessibility Initiative)[19] ja suositukset (Web Content Accessibility Guidelines, WCAG)[20] saavutettavien verkkosivujen suunnitteluperiaatteista[21] [22]. W3C on julkaissut erillisen ohjeistuksen[23] myös mobiililaitteille suunniteltuja saavutettavia sovelluksia ja web-sivuja varten. [19]

Vaikka saavutettavien palveluiden kohderyhmä ovat vammaiset henkilöt, niistä hyötyvät muutkin käyttäjät. Esimerkiksi vanhusten aistit ovat usein heikentyneet, jolloin mm. näkö- tai kuulovammaisille tarkoitetut apukeinot hyödyttävät heitäkin. Saavutettavuus tuo etua myös silloin, kun palvelun käyttäjällä on heikko lukutaito, vanha laitteisto tai ohjelmisto, tai huono verkkoyhteys. Saavutettavien palveluiden suunnittelu on kuitenkin tärkeää kohdentaa juuri vammaisille, jotta voidaan varmistaa, että heidän erityistarpeensa tulevat täytettyä. [19]

---

[18] https://www.oreilly.com/library/view/learning-react-native/9781491929049/ch01.html
[19] https://www.w3.org/WAI/
[20] https://www.w3.org/WAI/standards-guidelines/wcag/new-in-21/
[21] https://www.w3.org/TR/mobile-accessibility-mapping/
[22] https://www.w3.org/WAI/standards-guidelines/mobile/
[23] https://www.w3.org/TR/mobile-bp/

Saavutettavuus asettaa rajoituksia responsiivisten verkkosivujen suunnittelulle ja kehitykselle, sillä responsiiviset web-teknologiat eivät automaattisesti takaa saavutettavuutta. Jotta saavutettavuus toteutuu, sen periaatteita ja ohjeistusta on noudatettava sivuston tuotekehitysprosessin alusta saakka.

HTML:n semanttisuus on avainasemassa saavutettavaa verkkosivua suunniteltaessa. Semanttisuus tarkoittaa sitä, että oikeita HTML-elementtejä käytetään niiden oikeissa käyttötarkoituksissa niin paljon kuin mahdollista. Semanttisia HTML-elementtejä ovat esimerkiksi <button>, <form>, <table> tai <article>, jotka selkeästi määrittelevät elementin sisällön. ja ovat esimerkkejä ei-semanttisista elementeistä, sillä ne eivät millään tavalla kuvaa elementin sisältöä. Otsikkotagien oikea käyttö on tärkeää, ja niitä ei koskaan pidä käyttää muun tekstin muotoiluun. Kuvien vaihtoehdoksi voidaan määrittää vaihtoehtoinen teksti alt-attribuutilla, jotta ruudunlukuohjelma pystyy sen esittämään. Hakukoneita ja ruudunlukijoita varten on olennaista myös määrittää sivuston kieli HTML-dokumentissa. Linkkien otsikon tulisi kuvata selkeästi, mitä tietoa linkistä avautuu. Varsinainen sivuston tekstisisältö olisi hyvä kirjoittaa selkeästi lyhyitä ja ymmärrettäviä lauseita käyttäen. Lyhenteitä, erikoismerkkejä ja slangisanoja tulee välttää tekstissä[24]. Selaimiin on sisäänrakennettu tuki HTML:n saavutettavuusominaisuuksille, mutta tuen kattavuus vaihtelee jonkin verran selainkohtaisesti.[25]

CSS:n etu saavutettavuudelle on se, että sillä erotetaan sivun muotoilu sen rakenteen ja sisällön määrittelevästä HTML-dokumentista. Tällöin HTML-dokumentti voidaan pitää yksinkertaisena ja selkeänä, ja se toimii hyvin myös ruudunlukuohjelmien kanssa. Kun sisältö ja tyylit on erotettu, sivu on saavutettava ja sen sisältö on käytettävissä, vaikka tyyliasetukset eivät olisikaan voimassa. CSS:n tarjoamat tyylien muotoiluasetukset ehkäisevät sitä, ettei HTML:n elementtejä käytetä sivun ulkoasun muotoiluun, mikä aiheuttaisi ongelmia ruudunlukuohjelmia käytettäessä. CSS mahdollistaa käyttäjälle omien tyyliasetusten luomisen, jolloin käyttäjä voi valita esimerkiksi itselleen paremmin sopivat fontit tai värit. CSS:n tukema Aural style sheets (ACSS) tarjoaa mahdollisuuden määrittää miltä dokumentti kuulostaa ruudunlukuohjelman tulkitsemana. Sivuston tekijä tai käyttäjä voi muuttaa mm. luetun tekstin ja taustaäänien äänenvoimakkuutta, puhenopeutta tai sävelkorkeutta.[26]

2.5. Pankkisovelluksen kehityksen erityispiirteet

Pankkisovelluksen tuotantoonvienti on tarkkaan kontrolloitua. Laatuportit ja tarkistukset sekä korjaus- ja jatkuvuussuunnitelmat on oltava jo etukäteen. Finassivalvonnan nimeämät henkilöt valvovat, että kaikkia asetuksia noudatetaan. Kehittäjillä tai testaajilla ei ole pääsyä tuotantoympäristöön, ja tuotantoasennuksia tekemään on nimetty sitä varten eri henkilöt, eli on poistettu niin kutsutut vaaralliset työyhdistelmät. Tästä johtuen tarvitaan useita ympäristöjä, joissa sovelluksen kehitys ja testaus tapahtuu. Ympäristöjen välillä on eroja esimerkiksi testiaineiston kattavuuden ja eräajojen automatiikan suhteen, mikä aiheuttaa haasteita sekä kehittäjille että testaajille.

[24] https://www.w3schools.com/html/html_accessibility.asp
[25] https://www.html5accessibility.com/
[26] https://www.w3.org/TR/CSS-access

Pankkitoiminnan on oltava käytettävissä ympäri vuorokauden vuoden jokaisena päivänä. Häiriöiden varalta täytyy olla varasuunnitelma, sillä toiminnan jatkuvuus on taattava. Moninkertaiset suojaukset ja varmuuskopiot ovat kriittisen tärkeitä olla olemassa. Tuotantohäiriön syyt on selvitettävä välittömästi, ja usein kehittäjät ja testaajat osallistuvat selvitystyöhän.

3. PANKKISOVELLUSTEN MOBIILIALUSTAT

Natiivisovellus on käyttöjärjestelmäriippuvainen, joten yksi versio pankkisovelluksesta ei riitä sen asiakaskunnalle. Koska kaksi käyttöjärjestelmää, Android ja iOS, dominoivat tällä hetkellä markkinoita, myös pankkisovelluksesta täytyy tarjota oma versio näille kahdelle käyttöjärjestelmälle. Henkilöasiakkaalle suunnatun mobiilisovelluksen lisäksi tarjolla on oma sovellus myös yritysasiakkaille. OP:n mobiilisovelluksen käyttäjistä noin 70%:lla on Android-laite ja noin 30%:lla iOS-mobiililaite.

Mobiilialustojen keskeinen piirre on niiden jatkuva kehittyminen. Käyttöjärjestelmäversioiden nopea päivitystahti kuormittaa sekä sovelluksen kehittäjiä että testaajia. Taulukossa 2 on esitetty Androidin ja iOS:n käyttöjärjestelmäpäivitykset sekä niiden ensimmäisen tuotantoversion julkaisukuukausi loppuvuodesta 2011 alkaen[27] [28]. Käyttöjärjestelmistä on julkaistu lisäksi useita pienempiä päivityksiä, joiden aikataulua ei ole eritelty näissä taulukoissa.

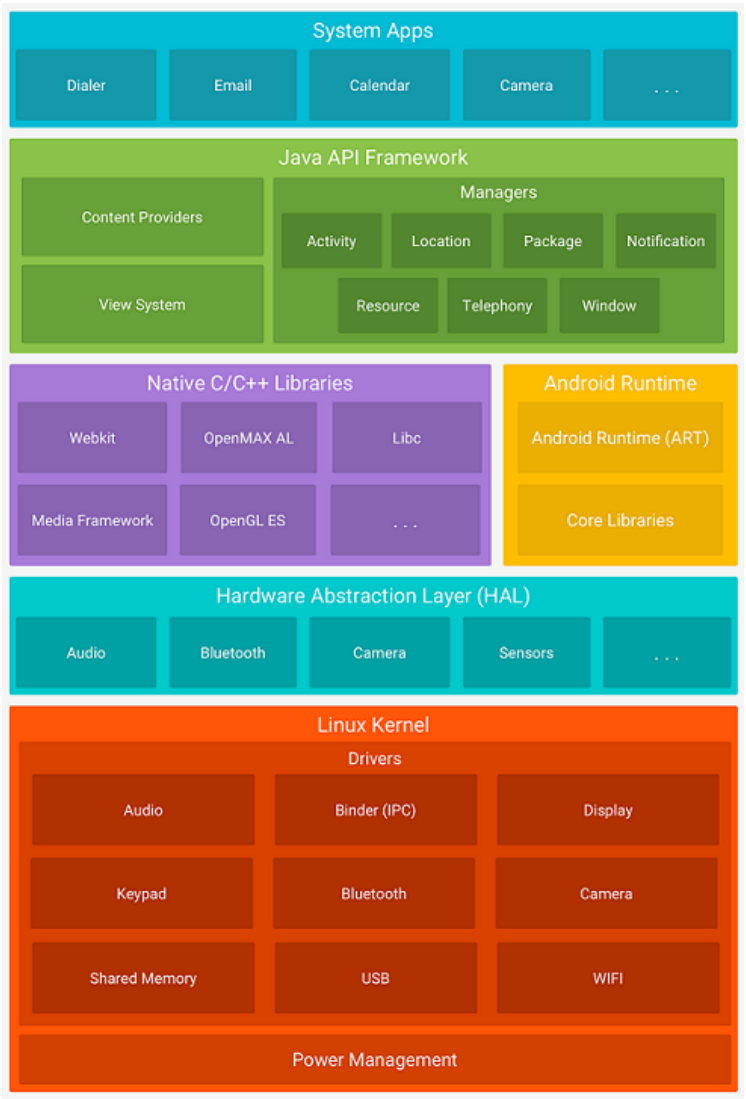
Taulukko 2. Android- ja iOS-käyttöjärjestelmistä julkaistut versiot vuodesta 2011 alkaen

Android		**iOS**	
Versionumero	**Ensimmäisen version julkaisukuukausi**	**Versionumero**	**Ensimmäisen version julkaisukuukausi**
4.0 - 4.0.4	10/2011	5.0.x - 5.1.x	10/2011
4.1 - 4.3.1	07/2012	6.0.x - 6.1.x	09/2012
4.4 - 4.4.4	10/2013	7.0.x - 7.1.x	09/2013
5.0 - 5.1.1	11/2014	8.0.x - 8.4.x	09/2014
6.0 - 6.0.1	10/2015	9.0.x - 9.3.x	09/2015
7.0 - 7.1.2	08/2016	10.0.x - 10.3.x	09/2016
8.0 - 8.1	08/2017	11.0.x - 11.4.x	09/2017
9.0	08/2018	12.0.x - 12.4.x	09/2018
10.0	09/2019	13.0 - 13.6.x	09/2019
11.0 Beta	06/2020	14.0 Beta	08/2020

3.1. Android-sovellusalustan rakenne

Android on Linuxiin perustuva avoimen lähdekoodin ohjelmisto. Androidin sovellusalusta koostuu kuudesta pääkomponentista, jotka ovat sovelluskerros (System Apps), sovelluskehys (Java API Framework), natiivikirjastot (Native C/C++ Libraries), Android Runtime (ART), abstraktiokerros (Hardware Abstraction Layer, HAL), ja Linux-ydin (Linux Kernel) kuvan 9 mukaisesti.[29]

[27] https://en.wikipedia.org/wiki/Android_version_history
[28] https://en.wikipedia.org/wiki/IOS_version_history
[29] https://developer.android.com/guide/platform



Kuva 9. Android-sovellusalustan arkkitehtuuri (Reused under the Apache License, Version 2.0).

Sovelluskerroksessa on Androidin mukana tulevat keskeiset sovellukset, kuten sähköposti, tekstiviestit, kalenteri ja yhteystiedot. Sovellukset ovat loppukäyttäjän käytössä, mutta myös sovelluskehittäjä voi hyödyntää niiden toiminnallisuutta omassa kehitystyössään. Androidin käyttöjärjestelmän kaikki ominaisuudet ovat käytettävissä sovelluskehyksen Java API-rajapintojen (Application Programming Interface) kautta. Rajapinnat muodostavat rakennusosat uusien Android-sovellusten tekemiseen, sillä ne yksinkertaistavat ytimen, modulaarijärjestelmien komponenttien sekä palveluiden käyttämisen. Sovelluskehittäjillä on pääsy kaikkiin samoihin rajapintoihin kuin Androidin järjestelmäsovelluksillakin.[29]

Monet Androidin järjestelmäkomponentit ja -palvelut on rakennettu natiivikoodilla, joka tarvitsee C- tai C++ -kielellä kirjoitetut kirjastot. Kirjastojen toiminnot ovat käytettävissä Java API-rajapintojen kautta. Android Runtime on sovellusten ja käyttöjärjestelmän välissä oleva ohjelmistokerros. Se luo jokaiselle sovellukselle oman virtuaalikoneen, mikä mahdollistaa useiden sovellusprosessien yhtäaikaisen ajamisen. Android Runtime muuntaa ART- tai Dalvik -virtuaalikoneen Android-

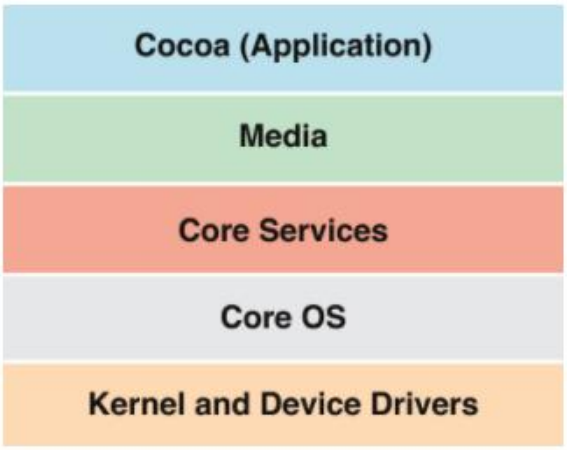
alustalla käytettävän Javan luokkatiedostojen DEX-tavukoodiksi. Suurin osa Androidin sovelluskehittäjien käyttämän Java-ohjelmointikielen toiminnallisuudesta sisältyy Android Runtimen ydinkirjastoon. Android 5.0 -käyttöjärjestelmäversiosta alkaen käytössä on ART-virtuaalikone, aikaisemmin käytettiin Dalvik-virtuaalikonetta.[29]

Abstraktiokerros mahdollistaa sujuvan kommunikoinnin Android-laitteen fyysisen laitteiston ja ohjelmistojen välillä. Kerros tarjoaa standardirajapinnan, joka piilottaa ulkoisten laitteiden erilaisuuden, ja jonka kautta laitteiston ominaisuudet tuodaan Java API -työkalukehyksen käyttöön. Abstraktiokerros muodostuu useista kirjastomoduuleista, joista jokainen toteuttaa rajapinnan laitteistokomponenttiin, kuten kameraan tai bluetooth-moduuliin.[29]

Androidin sovellusalustan perusta on Linux-ydin, joka käsittää arkkitehtuurin ylempien osien toiminnan mahdollistavat ajurit, sekä prosessien, muistin- ja virranhallinnan. Android hyötyy Linux-ytimen turvallisuusominaisuuksista, ja hyvin tunnettu ydin on etu, kun laitevalmistajat kehittävät uusia laitteistoajureita.[29]

3.2. iOS-sovellusalustan rakenne

iOS on yksinoikeudella Apple Inc:n mobiililaitteissa käytettävä mobiilikäyttöjärjestelmä. iOS-arkkitehtuuri koostuu neljästä kerroksesta, joita ovat Cocoa Touch, Media, Core Services ja Core OS [20]. Kuvassa 10 on esitetty OS X -käyttöjärjestelmän arkkitehtuuri, jota iOS mukailee.



Kuva 10. OS X -käyttöjärjestelmän arkkitehtuuri (Reused on the permission from Apple Inc.).

Cocoa Touch on arkkitehtuurin ylin kerros ja se vastaa iOS-käyttöjärjestelmän ja sovellusten käyttöliittymästä sekä sovellusten yleisimmin käyttämistä toiminnoista. Kerros sisältää Foundation- ja UiKit -työkalukehykset, joilla iOS-käyttöjärjestelmässä ajettavia sovelluksia voidaan kehittää. Foundation tarjoaa sovellusten ja työkalukehysten perustoiminallisuuden, kuten tiedon tallentamisen, tekstin prosessoinnin, päivämäärän ja ajan laskemisen, lajittelun ja suodatuksen sekä verkkoyhteydet. Cocoa Touchin Foundation-työkalukehys perustuu Core Services -

kerroksen Foundation-työkalukehykseen[30] [31]. UiKit vastaa mm. käyttöliittymän ikkuna- ja näyttöarkkitehtuurista sekä käyttäjän, sovelluksen ja käyttöjärjestelmän välisestä interaktiosta[13]. Cocoa Touch -kerros tukee mm. moniajoa, kosketusnäytön syötteiden vastaanottamista sekä push-ilmoituksia. Pääasialliset kehityskielet ovat C++ ja Swift. [20]

Media Layer tarjoaa monipuoliset rajapinnat ja työkalukehykset sovelluksen multimediaominaisuuksille, kuten äänelle, kuvalle, 2D- ja 3D -grafiikalle, animaatiolle, teksteille, videolle ja suoratoistolle verkon yli. Mm. värien hallinta, tulostustoiminnot, audiovisuaalisen datan tallentaminen ja käsittely sekä tuki pikaviestipalveluiden liitännäisille kuuluvat Media Layerin palveluihin.[32]

Core Services tarjoaa keskeiset teknologiat ja palvelut sovelluksille, mutta sillä ei ole suoraa yhteyttä sovelluksen käyttöliittymän kanssa. Teknologioilla on riippuvuus arkkitehtuurissa alempana olevan Core OS -kerroksen teknologioiden ja työkalukehysten kanssa. Core Services sisältää mm. palvelut ja rajapinnat sosiaalisen median integraatioon, iCloud-pilvipalvelun käyttöön, tiedostojen ja tietokantojen käyttöön, lokalisaatioon, tietoturvapalveluihin, verkkotoimintoihin ja karttoihin. Kerroksen tärkeimmät rajapinnat ovat Core Foundation ja Foundation, jotka vastaavat esimerkiksi tietotyypeistä, kokoelmista, Unicode-merkistöstä ja olioiden perustoiminnoista ja käyttäytymisestä.[33]

Core OS -kerros sisältää teknologiat ja työkalukehykset alimman tason tallennusjärjestelmään, turvallisuuteen, verkkoyhteyksiin sekä muihin kernelin toimintoihin liittyen. Kerros tarjoaa rajapinnat mm. rinnakkaislaskentaan ja raskaiden matemaattisten laskutoimitusten nopeuttamiseen.[34]

3.3. Android-sovelluksen komponentit

Android-sovellus rakentuu komponenteista. Jokainen komponentti voi toimia pisteenä, jonka kautta järjestelmä tai käyttäjä voi siirtyä sovellukseen. Osalla komponenteista voi olla riippuvuuksia toisten komponenttien kanssa. Sovelluskomponentteja on neljää eri tyyppiä, aktiviteetti (activity), palvelu (service), sisällöntarjoaja (content provider) ja tiedotusten vastaanottaja (broadcast receiver). Jokaisella komponentilla on selkeä yksilöllinen rooli ja elinkaari, mikä määrittelee, kuinka komponentti luodaan ja tuhotaan. Androidin erikoispiirre on se, että mikä tahansa sovellus voi käynnistää toisen sovelluksen komponentin. Sovelluskomponentti aktivoidaan lähettämällä sille asynkroninen viesti, aie (intent). Aie sitoo riippumattomat komponentit yhteen ajon ajaksi. Sisällöntarjoaja on ainoa komponenttityyppi, jota ei aktivoida aietta käyttämällä. Sisällöntarjoajan aktivoimiseksi tarvitaan sille lähetetty pyyntö sisällönratkaisijalta (content resolver).[35]

---

[30]https://developer.apple.com/documentation/foundation

[31]https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/Cocoa.html

[32]https://developer.apple.com/library/archive/documentation/MacOSX/Conceptual/OSX_Technology_Overview/MediaLayer/MediaLayer.html

[33]https://developer.apple.com/library/archive/documentation/MacOSX/Conceptual/OSX_Technology_Overview/CoreServicesLayer/CoreServicesLayer.html

[34]https://developer.apple.com/library/archive/documentation/MacOSX/Conceptual/OSX_Technology_Overview/CoreOSLayer/CoreOSLayer.html

[35]https://developer.android.com/guide/components/fundamentals

Aktiviteetti on piste, jonka kautta käyttäjän kanssa kommunikoidaan. Graafinen käyttöliittymä (Graphical User Interface, GUI) koostuu aktiviteeteista. Jokaista sovelluksen näkymää vastaa ainakin yksi aktiviteetti. Sovelluksen aktiviteetit ovat pinossa avaamisjärjestyksessä. Aktiviteetti suljetaan painamalla fyysistä tai ohjelmallista *takaisin* -painiketta, jolloin pinon päällimmäinen aktiviteetti poistetaan ja palataan pinossa seuraavaksi olevaan aktiviteettiin. Aktiviteetit toimivat yhteistyössä, mutta ne ovat riippumattomia toisista aktiviteeteista. Aktiviteetit mm. seuraavat mikä näyttö käyttäjällä on ruudulla avoinna ja varmistavat, että kyseistä näyttöä ylläpitävää prosessia ajetaan. Aktiviteetit myös priorisoivat ja auttavat palauttamaan prosesseja, jotka käyttäjä on pysäyttänyt tai lopettanut ja joihin hän saattaa palata uudelleen[35] [36]. Graafisen käyttöliittymän testit keskittyvät aktiviteetteihin sekä siirtymiin sovellusten eri näkymien välillä [21].

Palvelu on taustalla ajettava komponentti, joka suorittaa pitkäkestoisia tai etänä suoritettavia operaatioita. Palvelut jatkavat taustalla toimintaansa, vaikka kyseisen sovelluksen käyttöliittymä ei enää olisikaan aktiivisena. Taustalla tapahtuvia toimintoja ovat mm. musiikin toistaminen tai tiedostojen lataaminen. Palvelulla ei ole omaa käyttöliittymää.[35]

Sisällöntarjoaja hallinnoi sovelluksen tuottamaa ja tallentamaa tietoa. Muut sovellukset voivat pyytää sisällöntarjoajalta lupaa lukea tai muokata esimerkiksi laitteen muistiin, etäpalvelimelle tai SQLite-tietokantaan varastoitua tietoa. Sisällöntarjoajaa voidaan käyttää myös lukemaan ja kirjoittamaan sovelluksen yksityistä tietoa, jota ei jaeta muiden sovellusten kanssa.[35]

Tiedotusten vastaanottaja on komponentti, joka mahdollistaa tapahtumien välittämisen sovelluksille, myös niille, jotka eivät tapahtumahetkellä ole käynnissä. Tiedotuksen lähettäjä on useimmiten käyttöjärjestelmä, mutta myös sovellukset voivat lähettää tiedotuksia toisille sovelluksille.[35]

3.4. iOS-sovelluksen komponentit

UIKit tarjoaa peruskomponentit, joista iOS-sovellus rakentuu. UIKit:stä löytyy oliot mm. sisällön näyttämiseen ja sen kanssa vuorovaikuttamiseen sekä käyttöjärjestelmän kanssa tapahtuvan vuorovaikutuksen hallitaan.[37]

UIKit:in sisältämien käyttöliittymäelementtien pääkategoriat ovat palkit (bars), näkymät (views) ja kontrollit (controls). Palkkikategorian elementteihin kuuluu mm. tilapalkki, navigointipalkki, hakukenttä, välilehtipalkki ja työkalupalkki. Näkymät sekä sisältävät käyttöliittymän näytöllä esitettävän sisällön, kuten tekstin, grafiikat, animaatiot ja interaktiiviset elementit, että käsittelevät kaikki kosketuseleet. Näkymät mahdollistavat näytöllä tapahtuvat toiminnot, kuten vierittämisen, liittämisen tai poistamisen. Kontrollit toimivat tiedon välittämisessä käyttäjän ja käyttöjärjestelmän välillä. Mm. erilaiset näppäimet, tekstikentät, valikot, liukusäätimet sekä edistymistä kuvaavat elementit ovat kontrolleja.[37]

View Controllerit ovat iOS-sovelluksen sisäisen rakenteen perusta. Jokainen sovellus sisältää vähintään yhden view controllerin, mutta suurimmassa osassa sovelluksia niitä on useampia. Jokainen view controller hallinnoi omaa osaansa käyttöliittymästä sekä käyttöliittymän ja sen alla olevan datan vuorovaikutuksesta. View controllerit myös ohjaavat siirtymiä eri käyttöliittymän osien välillä. View

[36] https://developer.android.com/guide/components/activities/tasks-and-back-stack
[37] https://developer.apple.com/documentation/uikit/about_app_development_with_uikit

controllereita on kahta tyyppiä. Content view controller hoitaa yhtä erillistä osaa sovelluksen sisällöstä ja container view controller kerää tietoa muilta view controllereilta esimerkiksi navigaation ohjaamista varten.[38]

UIKit-sovellusten rakenne perustuu MVC-suunnittelumalliin (Model-View-Controller, malli-näkymä-käsittelijä), jossa oliot on jaettu niiden käyttötarkoituksen mukaan. Malliosaan kuuluvat oliot hallinnoivat sovelluksen dataa ja bisneslogiikkaa, näkymän oliot tarjoavat toiminnot datan visuaaliseen esitykseen, ja käsittelijä siirtää dataa mallin ja näkymän välillä.[37]

3.5. Huawei AppGallery

iOS:n ja Androidin rinnalle on tullut uusi ja merkittävä kolmas ekosysteemi, Android-pohjainen Huawei AppGallery. Maaliskuussa 2018 kiinalainen Huawei julkisti AppGalleryn yhdessä HUAWEI P20 -älypuhelimen kanssa. AppGallery on Huawein virallinen sovellusten jakelualusta, jossa käyttäjä voi etsiä, ladata, hallita ja jakaa mobiilisovelluksia Huawein mobiililaitteille.[39] [40]

AppGallery on saavuttanut nopeasti suosiota. Vuonna 2020 AppGallerylla on maailmanlaajuisesti jo noin 400 miljoonaa aktiivista käyttäjää kuukaudessa, ja se on kolmanneksi suurin sovellusmarkkina maailmassa. AppGalleryn sovellusvalikoima laajenee jatkuvasti, ja sen valikoimiin kuuluu jo yli miljoona sovellusta ympäri maailmaa.[40] [41]

AppGalleryn tietoturva on varmistettu nelikerroksisella turvajärjestelmällä. Jokaisen AppGalleryn sovelluksen on läpäistävä yksityisyysasetusten tarkastus, tietoturvahaavoittuvuuksien skannaaminen, haittaohjelmien etsintä ja manuaalinen sovelluksen kehittäjän oikeaan nimeen perustuva käyttäjän todentaminen. Asiakasta koskevaa sensitiivistä tietoa ei käsitellä Huawein laitteen ulkopuolella, ja kaikki käyttäjädata anonymisoidaan ja säilytetään paikallisesti. AppGallery noudattaa EU:n yleistä tietosuoja-asetusta.[39]

3.6. Pankkisovellusalustan arkkitehtuuri

OP:n mobiilisovelluksen yksinkertaistettu rakenne voidaan kuvata viisikerroksisella mallilla kuvan 11 mukaisesti. Digitaalisten pankkipalveluiden taustalla on massiivinen palveluista ja perusjärjestelmistä koostuva taustajärjestelmä, ja sovelluksen roolina on olla eräänlainen älykäs käyttöliittymä tähän järjestelmään. Taustajärjestelmän ja sovelluksen välissä on palvelusovitinkerros, jonka tehtävänä on muuntaa sovelluksen ja palvelukerroksen tieto toistensa kanssa yhteensopivaksi. Asiakasohjelma on mobiililaitteeseen asennettava natiivisovellus, ja palvelusovitin, liiketoimintasovellukset sekä perusjärjestelmät sijaitsevat palvelimella. Edustatunnistus on keskitetty ratkaisu, joka tarjoaa erilaisia palveluita käyttöoikeuksien hallintaan.

---

[38] https://developer.apple.com/library/archive/featuredarticles/ViewControllerPGforiPhoneOS/index.html

[39] https://www.prnewswire.com/in/news-releases/discover-a-wonderful-life-with-huawei-appgallery-680636951.html

[40] https://www.albawaba.com/business/pr/privacy-under-your-control-huawei-appgallery-1343382

[41] https://consumer.huawei.com/fi/campaign/appgallery-faq/

Kuva 11. Pankkisovelluksen kerrosmalli.

Pankkisovelluksen arkkitehtuurin kerrosmalli muistuttaa hieman OSI-mallia (Open Systems Interconnection), jonka konseptiin kuuluu, että kerrokset keskittyvät tiettyihin palveluihin ja kerrosten välissä on rajapinnat, joiden kautta palveluita käytetään. Puhdasoppista OSI-mallia pankkisovelluksen arkkitehtuuri ei kuitenkaan edusta. [22]

3.7. Pankkisovellusalustan testaaminen

Pankkisovelluksen jokainen kerros ja niiden väliset rajapinnat testataan erikseen, ja jokainen rajapinta tarvitsee oman testausautomaationsa. Kerrosten sisällä suoritettavat automaatiotestit rajaavat toisissa kerroksissa suoritettavia testejä. Esimerkiksi suorituskykytestaus tapahtuu palvelinsovitin- ja palvelukerroksessa, sitä ei suoriteta sovelluskerroksessa. Käyttäjän ja sovelluksen välissä olevan graafisen käyttöliittymän testaus puolestaan sisältää mm. responsiivisuuden ja saavutettavuuden testaamisen, mitä ei tarvitse tehdä muissa kerroksissa. Tietoturvatestaus poikkeaa muista testeistä, sillä se on suoritettava jokaisessa kerroksessa. Kerrosten ja niiden rajapintojen sisällä tapahtuvan testaamisen lisäksi suoritetaan vielä end-to-end-testaus kaikkien kerrosten läpi.

Digitaalisten pankkipalveluiden testaaminen on laaja-alaista. Mobiilisovelluksen käyttämä taustajärjestelmä ja edustatunnistus on käytössä sovelluksen lisäksi myös sekä selaimella käytettävässä verkkopalvelussa että toimihenkilökanavassa, mikä tarkoittaa sitä, että samaa tietoa voidaan tarkastella erikseen jokaisessa kanavassa ja sen on oltava yhdenmukaista kanavasta riippumatta.

Pankkisovelluksen kerrosmallin kahdessa alimmassa kerroksessa on paljon vanhaa legacy-ohjelmistoa, mikä tuo haasteita sekä sovelluksen kehitykseen että testaukseen. Alempien kerrosten testausta on mahdotonta suorittaa ylempää rajapintaa käyttäen, joten sovelluksen graafisen käyttöliittymän kautta ei alempia kerroksia pystytä testaamaan kattavasti.

4. MOBIILISOVELLUKSEN GRAAFISEN KÄYTTÖLIITTYMÄN TESTAAMINEN

Graafisella käyttöliittymällä tarkoitetaan hierarkkista interaktiivisten visuaalisten komponenttien muodostamaa järjestelmää. Visuaalisia komponentteja ovat mm. valikot, napit tai tekstikentät. [23]

Käyttöliittymän testaaminen on erityisen tärkeää, sillä vuorovaikutus käyttäjän ja sovelluksen välillä tapahtuu juuri käyttöliittymän kautta, ja käyttöliittymän toiminta ja ominaisuudet vaikuttavat sovelluksesta saatavaan mielikuvaan. Jotta koko ohjelmisto toimii oikein, on myös käyttöliittymän oltava kunnossa. Yleinen tapa GUI:n vikojen etsintään on suorittaa joko manuaalisia tai automaattisesti GUI:n mallinnuksen pohjalta luotuja testitapauksia. [23]

Erityisesti Android-sovellukset ovat keskimäärin heikosti testattuja. Joko niitä ei ole testattu ollenkaan, tai testit on suoritettu ainoastaan manuaalisesti. Loppukäyttäjiltä saatu palaute on usein manuaalitestauksen ohella ainoa käytetty tapa selvittää sovelluksen laatua. Menetelmä jättää testaukseen paljon aukkoja, eikä testitapaukset ole toistettavissa myöhemmin [24] [25]. Pankkisovellukselle luotettava toiminta ja helppokäyttöisyys on kuitenkin elintärkeää, myös graafisen käyttöliittymän osalta. Testausautomaation käyttö onkin pankkisovelluksen kattavan ja luotettavan testauksen kulmakivi.

Rajallinen akunkesto, muisti ja kaistanleveys, yhteystyypin nopeat muutokset, järjestelmän ja yhteystapahtumien aiheuttamat jatkuvat keskeytykset, tarve sovittaa käyttöliittymä skaalautumaan laajalle laitekannalle, jatkuva moniajo ja vuorovaikutus muiden sovellusten kanssa sekä käyttöjärjestelmän ja sovellusten lähettämien tapahtumien aiheuttamat keskeytykset tekevät sovellusten testaamisesta työlästä. Sovelluksen elinkaari markkinoilla on usein lyhyt, ja sovellus on kehitettävä lyhyessä ajassa, sillä se on saatava nopeasti levitykseen. Tästä johtuen testausautomaation käytöstä luovutaan, sillä automatisoitujen testien tekemisen ja ylläpitämisen katsotaan vievän liian paljon aikaa ja rahaa suhteessa siitä saatavaan hyötyyn. Yhteensopivuusongelmat, testityökalujen monimutkaisuus ja huono dokumentaatio myös osaltaan vähentävät kehittäjien kiinnostusta sovellusten testaamiseen. [21]

4.1. Automatisoidut GUI-testit

Eräs automatisoitujen GUI-testien jaottelu on seuraava: sumea tai satunnainen testaus, mallipohjaiset testausmenetelmät, nauhoita ja toista, sekä white box -skriptitestit. Useimmat näistä menetelmistä sallivat testiskriptien luomisen ja mahdollistavat nopeat ja toistettavat testiajot. [24]

Sumea tai satunnainen testi antaa aktiviteeteille satunnaisia syötesekvenssejä ja tutkii aiheuttavatko ne sovellukseen mahdollisia vikoja tai sen kaatumisen. Käyttöliittymä voidaan ensin mallintaa satunnaistestausta varten. Syötteet käyttöliittymälle annetaan luodun mallin mukaisesti. Menetelmä ei vaadi etukäteistietoa sovelluksesta, ja se on helppo valmistella käyttöön. [24] [25]

Mallipohjaiset testitekniikat hyödyntävät graafisesta käyttöliittymästä muodostettuja malleja, kuten äärellisiä automaatteja tai tapahtumavuokaavioita. Mallit voidaan tehdä manuaalisesti, tai ne voidaan luoda prosessilla, jota kutsutaan *GUI rippingiksi*. Näitä malleja käytetään apuna, kun luodaan testitapaukset graafisen käyttöliittymän systemaattiseen testaamiseen ja läpikäyntiin. [24]

Nauhoita ja toista -testaustyökalut nauhoittavat testaajan suorittaman manuaalisen testin tapahtumat ja luovat niistä uudelleenkäytettävät testisekvenssit. White box -skriptaus tarkoittaa yleensä manuaalisesti kirjoitettua testikoodia, joka sisältää erilaisia käyttöliittymälle annettavia tapahtumasekvenssejä. White box -testiskriptien tekeminen vaatii pääsyn sovelluksen koodeihin. [24]

4.2. GUI ripping

GUI ripping tarkoittaa dynaamista prosessia, jossa ohjelmiston graafinen käyttöliittymä käydään automaattisesti kokonaan läpi avaamalla kaikki sen ikkunat, niiden sisältämät widgetit, sekä widgettien asetukset ja asetuksien arvot [23]. Widgetit ovat pieniä taustalla koko ajan käynnissä olevia sovelluksia, jotka näyttävät sovelluksen oleellisen tiedon laitteen näytöllä. Esimerkiksi kello ja kalenteri ovat widgettejä[42]. GUI ripping-prosessi aloitetaan ohjelmiston ensimmäisestä ikkunasta ja sen jälkeen käyttöliittymän hierarkkisessa rakenteessa edetään alaspäin. Prosessi toistetaan rekursiivisesti kaikille luoduille ikkunoille. Tällä tavalla saatu tieto toimii pohjana automaattisesti luoduille testitapauksille. Graafinen käyttöliittymä siis takaisinmallinnetaan sekä rakenteen että käyttäytymisen osalta suorittamalla kaikki sen sisältämät toiminnot. Ennen testitapausten luomista testaajan on hyvä käydä visuaalisesti läpi GUI rippingin tuottama mallinnus käyttöliittymän rakenteesta ja korjata siinä mahdollisesti ilmenneet viat. Testaaja identifioi mahdolliset puutteet GUI ripperille, minkä jälkeen GUI ripping voidaan suorittaa puuttuville ikkunoille. Kun menetelmän tuottama graafisen käyttöliittymän malli on valmis, sitä käytetään luomaan tarvittavat testitapaukset automaattisesti. [23]

Graafisen käyttöliittymän rakenteen voi esittää GUI-metsänä (GUI forest), jossa jokainen metsän solmu edustaa yhtä ikkunaa sekä ikkunan sisältämiä widgettejä. Mikäli ikkunasta voidaan avata toinen ikkuna, nämäkin hierarkkiset yhteydet esitetään solmujen välillä. GUI-metsä ei yksinään riitä mallintamaan graafista käyttöliittymää, vaan lisäksi tarvitaan käyttöliittymän komponentit ja niiden tapahtumat kuvaava tapahtumavuokaavio sekä komponenttien välisen interaktion esittävä integraatiopuu (integration tree), jotka yhdessä kuvaavat suorituksen aikaista graafisen käyttöliittymän toimintaa. Jokainen ikkuna mallinnetaan sen sisältämien widgettien kokoelmana. Widgeteillä on asetuksia ja asetuksilla on arvot. Ikkunan suorituksen jokaista hetkeä voidaan kuvata sen perusteella mistä widgeteistä se sillä hetkellä koostuu ja mitä arvoja widgettien asetukset sisältävät. Testiä varten graafisesta käyttöliittymästä on tarpeellista löytää pienempiä osakokonaisuuksia (unit), joita voidaan analysoida erillään muista käyttöliittymän toiminnoista. [23]

GUI rippingin suorittamisessa on haasteita. Sovelluksen lähdekoodi ei ole aina saatavilla, jolloin tarvitaan algoritmi, jolla tarvittava tieto pystytään irrottamaan sovelluksen suoritettavasta tiedostosta. Eri ohjelmistoalustat ja toteutukset eivät käytä yhtenäistä GUI-standardia, jolloin tieto täytyy hakea alemman tason toteutuksesta riippuvista käyttöjärjestelmäkutsuista, jotka ovat monesti huonosti dokumentoituja. Jotkin osat toteutuksesta tarjoavat liian vähän tarpeellista tietoa, jotta automaatiotestejä pystytään suorittamaan. Tällöin tarvitaan heuristiikkaa tai ihmisen osallistumista, jotta puuttuvat osat saadaan katettua. Graafinen käyttöliittymä saattaa sisältää niin kutsuttuja mahdottomia polkuja, jotka myös estävät kokonaisvaltaisen testien automatisoinnin. Jokin ikkuna saattaa esimerkiksi avautua vasta salasanan

[42] https://en.wikipedia.org/wiki/Software_widget

syöttämisen jälkeen, eikä GUI ripperillä ole välttämättä pääsyä tarvittaviin salasanoihin. Muita prosesseja ja työkaluja tarvitaan, jotta GUI-malliin voidaan lisätä siitä puuttuvat osat. [23]

GUI rippingiä on sen kehittämisen jälkeen käytetty ja tutkittu laajasti, ja sen ympärille on tehty myöhemmin uusia automaatioprosesseja, testitapausten generointityökaluja sekä muita laajennuksia. Tutkimus- ja kehitystyö jatkuu edelleen. [26]

4.3. GUI-testien hauraus

Mobiilisovellukset kärsivät automaattisten GUI-testien hauraudesta. GUI-testin voi määritellä hauraaksi, jos testiluokat hajoavat tai tarvitsevat päivitystä, mikäli sovelluksen toiminnassa tai sen graafisessa käyttöliittymässä tapahtuu pienikin muutos. Yleisimpiä syitä testien rikkoutumiselle ovat käyttöliittymän tekstin tai grafiikan muutokset tai komponentin id:n vaihtuminen. Komponentteja on voitu myös poistaa tai sijoittaa eri paikkaan kuin aikaisemmin. Käyttöliittymän sivupohjan muutokset aiheuttavat ongelmia erityisesti silloin kun testausautomaatio perustuu hahmontunnistukseen. Myös muutokset tapahtumavuossa ja tapahtumien suoritusajassa saattavat aiheuttaa testien epäonnistumisen. [21]

Hauraus on merkittävä ongelma, sillä hylätty testi voi aiheuttaa laajan selvitystyön, jotta sen alkuperä löydetään. Usein iso joukko testitapauksia vaatii korjaamista, ja se on työlästä. Tämä johtaa joskus siihen, että automaatiotestien suorittamisesta luovutaan kokonaan. [21]

4.4. Testausympäristöt

Mobiilisovelluksia voi testata useissa erityyppisissä ympäristöissä. Neljä suosittua lähestymistapaa testaukseen on käyttää testihenkilöistä koostuvaa joukkoa, emulaattoreita, fyysisiä laitteita tai pilvipalveluita. Jokaisessa testausympäristössä on sekä etuja että haasteita [27]. Testausympäristöt soveltuvat GUI-testauksen lisäksi sovelluksen kokonaisvaltaiseen testaamiseen. Testausympäristön valinta riippuu mm. käytettävissä olevasta budjetista, testausresurssien määrästä sekä sovelluksen tyypistä, käyttötarkoituksesta ja kompleksisuudesta.

Joukkoon perustuva testaus (crowd-based testing, crowd testing) tarkoittaa sitä, että testiä varten hankitaan joukko loppukäyttäjiä edustavia testihenkilöitä, jotka käyttävät testattavaa ohjelmistoa ja raportoivat havaitsemistaan vioista ja ongelmista. Menetelmän etuna on se, että sovelluksen tekijän ei tarvitse investoida testaustiloihin eikä testilaitteistoon. Ryhmässä on usein edustettuna useita käyttöjärjestelmiä eri versioineen, erilaisia laitteita sekä sijainteja ja verkkoyhteyksiä, jolloin testitulosten kattavuus paranee. Iso joukko voi testata sovellusta yhtä aikaa, mikä nopeuttaa testitulosten saamista. Huonona puolena on se, että testitulosten laatu voi vaihdella paljonkin, sillä ryhmä ei koostu testauksen ammattilaisista, ja testien aikataulu saattaa venyä. Luottamuksellisuus täytyy varmistaa silloin kun testauksessa käytetään ulkopuolisia resursseja [27]. Pankkisovelluksen kriittisen luonteen johdosta ulkopuolisen joukon käyttäminen sovelluksen testaamiseen onnistuu vain tuotannossa olevan version osalta, ja sovellus on siihen mennessä testattu huolellisesti testausammattilaisten toimesta. Myös pieni ryhmä loppukäyttäjiä on voitu kutsua tekemään käytettävyystestejä ennen sovelluksen julkaisua.

Emulaatiopohjaisessa testauksessa emulaattori luo mobiililaitteen virtuaalikoneversion tietokoneelle. Emulaattori sisältyy usein ohjelmistoalustan mukana tulevaan ohjelmistotyökalupakettiin (Software Development Kit, SDK). Emulaattorilla testaaminen on edullista verrattuna fyysisen testilaitteiston ylläpitämiseen tai vuokraamiseen, mutta siinä on useita rajoituksia. Emulaattorit eivät tue esimerkiksi kaikkia laitteita, laitteiden kaikkia ominaisuuksia tai kosketuseleitä, tai palveluiden, kuten verkkoyhteyksien, testaamista. Emulaattorit eivät siis koskaan täysin vastaa fyysistä laitteistoa, ja niiden käytön yhteydessä on arvioitava riittääkö emulaattorin käyttö täyttämään sovelluksen kaikki testaustarpeet [27]. Pankkisovellusta voi kehitysvaiheessa testata emulaattorilla, mutta sen lisäksi tarvitaan täydentävää testausta fyysisillä laitteilla joko omissa tiloissa tai pilviympäristössä.

Laitepohjainen testaus vaatii omaan fyysiseen laitteistoon investoimista ja sen ylläpitämistä. Laitteiston hankkiminen on kallista ja sen ylläpitäminen on työlästä, mutta sillä voi testata kaikki laitteistosta riippuvat toiminnot sekä verkkoyhteydet ja muut palvelut. Markkinoilla oleva laitekanta päivittyy kuitenkin hyvin nopeasti, joten testilaitteiston pitäminen ajan tasalla on merkittävä haaste ja kustannuserä. Näistä ongelmista johtuen oma testilaitteisto onkin yleensä selvästi suppeampi verrattuna emulaattori- ja pilvilaitevalikoimaan. [27]

Pilvipalvelun laitepilvi (device cloud) on mobiilitestausympäristö, jossa palveluntarjoajan ylläpitämiä fyysisiä laitteita voi käyttää verkon yli manuaali- tai automaatiotestien suorittamiseen. Laitepilven kautta on mahdollista saada käyttöön laaja valikoima eri laitteita ja käyttöjärjestelmiä sekä selaimia eri versioineen. Osa pilvipalveluista tarjoaa fyysisten laitteiden lisäksi myös emulaattoreita ja simulaattoreita[43] [44]. Emulaattorin ja simulaattorin välinen ero on se, että emulaattori jäljittelee sekä laitteiston että ohjelmiston toimintaa, simulaattori jäljittelee vain ohjelmistoa. Jotkin palveluntarjoajat mahdollistavat myös oman yksityisen pilven, hybridipilven tai paikallisen pilven muodostamisen halutulla laitekokoonpanolla. Testit voi ajaa useassa laitteessa yhtä aikaa. Pilvipalvelut tukevat mm. useita testaustyökalukehyksiä ja CI/CD-integraatiota (Continuous Integration/Continuous Delivery), ja tarjoavat esimerkiksi erilaisia testausraportteja, lokeja, kuvankaappauksia ja videoita sovelluksen testiajoista.[43] [44]

Laitepilven käyttö on maksullista, ja laitevalikoima sekä uusien laitteiden saaminen valikoimaan ja käyttöjärjestelmäversioiden päivitysnopeus vaihtelee eri palveluntarjoajien välillä. Laitteita saattaa joutua jonottamaan ja testien ajaminen pilvessä on hitaampaa verrattuna omissa laitteissa ajamiseen. Verkkoyhteysongelmat saattavat toisinaan keskeyttää testiajojen suorittamisen.

4.5. Pankkisovelluksen GUI-testauksen haasteet

Yleiset graafisen käyttöliittymän ongelmat koskevat myös pankkisovellusta, mutta niiden lisäksi sillä on useita erityispiirteitä, jotka tuovat testaukseen lisähaasteita. Tietoturva on erittäin tärkeää ja on varmistettava, ettei käyttöliittymän kautta ole mahdollista päästä käsiksi pankkijärjestelmään. Kaikki käyttöliittymältä eteenpäin lähtevä tieto on oltava kryptattua, ja on tarkasti määritelty missä muodossa mikäkin tieto on oltava. Mikään käyttäjää koskeva tieto ei saa jäädä laitteeseen, eli esimerkiksi

[43] https://kobiton.com/
[44] https://saucelabs.com/

sormenjälkeä ei tallenneta tietokantaan. Laitteen kadottaminen ei myöskään saa vaarantaa käyttäjän tietoturvaa ja mahdollistaa sovelluksen väärinkäyttöä.

Sovelluksen käyttö vaatii vahvan tunnistautumisen, esimerkiksi avainlukulistaa tai mobiiliavainta käyttämällä, ja tunnistautumistapa saattaa vaihtua kesken asioinnin, kun siirrytään vaikka sovelluksen ja verkkokaupan välillä. Osa sovelluksessa aloitetuista ostopoluista vie selaimessa olevaan verkkopankkiin, jolloin ostopolkujen täytyy olla saumattomia, eikä mikään tieto saa kadota sovelluksen ja selaimen välisessä siirtymässä.

Osa käyttöjärjestelmän käyttöliittymäkomponenteista on korvattu itse tehdyillä komponenteilla ja päinvastoin, mistä aiheutuu ongelmia. Komponenttien version vaihtuessa täytyy kartoittaa, kuinka laaja testaus on tarpeen. Kun uusi alustaversio julkaistaan, täytyy kaikki testit suorittaa, vaikka koodi ei olisi muuttunut edellisellä alustaversiolla suoritettujen testien jälkeen.

Pankkisovelluksen rakenne ei ole puumainen hierarkia, vaan verkko, jossa on silmukoita. Automaatiotestejä rakennettaessa täytyy kiinnittää huomiota, ettei testin suoritus jää ikuiseen silmukkaan. Sovelluksessa on myös oikopolkuja näkymien välillä, mikä tuo haastetta testausautomaatiolle. Erityisesti saavutettavassa versiossa on mahdollista päätyä samalta näyttävään, mutta koodiltaan erilaiseen näkymään riippuen siitä onko siirtyminen tapahtunut polkua pitkin vai suoran linkin kautta.

4.6. Pankkisovelluksen testausautomaation vaatimukset

Testausautomaatioratkaisun valintaan vaikuttavat monet seikat. Mobiililaitteiden ja niiden käyttöjärjestelmien kehityssykli on nopea, millä on merkittävä vaikutus myös mobiilisovellusten kehitykseen ja testaukseen. Uusi käyttöjärjestelmäversio pitäisi saada käyttöön nopeasti, ja toisaalta vanhimmat versiot olisi hyvä saada pois asiakkaiden käytöstä, sillä niiden tukeminen on työlästä ja hankalaa.

Sovelluksen graafiseen käyttöliittymään saattaa tulla muutoksia usein, ja pienetkin muutokset voivat rikkoa automaatiotestit. Korjaukset testeihin on tehtävä nopealla aikataululla, ja se vaatii työkalulta joustavuutta. Koska kyseessä on natiivisovelluksen käyttöliittymän testausautomaation kehittäminen, täytyy testausratkaisun soveltua käyttöliittymän responsiivisuuden testaamiseen ja tukea sekä Android- että iOS-sovellusta. Tärkeää on, että työkalu on helppo ottaa käyttöön sen ohjelmistoalustan kanssa, jolle sovellusta kehitetään ja testausautomaatiota rakennetaan. Testausautomaatiota kehitetään sekä Windows- että Mac -ympäristöissä, joten työkalun tulisi tukea molempia käyttöjärjestelmiä.

Työkalun oppimissyklin tulisi olla nopea ja helppo, sillä muutoin työkalun käyttöönotto ja konfigurointi sekä työntekijöiden koulutus sen käyttöön vie liikaa aikaa ja resursseja. Testausautomaation tekemisen pitäisi olla vaivatonta ja nopeaa. Työkalun on tuettava menetelmiä, joilla riittävä testikattavuus on mahdollista luoda.

Testien suorittamisen tulisi olla nopeaa ja stabiilia, ja suorituksesta olisi hyvä saada erilaisia raportteja, kuten kuvankaappauksia, videoita tai lokeja. Testien ajaminen rinnakkain useassa laitteessa tai emulaattorissa on etu, sillä se lyhentää testien suoritusaikaa. Testausratkaisun tulee tarjota riittävän kattava valikoima emulaattoreita ja fyysisiä mobiililaitteita, joissa testiajot voidaan suorittaa.

Testausautomaatio voidaan toteuttaa usealla eri kielellä. Eri työkalut tukevat eri kieliä, ja automaation käyttöönottoa helpottaa ja nopeuttaa jos kieli on entuudestaan tuttu testien tekijälle. On mahdollista, että aikaisemmin on jo toteutettu testausautomaatiota, jolloin täytyy harkita, olisiko syytä valita niiden kanssa

yhteensopiva ratkaisu, vai onko työkalun ja kielen vaihtaminen perusteltua, ja kuinka iso työmäärä vaaditaan, jotta olemassa olevat automaatiotestit konvertoidaan uuden testausratkaisun kanssa yhteensopiviksi.

Pilvipohjaisen mobiilitestausalustan käyttöönottoa punnitessa täytyy ottaa huomioon muiden ominaisuuksien lisäksi myös palvelun tarjoama laitevalikoima. Pankkisovellusta, jolle tässä työssä testausautomaatioratkaisua haetaan, testataan tietyllä laitekokoonpanolla, joka sisältää suosituimpia ja uusimpia laitemalleja sekä käyttöjärjestelmäversioita. Jotta paikallinen testilaboratorio voidaan korvata pilvipohjaisella ratkaisulla, täytyy pilvipalvelun laite- ja käyttöjärjestelmävalikoiman olla riittävän kattava.

Pankkisovelluksen kehityksessä on käytössä ketterät menetelmät sekä DevOps-toimintamalli, joten testausautomaatiotyökalujen tulisi olla yhteensopivia ohjelmistokehityksessä käytettävien DevOps-työkalujen, kuten Jenkinsin, kanssa. Myös työkalujen ylläpito ja tuki on yksi valintakriteeri. Pilvipalveluiden osalta niiden fyysinen sijainti saattaa vaikuttaa tuen saatavuuteen. Valituista työkaluista ja testausratkaisusta riippumatta kaikkea pankkisovelluksen testausta ei pystytä automatisoimaan, ja manuaalista testausta tarvitaan jonkin verran täydentämään automaatiotestausta. Ihmiset myös priorisoivat testauksessa käytetyt laitteet ja ylläpitävät omaa testilaitelaboratoriota. Saavutettavuuden kattava testaaminen on haastavaa pelkkää automaatiota käyttämällä, esimerkiksi käyttöliittymän testaaminen ruudunlukijalla on paras tehdä manuaalisesti.

Testausratkaisun valintaan vaikuttaa myös siitä aiheutuvat kustannukset. Maksullinen työkalu voidaan valita, mikäli se tuo jotakin tarpeellista lisäarvoa maksuttomiin työkaluihin verrattuna.

5. MOBIILISOVELLUKSEN GRAAFISEN KÄYTTÖLIITTYMÄN TESTAUSTYÖKALUT

Ratkaisu graafisen käyttöliittymän testausautomaatio-ongelmaan voi koostua yhdestä tai useammasta työkalusta. Tarjolla on sekä yhdelle alustalle suunnattuja että monialustayhteensopivia työkalukehyksiä, joilla testausautomaatio voidaan toteuttaa. Näiden lisäksi ratkaisua voidaan laajentaa vielä pilvipohjaisella mobiilitestausalustalla, jolloin automaatiotestit voidaan ajaa esimerkiksi testausalustan tarjoamissa emulaattoreissa, simulaattoreissa tai oikeissa mobiililaitteissa.

Tässä tutkielmassa työkalujen vertailu on jaoteltu neljään osaan, joita ovat Androidille soveltuvat työkalukehykset, iOS:lle soveltuvat työkalukehykset, monialustayhteensopivat työkalukehykset sekä pilvipohjaiset mobiilitestausalustat.

On yleistä, että ohjelmointikielillä käytettäviä työkalukehyksiä muokataan esimerkiksi automaatiotestien erilaisten suunnittelumallien (Design Pattern), kuten Page Object Patternin tai BDD:n mukaisesti. Työkalukehystä täydennetään ja uusia luokkia luodaan vastaamaan testausautomaation tarpeita. Suunnittelumallit ovat alustariippumattomia, joten on mahdollista rakentaa samanlainen testausautomaatio sekä Androidille että iOS:lle, edellyttäen että käytetyt ohjelmointikielet soveltuvat valittuun suunnittelumalliin.

5.1. Automaatiotestaustyökalukehykset Android-sovelluksille

Androidille suunnattujen työkalukehysten vertailussa on mukana kuusi testausautomaatiotyökalukehystä, joita ovat Espresso, Robotium, MonkeyRunner, UI Automator, Selendroid ja Monkey, ja joista suurin osa tulee Android SDK:n mukana. Työkalukehysten tärkeimmät ominaisuudet tämän työn tutkimusongelman kannalta on esitetty taulukoissa 3a, 3b ja 3c.

Taulukko 3a. Android-sovelluksen graafisen käyttöliittymän testaustyökalut

Työkalu	**Saatavuus**	**Testien ajaminen**	**Testien luominen**
Espresso	SDK:n mukana	Emulaattorit, fyysiset laitteet, pilvilaitteet	Nauhoitus, minkä jälkeen automaattinen testiskriptin luonti. Testiskriptin voi myös kirjoittaa itse.
Robotium	Maksuton avoimen lähdekoodin ohjelmisto	Emulaattorit, fyysiset laitteet	Testiskriptin kirjoittaminen.
MonkeyRunner	Android SDK:n mukana	Emulaattorit, fyysiset laitteet	Testiskriptin kirjoittaminen.
UI Automator	Android SDK:n mukana uiautomator Java-kirjasto	Emulaattorit, fyysiset laitteet	Testiskriptin kirjoittaminen.
Selendroid	Maksuton avoimen lähdekoodin ohjelmisto	Emulaattorit, fyysiset laitteet, pilvilaitteet	Selenium 2 client API:n kautta, testiskriptin kirjoittaminen, klikkausten nauhoitus.
Monkey	SDK:n mukana	Emulaattorit, fyysiset laitteet	Komentorivikomennoilla.

Taulukko 3b. Android-sovelluksen graafisen käyttöliittymän testaustyökalut

Työkalu	**Ohjelmointikielet**	**Testin tyyppi**	**Muuta**
Espresso	Java, Kotlin	White box	Googlen virallinen testityökalukehys Android-sovellusten UI:n funktionaalista testausta varten. Hyvä testiskriptin suoritusnopeus.
Robotium	Java	Grey box/black box	Automaattiset ajoitukset ja viivästykset. Ei tue Flash- tai Web-komponentteja. Käsittelee vain yhtä sovellusta kerrallaan.
MonkeyRunner	Python	Black box	Jokaiselle laitteelle täytyy kirjoittaa oma testiskripti. Testit on päivitettävä joka kerta kun UI muuttuu.
UI Automator	Java, Kotlin	Black box	Vain natiivisovelluksille, ei tue WebViewiä.
Selendroid	WebDriver-yhteensopivat kielet, kuten Java, Ruby, Python, Perl, C#	Black box/ white box	Testit voi ajaa yhtä aikaa useissa laitteissa/emulaattoreissa. iOS-sovelluksia varten on oma ajuri. Tukee Android-versioita API 10:sta alkaen.
Monkey	-	Black box	Stressitestaukseen. Generoi ja lähettää järjestelmälle tapahtumia pseudo-satunnaisessa järjestyksessä.

Taulukko 3c. Android-sovelluksen graafisen käyttöliittymän testaustyökalut

Työkalu	**Soveltuu automaatiotestaukseen**	**Soveltuu responsiivisuuden testaukseen**	**Soveltuu pankkisovelluksen testaukseen**
Espresso	Kyllä	Kyllä	Kyllä
Robotium	Kyllä	Kyllä	Kyllä
MonkeyRunner	Kyllä	Kyllä	Kyllä
UI Automator	Kyllä	Kyllä	Kyllä
Selendroid	Kyllä	Kyllä	Kyllä
Monkey	Rajoitetusti	Rajoitetusti	Ei

5.1.1. Espresso

Espresso on Googlen luoma natiivi työkalukehys Android-sovellusten käyttöliittymän testausautomaation tekemiseen. Espresso on osa Androidin SDK:ta, ja se on integroitu Android Studioon, joka on Androidin natiivi kehitysympäristö. Tästä johtuen Espresson käyttöönotto on vaivatonta ja nopeaa. Muita Espresson etuja ovat sen stabiilius ja testiskriptien hyvä suoritusnopeus. Työkalukehys synkronoi testit automaattisesti sovelluksen käyttöliittymän kanssa. Se suorittaa testit vasta siinä vaiheessa, kun käyttöliittymän elementit ovat näkyvissä ja taustan tapahtumat on saatu päätökseen, mistä johtuen ennenaikaisesti ajettujen testien aiheuttamia virheellisiä testituloksia on vähemmän. Automaatiotestit voi suorittaa joko oikeissa laitteissa tai Android Studion emulaattorissa. [45] [46] [47] [48]

Espresso sisältää yksinkertaisen kolmesta komponentista koostuvan ja kustomoitavan API-rajapinnan, mikä tekee automaatiotestien tekemisestä ja ylläpitämisestä helppoa [28]. Espresso Test Recorderilla voi nauhoittaa

manuaalitestejä ja työkalu luo nauhoituksesta suorituskelpoista Java-lähdekoodia automaatiotestien pohjaksi. Espresso on sovelluksen sisällä ja sillä on pääsy sovellusta ajavaan koodiin, mikä mahdollistaa elementtien perusteellisen testaamisen. Espressolla voi testata vain yhtä sovellusta kerrallaan.[45] [46] [47] [48]

Graafisen käyttöliittymän automaatiotestaustyökalukehyksiä voi käyttää myös sovelluksen energiankulutuksen tutkimiseen. Työkalukehyksellä simuloidaan käyttäjän suorittamia kosketuseleitä samalla kun sovelluksen kuluttamaa energiaa mitataan siihen soveltuvilla työkaluilla. Työkalukehyksen suorittamat toiminnot kosketuseleiden simuloinnissa kuitenkin kuluttavat enemmän energiaa ihmisen suorittamiin kosketuseleisiin verrattuna, ja työkalukehysten välillä on energiatehokkuudessa suuria eroja. Eri kosketuseleiden vaatima energia vaihtelee, ja toiset työkalukehykset suoriutuvat toisista kosketuseleistä tehokkaammin kuin toisista. Kaikki kosketuseleet huomioiden Espresson energiajalanjälki on pieni, ja se soveltuu hyvin sovelluksen energiankulutuksen testaamiseen. [29]

Espresson rajoitteena voi pitää sitä, että se tukee ohjelmistokielistä vain Javaa ja Kotlinia. Lisäksi työkalukehys soveltuu vain Android-sovellusten käyttöliittymän testaamiseen, joten toinen työkalukehys tarvitaan iOS-sovelluksen käyttöliittymän automaatiotestien luomista varten.[45] [46] [47] [48]

5.1.2. Robotium

Robotium on avoimen lähdekoodin työkalukehys, joka soveltuu Androidin natiivi- ja hybridisovellusten käyttöliittymän automaatiotestien luomiseen. Työkalukehys tukee myös toiminnallisten testien sekä järjestelmä- ja hyväksyntätestien tekemistä. Robotium käsittelee vain yhtä sovellusta kerrallaan. Robotium on Androidin testityökalukehyksen lisäosa, ja sen saa käyttöön lataamalla Robotium.jar-tiedoston ja tallentamalla sen projektin libs-kansioon.[49] [50] [51]

Robotium-testien luomista varten tarvitaan joko sovelluksen lähdekoodi tai APK-tiedosto (Android Application Package). Testit ovat black box tai grey box -testejä, eikä niiden tekeminen vaadi lähdekoodin tarkempaa tuntemista tai Android-sovellusalustan muokkaamista. Testien suorittaminen on nopeaa, työkalukehys on helppokäyttöinen ja testien luettavuus on hyvä. Robotium käyttää testien suorittamisen aikana graafisen käyttöliittymän komponenttien sidontaa ja ajoittaa automaattisesti testien suoritusta, mikä tekee testeistä robusteja. Automaatiotesteillä voi simuloida käyttäjän kaikkia kosketuseleitä, kuten klikkauksia tai tekstinsyöttöä. Testit voi suorittaa joko emulaattorissa tai oikeissa laitteissa.[49] [52]

Robotiumilla on muutamia heikkouksia. Testattavan sovelluksen lisäksi täytyy tehdä erillinen testisovellus, mikä aiheuttaa lisätyötä ja mahdollistaa sen, että varsinaiseen sovellukseen päätyy ominaisuuksia ja virheitä, jotka eivät ole testisovelluksessa mukana, ja siten jäävät testauksen ulkopuolelle. Robotium tukee vain Javalla kirjoitettuja testejä, minkä monet automaatiotestien tekijät kokevat

[45] https://developer.android.com/training/testing/espresso
[46] https://bitbar.com/blog/appium-vs-espresso-which-framework-to-use-for-automated-android-testing/
[47] https://saucelabs.com/blog/appium-vs-espresso
[48] https://bitbar.com/blog/android-espresso-tutorial-for-mobile-app-testing/
[49] https://saucelabs.com/resources/articles/open-source-tools-robotium-android-appium
[50] https://github.com/robotiumtech/robotium
[51] http://www.testingtoolsguide.net/tools/robotium/

rajoittavana. Robotium ei pysty käsittelemään Flash- tai Web-komponentteja eikä vuorovaikuttamaan Status Bar -ilmoitusten kanssa. Robotium ei tue testien nauhoittamista, mutta sen pohjalta on tehty Robotium Recorder, joka on Eclipsen ja Android Studion liitännäinen. Robotium Recorder ei ole avoimen lähdekoodin työkalu, vaan se tarvitsee toimiakseen Eclipsen ja Android Studion maksullisen lisenssin. Espresson tavoin Robotium tukee vain Android-sovelluksia, eikä se käy iOS-sovellusten automaatiotestaukseen.[52] [53]

5.1.3. Monkeyrunner

Monkeyrunner on Android SDK:n osa, joka tarjoaa API-rajapinnan Android-laitteen tai emulaattorin ohjaamiseen tarkoitetun ohjelman kirjoittamiseen. Monkeyrunner toimii sovelluksen ulkopuolella, eikä sen lähdekoodia tarvita testausautomaation luomiseen. Monkeyrunnerin avulla voi kirjoittaa Python-ohjelman, joka esimerkiksi asentaa Android-sovelluksen tai testipaketin, suorittaa sen, lähettää sille kosketuseleitä, ottaa kuvankaappauksia sen käyttöliittymästä ja tallentaa ne työasemalle. Sovelluksen ohjaamista varten on mahdollista kehittää kokonainen järjestelmä moduuleista ja ohjelmista, jotka perustuvat Pythoniin. API-rajapintaa voi laajentaa Java-ohjelmointikielellä kirjoitetuilla luokilla, jotka käännetään JAR-paketeiksi (Java ARchive). Monkeyrunner käyttää Jythonia, joka on Javalla tehty toteutus Pythonista.[54]

Monkeyrunner on ensisijaisesti tarkoitettu sovelluksen syvemmän toiminnallisen ja työkalukehystason testaamiseen sekä yksikkötestien suorittamiseen, mutta se soveltuu muuhunkin, kuten käyttöliittymän testaamiseen. Testit ovat black box -testejä, eli sovelluksen lähdekoodin tuntemusta ei tarvita. Työkalua voi käyttää myös regressiotestaukseen, sillä se voi verrata ottamiaan kuvankaappauksia oikeiksi tiedettyihin referenssikuviin. Monkeyrunnerin API-rajapinta mahdollistaa testien suorittamisen useissa laitteissa ja emulaattoreissa yhtä aikaa[54]. Monkeyrunner sopii myös sovelluksen energiankulutuksen testaamiseen, vaikka sen energiajalanjälki kaikki kosketuseleet huomioiden onkin Espressoa suurempi [29].

Monkeyrunnerin vajavuutena on se, että jokaiselle laitteelle täytyy kirjoittaa oma testiskripti, ja automaatiotestit vaativat hienosäätöä joka kerta kun sovelluksessa tai sen käyttöliittymässä tapahtuu muutoksia[55].

5.1.4. UI Automator

UI Automator on Android SDK:n mukana tuleva käyttöliittymätestaustyökalukehys, joka soveltuu funktionaaliseen monisovellustestaukseen sekä käyttäjän itse asentamille sovelluksille että järjestelmäsovelluksille. UI Automator tarjoaa joukon API-rajapintoja, joiden kautta testit pystyvät vuorovaikuttamaan sovellusten komponenttien ja laitteen fyysisten näppäinten kanssa. Työkalukehys soveltuu black box -testien kirjoittamiseen, sillä testikoodi perustuu ainoastaan käyttöliittymän

[52] https://www.tutorialspoint.com/mobile_testing/mobile_testing_robotium_framework.htm
[53] https://saucelabs.com/blog/things-robotium-can-do-that-selendroid-cant
[54] https://developer.android.com/studio/test/monkeyrunner
[55] https://www.guru99.com/mobile-testing-tools.html

elementteihin, eikä se ole riippuvaista sovelluksen sisäisestä toteutuksesta ja rakenteesta.[56] [57]

UI Automator sisältää Ui Automator viewerin, joka tarjoaa graafisen käyttöliittymän Android-laitteen näytöllä näkyvien komponenttien skannaamiseen ja analysointiin. Työkalua voi käyttää näkymän hierarkian tutkimiseen ja näkyvillä olevien käyttöliittymäkomponenttien ominaisuuksien tarkasteluun. Työkalukehys tarjoaa luokat, jonka avulla voidaan mm. kommunikoida sovellusta ajavan laitteen kanssa, etsiä käyttöliittymän elementtejä näytöltä, valita useita elementtejä yhtä aikaa tai vierittää näyttöä. Ui Automator -testit voi suorittaa joko emulaattorissa tai oikeissa laitteissa. [28] [56] [58] [59]

Energiankulutuksessa Ui Automator päihittää joidenkin kosketuseleiden osalta Espresson, vaikka kaikkien kosketuseleiden yhteenlaskettu energian kokonaiskulutus on Espressoon verrattuna suurempi. Ui Automator onkin varteenotettava vaihtoehto, mikäli sovelluksen energiankulutusta halutaan testata. [29]

UI Automator tukee vain Java- ja Kotlin-ohjelmointikieliä. Testien suoritusnopeus on hitaampaa verrattuna Espressoon[57]. WebView on näkymä, joka näyttää web-sivut sovelluksen sisällä[60]. Ui Automator ei tue WebView-komponentteja, joten se soveltuu vain Androidin natiivisovellusten testaamiseen. Työkalukehys vaatii toimiakseen Android-version 4.3 (API-level 18) tai uudemman.[57]

5.1.5. Selendroid

Selendroid on Seleniumiin pohjautuva avoimen lähdekoodin työkalukehys natiivi-, hybridi- ja web-sovellusten käyttöliittymän testaamiseen. Testit kirjoitetaan Selenium 2 client API-rajapintaa käyttämällä, eikä sovelluksen koodia tarvitse muokata. Sovelluksen APK-tiedosto täytyy olla asennettuna sille koneelle, johon Selendroid-palvelin käynnistetään. Testit voi kirjoittaa kaikilla kielillä, joihin Selenium tarjoaa oman kirjaston. Näitä ohjelmointikieliä ovat esimerkiksi Ruby, Python ja Java. Selendroid on täysin yhteensopiva JSON wire protokollan kanssa.[61] [63]

Selendroidin neljä pääkomponenttia ovat Web Driver Client, Selendroid server, Android Driver-App ja Selendroid-standalone. Web Driver Client on Seleniumiin perustuva Java client -kirjasto, joka asennetaan koneelle, jossa testausautomaatiota kehitetään. Se lähettää HTTP- ja WebView -kutsut Selendroid-standalone-palvelimelle. Selendroid server on palvelin, jota ajetaan testattavan sovelluksen kanssa Android-laitteessa. Android Driver-App on sisäänrakennettu Android WebView -ajuri web-sovelluksen testaamiseen. Selendroid-standalone-palvelin vastaa Android-laitteiden hallinnasta ja APK-tiedostoista sekä Selendroid serverin ja testattavan sovelluksen asentamisesta. HTTP-kutsut mobiililaitteeseen asennetulle sovellukselle lähetetään Selendroid-standalone-palvelimelta.[62]

Selendroid tarjoaa uuden *Advanced User Interactions* API-rajapinnan kattavampaan käyttäjän kosketuseleiden hallintaan. Tuettuja eleitä on mm. raahaaminen, useiden komponenttien valitseminen ja pyyhkäisy. Selendroid voi

[56] https://developer.android.com/training/testing/ui-automator
[57] https://saucelabs.com/blog/the-top-5-android-ui-frameworks-for-automated-testing
[58] https://www.vogella.com/tutorials/AndroidTestingUIAutomator/article.html
[59] https://developer.android.com/training/testing/ui-testing/uiautomator-testing
[60] https://www.tutorialspoint.com/android/android_webview_layout.htm
[61] http://selendroid.io/quickStart.html
[62] https://www.softwaretestinghelp.com/selendroid-tutorial-1/

kommunikoida useiden emulaattoreiden ja oikeiden laitteiden kanssa, ja sen voi integroida solmuksi Selenium Gridiin, jolloin testejä voi ajaa rinnakkain useissa laitteissa. Laitteet voi kytkeä tai poistaa testien suorituksen aikana testiajoa keskeyttämättä. Selendroid-testit voi ajaa vain yhdelle sovellukselle kerrallaan. Selendroid tukee myös Androidin vanhempia versioita API 10:sta alkaen.[63] [64]

Selendroidin mukana tuleva Selendroid Inspector on pieni web-sovellus, joka on sulautettu osaksi Selendroid-palvelinta. Selendroid Inspector mahdollistaa sovelluksen käyttöliittymän kulloisenkin tilan ja elementtien tutkimisen. Sen avulla pääsee mm. näkemään näkymän hierarkian ja käyttöliittymän elementtien ominaisuudet, ottamaan kuvankaappauksia ja nauhoittamaan klikkauksia.[65]

Selendroidin huonoina puolina on testien hitaus, ja se on käyttökelvoton laitteissa, joissa on alle neljä gigatavua RAM-muistia (Random Access Memory). Se ei mahdollista sovelluksen ulkopuolisten laitteiden tai sovellusten, kuten kameran tai karttojen, käyttöä. Selendroid soveltuu vain Android-sovellusten testaamiseen, mutta erillinen iOS-ajuri on saatavilla myös iOS-sovellusten testaamista varten.[63] [66] [67]

5.1.6. Monkey

Monkey on Android SDK:n mukana tuleva komentorivityökalu, jota voi ajaa emulaattorissa tai oikeassa laitteessa. Monkey lähettää sovellukselle kirjoitetun testiskriptin sijaan pseudo-satunnaisen jonon sekä käyttäjän toimintoja simuloivia syötteitä että järjestelmätason tapahtumia, mikä toimii satunnaisen kaltaisena, mutta toistettavana stressitestinä.[68]

Tapahtumien generoimisen ja lähettämisen lisäksi Monkey tarkkailee testattavaa järjestelmää. Se estää navigoinnin muihin paketteihin, mikäli se on rajoitettu asetuksissa. Mikäli sovellus kaatuu, vastaanottaa käsittelemättömän virhesanoman tai ei vastaa, Monkey pysäyttää testien suorittamisen ja raportoi virheen.[68]

Monkey pitää sisällään useita eri asetuksia, jotka jakautuvat neljään pääkategoriaan. Kategorioita ovat perusasetukset, kuten tapahtumien lukumäärä, toiminnalliset asetukset, kuten testien rajoittaminen yhteen pakettiin, tapahtumien tyypit ja esiintymistiheys, sekä debuggausasetukset.[68]

Monkey-testien heikkoutena on niiden satunnaisuus. Sovelluksen virheet voi jäädä löytymättä, tai niiden löytyminen on hidasta järjestelmällisten testien puuttuessa. Satunnaisesti löydettyjen bugien alkuperän selvittäminen voi olla hidasta ja työlästä. Monkey-testit poikkeavat muista automaatiotesteistä, eivätkä ne riitä yksinään sovelluksen käyttöliittymän kokonaisvaltaiseen ja järjestelmälliseen testaamiseen.[69]

5.2. Automaatiotestaustyökalukehykset iOS-sovelluksille

iOS-sovellusten graafisen käyttöliittymän testaukseen soveltuvien työkalukehysten vertalussa on mukana kolme työkalukehystä, XCUITest, Earl Grey ja KIF. Frank on

---

[63] http://selendroid.io/
[64] https://saucelabs.com/blog/selendroid-vs-appium-which-to-choose-for-your-mobile-app-testing
[65] http://selendroid.io/inspector.html
[66] https://www.professionalqa.com/what-is-selendroid-framework
[67] https://whataftercollege.com/uncategorized/mobile-testing-tools/
[68] https://developer.android.com/studio/test/monkey
[69] https://medium.com/skyshidigital/monkey-testing-ade9ca69205c

myös eräs iOS-sovellusten testaukseen soveltuva työkalukehys, mutta koska uusimmat löydetyt päivitykset ja tiedot työkalusta ovat usean vuoden takaa, jäi Frank tämän vertailun ulkopuolelle[70]. Vertailussa mukana olevien työkalukehysten ominaisuudet on esitetty taulukoissa 4a, 4b ja 4c.

Taulukko 4a. iOS-sovelluksen graafisen käyttöliittymän testaustyökalut

Työkalu	**Saatavuus**	**Testien ajaminen**	**Testien luominen**
XCUITest	Xcoden mukana	Emulaattorit, fyysiset laitteet	Nauhoitus ja automaattinen testiskriptin luonti. Testiskriptin voi myös kirjoittaa itse.
Earl Grey	Maksuton avoimen lähdekoodin ohjelmisto	Simulaattori, fyysiset laitteet	Testiskriptin kirjoittaminen.
KIF	Maksuton avoimen lähdekoodin ohjelmisto	Simulaattori, fyysiset laitteet	Testiskriptin kirjoittaminen.

Taulukko 4b. iOS-sovelluksen graafisen käyttöliittymän testaustyökalut

Työkalu	**Ohjelmointikielet**	**Testin tyyppi**	**Muuta**
XCUITest	Swift tai C++	Black box	Applen virallinen UI-testaustyökalukehys iOS-sovellusten käyttöliittymän testaamiseen. Hyvä testiskriptin suoritusnopeus.
Earl Grey 1.0 ja Earl Grey 2.0	Swift tai C++	Earl Grey 1.0 White box, Earl Gray 2.0 White box/black box	Googlen vastine XCUITestille. Earl Grey 1.0 kerää käyttäjädataa ja lähettää sen Google Analyticsiin toisin kuin Earl Grey 2.0, joka ei kerää eikä lähetä dataa Googlelle. Earl Gray 2.0 käyttää XCUITest -työkalukehystä.
KIF	C++, myös Swiftillä kirjoitetut testit toimivat, mutta tarvitsevat pienen lisäosan C-esiprosessorimakroja varten	Black box	KIF käyttää dokumentoimattomia API-rajapintoja, minkä takia se on poistettava ennen sovelluksen julkaisua. Mahdollistaa mock-luokkien käytön.

Taulukko 4c. iOS-sovelluksen graafisen käyttöliittymän testaustyökalut

Työkalu	**Soveltuu automaatiotestaukseen**	**Soveltuu responsiivisuuden testaukseen**	**Soveltuu pankkisovelluksen testaukseen**
XCUITest	Kyllä	Kyllä	Kyllä
Earl Grey 1.0 ja Earl Grey 2.0	Kyllä	Kyllä	Kyllä
KIF	Kyllä	Kyllä	Kyllä

5.2.1. XCUITest

XCUITest, joka tunnetaan myös nimellä Xcode UI Testing, on osa Applen XCTest -testaustyökalukehystä, ja se tulee Xcode-kehitysympäristön mukana. XCUITest mahdollistaa käyttöliittymätestien kirjoittamisen suoraan Xcoden sisällä Swift- tai C++ -ohjelmointikieliä käyttäen.[71]

[70] https://github.com/TestingWithFrank/Frank
[71] https://bitbar.com/blog/xcuitest-101-basics-best-practices/

XCUITestillä tehdyt käyttöliittymätestit ovat black box -testejä, ne ajetaan sovelluksesta erillään olevassa prosessissa, eikä niillä ole pääsyä sovelluksen sisäisiin tietoihin, metodeihin tai API-rajapintoihin[71]. Ennen kuin XCUITest aloittaa vuorovaikutuksen sovelluksen kanssa se varmistaa epäsuorasti, että sovellus on valmiustilassa. Testien suoritus on siten mahdollista synkronoida sovelluksen tilan kanssa, mikä parantaa testiajojen luotettavuutta [30].

XCUITest käyttää saavutettavia teknologioita iOS-sovelluksen kanssa kommunikoimiseen, mikä tarkoittaa sitä, että XCUITest-työkalukehystä käytettäessä sovellus ja sen kaikki käyttöliittymäkomponentit täytyy tehdä saavutettavaksi. UI-testitapaukset käyttävät Accessibility coren toimintoja testien suorittamiseen ja validointiin.[71] [72]

XCUITest tarjoaa kolme luokkaa käyttöliittymätestien kirjoittamiseen. Automatisoitujen testien suorittamisen ensimmäinen askel on määritellä mille sovellukselle testit ajetaan. Tämä tapahtuu luomalla XCUIApplication-luokan instanssi ja kutsumalla luokan käynnistysmetodia. Metodia käytetään sekä sovelluksen käynnistämiseen ennen testien suorittamista että sen sulkemiseen, kun testiajo on suoritettu loppuun.[72]

XCUIElement-luokka mahdollistaa testitapauksille minkä tahansa käyttöliittymäkomponentin kontrolloimisen ja se tarjoaa metodit komponenttien toimintojen suorittamiseen. Toiminnot vastaavat sovelluksen käyttäjän vuorovaikutusta sovelluksen kanssa kosketuseleillä, esimerkiksi pyyhkäisyillä ja erilaisilla painalluksilla. XCUIElementQuery-luokkaa käyttämällä voi etsiä tietyn käyttöliittymäelementin ja suorittaa jonkin toiminnon haetulle elementille.[72]

XCUITest mahdollistaa käyttöliittymän ohjaamisen lisäksi myös laitteen joidenkin fyysisten ominaisuuksien simuloinnin. XCUIDevice-luokka simuloi laitteen fyysisiä näppäimiä, laitteen suuntausta sekä vuorovaikutusta Sirin kanssa ja XCUISiriDevice-luokka vastaa Sirin käyttöliittymän simuloinnista.[73]

XCUITest mahdollistaa kuvankaappausten ottamisen ja tallentamisen testiajojen aikana. Xcode soveltuu testitapausten nauhoittamiseen emulaattorissa tai oikeassa laitteessa. Nauhoituksen aloittaminen käynnistää sovelluksen emulaattoriin tai oikeaan laitteeseen, ja kaikki käyttöliittymässä suoritetut tapahtumat generoivat koodia sovelluksen testimetodiin.[72]

XCUITest on suunniteltu nimenomaisesti iOS-sovelluksen käyttöliittymän testaamiseen, minkä vuoksi testit ovat stabiileja, luotettavia ja niiden suoritus on nopeaa. Työkalukehyksen käyttöönotto on melko helppoa, sillä se on osa Xcode-kehitysympäristöä. Testikoodia ja sovelluksen lähdekoodia voidaan työstää samassa koodipohjassa, minkä ansiosta kehittäjä voi nopeasti testata sovelluksen toiminnallisuutta. XCUITestin huonoina puolina ovat ohjelmointikielten rajoittuminen ainoastaan Swiftiin ja C++:aan, sekä työkalukehyksen soveltuvuus ainoastaan iOS-sovellusten testaamiseen.[74]

5.2.2. Earl Grey

Earl Grey on UI-testausautomaatiotyökalukehys, joka on Googlen vastine Applen XCUITest-työkalukehykselle. Toisin kuin XCUITest, Earl Grey jakaa saman

[72] https://www.browserstack.com/guide/getting-started-xcuitest-framework
[73] https://developer.apple.com/documentation/xctest/user_interface_tests
[74] https://www.browserstack.com/guide/appium-vs-xcuitest

prosessin testattavan sovelluksen kanssa, joten sillä on pääsy sovelluksen käyttämään muistiin. Earl Grey synkronoi testien suorituksen Espresson tapaan automaattisesti käyttöliittymän ja verkkopyyntöjen kanssa, sekä sallii niiden lisäksi käyttäjän asettamat omat ajastukset Synchronization API -rajapinnan kautta[75] [76] [77]. Ajastusten kustomoinnin ansiosta Earl Greyn synkronointiominaisuudet ovat kattavammat XCUITestiin verrattuna [30]. Synkronointi varmistaa, että käyttöliittymän tila on testien suorituksen aikana vakaa, mikä parantaa testien stabiiliutta, mutta toisaalta kasvattaa niiden suoritusaikaa.[75] [76] [77]

Earl Grey käyttää kuvankaappauksia tarkistaakseen käyttöliittymän komponenttien näkyvyyden ennen kuin se alkaa vuorovaikuttaa niiden kanssa, ja kaikki vuorovaikutus tapahtuu vain näkyvillä olevien komponenttien kanssa. Earl Greyn UI-interaktiot simuloivat todellisen käyttäjän toimia, mikä auttaa löytämään samat käyttöliittymän viat mitkä näkyisivät myös loppukäyttäjälle. [30] [75] [76]

Earl Grey toimii yhdessä XCTest-työkalukehyksen ja Xcoden Test Navigatorin kanssa, joten testit voi suorittaa suoraan Xcodesta tai komentoriviltä. Earl Grey ei pysty käynnistämään tai sulkemaan sovellusta testitapauksesta käsin toisin kuin XCUITest. Testit voi suorittaa joko simulaattorissa tai oikeassa laitteessa. Earl Grey soveltuu sekä funktionaalisten testien että yksikkötestien tekemiseen.[77] [78]

Earl Greystä on saatavilla kaksi versiota, Earl Grey 1.0 ja Earl Grey 2.0. Earl Grey 2.0 on yhdistelmä Earl Grey 1.0:aa ja XCUITestiä, joten mm. järjestelmän varoitukset, sovellusten välinen kommunikaatio ja saavutettavuusominaisuudet ovat tuettuina. Earl Grey 1.0 on white box -testaustyökalu, mutta XCUITest on black box -testaustyökalu, ja siitä johtuen myöskään Earl Grey 2.0:lla ei ole suoraa pääsyä testattavaan sovellukseen toisin kuin Earl Grey 1.0:lla, vaan sen on käytettävä sitä varten eDistantObject (eDO) -kirjastoa. [30] [79]

XCUITest käynnistää sovelluksen ennen testien aloittamisen, mikä tekee Earl Grey 2.0 -testien suorituksesta hitaampaa verrattuna Earl Grey 1.0 -testien suoritukseen. Earl Gray 1.0 kerää käyttäjädataa ja lataa sen Google Analyticsiin toisin kuin Earl Grey 2.0, joka ei kerää dataa eikä lähetä sitä eteenpäin[79]. Muiden natiivien testaustyökalukehysten tapaan Earl Greyn puutteina voi pitää sitä, että se tukee vain natiiveja ohjelmointikieliä ja soveltuu ainoastaan iOS-sovellusten testaukseen[78].

5.2.3. KIF

KIF (Keep It Funktional) on iOS-sovellusten integraatiotestaukseen käytetty avoimen lähdekoodin työkalukehys, joka soveltuu hyvin myös graafisen käyttöliittymän testaamiseen. Se käyttää iOS-käyttöjärjestelmän saavutettavuusominaisuuksia iOS-sovellusten testien automatisointiin. KIF tukee kosketuseleiden simulointia, ja kaikki toiminnot ja vuorovaikutus testien ja sovelluksen välillä jäljittelee oikeiden käyttäjien toimia.[80]

KIF integroituu suoraan Xcode-projektiin ja Xcoden testaustyökaluihin, joten sen konfigurointi ja käyttöönotto on helppoa, eikä muita palvelimia tai pakettien asennusta tarvita. Xcoden hyödylliset ominaisuudet, kuten Test Navigator ja Bots ovat myös

[75] https://github.com/google/EarlGrey/blob/master/docs/features.md
[76] https://bitbar.com/blog/how-to-get-started-with-earlgrey-ios-functional-ui-testing-framework/
[77] https://github.com/google/EarlGrey/blob/master/docs/faq.md
[78] https://github.com/google/EarlGrey
[79] https://github.com/google/EarlGrey/tree/earlgrey2
[80] https://github.com/kif-framework/KIF

KIF:n käytössä. Testit kirjoitetaan C++ -ohjelmointikielellä, mutta Swiftin käyttö on myös mahdollista, kunhan C-kielisten esiprosessorimakrojen käyttöä varten tehdään pieni lisäosa esimerkiksi XCTestCase-luokkaan.[80] [81]

KIF käyttää dokumentoimattomia API-rajapintoja, joten on tärkeää, ettei mitään KIF-testeihin liittyvää koodia sisälly sovelluksen tuotantoversioon, sillä se johtaisi sovelluksen hylkäämiseen Applen taholta. Kyseisten API-rajapintojen käyttö on kuitenkin turvallista testaustarkoituksiin käytettynä. KIF-työkalu eroaa XCUITestistä siten, että se pystyy käyttämään mock-luokkia. KIF:stä puuttuu XCUITestin kyky avata sovellus ennen testin suorittamista ja lopettaa se testiajon päättymisen jälkeen. KIF-testit voidaan ajaa simulaattorissa tai oikeissa laitteissa.[80] [82] [83]

KIF-työkalukehyksellä on samat puutteet kuin XCUITestillä ja Earl Greyllä, eli se tukee vain natiiveja ohjelmointikieliä ja soveltuu vain iOS-sovellusten testaamiseen. Huonona puolena voi pitää myös sitä, että dokumentoimattomien API-rajapintojen takia KIF-työkalukehys täytyy poistaa sovelluksesta ennen sen julkaisua.[83]

5.3. Automaatiotestaustyökalukehykset monialustaympäristöön

Monialustatestaukseen soveltuvista työkalukehyksistä vertailussa on mukana Appium, Calabash, eggPlant, SeeTest Automation, TestComplete, Ranorex Studio, Detox ja Applitools Eyes. Työkalukehysten vertailu on esitetty taulukoissa 5a, 5b ja 5c.

Taulukko 5a. Monialustayhteensopivat graafisen käyttöliittymän testaustyökalut

Työkalu	**Saatavuus**	**Testien ajaminen**	**Testien luominen**
Appium	Maksuton avoimen lähdekoodin ohjelmisto	Emulaattorit, simulaattorit, fyysiset laitteet	Testiskriptin kirjoittaminen, testien nauhoitus Recorderilla.
Calabash	Maksuton avoimen lähdekoodin ohjelmisto	Emulaattorit, fyysiset laitteet	Testiskenaarioiden ja testiskriptin kirjoittaminen.
eggPlant	Maksullinen lisenssi, maksuton kokeilujakso	Emulaattorit, fyysiset laitteet	Tekoäly, testiskriptien generoiminen ja kirjoittaminen.
SeeTest Automation	Maksullinen lisenssi, maksuton kokeilujakso	Emulaattorit, simulaattorit, fyysiset laitteet paikallisesti tai pilvessä	Nauhoitus ja testiskriptin generoiminen sekä kirjoittaminen.
Test Complete Mobile	Maksullinen lisenssi, maksuton kokeilujakso	Emulaattorit, fyysiset laitteet, virtuaalikoneet	Nauhoitus ja toistaminen, testiskriptin kirjoittaminen, keyword-testit.
Ranorex Studio	Maksullinen lisenssi, maksuton kokeilujakso	Emulaattorit, fyysiset laitteet	Nauhoitus ja toistaminen, testiskriptin kirjoittaminen.
Detox	Maksuton avoimen lähdekoodin ohjelmisto	Emulaattorit, fyysiset laitteet osittain tuettuna	Testiskriptin kirjoittaminen.
Applitools Eyes	Maksullinen lisenssi, maksuton lisenssi rajatuilla ominaisuuksilla, tarjouksia open source-projekteille ja uusille yrityksille	Emulaattorit, fyysiset laitteet	Testiskriptin kirjoittaminen, aikaisempien testiskriptien automaattinen hyödyntäminen.

[81] https://bitbar.com/blog/getting-started-with-kif-for-functional-ios-ui-testing/
[82] https://medium.com/supercharges-mobile-product-guide/ui-testing-on-ios-kif-vs-xcuitests-caf31a254428
[83] https://saucelabs.com/resources/articles/is-kif-ios-in-line-with-mobile-testing-best-practices

Taulukko 5b. Monialustayhteensopivat graafisen käyttöliittymän testaustyökalut

Työkalu	Ohjelmointikielet	Testin tyyppi	Muuta
Appium	Mm.Java, C++, PHP, JavaScript, Ruby, Python, C#	Black box	Tunnetuin monialustayhteensopiva mobiilitestaustyökalukehyksistä.
Calabash	Gerkhin-syntaksi, Ruby	Black box	Työkalukehys BDD-hyväksymistestaukseen. Työkalua ei ylläpidetä tällä hetkellä.
eggPlant	SenseTalk	Black box	Tarjoaa tekoälyyn perustuvan alustan älykkään testausautomaation tekemiseen.
SeeTest Automation	Mm. Java, C#, Perl, Ruby, Python, Visual Basic, JavaScript	Black box	Sekä objekti- että kuvapohjainen elementtien tunnistus.
Test Complete Mobile	JavaScript, JScript, Python, VBScript, DelphiScript, C#Script, C++Script	White box	BDD- ja DDT-testaus, checkpoint-operaatiot, objektipohjainen elementtien tunnistus. Sensoridatan lukeminen mobiililaitteesta. Asennus vain Windows-ympäristöön.
Ranorex Studio	C#, VB.NET	Black box	BDD- ja DDT-testaus, objektipohjainen elementtien tunnistus. Laitteen tietojen lukeminen. Asennus vain Windows-ympäristöön.
Detox	JavaScript	Gray box	Natiivi- ja React Native -sovellusten testaamiseen. Testien suoritusnopeus on hyvä. Android-sovellusten tuki on vielä puutteellinen ja tuki iOS-laitteissa testaamiseen puuttuu kokonaan.
Applitools Eyes	Java, JavaScript, C#, Ruby, Python, PHP	Black box	Kuvankaappauksia verrataan tekoälyn avulla aikaisemmin otettuihin referenssikuviin.

Taulukko 5c. Monialustayhteensopivat graafisen käyttöliittymän testaustyökalut

Työkalu	Soveltuu automaatiotestaukseen	Soveltuu responsiivisuuden testaukseen	Soveltuu pankkisovelluksen testaukseen
Appium	Kyllä	Kyllä	Kyllä
Calabash	Kyllä	Kyllä	Kyllä
eggPlant	Kyllä	Kyllä	Kyllä
SeeTest Automation	Kyllä	Kyllä	Kyllä
TestComplete Mobile	Kyllä	Kyllä	Ei, sillä asennus myös Mac-ympäristöön tarvitaan
Ranorex Studio	Kyllä	Kyllä	Ei, sillä asennus myös Mac-ympäristöön tarvitaan
Detox	Kyllä	Kyllä	Ei, sillä Android- ja iOS -tuki ei ole toistaiseksi riittävällä tasolla
Applitools Eyes	Kyllä	Kyllä	Kyllä

5.3.1. Appium

Appium on monialustayhteensopivista mobiilitestaustyökalukehyksistä tunnetuin. Se on avoimen lähdekoodin työkalu natiivi-, hybridi- ja web-sovellusten graafisen käyttöliittymän testaamiseen, ja se soveltuu sekä iOS- että Android- sovelluksille, sillä se käyttää samaa WebDriver API-rajapintaa kummallekin sovellusalustalle. Appiumin

voi yhdistää muiden testaustyökalukehysten kanssa[84]. Appiumin energiankulutus kosketuseleiden simuloinnissa on verraten pieni, joten se sopii myös sovelluksen energiankulutuksen testaamiseen [29].

Testausautomaation tekeminen Appiumilla ei vaadi sovelluksen uudelleenkääntämistä tai muokkaamista, joten testaaminen kohdistuu julkaistavaan sovellukseen ilman muutoksia. Appium käyttää iOS-versiosta 9.3 ylöspäin XCUITest-työkalukehystä ja vanhemmille versioille UIAutomation-työkalukehystä. Android-versiosta 4.3 eteenpäin käytössä on UiAutomator tai UiAutomator2. Appium sisältää työkalukehyksen, joka kääntää Seleniumin WebDriver-komennot XCUITest, UIAutomation- tai UiAutomator -komennoiksi testattavan laitteen tyypistä riippuen.[84] [85]

Testien tekeminen tai suorittaminen ei ole rajattu tiettyihin ohjelmointikieliin tai testaustyökalukehyksiin. Appium tukee kaikkia ohjelmointikieliä, joille Selenium client -kirjasto on saatavilla, kuten Javaa, C++:aa, JavaScriptiä, PHP:tä, Rubya, Pythonia ja C#:ia. [28] [85]

Appium on eräänlainen node.js -alustalle kirjoitettu HTTP-palvelin, joka luo istunnon Webdriver JSON wire -protokollaa käyttäen. Tästä johtuen Node.js on asennettava ennen Appium-palvelimen käyttöönottoa. Kun Appium on asennettu, käynnistetään koneelle palvelin, jonka kautta WebDriver API-rajapinnan saa käyttöön testausautomaation tekemistä varten. Appium vastaanottaa sovellukselta pyyntöjä, suorittaa ne mobiililaitteessa ja lähettää komennon suorituksesta saadun tuloksen HTTP-vastauksena takaisin sovellukselle valittua testiautomaatiotyökalukehystä käyttäen. Appium-testejä voi suorittaa emulaattoreissa, simulaattoreissa sekä oikeissa laitteissa. [28] [84]

Appium Desktop on avoimen lähdekoodin GUI-sovellus, joka sisältää kaiken tarpeellisen palvelimen asentamista ja käyttöönottoa varten. Sen mukana tulee Inspector, joka tarjoaa graafisen käyttöliittymän Appium-palvelimeen sekä sen toimintoihin ja asetuksiin. Inspectorilla voi tutkia sovelluksen hierarkiaa ja elementtejä, sekä vuorovaikuttaa käyttöliittymän kanssa. Appium Desktop sisältää myös Recorderin, työkalun, jolla voi nauhoittaa Appium Desktopin perustapahtumia ja luoda niiden perusteella ohjelmointikieli- ja testaustyökalukehys-spesifistä koodia. Recorderia ei kuitenkaan ole tarkoitettu valmiin testikoodia tuottamiseen, vaan se on työkalu API-rajapinnan tutkimista varten.[86]

Myös Appiumissa on heikkouksia. Appiumin asentaminen voi olla haastavaa ja monimutkaista. Asiakas-palvelin-rakenteesta johtuen testien suorittaminen on hidasta. Testien suorituksen synkronointiominaisuudet puuttuvat Appiumista ja testit saattavatkin olla epästabiileja ja epäluotettavia.[74] [87]

5.3.2. Calabash

Calabash on Microsoftin omistama avoimen lähdekoodin työkalukehys iOS- ja Android-mobiilisovellusten hyväksymistestaukseen. Työkalu soveltuu sekä natiivi- että hybridisovellusten testausautomaation kehittämiseen. Calabash käyttää yksityisiä API-rajapintoja sovelluksen näkymähierarkian tutkimiseen. Kyseisistä rajapinnoista

[84] http://appium.io/docs/en/about-appium/intro/
[85] https://www.guru99.com/introduction-to-appium.html
[86] https://github.com/appium/appium-desktop
[87] https://medium.com/reactive-hub/detox-vs-appium-ui-tests-in-react-native-2d07bf1e244f

johtuen Calabash-palvelin on poistettava iOS-sovelluksen tuotantoversiosta, muutoin Apple estää sen julkaisun AppStoressa.[88]

Calabash on työkalukehys BDD-testien (Behavior Driven Development) tekemiseen. BDD on ketterä ohjelmistokehityksen menetelmä, jossa hyväksymistestit kirjoitetaan luonnollisella kielellä ohjelmistokehityksen alkuvaiheessa, eikä niiden tekemiseen tarvita varsinaista ohjelmointitaitoa. Menetelmän periaatteena on, että määrittelijä, testaaja ja kehittäjä yhteistyössä määrittelevät ja kuvaavat hyväksymiskriteerit skenaarioiksi ohjelmiston käyttäytymisen perusteella. Skenaariot ohjaavat ohjelmistokehittäjän tekemää toteutusta ja ovat pohjana testausautomaation tekemiseen. Helppolukuisuutensa ansiosta BDD-hyväksymistestiskenaariot lisäävät kaikkien projektissa työskentelevien jäsenten ymmärrystä ohjelmiston toiminnasta [31]. Calabash käyttää Cucumber-työkalua, jossa testiskenaariot kirjoitetaan Gerkhin-syntaksia käyttäen. Testiskripti voidaan kirjoittaa Ruby-ohjelmointikielellä. Ruby on asennettava koneelle ennen Calabashin käyttöönottoa.[88]

Calabash mahdollistaa automaattisen vuorovaikutuksen sovelluksen graafisen käyttöliittymän kanssa. Vuorovaikutusta on esimerkiksi nappien painaminen, tekstikenttien täyttäminen tai vastausten validointi [28]. Calabash pystyy ohjaamaan ainoastaan testattavan sovelluksen käyttöliittymää, esimerkiksi notifikaatioiden käsittely puuttuu kokonaan.[89] [90]

Calabashin kosketuseleiden simuloinnin energiajalanjälki on suuri, joten se ei sovi sovelluksen energiankulutusen testaamiseen [29]. Työkalukehyksen haasteina pidetään testiskriptin ylläpidon ja debuggauksen työläyttä. Työkalukehys ei tue testitapausten nauhoitusta eikä koodin automaattista generointia. Calabashin ongelmana on myös se, että Microsoft on lakkauttanut työkalukehyksen kehityksen tukemisen, eikä työkalulla ole tällä hetkellä kehittäjää ja ylläpitäjää.[88] [91] [92]

5.3.3. eggPlant

eggPlant on tekoälypohjainen työkalu mobiilisovellusten testaamiseen. eggPlant soveltuu toiminnalliseen testaukseen ja kuormitustestaukseen laitteesta, käyttöjärjestelmästä ja ohjelmointikielestä riippumatta. Testit voi suorittaa rinnakkain useissa laitteissa ja käyttöjärjestelmissä. Työkalu voi testata minkä tahansa ohjelmiston kerroksen graafisesta käyttöliittymästä REST API -rajapintoihin (Representational State Transfer) ja tietokantoihin saakka[93]. eggPlant on maksullinen, mutta maksuton kokeilujakso on saatavilla[94].

eggPlant Functional on eggPlantin kuvapohjainen testiautomaatiotyökalu. Testien luominen perustuu kuvankaappauksiin sovelluksen käyttöliittymän elementeistä, joihin testit halutaan kohdistaa. Kuvankaappausten perusteella voidaan testiskriptiä luoda automaattisesti. Testiskriptit kirjoitetaan SenseTalk-skriptikielellä, jonka syntaksi on puhutun kielen kaltaista, joten testausautomaation tekeminen ei vaadi

[88] https://github.com/calabash/calabash-ios
[89] https://bitbar.com/blog/calabash-tutorial-for-mobile-app-testing/
[90] https://badoo.com/techblog/blog/2017/01/24/break-limitations-with-calabash-android/
[91] https://www.guru99.com/calabash-android-ios-testing.html
[92] https://www.qamadness.com/comparing-mobile-automation-tools-appium-selendroid-and-calabash/
[93] https://www.eggplantsoftware.com/digital-automation-intelligence
[94] https://www.getapp.com/it-management-software/a/eggplant/pricing/

syvällistä ohjelmointiosaamista[95]. eggPlant Functionalia voi käyttää myös objektipohjaiseen testaukseen Selenium WebDriverin kautta[96].

eggPlant Functional käyttää toimintaansa kaksiosaista mallia. Ensimmäinen osa on laite, johon testaustyökalu on asennettu, ja toinen osa on testattava järjestelmä (system under test, SUT), eli laite, jossa testattavan sovelluksen suoritus tapahtuu. Työkalu luo yhteyden näiden kahden osan välille, minkä jälkeen käyttäjä voi vuorovaikuttaa testattavan järjestelmän kanssa ja hallita sitä. Yhteyden luomiseen käytetään VNC-protokollaa (Virtual Network Computing). iOS-laitteiden yhdistämiseen tarvitaan iOS Gateway -sovellus, joka toimii VNC-palvelimena, ja Android-laitteiden yhdistämistä varten eggPlant sisältää sisäänrakennetun VNC-palvelimen. Myös Android Gatewayn käyttö on mahdollista. Testiskriptin voi ajaa sekä simulaattorissa että oikeissa laitteissa.[97] [98]

eggPlant tarjoaa Digital Automation Intelligence (DAI) -alustan, joka sisältää tekoälypohjaisia ominaisuuksia testausautomaation luomiseen. Tekoälyä voi käyttää mm. testitapausten generointiin, testien optimointiin, älykkäiden raporttien luomiseen ja testitulosten analysointiin. Tekoälyn avulla voi laajan ja monimutkaisen ohjelmiston jakaa pienempiin malleihin, joiden avulla sitä on helpompi käyttää ja hallita.[93]

Syväoppimisalgoritmejä käyttämällä käyttöliittymän elementit pystytään tunnistamaan ja luokittelemaan ohjelmistoalustasta riippumatta. Tekoälyn avulla voi myös kerätä tietoa sovelluksen todellisten käyttäjien toimista ja eniten käytetyistä poluista. Kerätyn tiedon perusteella testauksen voi kohdistaa sovelluksen eniten käytettyihin osiin ja siten parantaa laatua niillä osa-alueilla, jotka eniten vaikuttavat sovelluksen käyttäjäkokemukseen. eggPlant tarjoaa tekoälyn ominaisuuksia myös DevOps-tiimien käyttöön nopeuttamaan ongelmanratkaisua ja lyhentämään tuotteen toimitusaikaa asiakkaille.[93] [99] [100] [101]

eggPlantin huonona puolena on se, että se on maksullinen suljetun lähdekoodin ohjelmisto. Käyttäjäarvosteluissa on myös mainittu, että runsas kuvankaappausten määrä voi hidastaa työkalun toimintaa. Vaikka työkalun käyttö ja yksinkertaisten testiskriptien tekeminen on helppo oppia, vaatii kompleksisten ohjelmistojen testausautomaation suunnitteleminen ja toteuttaminen asiantuntemusta ja kokemusta. Työkalun joustavuus mahdollistaa myös huonojen ratkaisujen ja testitapojen käyttämisen.[102] [103]

5.3.4. SeeTest Automation

SeeTest Automation on Experitestin testausautomaatiotyökalu mobiilisovellusten käyttöliittymän testaamiseen. SeeTest Automation tukee sekä iOS- että Android-

[95] http://docs.eggplantsoftware.com/ePF/using/epf-capturing-images.htm
[96] http://docs.eggplantsoftware.com/ePF/using/epf-selenium-webdriver-testing.htm
[97] https://searchsoftwarequality.techtarget.com/feature/Examining-the-eggPlant-Functional-app-testing-tool
[98] http://docs.eggplantsoftware.com/ePF/using/epf-connecting-to-suts.htm
[99] https://www.businesswire.com/news/home/20181106005127/en/Eggplant-Strengthens-AI-Powered-Digital-Automation-Intelligence
[100] https://blog.eggplantsoftware.com/press-releases/eggplant-enhances-its-digital-automation-intelligence-platform-with-deep-learning
[101] https://www.eggplantsoftware.com/devops
[102] https://www.gartner.com/reviews/market/software-test-automation/vendor/eggplant/product/egg-plant/reviews
[103] https://www.capterra.com/p/176402/Eggplant/

sovelluksia, ja se tarjoaa kirjastot mm. Java-, C#-, Perl- ja Python -ohjelmointikielille testiskriptin kirjoittamista varten. Testit voi suorittaa rinnakkain useissa emulaattoreissa, simulaattoreissa tai oikeissa laitteissa, jotka sijaitsevat joko pilvipalvelussa tai paikallisessa testilaboratoriossa. Työkalu soveltuu natiivi-, hybridi- ja web-sovellusten testaamiseen.[104] [105]

Automaatiotestien luominen aloitetaan kytkeytymällä joko emulaattoriin tai fyysiseen mobiililaitteeseen. Tämän jälkeen sovellus valmistellaan komponenttien tunnistamista varten. Käyttöliittymän testitapaukset voidaan nauhoittaa samalla kun ne suoritetaan manuaalisesti. Työkalu tarjoaa komennot, joilla simuloidaan mm. erilaisia graafisen käyttöliittymän kosketuseleitä. Nauhoituksesta generoidaan testiskripti, jota voi muokata ja täydentää tarpeen mukaan. Testiskripti suoritetaan emulaattorissa tai fyysisessä laitteessa, ja suorituksesta luodaan raportti [32]. Testiskripti viedään tämän jälkeen haluttuun testaustyökalukehykseen, jossa testien suoritus ja raporttien generointi tapahtuu. SeeTest Automation tukee mm. C#-, Perl-, Python- ja JUnit -työkalukehyksiä. Työkalu tukee automaatiotestien jatkuvaa integraatiota, ja sen voi yhdistää esimerkiksi Jenkinsin kanssa.[105] [106] [107]

SeeTest Automation käyttää sekä objekti- että kuvapohjaista elementtien tunnistusta. Menetelmiä voi käyttää yhdessä, mikäli elementin tunnistaminen ei onnistu pelkästään toisella tunnistustavalla. Testien suoritus oikeassa laitteessa saattaa keskeytyä odottamatta esimerkiksi notifikaation, saapuvan puhelun tai tekstiviestin takia, mikä sekoittaa testitapauksen kulun ja aiheuttaa testin hylkäämisen. SeeTest Automation tarjoaa keinon näiden keskeytysten käsittelemiseen client-kirjastojen kautta. Ominaisuus on käytössä, kun testiskripti on kirjoitettu SeeTest Automationin tukemilla ohjelmointikielillä.[105]

SeeTest Automationin tärkeimmät komponentit ovat Application Manager, Device Manager, Object Repository ja Device Reflection. Application Managerin avulla sovellus voidaan mm. asentaa laitteeseen, käynnistää, sulkea sekä poistaa laitteesta. Device Manager yhdistää emulaattoriin tai laitteeseen, näyttää laitteen tiedot ja avaa Device Reflection-komponentin. Object Repository toimii säilytyspaikkana niille kuville ja teksteille, joita käytetään elementtien tunnistamiseen. Device Reflection näyttää tietokoneen ruudulla näkymän testattavan laitteen käyttöliittymästä ja hallintanäppäimistä sekä mahdollistaa laitteen ohjaamisen näkymän kautta.[108]

SeeTest Automation on maksullinen työkalu, mutta se tarjoaa 30 päivän maksuttoman kokeilujakson. SeeTestin valikoimaan kuuluu myös SeeTest Manual, joka on työkalu manuaalitestausta varten. SeeTest Automationin huonona puolena on hinnan lisäksi se, ettei se ole avoimen lähdekoodin ohjelmisto, eikä siten käyttäjän muokattavissa.[104] [109]

5.3.5. TestComplete Mobile

TestComplete on SmartBear Softwaren työkalu sovelluksen käyttöliittymän testausautomaation tekemiseen. TestComplete koostuu TestComplete Platformista ja kolmesta moduulista, joita ovat Desktop, Web ja Mobile. TestComplete Platform

[104] https://www.softwaretestinghelp.com/seetest-automation-tutorial/
[105] https://www.softwaretestinghelp.com/advanced-usage-of-seetest-automation/
[106] https://docs.experitest.com/display/TDB/SeeTestAutomation+-+Overview
[107] https://www.softwaretestinghelp.com/seetest-automation-commands/
[108] https://docs.experitest.com/display/TDB/SeeTestAutomation-+SeeTest+Main+Components
[109] https://seeteststudy.wordpress.com/

tarjoaa ohjelmointiympäristön, joka sisältää peruskomponentit testausautomaation luomiseen sekä nauhoitustyökalun kaikille sovellustyypeille. Moduulit ovat TestComplete Platformin laajennusosia, ja ne sisältävät teknologiaspesifisiä ominaisuuksia erityyppisten sovellusten automaatiotestaukseen. Mobile-moduuli on tarkoitettu iOS- ja Android-sovellusten testausautomaation kehittämiseen. TestComplete Mobile tukee sekä natiivi- että hybridisovelluksia.[110]

TestComplete tukee lukuisia skriptikieliä, kuten JavaScriptiä, JScriptiä, Pythonia, VBScriptiä, DelphiScriptiä, C#Scriptiä and C++Scriptiä. Skriptikielen voi valita vapaasti sovelluksen kehityskielestä riippumatta. TestComplete tukee myös niin kutsuttuja keyword-testejä, joissa yksi keyword, avainsana, vastaa yhtä toimintoa, kuten klikkausta, näppäimistön painallusta, ikkunan avaamista tai sulkemista. [32] [111] [112]

TestCompletella voi myös nauhoittaa testejä ja toistaa niitä emulaattorissa, virtuaalikoneissa ja oikeissa laitteissa. Kosketuseleiden käyttöä voi nauhoittaa yhdessä laitteessa, ja ajaa nauhoituksen tämän jälkeen muissa laitteissa. TestComplete tunnistaa sovelluksen objektit ja ohjaukset, ja tarjoaa komennot, joilla voi simuloida käyttäjän toimia. Tarkistuspisteet (checkpoints) ovat yksi TestCompleten ominaisuus, niiden avulla voidaan tehdä erilaisia tarkastuksia ja vertailuja testien suorituksen aikana. TestCompletella voi lukea myös mobiililaitteiden sensoridataa esimerkiksi kiihtyvyysanturista, valosensorista ja magnetometristä.[113] [114] [115]

TestComplete on integroitunut API-testaustyökalujen, kuten SoapUI:n ja SoapUI Pron, kanssa. Työkalut soveltuvat eri laitteiden ja käyttöjärjestelmien API-rajapintojen toiminnan validoimiseen.[115]

TestComplete soveltuu sekä toiminnalliseen testaukseen, yksikkötestaukseen ja regressiotestaukseen. TestComplete sisältää Debug Info Agent -teknologian, joka mahdollistaa sovelluksen white box -testauksen. Työkalu tukee BDD-testien lisäksi DDT (Data-Driven Testing) -testejä, mikä tarkoittaa sitä, on olemassa joukko määriteltyjä syötteitä, joihin kaikkiin hyödynnetään samaa testauslogiikkaa ja siten laajennetaan testauksen kattavuutta. Automaatiotestejä voi ajaa rinnakkain useissa laitteissa ja käyttöjärjestelmissä. Testit voi suorittaa TestComplete -työkalussa, tai ne voi viedä toiseen työkaluun ja suorittaa siellä. TestComplete tukee integraatiota CI/CD-työkalujen, kuten Jenkinsin, ja versionhallintatyökalujen, kuten Gitin kanssa.[115] [116] [117]

TestCompleten käyttöön tarvitaan maksullinen lisenssi. Lisenssiä on kahta tyyppiä. Node-Locked-lisenssillä voi ajaa yhtä TestCompleten instanssia yhdellä koneella, ja Floating User -lisenssi mahdollistaa yhden tai useamman instanssin ajamisen eri koneilla. TestCompleteen on myös mahdollista tutustua maksuttoman kokeilujakson ajan.[118]

---

[110] https://support.smartbear.com/testcomplete/docs/general-info/testcomplete-modules.html
[111] https://support.smartbear.com/testcomplete/docs/scripting/selecting-the-scripting-language.html
[112] https://support.smartbear.com/testcomplete/docs/testing-approaches/different-ways-of-testing.html
[113] https://support.smartbear.com/testcomplete/docs/general-info/introducing-testcomplete.html
[114] https://support.smartbear.com/testcomplete/docs/app-testing/mobile/common-tasks/sensors/getting-data.html
[115] https://smartbear.com/product/testcomplete/mobile-testing/
[116] https://smartbear.com/learn/automated-testing/introduction-to-data-driven-testing/
[117] https://support.smartbear.com/testcomplete/docs/testing-approaches/white-box-testing.html
[118] https://support.smartbear.com/testcomplete/docs/licensing/about.html

Hinnan lisäksi TestCompleten heikkoutena on se, että sen voi asentaa vain koneeseen, jossa on Windows-käyttöjärjestelmä. TestComplete on suljetun lähdekoodin ohjelmisto.[119]

5.3.6. Ranorex Studio

Ranorex Studio on Ranorex GmbH:n testiautomaatiotyökalukehys sovellusten testaamiseen. Työkalukehys tarjoaa joukon työkaluja sekä testausautomaation tekoon ilman ohjelmointia että täyden ohjelmointiympäristön ja avoimen API-rajapinnan testiskriptien kirjoittamiseen. Työkalu tukee natiivi-, hybridi- ja web-sovellusten testaamista, ja automaatiotestit voi suorittaa emulaattorissa tai oikeissa laitteissa. Testiskriptit ovat alustariippumattomia, samaa skriptiä voi käyttää niin työpöytä-, web-, hybridi- kuin natiivisovellustenkin testaamiseen. [32] [120]

Testit voi luoda yhdessä laitteessa ja suorittaa sen jälkeen muissa laitteissa resoluutiosta ja näytön koosta riippumatta. Testiajon voi suorittaa myös rinnakkain saman käyttöjärjestelmän laitteissa. Testien nauhoituksen aikana on mahdollista siirtyä työpöytä-, web- ja mobiilisovellusten välillä kattavamman testikokonaisuuden aikaansaamiseksi.[120]

Ranorex Studio sisältää Ranorex Spy -komponentin sovellusten käyttöliittymän tutkimiseen ja analysointiin. Ranorex Spy tutkii kaikki laitteessa käynnissä olevat sovellukset ja kuvaa niiden elementit puurakennenäkymään. Ranorex Studio käyttää XPath-kyselykieleen perustuvaa objektipohjaista elementtien tunnistusta, ja se pystyy tunnistamaan myös dynaamiset käyttöliittymän elementit, mikä tekee käyttöliittymätesteistä robusteja ja uudelleenkäytettäviä laitteesta riippumatta. Työkalu tukee kosketuseleiden simulointia ja näytön orientaation vaihtamista.[120] [121] [122]

Työkalukehys mahdollistaa sovellusta ajavan laitteen tietojen, kuten akun latauksen, muistin ja prosessorin tilan, sekä laitteeseen tallennettujen tekstiviestien ja puhelutietojen tutkimisen. Tietoa voi hyödyntää sovelluksen paremman käyttäjäkokemuksen saavuttamiseen.[120]

Ranorex Studio soveltuu UI-testien, funktionaalisten testien ja end-to-end-testien tekemiseen. Työkalukehys mahdollistaa testausautomaation luomisen BDD- ja DDT-menetelmiä käyttäen. Testiskriptit kirjoitetaan C#- ja VB.NET -ohjelmointikielillä, ja testit ovat tyypiltään black box -testejä. Ranorex Studio tukee integraatiota CI/CD-työkalujen, kuten Jenkinsin, ja versionhallintatyökalujen, kuten Gitin, kanssa.[120] [123] [124] [125]

Työkalun asennus on helppoa, eikä sen käyttöönotto vaadi mobiililaitteiden modifikaatiota. Nauhoitustyökalun käyttö mahdollistaa automaatiotestien luomisen myös ilman ohjelmointikokemusta. Jokaisesta testiajosta luodaan raportti, jota voi

---

[119] https://support.smartbear.com/testcomplete/docs/general-info/system-requirements.html
[120] https://www.ranorex.com/mobile-automation-testing/
[121] https://www.ranorex.com/help/latest/web-mobile-testing/advanced-mobile-testing/cross-device-mobile-tests/
[122] https://www.ranorex.com/help/latest/ranorex-studio-advanced/ranorex-spy/introduction/
[123] https://www.ranorex.com/blog/how-to-use-ranorex-studio-in-your-bdd-process/
[124] https://www.ranorex.com/data-driven-testing/
[125] https://www.ranorex.com/integrations/

tarkastella jo ajon aikana. Raporttien rakennetta ja sisältöä on mahdollista muokata vastaamaan käyttäjän tarpeita ja mieltymyksiä.[120] [126]

Kuten TestComplete Mobile, myös Ranorex Studio on maksullinen suljetun lähdekoodin ohjelmisto. Työkalusta on saatavilla kaksi lisenssivaihtoehtoa, konekohtainen Node-Locked-lisenssi ja käyttäjäkohtainen Floating-lisenssi. Työkalukehyksen asentaminen on mahdollista vain koneeseen, jossa on Windows-käyttöjärjestelmä. Työkalukehyksen käyttäjäkunta on pieni verrattuna esimerkiksi Appiumin käyttäjäkuntaan, joten tuki ongelmatilanteissa ei ole yhtä kattavaa kuin monen muun työkalun kohdalla.[127] [128] [129]

5.3.7. Detox

Detox on avoimen lähdekoodin testaustyökalukehys natiivi- ja React Native -sovellusten end-to-end-testaukseen. Työkalu tukee sekä iOS- että Android-sovelluksia.[130]

Toisin kuin monet muut mobiilisovellusten testaustyökalukehykset, Detox ei käytä WebDriver-työkalukehystä, vaan sen sijaan se integroituu suoraan sovelluksen natiivikerroksiin ja kontrolloi laitetta alempien kerrosten API-rajapintojen kautta. Detox on grey box -testaustyökalu, mikä mahdollistaa sovelluksen ohjaamisen sisältäpäin.[131]

Detox käyttää Espresso- ja Earl Grey -työkalukehysten natiivimetodeja JSON-pohjaisen reflektiomekanismin kautta. Kuten Espresso ja Earl Grey, myös Detox synkronoi testien suorituksen sovelluksen tilan kanssa niin, että testit ajetaan vain silloin kun mitään toimintoja ei sovelluksessa ole käynnissä. Synkronointi parantaa testien stabiiliutta ja luotettavuutta. Testit voi ajaa rinnakkain useissa emulaattoreissa ja testien suoritus on nopeaa esimerkiksi Appiumiin verrattuna. Testien suorituksesta luodaan testiraportti. Detox tukee integraatiota CI/CD-työkalujen kanssa.[87] [131] [132] [133]

Detox pohjautuu JavaScriptiin, ja myös testiskriptien kirjoitus tapahtuu JavaScript-ohjelmointikielellä. Testit voi suorittaa missä tahansa JavaScript-testaustyökalukehyksessä, esimerkiksi Mochassa tai AVA:ssa. Sekä Node.js-ympäristössä suoritettava testiskripti, että laitteessa ajettava sovellus ovat asiakasohjelmia eli clienteja. Erillinen websocket-välityspalvelin muodostaa yhteyden asiakasohjelmien välille ja kumpikin osapuoli voi katkaista yhteyden ilman että se vaikuttaa toiseen asiakasohjelmaan. Websocket-yhteyden etuna on mm. sen kaksisuuntaisuus ja nopeus.[131]

Detoxin heikkoutena on, että esimerkiksi Appiumiin verrattuna siinä on vähemmän ominaisuuksia ja työkalun käyttäjäkunta on pieni. Vaikka Detox tukee monialustatestausta, tuki Android-sovelluksille on vielä puutteellinen ja tuki testien suorittamiseen iOS-laitteissa puuttuu toistaiseksi kokonaan. Testiskriptien ohjelmointikielivalikoima rajoittuu JavaScriptiin. Testien nauhoitusmahdollisuutta ei

---

[126] https://www.ranorex.com/help/latest/ranorex-studio-fundamentals/reporting/introduction/
[127] https://www.ranorex.com/prices/
[128] https://www.ranorex.com/help/latest/ranorex-studio-system-details/system-requirements/
[129] https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-ranorex-gui-test-automation-tool/
[130] https://github.com/wix/Detox
[131] https://github.com/wix/Detox/blob/master/docs/More.DesignPrinciples.md
[132] https://github.com/wix/Detox/blob/master/docs/Guide.ParallelTestExecution.md
[133] https://bitbar.com/blog/detox-superfast-e2e-react-native-ui-testing/

ole. Työkalun kyvykkyyttä muodostaa testiraportteja ja ottaa kuvankaappauksia testien suorituksesta on kritisoitu. Detox on lupaava, mutta vielä keskeneräinen ja monilta osin puutteellinen työkalu.[87] [131] [133]

5.3.8. Applitools Eyes

Applitools Eyes on tekoälyyn perustuva työkalu automatisoitujen visuaalisten käyttöliittymätestien tekemiseen. Applitools Eyes soveltuu työpöytä-, web- ja natiivisovellusten toiminnalliseen testaamiseen. Työkalu käyttää edistynyttä kognitiivista näköteknologiaa, jonka avulla se havaitsee kaikki eroavaisuudet kahden näkymän välillä näytön koosta ja resoluutiosta riippumatta, ja se soveltuu hyvin responsiivisuuden testaamiseen. Applitools Eyes on automaatiotestaustyökalu, joka testaa sovelluksen samalla tavalla kuin manuaalitestaaja, mutta nopeammin ja tarkemmin.[134]

Visuaalinen testaus on eräänlainen regressiotestauksen muoto, jonka tarkoituksena on tarkistaa, ettei aikaisemmin virheetön käyttöliittymän näkymä ole muuttunut odottamatta. Sovelluksen visuaalinen testaus toimii niin, että sovellukseen määritellään tarkistuspisteitä, joiden kohdalla näytöstä otetaan kuvankaappaus, jota verrataan ensimmäisen testiajon aikana tallennettuihin referenssikuviin. Jos kuvista löytyy vertailun perusteella merkittäviä eroja, tuloksesta raportoidaan testaajalle. Mikäli referenssikuvien ottamisen jälkeen sovelluksen graafisessa käyttöliittymässä on tapahtunut jokin hyväksytty muutos, referenssikuva päivitetään vastaamaan käyttöliittymän uutta ulkoasua.[135]

Testausjärjestelmä koostuu kuudesta pääkomponentista. AUT (Application Under Test) on testattava sovellus, testijoukko (test suite) koostuu sovellukselle suoritettavista testeistä ja Eyes SDK on kehitysympäristö, jota testijoukko kutsuu kuvankaappauksien ottamista ja niiden vertaamista varten. Sovellusajuri, kuten Appium tai Selenium, tarjoaa testijoukolle työkalun, jonka avulla vuorovaikutus sovelluksen kanssa voidaan suorittaa. Eyes Server on palvelin, joka vastaanottaa ja tallentaa SDK:lta tulevat kuvankaappaukset, vertaa niitä referenssikuviin ja raportoi kuvien väliltä löydetyt eroavaisuudet. Eyes Test Managerin kautta käyttäjä näkee testitulokset, raportoi löydetyt viat ja hallinnoi referenssikuvia.[136]

Applitools Eyes on suunniteltu integroitumaan jo olemassa oleviin testeihin, joten uusia testejä ei tarvitse kirjoittaa eikä uutta ohjelmointikieltä opetella, mikäli sovellukselle on automaatiotestejä jo kattavasti olemassa. Työkalu käyttää algoritmejä, joiden avulla on mahdollista analysoida kaikki testiajoissa löydetyt eroavaisuudet nopeasti ja muodostaa raportti merkittävistä eroista. Testeissä tapahtuneet muutokset on mahdollista hyväksyä tai hylätä, ja päätöksen voi automaattisesti asettaa koskemaan kaikkia vastaavia testeissä esiintyviä muutoksia. Testeille on mahdollista osoittaa komponentteja, joiden on sallittua vaihtaa paikkaa, tai joiden olemassaolo voidaan sivuuttaa. Testiraportit perustuvat kuvankaappauksiin, ja raportteja on mahdollista räätälöidä omien tarpeiden mukaisiksi.[139]

Ultrafast Grid on vapaavalintainen palvelu, joka merkittävästi lyhentää visuaalisten testien suoritusaikaa. Palvelu siirtää tarkistuspisteissä otettujen kuvien generoimisen

[134] https://applitools.com/blog/applitools-eyes-introduction-to-automated-visual-ui-testing/
[135] https://applitools.com/docs/topics/overview/overview-visual-testing.html
[136] https://applitools.com/docs/topics/overview/overview-system-overview.html

pilvipalveluun, jossa kuvien prosessointi ja testitapaukset voidaan suorittaa rinnakkain useilla palvelimilla.[137]

Applitools Eyes tukee mobiiliympäristössä Java-, JavaScript-, C#-, Ruby-, Python- ja PHP-ohjelmointikieliä. Muista testaustyökalukehyksistä Applitools Eyes tukee Calabashia, Espressoa, EarlGreytä ja XCUI:ta. XCUI mahdollistaa testiskriptin kirjoittamisen Swift- ja C++ -ohjelmointikielillä. Applitools Eyes voi integroitua useiden pilvipohjaisten palveluiden, kuten SauceLabsin, Perfecton tai BrowserStackin kanssa. Työkalu tukee integraatiota CI/CD-työkalujen, kuten Jenkinsin ja Jiran kanssa. EyesSelenium on Applitoolsin Python-ohjelmointiympäristö, jota Robot Frameworkin EyesLibrary -kirjasto käyttää. EyesLibrary tarjoaa automatisoinnin visuaalisen testauksen varmentamiselle.[138] [139] [140]

Applitools Eyesissa on kolme asennusvaihtoehtoa. Enterprise Cloud on julkinen pilvi, johon jokainen asiakas saa oman säiliön testitulosten ja referenssikuvien säilyttämiseen. Private Cloud tarkoittaa sitä, että pilvipalvelusta erotetaan asiakkaalle oma kone, johon vaaditaan jokaiselta käyttäjältä erillinen autentikointi. On-Premise-versio on paikallisesti asiakkaan omissa tiloissa ja tietoverkossa.[141]

Applitools Eyes on maksullinen suljetun lähdekoodin ohjelmisto. Siitä on kuitenkin mahdollista saada käyttöön maksuton kokeilujakso sekä maksuton lisenssi rajoitetuilla ominaisuuksilla, jossa käyttö on rajattu vain yhteen käyttäjään ja sataan tarkistuspisteeseen kuukaudessa. Applitools tarjoaa maksuttoman lisenssin avoimen lähdekoodin projekteille sekä tarjouksia vastaperustetuille startup-yrityksille.[142]

5.4. Pilvipohjaiset mobiilitestausalustat

Pilvipohjaiset testausalustat tarjoavat valikoiman oikeita mobiililaitteita, käyttöjärjestelmiä, selaimia ja emulaattoreita sovellusten ja verkkosivujen testaamista varten. Pilvessä sijaitsevat testausalustat ovat varteenotettava vaihtoehto paikalliselle fyysisistä mobiililaitteista koostuvalle testilaboratoriolle, jonka ylläpitäminen ja päivittäminen on sekä työlästä että kallista.

Vertailussa mukana olevissa pilvipohjaisissa testausalustoissa on useita kaikille yhteisiä ominaisuuksia. Jokainen pilvipalvelu tarjoaa testauskäyttöön oikeita mobiililaitteita, jotka sijaitsevat palveluntarjoajan omassa datakeskuksessa ja muodostavat niin kutsutun julkisen pilven, jonka sisältämät laitteet ovat varattavissa kaikkien asiakkaiden käyttöön testien suorittamista varten. Pilvipalveluiden tarjoama laitekanta vaihtelee mallien, käyttöjärjestelmien ja niiden versioiden osalta, mutta kaikki laitepilvet tukevat sekä Android- että iOS-sovellusten testaamista.[143] [184] [185] [190] [197] [204] [209] [216] [224]

Pilvipalvelut soveltuvat natiivisovellusten automaatiotestaukseen, ja testiajoja voi suorittaa useissa laitteissa rinnakkain. Laitepilviä voi käyttää graafisen käyttöliittymän testaamiseen, ja ne tukevat kosketuseleiden käyttöä fyysisissä laitteissa. Pilvipalvelut

[137] https://applitools.com/docs/topics/overview/using-the-ultrafast-grid.html
[138] https://applitools.com/docs/topics/general-concepts/supported-platforms.html
[139] https://applitools.com/features/test-automation/
[140] https://pypi.org/project/robotframework-eyeslibrary/
[141] https://help.applitools.com/hc/en-us/articles/360007189231-The-different-deployment-modes
[142] https://applitools.com/pricing/
[143] https://www.browserstack.com/list-of-browsers-and-platforms/live

tukevat integraatiota testaustyökalukehysten ja CI/CD-työkalujen, kuten Jenkinsin kanssa.[170 182 187 190 192 197 201 204 144 145 146 147 220 224 227]

Pilvipohjaisten mobiilitestausalustojen lisenssit ovat maksullisia, mutta kaikista on mahdollista saada joko maksuton kokeilulisenssi, joka on voimassa keskimäärin 14-30 vuorokautta, tai maksuton free tier -lisenssi rajatuilla ominaisuuksilla. Laitepilvien heikkoutena on niiden käytön hitaus ja yhteysongelmat verrattuna paikallisen testilaitelaboratorion käyttöön. Yhdenkään vertailussa mukana olevan pilvipalvelun mobiililaitevalikoima ei täysin vastaa OP:n mobiilisovelluksen testauksen tarpeita laitteiden mallien tai käyttöjärjestelmäversioiden osalta.[143 148 149 150 151 152 153 154 155 156 157 158 159 160 178 180 189 217 223]

Pilvipalveluita tarjoavien yritysten hallinto keskittyy Yhdysvaltoihin. Tässä työssä käsitellyistä pilvipohjaisista mobiilialustoista kaikilla on pääkonttori Yhdysvalloissa, ja ainoastaan BrowserStackilla on alueelliset pääkonttorit myös Yhdysvaltojen ulkopuolella, Dublinissa ja Mumbaissa[161 162 163 164 165 166 167 168 169].

Pilvipohjaisten mobiilitestausalustojen vertailussa on mukana BrowserStack, Kobiton, Bitbar, SauceLabs, AWS Device Farm, Firebase TestLab, Visual Studio App Center, Perfecto ja Experitest. Testausalustojen vertailu on esitetty taulukossa 6a ja 6b.

---

144 https://wiki.jenkins.io/display/JENKINS/App+Center+Plugin
145 https://docs.microsoft.com/vi-vn/appcenter/test-cloud/frameworks/uitest/features/gestures
146 https://www.perfecto.io/platform/test-creation
147 https://developers.perfectomobile.com/display/PD/Gesture
148 https://kobiton.com/pricing/
149 https://docs.kobiton.com/devices-list/
150 https://bitbar.com/pricing/
151 https://saucelabs.com/pricing
152 https://saucelabs.com/platform/supported-browsers-devices
153 http://awsdevicefarm.info/
154 https://aws.amazon.com/device-farm/pricing/
155 https://firebase.google.com/pricing
156 https://firebase.google.com/docs/test-lab/android/available-testing-devices
157 https://visualstudio.microsoft.com/app-center/pricing/
158 https://docs.microsoft.com/en-us/appcenter/test-cloud/devices/android
159 https://experitest.com/pricing/
160 https://experitest.com/real-devices/
161 https://www.browserstack.com/company/fact-sheet
162 https://craft.co/kobiton/locations
163 https://craft.co/bitbar/locations
164 https://saucelabs.com/contact
165 https://craft.co/amazon
166 https://craft.co/firebase/locations
167 https://baxtel.com/data-centers/microsoft-azure
168 https://www.perfecto.io/company/worldwide-locations
169 https://experitest.com/contact-us/

Taulukko 6a. Pilvipohjaiset mobiilitestausalustat

Pilvi-palvelu	**Lisenssit**	**Testauksen tyyppi**	**Integraatiot**	**Testien suoritus**
Browser Stack	Maksullinen Maksuton open source-lisenssi Maksuton kokeilujakso	Manuaali- ja automaatiotestaus	Appium, Espresso, XCUITest, EarlGrey, Jenkins, Jira	Oikeat laitteet, emulaattorit
Kobiton	Maksullinen Maksuton kokeilujakso	Manuaali- ja automaatiotestaus	Appium, Katalon Studio, Jenkins, Jira, GitHub	Oikeat laitteet
Bitbar	Maksullinen Veloitus käytön mukaan Maksuton kokeilujakso	Manuaali- ja automaatiotestaus, tekoälypohjainen kooditon testaus	Lähes kaikki testiautomaatio-työkalukehykset ja CI/CD-työkalut	Oikeat laitteet
Sauce Labs	Maksullinen Maksuton kokeilujakso	Manuaali- ja automaatiotestaus, koodittomat UI-testit React-, Vue- ja Angular-komponenteille	Appium, Espresso, XCUITest, Robotium, Jenkins	Oikeat laitteet, emulaattorit, simulaattorit
AWS Device Farm	Maksullinen Maksuton Free Tier -lisenssi	Manuaali- ja automaatiotestaus, kooditon testaus AWS:n sisäänrakennetulla työkalukehyksellä	Appium, Calabash, UI Automator, UI Automation, XCUITest, Jenkins	Oikeat laitteet
Firebase	Ominaisuuksiltaan rajattu ilmainen lisenssi, minkä jälkeen maksu käytön mukaan	Automaatiotestaus, Robo skriptit ja koodittomat Robo-testit (Androidille)	Espresso, UI Automator 2.0, XCTest, Jenkins	Oikeat laitteet, simulaattorit
Visual Studio App Center	Ominaisuuksiltaan rajattu ilmainen Free Tier, minkä jälkeen maksu käytön mukaan	Automaatiotestaus	Appium, Espresso, Xamarin.UITest, XCUITest, Jenkins	Oikeat laitteet
Perfecto	Maksullinen Ilmainen kokeilujakso	Manuaali- ja automaatiotestaus	Appium, Espresso, XCUITest KIF, Jenkins, Jira	Oikeat laitteet, emulaattorit, simulaattorit
Experitest	Maksullinen Ilmainen kokeilujakso	Manuaali- ja automaatiotestaus	Appium, Espresso, XCUITest, Junit, Jenkins	Oikeat laitteet, emulaattorit, simulaattorit

Taulukko 6b. Pilvipohjaiset mobiilitestausalustat

Pilvipalvelu	**Laitekanta täyttää pankkisovelluksen testausvaatimukset**	**Soveltuu responsiivisuuden testaukseen**	**Soveltuu pankkisovelluksen testaukseen**
BrowserStack	Ei	Kyllä	Kyllä, mutta ei toistaiseksi pysty kilpailemaan oman testilaitelaboratorion kanssa.
Kobiton	Ei	Kyllä	Kyllä, mutta ei toistaiseksi pysty kilpailemaan oman testilaitelaboratorion kanssa.
Bitbar	Ei	Kyllä	Kyllä, mutta ei toistaiseksi pysty kilpailemaan oman testilaitelaboratorion kanssa.
Sauce Labs	Ei	Kyllä	Kyllä, mutta ei toistaiseksi pysty kilpailemaan oman testilaitelaboratorion kanssa.
AWS Device Farm	Ei	Kyllä	Kyllä, mutta ei toistaiseksi pysty kilpailemaan oman testilaitelaboratorion kanssa.
Firebase	Ei	Kyllä	Ei sovellu testilaite-laboratorion korvaamiseen manuaalitestaustuen puuttumisen vuoksi.
Visual Studio App Center	Ei	Kyllä	Ei sovellu testilaite-laboratorion korvaamiseen manuaalitestaustuen puuttumisen vuoksi.
Perfecto	Ei	Kyllä	Kyllä, mutta ei toistaiseksi pysty kilpailemaan oman testilaitelaboratorion kanssa.
Experitest	Ei	Kyllä	Kyllä, mutta ei toistaiseksi pysty kilpailemaan oman testilaitelaboratorion kanssa.

5.4.1. BrowserStack

BrowserStack tarjoaa testausalustan sekä web-sivuille että natiivi-, hybrdi- ja web-sovelluksille julkisessa tai yksityisessä pilvessä. Testausalusta soveltuu automaatiotestauksen lisäksi sovelluksen manuaaliseen testaamiseen emulaattoreissa ja oikeissa laitteissa. BrowserStack on yhteensopiva Appium-, Espresso-, XCUITest- ja EarlGrey -työkalukehyksillä tehdyn testausautomaation kanssa. BrowserStack soveltuu toiminnalliseen testaukseen, regressiotestaukseen sekä lokalisaatiotestaukseen.[170] [171] [172] [173] [174] [175] [177]

BrowserStackilla voi testata sovelluksen natiiviominaisuuksia, kuten GPS-paikannusta (Global Positioning System), Push-notifikaatioita, aikaisemmin tallennettuja kuvia, verkkoyhteyden simulointia ja aikavyöhykettä. Sovellusta on mahdollista debugata reaaliajassa ja tarkastella testien suorituksesta saatavia laitelokeja, tekstilokeja, videotallenteita ja kuvankaappauksia.[176] [177]

[170] https://www.browserstack.com/integrations/app-automate
[171] https://www.browserstack.com/question/476
[172] https://www.browserstack.com/app-automate
[173] https://www.browserstack.com/private-device-cloud
[174] https://www.browserstack.com/automated-functional-testing
[175] https://www.browserstack.com/automated-regression-testing
[176] https://www.browserstack.com/app-automate/features
[177] https://www.browserstack.com/app-automate/device-features

BrowserStackin Local Testing tarjoaa kyvykkyyden testata laitepilvessä myös sovellusta, joka sijaitsee yksityisellä tai sisäisellä palvelimella. Local Testing muodostaa testien suoritusta varten suojatun yhteyden paikallisen koneen ja BrowserStackin etäkoneen välille, ja kaikki sovelluksen data kulkee tätä yhteyttä pitkin.[178]

Pilvipohjaisten palveluiden tietoturvallisuus on äärimmäisen tärkeää. BrowserStackin jokainen virtuaalikone on uusi instanssi, johon alkuperäiset asetukset on palautettu, rekisterit ja välimuisti tyhjennetty sekä evästeet poistettu, minkä jälkeen virtuaalikoneelle vielä ajetaan erilaisia tarkistuksia ja se poistetaan käytöstä, mikäli yksikin tarkistus antaa virheellisen tuloksen. Virtuaalikoneet ovat lisäksi vahvan palomuurin takana.[179]

Fyysisten laitteiden tehdastila palautetaan jokaisen testisession jälkeen. Mobiililaitteet on hankittu palveluun uutena ja käyttämättömänä, ja niiden sijoituspaikka on salainen. Ainoastaan valtuutetuilla henkilöillä on lupa käsitellä ja ylläpitää laitteita. Enkryptattu data siirretään BrowserStackin palvelimelle VNC-protokollaa käyttäen. Local Testing mahdollistaa sovelluksen testaamisen yksityiseltä tai paikalliselta palvelimelta käsin. BrowserStack käyttää suojattua HTTPS-protokollaa. Käyttäjän kirjautumistiedot säilytetään salatussa formaatissa BrowserStackin järjestelmässä, eikä palvelun käytöstä kerätty lokitieto sisällä käyttäjän henkilötietoja. Testien suorituksesta muodostetut lokit ja kuvankaappaukset säilytetään varmennetussa tietokannassa, ja niitä pääsee näkemään vain oman käyttäjätilin kautta, jonka kirjautumismekanismi on vahvasti enkryptattu.[179]

BrowserStack tarjoaa maksuttoman kokeilujakson ja maksullisten lisenssien lisäksi maksuttoman lisenssin avoimen lähdekoodin projekteille[180].

5.4.2. Kobiton

Kobiton on testausalusta, joka mahdollistaa mobiili- ja web-sovellusten manuaali- ja automaatiotestauksen pilvessä sijaitsevissa oikeissa iOS- ja Android-mobiililaitteissa. Kobiton tukee laitteen orientaation vaihtamista sekä GPS-paikannuksen ja verkkoyhteyden testaamista. Testeistä luodaan yksityiskohtaiset raportit, jotka sisältävät videoita, kuvankaappauksia, käytetyt kosketuseleet, järjestelmän metriikkatietoja esimerkiksi muistin käytöstä ja akun suorituskyvystä, laitteen lokit sekä tulokset testiskriptin suorituksesta.[181] [182]

Kobiton mahdollistaa Appium-testiskriptin luomisen iOS- tai Android-laitteessa suoritetun manuaalisen testin perusteella. Ohjelmointikielistä Kobiton tukee Javaa, Pythonia, C#:ia, Rubya ja PHP:ta sekä lisäksi Node.js-ympäristöä. Testausalusta soveltuu sekä toiminnalliseen testaamiseen että suorituskykytestaukseen. Kobiton pystyy havaitsemaan visuaaliset poikkeamat automaattisesti testiajon aikana, ja koneoppimisteknologian käyttö parantaa testien vakautta ja luotettavuutta. Kobiton tarjoaa tuen Appiumille ja Seleniumille, ja tekee myös yhteistyötä Katalon Studion kanssa.[181] [182] [183]

---

[178] https://www.browserstack.com/local-testing/app-live
[179] https://www.browserstack.com/policy/security
[180] https://www.browserstack.com/pricing
[181] https://kobiton.com/book/chapter-10-test-execution-on-remote-devicekobiton/
[182] https://kobiton.com/real-device-testing/
[183] https://kobiton.com/intelligent-test-automation/

Kobiton tarjoaa julkisessa pilvessä olevan laitevalikoiman lisäksi mahdollisuuden muodostaa yksityisen pilvilaitekokoelman vain omaan käyttöön. Myös paikallisen testilaboratorion laitteet voi yhdistää hybridipilveksi Kobitonin pilvilaitekokoelmaan device lab -hallintaohjelmalla. Kobiton tarjoaa turvallisen tunneliyhteyden paikallisen tai yrityksen palomuurin takana olevan tietokoneen ja testauksessa käytetyn pilvilaitteen välille, jotta vielä julkaisemattomien ja salassa pidettävien sovellusten testaaminen on riskitöntä. Org Management -työkalu mahdollistaa laitteiden ja sovellusten käyttöoikeuksien hallinnan käyttäjä-, tiimi- ja organisaatiotasolla.[43] [184]

5.4.3. Bitbar

Bitbar on pilvipohjainen mobiilitestausalusta, joka mahdollistaa rinnakkaisen automaatiotestauksen millä tahansa natiivilla testaustyökalukehyksellä ja ohjelmointikielellä. Manuaalitestauksen voi suorittaa yhdellä laitteella kerrallaan ja tekoälypohjaisen koodittoman testauksen rinnakkain suuressa joukossa laitteita. AI Testbot on Android-laiteissa toimiva työkalu, joka tutkii sovelluksen ja vuorovaikuttaa sen elementtien kanssa. Video, kuvankaappaukset ja testiaskeleiden kuvaukset on saatavilla jokaisesta AI Testbot-testiajosta.[185] [186] [187]

Bitbarista on saatavilla kolme versiota, julkinen pilvi (Bitbar Public Cloud), yksityinen pilvi (Bitbar Private Cloud) sekä paikallinen pilvi (Bitbar On-premise Cloud). Julkinen pilvi on yhteinen palvelu, joka tarjoaa laajan valikoiman oikeita iOS- ja Android -mobiililaitteita natiivi-, hybridi- ja web-sovellusten testaamiseen. Julkisessa pilvessä on mahdollista suorittaa automaatiotestausta, manuaalitestausta sekä tekoälypohjaista testausta. Bitbar tarjoaa REST API -rajapinnan käyttöoikeuksien, testien sekä testiajojen hallintaan. Julkinen pilvi tukee mm. Appiumia, Robotiumia, Espressoa, uiautomatoria, XCUITestiä ja Seleniumia.[185]

Yksityinen pilvi tarjoaa kokoelman yksityiseen käyttöön varattuja laitteita, joita Bitbarin työntekijät hallinnoivat Bitbarin tiloissa. Yksityinen pilvi varmistaa laitteiden saatavuuden ympäri vuorokauden, esimerkiksi jatkuvan integroinnin käyttöön. Yksityisen pilven laitteiden käyttö tapahtuu ja SSL-salatun (Secure Sockets Layer) VPN-yhteyden (Virtual Private Network) välityksellä. Yksityinen pilvi voidaan muodostaa varaamalla oma pilvi-instanssi Bitbarin hallinnassa olevista laitteista. Toinen vaihtoehto on määrittää käyttöön asiakkaan oma virtuaalinen yksityinen pilvi-instanssi, esimerkiksi AWS Virtual Private Cloud, Microsoft Azure, IBM Cloud tai Google Cloud. Yksityinen pilvi on räätälöity ympäristö, ja se tukee lähes kaikkia testiautomaatiotyökalukehyksiä sekä Docker-konttiteknologiaa. On-premise on yksityisen pilven kaltainen, mutta asiakkaan omissa tiloissa sijaitseva laitepilvi, jonka asiakas omistaa ja jonka kaikkia toimintoja ja sisältöä se hallinnoi.[185]

Bitbarin pilvipalvelut on SSL-salattuja, ja kaikki palveluun ladattu sisältö sekä testitulokset ovat näkyvillä vain valtuutetuille käyttäjille. Kaikki käyttäjädata sekä käyttäjän asentamat sovellukset poistetaan jokaisen testiajon jälkeen, muistikortit formatoidaan ja laitteet käynnistetään uudelleen automaattisesti, jotta mitään testidataa ei jää näkyviin ja laitteisto on identtisessä tilassa ennen jokaista uutta testiajoa.

---

[184] https://kobiton.com/device-lab-management/
[185] https://support.smartbear.com/bitbar/docs/about-bitbar.html
[186] https://support.smartbear.com/bitbar/docs/testing-with-bitbar/ai-driven-testing.html
[187] https://bitbar.com/

Palveluun ladattua dataa säilytetään vain rajattu aika, minkä jälkeen sen palauttaminen Bitbarin toimesta on mahdotonta.[188]

Bitbarin tarjoamien iOS-laitteiden ohjelmistoversioita ei ole näkyvillä, joten niiden soveltuvuutta ei pysty arvioimaan pelkän laitelistan perusteella[189].

5.4.4. Sauce Labs

Sauce Labs on pilvipohjainen testausalusta, joka tarjoaa oikeita iOS- ja Android-laitteita, emulaattoreita ja simulaattoreita natiivi- ja hybrid-sovellusten sekä web-sivujen manuaali- ja automaatiotestaukseen julkisessa tai yksityisessä pilvessä. Sauce Labs tukee integraatiota Appiumin, Espresson, XCUITestin ja Robotiumin kanssa.[44] [190] [191]

Sauce Labs mahdollistaa sovelluksen jatkuvan visuaalisen regressiotestaamisen. Visuaaliset testit voi suorittaa jokaisen versionhallintaan tehdyn muutoksen jälkeen. Testeistä voi ottaa kuvankaappauksia ja verrata niitä aikaisemmista testiajoista otettuihin kuvankaappauksiin, jotta muutokset on helppo havaita. Kuvankaappausten lisäksi vertailua voi tehdä myös DOM-tiedostojen (Document Object Model) välillä. Sauce Labs tarjoaa Storybook-ympäristön, jossa koodittomia UI-testejä voi luoda React-, Vue- ja Angular-komponenteille.[192] [193]

Sauce Labs käyttää koneoppimiseen perustuvaa analytiikkaratkaisua testien laadun parantamiseen. Asiakkaan testiajojen tuloksista kerätty data analysoidaan ja siitä pyritään löytämään yleisimmät syyt, jotka aiheuttavat hylätyn testituloksen. Analyysin perusteella sovelluksen kehittäjien ja testaajien on mahdollista saada näkyvyys samantyyppisten virheiden toistumistiheyteen ja tehdä huomattavia parannuksia sovelluksen toimintaan sekä parantaa testauksen laatua.[194]

Sauce Connect Proxy™ on sisäänrakennettu HTTP-välityspalvelin, joka avaa TLS-salatun (Transport Layer Security) turvallisen tunneliyhteyden Sauce Labsin pilvipalvelussa olevan mobiililaitteen tai virtuaalikoneen ja paikallisella koneella tai yrityksen palomuurin takana sijaitsevan testattavan sovelluksen välillä. Sauce Labs tarjoaa IPSec VPN -yhteyden (Internet Protocol Security) Sauce Labsin pilven ja asiakkaan laitteen välille.[195] [196]

5.4.5. AWS Device Farm

AWS Device Farm on Amazonin natiivi-, hybridi- ja web-sovellusten pilvipohjainen testauspalvelu. AWS Device Farmissa on sisäänrakennettu testaustyökalukehys, joka mahdollistaa testien suorittamisen ilman testiskriptien kirjoittamista tai ylläpitoa. AWS Device Farm tukee myös useita automaatiotestaustyökalukehyksiä, kuten Appiumia, Calabashia, UI Automatoria, UI Automationia sekä XCUITestiä. AWS

---

[188] https://support.smartbear.com/bitbar/docs/general-info/handling-user-data.html
[189] https://cloud.bitbar.com/#public/devices
[190] https://saucelabs.com/platform/real-device-cloud
[191] https://wiki.saucelabs.com/display/DOCS/Mobile+Testing+Frameworks
[192] https://saucelabs.com/platform/visual-testing
[193] https://wiki.saucelabs.com/display/DOCS/Setting+Up+Sauce+Labs+with+Jenkins
[194] https://saucelabs.com/news/sauce-labs-launches-new-machine-learning-based-analytics-solution-to-improve-test-quality
[195] https://wiki.saucelabs.com/display/DOCS/Sauce+Connect+Proxy
[196] https://wiki.saucelabs.com/display/DOCS/IPSec+VPN

mahdollistaa standardien testaustyökalukehysten lisäksi myös kustomoitujen ympäristöjen käytön testausautomaatiossa. Myös manuaalinen testaus on mahdollista AWS:n pilvessä etäyhteyden välityksellä.[197] [198] [199]

Testiajoista luodaan videot, kuvankaappaukset, lokit ja suorituskykydata sovelluksen vikojen selvittämistä varten. Automatisoiduista testeistä löydetyille vioille AWS suorittaa identifioinnin ja ryhmittelyn, jonka perusteella kehittäjien on helpompi keskittyä sovelluksen merkittävimpiin ongelmakohtiin.[200]

AWS mahdollistaa testiympäristön hienosäädön esimerkiksi sijainnin, kielen, verkkoyhteyden ja sovellusdatan konfiguroinnin osalta, sekä tarvittavien sovellusten esiasentamisen, jotta testaus voidaan tehdä mahdollisimman hyvin todellisten asiakkaiden tilannetta vastaavissa olosuhteissa[197].

AWS tarjoaa liitännäisiä sekä API-rajapinnan, jotka mahdollistavat testiajojen automaattisen aloittamisen ja testitulosten vastaanottamisen ohjelmointiympäristöjen ja CI/CD-työkalujen kautta. Julkisen pilven lisäksi on mahdollista ottaa käyttöön myös yksityisessä käytössä oleva laitepilvi.[197] [201]

AWS käyttää tietoturvan osalta jaettua mallia. AWS vastaa kaikkien AWS:n pilvessä tarjottujen palveluiden infrastruktuurin, kuten pilvipalvelun muodostavan laitteiston, välineistön, ohjelmiston ja verkon suojaamisesta.[202]

Asiakkaan vastuu määräytyy sen mukaan mitä AWS:n palveluita asiakas valitsee käyttöönsä. Valinnan perusteella määräytyy, kuinka paljon konfigurointityötä asiakkaan on tehtävä riittävän tietoturvan tason varmistamiseksi. Ympäristöä koskevien vastuiden lisäksi myös IT-kontrollien hallinta-, operaatio- ja verifiointivastuut jaetaan AWS:n ja asiakkaan kesken.[202]

AWS Device Farmin käytössä on rajoituksia. Testiajossa käytettävien laitteiden määrää ei ole rajoitettu, mutta testit voi suorittaa rinnakkain korkeintaan viidessä laitteessa kerrallaan, tosin laitteiden määrää voi kasvattaa erikseen anomalla. Sekä manuaalitestauksen etäyhteyden että automaatiotestien suorituksen aikaraja on 150 minuuttia. Palveluun ladattavan sovellustiedoston koko voi olla enintään 4GB.[203]

5.4.6. Firebase

Firebase Test Lab on Googlen Firebase-mobiilisovellusalustan palvelu, joka tarjoaa oikeiden laitteiden lisäksi virtuaalikoneita Android- ja iOS-sovellusten testaamiseen. Testilaitteisiin asennetaan ajan tasalla olevat API-rajapinnat ja laitteiden lokalisaatioasetuksia voi kustomoida vastaamaan asiakkaiden todellista tilannetta. Firebase Test Lab integroituu Firebasen konsolin, Android Studion ja gcloud-komentorivityökalun kanssa.[204]

Test Lab suorittaa Espresso-, XCTest- sekä UI Automator -testejä. Test Lab tuki aikaisemmin myös Robotium-testaustyökalukehyksellä tehtyjä testejä, mutta tuki on sittemmin lopetettu. Test Labiin on integroitu Robo-testaustyökalu Android-

[197] https://aws.amazon.com/device-farm/
[198] https://docs.aws.amazon.com/devicefarm/latest/developerguide/test-types.html
[199] https://docs.aws.amazon.com/devicefarm/latest/developerguide/custom-test-environments.html
[200] https://docs.aws.amazon.com/devicefarm/latest/developerguide/welcome.html
[201] https://docs.aws.amazon.com/devicefarm/latest/developerguide/continuous-integration-jenkins-plugin.html
[202] https://aws.amazon.com/compliance/shared-responsibility-model/
[203] https://docs.aws.amazon.com/devicefarm/latest/developerguide/limits.html
[204] https://firebase.google.com/docs/test-lab

sovellusten testaamista varten. Robo-testi analysoi sovelluksen käyttöliittymän rakenteen ja käy sen järjestelmällisesti läpi simuloiden samalla käyttäjän aktiviteetteja. Robo simuloi aina samat käyttäjän aktiviteetit samassa järjestyksessä, kun sitä ajetaan samassa laitekonfiguraatiossa ja asetuksissa, minkä ansiosta sitä voi käyttää korjattujen virheiden verifiointiin sekä regressiotestaukseen. Robo-testistä luodaan lokitiedostot, tallennetaan kommentoituja kuvankaappauksia ja luodaan kuvankaappauksista video, josta nähdään testien aikana suoritetut käyttäjätoiminnot.[204] [205] [206]

Mikäli on tarpeen testata jotain tiettyä käyttäjän suorittamaa sovelluksen polkua, voidaan sen manuaalinen suoritus tallentaa Robo-skriptiksi ja ladata se Firebasen konsoliin Robo-testien suoritusta varten. Robo ensin ajaa Robo-skriptin ja siirtyy sen jälkeen tutkimaan sovellusta tavanomaiseen tapaan. Robo testaa ainoastaan Android UI työkalukehyksen elementtejä, kuten View- ja ViewGroup-objekteja. Mikäli sovellus käyttää muita UI työkalukehyksiä, Robo saattaa päättää testauksen käyttöliittymän ensimmäisen näkymän jälkeen.[204] [205]

Automaatiotestejä voi ajaa oikeassa laitteessa enintään 45 minuuttia ja virtuaalikoneessa 60 minuuttia kerrallaan. Android-sovelluksen automaatiotestejä varten APK-tiedostosta luodaan toinen APK-testitiedosto, joka ladataan Test Labiin yhdessä sovelluksen varsinaisen APK-tiedoston kanssa. iOS-sovelluksia varten testit käännetään, pakataan zip-tiedostoon ja ladataan Test Labiin.[206] [207]

Test Lab ajaa testejä erissä, joita kutsutaan testimatriiseiksi. Jokainen matriisi sisältää useita laitekonfiguraatioita ja niiden kombinaatioita, joita vasten sovellus testataan. Matriisien käytöllä pyritään varmistamaan hyvä testikattavuus eri laitteissa. Testikokonaisuudet voidaan jakaa useampiin osakokonaisuuksiin, joita suoritetaan rinnakkain useissa laitteissa.[206] [208]

Test Labin oikeiden laitteiden osalta Google noudattaa alan standardien parhaita käytäntöjä. Sovellusdata poistetaan laitteista jokaisen testiajon jälkeen ja varmistetaan, että laitteet ovat valmiina seuraavaa testiajoa varten. Mikäli laitteelle voi ajaa erityisen palautustiedoston, se tehdään testiajojen välissä. Virtuaalikoneiden instanssit poistetaan testiajon jälkeen, joten jokainen uusi testiajo käyttää uutta virtuaalikoneinstanssia. Laitteet sijaitsevat Googlen data centerissä.[204] [206]

Firebasen dokumentaatiosta ei löydy tietoa tukeeko palvelu sovellusten manuaalitestausta. Robon heikkoutena on se, että se on käytössä vain Android-sovelluksille.

5.4.7. Visual Studio App Center

App Center Test, joka tunnetaan myös Test Cloudina, on Microsoft Azureen perustuva testausautomaatiopalvelu natiivi- ja hybridisovelluksille. App Center Test on osa Microsoftin Visual Studio App Centeriä, joka on useista palveluista koostuva pilviympäristö mm. iOS-, Android-, Windows- ja macOS-sovellusten kehitykseen,

[205] https://firebase.google.com/docs/test-lab/android/robo-ux-test
[206] https://firebase.google.com/docs/test-lab/android/overview
[207] https://firebase.google.com/docs/test-lab/ios/command-line#build_xctests_for_your_app
[208] https://firebase.google.com/docs/test-lab/android/instrumentation-test

testauksen, toimitukseen ja analysointiin. Visual Studio App Center on uuden sukupolven versio entisestä Xamarin Test Cloudista.[209] [210] [211]

App Center Test tukee integraatiota Appiumin, Espresson, Xamarin.UITestin sekä XCUITestin kanssa. Testiajoista otetaan automaattisesti kuvankaappaukset, jotka liitetään mukaan testiaskeleisiin. Kaikkien testiajossa mukana olleiden laitteiden kuvankaappauksia voi vertailla rinnakkain samassa näkymässä, ja laitelokit on nähtävissä testeissä ilmenneiden vikojen selvittämistä varten.[212] [209]

Sovelluksen binääritiedostot ja testitiedostot ladataan palvelimelle ja testit suoritetaan datakeskuksissa sijaitsevissa mobiililaitteissa. Testitulokset säilytetään kuusi kuukautta, ja sitä vanhemmat raportit poistetaan. Testiajon jälkeen kaikki asennetut sovellukset poistetaan, muisti tyhjennetään ja laitteen oletusasetukset palautetaan. App Center Test ei tällä hetkellä tue manuaalitestausta, verkkoyhteyksien simulointia, soluverkkoyhteyksiä, VPN-yhteyksiä tai kuormitustestausta.[209] [211]

App Center Testin dokumentaatio ei tarjoa tietoa onko julkisen laitepilven lisäksi mahdollista muodostaa yksityistä pilveä tai paikallista pilveä, tai tukeeko palvelu manuaalitestausta.

5.4.8. Perfecto mobile

Perfecto on jatkuvan testaamisen alusta natiivi-, hybridi- ja web-sovelluksille oikeissa laitteissa, emulaattoreissa ja simulaattoreissa. Perfecto Smart Lab on laitepilvi, joka mahdollistaa todellisten olosuhteiden simuloinnin esimerkiksi verkkoyhteyksien, sensorien, sovellusten aiheuttamien keskeytysten, akun varauksen sekä laitteen suuntauksen osalta manuaali- ja automaatiotestauksessa. Julkisen pilven lisäksi Perfecton laitteista voi muodostaa myös yksityisen pilven tai hybridipilven. Perfecto mahdollistaa myös monimutkaisempien testitapausten, kuten kasvontunnistuksen, suorittamisen. Avoimen API-rajapinnan kautta Perfecto tukee integraatiota mm. Appiumin, Espresson, XCUITest KIF:n, Detoxin, Jenkinsin ja Jiran kanssa.[213] [214] [215] [216] [217]

Quantum on Perfecton monialustatestaukseen soveltuva automaatiotyökalukehys. Quantumilla on mahdollista luoda testiskriptejä eri ohjelmistonkehitystyökaluja, kuten Eclipseä tai IntelliJ IDEA:aa käyttämällä. Quantum tukee Javaa ja JavaScriptiä sekä Gerkhin-syntaksia. Testit voi luoda BDD-menetelmän mukaisesti. Työkalukehys integroituu CI/CD-työkalujen kanssa sekä sisältää sisäänrakennetun integraation Perfecton laitepilven kanssa. Quantum muodostaa testiajoista monipuoliset testiraportit ja tukee vikojen juurisyiden analysointia. Quantum on avoimen lähdekoodin ohjelmisto.[218] [219]

---

[209] https://docs.microsoft.com/en-us/appcenter/test-cloud/
[210] https://www.azuredevopslabs.com/labs/vstsextend/appcenter/
[211] https://visualstudio.microsoft.com/app-center/faq/
[212] https://docs.microsoft.com/en-us/appcenter/test-cloud/supported-frameworks
[213] https://www.perfecto.io/products/platform/cloud-based-testing-lab
[214] https://www.perfecto.io/feature-page-real-user-simulation
[215] https://www.perfecto.io/integrations
[216] https://www.perfecto.io/automation-testing
[217] https://www.perfecto.io/pricing/subscription-plans
[218] https://www.perfecto.io/integrations/quantum
[219] https://developers.perfectomobile.com/display/PSC/Quantum

Perfecton testiraportit sisältävät videoita, kuvankaappauksia, lokit sovelluksen suorituksen epäonnistumisesta sekä laitteen tilasta. Testiraportit mahdollistavat mm. vikojen juurisyiden analysoinnin, testitulosten yhdistämisen yhdeksi raporttialustaksi avoimen API-rajapinnan kautta, sovelluksen ulkoasun visuaalisen validoinnin, lämpökartat, väärien testitulosten suodattamisen ja CI-putkessa olevien testien suorituksen tarkastelun yhdessä näkymässä.[220]

Perfecto soveltuu graafisen käyttöliittymän testaamisen lisäksi mm. toiminnalliseen testaamiseen ja saavutettavuuden testaamiseen. Saavutettavuustesteistä Perfecto luo saavutettavuustestausraportin, jonka avulla kehittäjien on helpompi löytää saavutettavuuteen liittyvät ongelmat sovelluksesta.[221] [222]

Perfecton pilvilaitteista ei ole nähtävillä laitteiden ohjelmistoversiotietoa, minkä vuoksi on vaikea luotettavasti arvioida laitevalikoiman soveltuvuutta pankkisovelluksen testaamiseen.[223]

5.4.9. Experitest

Experitest on palvelu, joka tarjoaa työkaluja natiivi-, hybridi- ja web-sovellusten ja web-sivujen manuaali- ja automaatiotestaukseen. Experitest soveltuu smoke-, sanity- ja yksikkötestaukseen sekä toiminnalliseen ja ei-toiminnalliseen testaukseen, suorituskykytestaukseen ja saavutettavuustestaukseen. SeeTest Digital Assurance Lab on Experitestin laitepilvi, joka sisältää oikeiden laitteiden lisäksi emulaattoreita ja simulaattoreita. Julkisen laitepilven lisäksi käyttöön on mahdollista saada myös yksityinen pilvi tai paikallinen testilaboratorio.[224] [225] [226] [227]

SeeTest tukee integraatiota automaatiotestaustyökalukehysten, kuten Appiumin, XCUITestin, Espresson ja JUnitin kanssa. Sovelluskoodia voi ajaa ja debugata pilvilaitteissa suoraan kehitysympäristöstä, kuten XCodesta tai Android Studiosta.[225] [227] [228]

SeeTestin raporttityökalussa on mahdollista nähdä samassa näkymässä kaikki omien sovellusten tiedot, kuten testikattavuuden tai testin suorituksen tilanteen, sekä hallinnoida projekteja ja testiraportteja. SeeTest tarjoaa automaattisen juurisyiden analyysin, minkä perusteella testitulokset voidaan ryhmitellä esimerkiksi testin hylkäämisen syyn, laitteen tyypin tai käyttöjärjestelmän version perusteella. SeeTest automaattisesti tunnistaa ja luokittelee sovelluksen epäonnistuneen testiajon syyt, kuten ongelmat stabiiliudessa, suorituksessa tai toiminnallisuudessa.[229]

SeeTest tukee kosketuseleitä, ja niiden lisäksi käyttäjän on mahdollista mm. uudelleenkäynnistää laite, palauttaa USB-yhteyden alkuasetukset tai katkaista yhteys, ja hallita sekä virtuaalisia että fyysisiä näppäimiä, soittaa tai lähettää tekstiviestejä. Testauksessa voi simuloida eri lokaatioita ja verkkoyhteysprofiileja tai testata sisäänkirjautumista esimerkiksi sormenjäljellä, salasanalla, PIN-koodilla tai

---

[220] https://www.perfecto.io/platform/analytics-reporting
[221] https://www.perfecto.io/functional-testing-web-mobile-apps
[222] https://www.perfecto.io/accessibility-testing
[223] https://www.perfecto.io/supported-devices
[224] https://digital.ai/experitest
[225] https://experitest.com/mobile-cloud-testing/seetestcloud-online/
[226] https://experitest.com/mobile-device-browser-lab/
[227] https://experitest.com/datasheet/
[228] https://experitest.com/android-studio-and-xcode/
[229] https://experitest.com/mobile-cloud-testing/seetest-reporter/

testikuvalla, kuten viivakoodilla. Laitteiden prosessorin, muistin ja akun tilaa voi tarkkailla testiajon aikana.[227]

Ohjelmointikielillä kirjoitettujen testien lisäksi on mahdollista nauhoittaa ja toistaa manuaalisia testejä ja generoida testiskriptiä nauhoituksen perusteella automaattisesti. Experitest tukee sovelluksen elementtirakenteen tutkimista ja identifioimista sekä tunnisteiden luomista elementeille. Testiskriptiä voi verifioida ja analysoida erilaisten näkymien avulla.[227]

SeeTest on ISO 27001- ja SOC 2 -sertifioitu palvelu. Se tukee mm. TLS- ja SSL-protokollia, monivaiheista todentamista, kustomoitavaa laitteiden tyhjennysprosessia, pääsyn rajoittamista palveluun, laitteiden ei-toivotun päivittämisen estämistä, LDAP (Lightweight Directory Access Protocol) -protokollaa sekä salasanan vanhenemista.[227]

6. POHDINTA

Tässä tutkielmassa etsittiin ratkaisua siihen, *mitä vaatimuksia graafisen käyttöliittymän automaatiotestaamiselle on pankkitoiminnan kaltaisten monimutkaisten järjestelmien kehittämisessä, millaisia työkaluja on käytettävissä mobiilitestauksen automatisointiin, ja kuinka responsiivisen mobiilisovelluksen automaattinen testaaminen on toteutettavissa.* Tutkielmassa pyrittiin mobiilisovelluksen testaamiseen soveltuvia testaustyökaluja ja pilvipohjaisia mobiilitestausalustoja vertailemalla löytämään työkalut ja mobiililaiteympäristö, joita käyttämällä pankin natiivin mobiilisovelluksen graafisen käyttöliittymän testiautomaatio on mahdollista toteuttaa ja suorittaa sekä iOS- että Android-käyttöjärjestelmille.

Työkalut ovat kehityksessään eri vaiheissa. Osa on vielä kehityksen alkuvaiheessa ja ominaisuuksiltaan puutteellisia mutta kehityskelpoisia, toiset jo pitkälle kehittyneitä ja monipuolisia. Työkalujen valinnassa täytyy nykyhetken lisäksi pohtia myös testausratkaisun soveltuvuutta tulevaisuuteen.

Mobiilisovellusten automaatiotestaustyökaluja on paljon ja ne sopivat eri tarkoituksiin. Olemassa olevien työkalujen ominaisuudet kehittyvät koko ajan ja uusia työkaluja tulee markkinoille jatkuvasti. Testausratkaisun ajan tasalla pitäminen vaatii työkalujen tarjonnan ja kehityksen säännöllistä seuraamista. Työkalujen vertaileminen ja analysointi aiheuttaa haasteita isollekin yritykselle, sillä se vie paljon aikaa ja resursseja. Testausratkaisun evaluointi on kuitenkin tärkeää, sillä mobiiliympäristöjen ja työkalujen muutos on jatkuvaa.

Työkalujen arviointia varten on tunnettava pankkisovelluksen monimutkainen teknologia ja sen asettamat vaatimukset. Mobiilialustan ja -laitteiden nopea kehityssykli vaatii testaustyökaluilta joustavuutta. Vaivaton integrointi sovelluskehitystyökalujen kanssa on oleellista, ja työkalujen on tuettava natiivien Android- ja iOS-sovellusten graafisen käyttöliittymän responsiivisuuden testaamista. Responsiivisuustestauksen kattavuus täytyy varmistaa riittävän laajalla mobiililaitevalikoimalla, joka toimii automaatiotestien ajoympäristönä.

Mobiilisovelluksen testauksen automatisointi voidaan toteuttaa monella tavalla. Automaatiotestit voi luoda alustakohtaisesti tai monialustayhteensopivaa ratkaisua käyttäen. Testien suoritus voi tapahtua oman testilaboratorion mobiililaitteissa, pilvipalvelun tarjoamissa laitteissa tai simulaattoreissa ja emulaattoreissa.

OP:n mobiilisovelluksen tapauksessa on järkevintä käyttää kummallekin sovellusalustalle omaa automaatiotestaustyökalukehystä. Alustaspesifisillä työkalukehyksillä on keskimäärin erittäin hyvä testien suoritusnopeus, ja ne ovat helposti integroitavissa ja konfiguroitavissa käytettävän ohjelmistoalustan ja kehitystyökalujen kanssa. Testiskriptien luominen tapahtuu samalla ohjelmointikielellä kuin sovelluksen kehittäminen, joten testausautomaation tekemisen oppimissykli on nopea, varsinkin jos testien kehittäjällä on kokemusta sovelluksen koodin parissa työskentelystä. Testaajien lisäksi myös sovelluskehittäjät tekevät automaatiotestejä, jolloin käytettyjen työkalujen ja ohjelmointikielien tulisi olla yhteensopivia sovelluksen kehitysympäristön kanssa. Natiivien työkalujen etuna on lisäksi se, että ne tukevat sovelluksen natiivien ominaisuuksien testaamista ja ovat vakaita ja luotettavia. Testit ovat osa sovelluksen lähdekoodia, eikä niitä varten tarvitse ylläpitää erillistä tietovarastoa.

Vaikka alustaspesifistä työkalukehystä on suositeltavaa käyttää pankkisovelluksen testaamiseen, on myös monialustatestaukseen soveltuvissa testaustyökalukehyksissä

useita vahvuuksia, ja ne sopivat ominaisuuksiltaan pankkisovelluksen testausautomaation tekemiseen. Kolmannen osapuolen työkalut voivat kuitenkin lakata toimimasta silloin kun kehittäjät ottavat Googlen ja Applen tekemät SDK:n päivitykset käyttöönsä, ja tilanne korjaantuu vasta siinä vaiheessa, kun työkalujen toimittajat saavat korjaukset tehtyä ja julkaistua.

Android- ja iOS-sovellusversiot eroavat jonkin verran toisistaan esimerkiksi komponenttien osalta, joten yhteisen työkalun käyttö vaatisi mm. nimeämiskäytäntöjen yhtenäistämistä. Monialustayhteensopivat työkalukehykset eivät suoraan tue sovellusten natiivien ominaisuuksien testausta, mutta osa niistä on ratkaissut ongelman käyttämällä natiivien työkalukehysten metodeja. Kaikki ylimääräiset työkalukerrokset kuitenkin hidastavat testien suoritusta. OP:n mobiilisovelluksen tapauksessa monialustatestaustyökalukehysten edut eivät toistaiseksi ole kilpailukykyisiä alustakohtaisten työkalukehysten etujen kanssa, ja huonot puolet hankaloittavat testien kehittämistä ja suorittamista.

Responsiivisuustestauksen laitekattavuuden varmistamisen osalta pilvipohjaiset mobiilitestausalustat eivät tällä hetkellä pysty kilpailemaan oman mobiililaitelaboratorion kanssa pankkisovelluksen testauksessa. Pilvipalveluun tulee uusia laitteita ja käyttöjärjestelmäversioita hitaasti, eikä kaikkia haluttuja laitemalleja ja käyttöjärjestelmiä ole saatavilla. Varsinkin vanhempien laite- ja ohjelmistoversioiden valikoima vaihtelee suuresti eri laitepilvien välillä. Laitepilven käyttö on hidasta omaan testilaboratorioon verrattuna, sillä yhteyden muodostaminen pilvipalvelun laitteisiin vie aikaa, yhteys saattaa olla epästabiili, ja laitteita voi joutua jonottamaan. Pilvipalvelun lisenssin tyypistä riippuen myös esimerkiksi testien ajoaikaa ja rinnakkaisessa testiajossa käytettävien laitteiden määrää on voitu rajoittaa.

Omaa mobiililaitteistoa voi päivittää ja uudistaa nopeasti aina kun uusia laitteita ja ohjelmistoversioita tulee markkinoille. Kustannuksiltaan ja ylläpidoltaan järkevä testilaitteiden määrä on kuitenkin melko rajallinen, ja laitteet valitaan suosituimpien ja uusimpien markkinoilla olevien iOS- ja Android -mobiililaitteiden perusteella.

Omaa rajallista testilaitelaboratoriota on suositeltavaa täydentää simulaattoreilla ja emulaattoreilla, eli virtuaalisilla mobiililaitteilla. Simulaattorit ja emulaattorit tulevat saataville nopealla aikataululla, mutta ne eivät kuitenkaan täysin vastaa fyysisiä laitteita, joten oman testilaitteiston käyttö pankkisovelluksen testaamisessa on perusteltua.

Jatkossa olisi hyödyllistä luoda automaatiotestauksen menetelmien kehittämiselle oma prosessi ja esimerkiksi nimetä vastuuhenkilö, joka seuraa työkalujen tarjonnassa ja kehityksessä tapahtuvia muutoksia. Kansainvälisiä testiautomaatiokonferensseja järjestetään säännöllisesti, ja niihin osallistuminen voisi tuoda uusia näkökulmia ja ideoita prosessien ja toiminnan kehittämiseen sekä näkyvyyttä työkalujen evoluutioon.

Mikäli tulevaisuudessa löytyy käytössä olevaan työkalukehykseen verrattuna ominaisuuksiltaan parempi työkalukehys, tulisi sen käyttöönoton aiheuttamia vaikutuksia kartoittaa perusteellisesti. Työkalua tulisi myös testata pankkisovellusympäristössä ennen kuin päätöksiä käyttöönotosta tehdään.

Työkaluvertailun tekemisessä on muutamia haasteita. Kun vertailussa on mukana useita työkaluja, on niiden testaaminen monimutkaisessa pankkijärjestelmäympäristössä erittäin työlästä ja haastavaa. Puolueettomia vertailuja löytyy vain muutama, vertailuissa on mukana vain suppea valikoima työkalujen ominaisuuksista, eikä työkaluvalikoima vastaa tässä tutkielmassa mukana olevia työkaluja. Tästä johtuen tutkielmassa tehty vertailu perustuu suurelta osin työkalujen valmistajien tarjoamiin tietoihin ja satunnaisiin internetistä löytyneisiin

käyttäjäarvosteluihin. Työkaluihin pitäisi kuitenkin tutustua perusteellisemmin ja niitä pitäisi käyttää pankkisovelluksen oikean testausautomaation toteuttamiseen, jotta niiden vahvuudet ja heikkoudet tulisivat kunnolla ilmi.

Erityisesti työkalujen huonoista puolista ja puutteista on vaikea löytää luotettavaa ja objektiivista tietoa. Työkalujen omien sivujen sisältämiin kehuihin kyseisen tuotteen ylivoimaisuudesta verrattuna kilpailijoihin on syytä suhtautua kriittisesti. Työkalujen testauksen puuttuminen on tässä tutkielmassa tehdyn vertailun heikkous, mutta testauksen vaatima aika ja resurssit on todettu yhdessä työn teettäjän kanssa liian suuriksi, jotta sitä voisi toteuttaa tämän tutkielman puitteissa.

Saatavilla olevan tiedon kattavuus ja tarkkuus vaihtelee huomattavasti työkalujen välillä, esimerkiksi tietoturvan kuvauksen tasossa on selviä eroja. Myös ominaisuuksien painotuksessa on työkalukohtaisia eroja. Työkalujen dokumentaatiota tai niistä kertovia artikkeleita ei useinkaan ole päivätty, joten on mahdotonta sanoa mitä työkaluversiota kyseinen esittely koskee ja kannattaako sitä käyttää tutkielman lähdemateriaalina. Toisinaan eri lähteissä esitetty tieto voi olla ristiriidassa keskenään, jolloin on vaikea saada selville mikä lähde on validi.

Eri pankkien sovellusten tekniset ratkaisut ja palveluiden rakenne vaihtelevat. Eri tiimeillä on erilaiset vaatimukset testausautomaatiotyökaluille. Teknisten vaatimusten lisäksi voi olla muitakin ehtoja, jotka on otettava huomioon työkalujen valinnassa. Tästä johtuen on erittäin hankalaa kehittää täsmällistä artefaktia, jonka perusteella testausratkaisun valinta on parasta tehdä. Yksi työkalu ei sovellu kaikkien ongelmien ratkaisuun, ja jokaisessa työkalussa on sekä hyviä että huonoja puolia. Parhaan ratkaisun löytäminen on aina kontekstista ja yrityksen kulttuurista riippuvainen.

Tässä tutkielmassa on keskitytty ennen kaikkea työkalujen ominaisuuksiin, jotka koskevat natiivin mobiilisovelluksen automaatiotestausta. Työkalut ovat ominaisuuksiltaan tässä esitettyä monipuolisempia. Hybridi- ja web-sovelluksiin sekä manuaalitestaukseen liittyvät asiat on jätetty useimmiten käsittelemättä. Työkaluvertailuun on valittu mukaan muutamia yleisesti käytössä olevia, tunnettuja ja tarkoitukseen sopivia työkaluja ja pilvipalveluita, mutta kaikkia olemassa olevia mobiilisovellusten automaatiotestaustyökaluja tämä tutkielma ei käsittele.

Muiden pankkien sovellukset, palvelurakenteet ja työkulttuurit eroavat OP:n mobiilisovelluksesta, mutta natiivisovelluksen teknologia on kaikille yhteinen. Tässä tutkielmassa yhden pankin mobiilisovelluksen kautta lähestyttiin yleistä ongelmaa. Tutkielman näkökulmaa olisi voinut laajentaa ottamalla toisten pankkien mobiilisovellusten testausautomaatio mukaan, mutta tehdyistä rajauksista huolimatta tutkielman tulokset ja johtopäätökset ovat kohtuudella toistettavissa ja yleistettävissä myös muiden pankkien mobiilisovellusten automaatiotestaukseen.

7. YHTEENVETO

Tässä tutkielmassa etsittiin ratkaisua pankin mobiilisovelluksen graafisen käyttöliittymän responsiivisuustestauksen ongelmaan. Mobiililaitteiden eri mallit, käyttöjärjestelmät, käyttöjärjestelmäversiot, näyttöjen koot ja resoluutiot muodostavat valtavan määrän erilaisia asiakkaiden käytössä olevia laiteprofiileja, joihin sovelluksen tulisi adaptoitua. Manuaalitestaus on hidasta ja työlästä, ja se on suoritettava uudestaan aina kun siirrytään käyttämään uutta käyttöjärjestelmäversiota tai mobiilisovellukseen tehdään muutoksia. Laitekirjosta johtuen mobiilisovelluksen toiminta on verifioitava useilla laitemalleilla ja käyttöjärjestelmäversioilla, mikä kuormittaa testaajia huomattavasti. Oman testauslaitteiston hankkiminen ja ylläpito aiheuttaa lisäksi kustannuksia ja vaatii resursseja.

Tutkielmassa perehdytään mobiilisovelluksen kehittämisen ja testaamisen periaatteisiin ja tutustutaan pankkijärjestelmän erityispiirteiden asettamiin vaatimuksiin testausratkaisun valinnassa. Ratkaisua testauksen ongelmaan haetaan vertailemalla sekä yhdelle ohjelmistoalustalle suunnattuja että monialustayhteensopivia automaatiotestaustyökalukehyksiä ja pilvipohjaisia mobiilitestausalustoja.

Tämän tutkielman johtopäätöksenä on, että pankin mobiilisovelluksen testauksen automatisoinnissa paras ratkaisu on testausautomaation kehittäminen alustakohtaista testaustyökalukehystä käyttämällä, ja pilvipohjaiset mobiilitestausalustat eivät toistaiseksi pysty kilpailemaan oman testilaitelaboratorion kanssa. Omien testilaitteiden määrä on kuitenkin melko pieni, joten testien laitekattavuutta on suositeltavaa täydentää emulaattoreilla ja simulaattoreilla.

Testaustyökalut ja pilvipohjaiset palvelut kehittyvät jatkuvasti, ja uusia tulee markkinoille koko ajan. Jatkossa onkin tärkeää, että työkalujen kehitystä seurataan ja testausratkaisua evaluoidaan säännöllisesti.

Tässä tutkielmassa tehdyt analyysit antavat pohjan pankin mobiilisovelluksen graafisen käyttöliittymän testausautomaatioratkaisun valinnalle. Valintakriteerit on koottu taulukoihin, joiden perusteella työkaluvertailua on mahdollista tehdä. Tiimien ja organisaatioiden tarpeet vaihtelevat, joten yksiselitteisten kriteerien kehittäminen testausratkaisun valintaan on haasteellista. Paras ratkaisu on aina kontekstista, projektin vaatimuksista ja työyhteisön kulttuurista riippuvainen.

Tässä tutkielmassa työkaluvertailun motivaattorina toimii OP:n mobiilisovellus, josta on kehitetty oma sovellusversio sekä Android- että iOS-käyttöjärjestelmille. Muiden pankkien mobiilisovellusten testausautomaatioratkaisuja ei käytetty vertailun pohjana. Tutkielman tulokset ja johtopäätökset ovat kuitenkin toistettavissa ja yleistettävissä myös muiden pankkien mobiilisovellusten testauksen automatisointiin.

8. LÄHTEET

[1] D. Knott, Hands-On Mobile App Testing: A guide for mobile testers and anyone involved in the mobile app business, New York: Addison-Wesley Professional, 2015.

[2] N. A. Sulaiman ja M. Kassim, ”Developing a customized software engineering testing for Shared Banking Services (SBS) System,” tekijä: *2011 IEEE International Conference on System Engineering and Technology, pp. 132-137*, Shah Alam, 2011.

[3] D. Kumar ja K. K. Mishra, ”The Impacts of Test Automation on Software’s Cost, Quality and Time to Market,” Procedia Computer Science, pp. 8-15, Mumbai, 2016.

[4] Z. Xu, K. Gao, T. M. Khoshgoftaar ja N. Seliya, ”System regression test planning with a fuzzy expert system,” osa/vuosik. 259, pp. 532-543, 2014.

[5] K. Piirainen and R. A. Gonzalez, "Seeking Constructive Synergy: Design Science and the Constructive Research Approach," in *Design Science at the Intersection of Physical and Virtual Design, pp. 59-72*, Helsinki, 2013.

[6] A. R. Hevner, S. T. March, J. Park and S. Ram, "Design Science in Information Systems Research," MIS Quarterly, pp. 75-105, 2004.

[7] M. Nagappan ja E. Shihab, ”Future Trends in Software Engineering Research for Mobile Apps,” tekijä: *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), pp. 21-32*, Suita, 2016.

[8] S. L. Lim, P. J. Bentley, N. Kanakam, F. Ishikawa ja S. Honiden, ”Investigating Country Differences in Mobile App User Behavior and Challenges for Software Engineering,” *IEEE Transactions on Software Engineering,* osa/vuosik. 41, nro 1, pp. 40-64, 2015.

[9] L. Delia, N. Galdamez, P. Thomas, L. Corbalan ja P. Pesado, ”Multi-Platform Mobile Application Development Analysis,” tekijä: *2015 IEEE 9th International Conference on Research Challenges in Information Science (RCIS), pp. 181-186*, Ateena, 2015.

[10] J. Fonseca ja M. Vieira, ”A Practical Experience on the Impact of Plugins in Web Security,” tekijä: *2014 IEEE 33rd International Symposium on Reliable Distributed Systems, pp. 21-30*, Nara, 2014.

[11] I. Malavolta, ”Beyond native apps: web technologies to the rescue! (keynote),” tekijä: *Mobile! 2016- Proceedings of the 1st International Workshop on Mobile Development, pp. 1-2*, Amsterdam, 2016.

[12] I. Malavolta, G. Procaccianti, P. Noorland ja P. Vukmirovic´, ”Assessing the Impact of Service Workers on the Energy Efficiency of Progressive Web Apps,” tekijä: *MOBILESoft '17 Proceedings of the 4th International Conference on Mobile Software Engineering and Systems, pp. 35-45*, Buenos Aires, 2017.

[13] N. Serrano, J. Hernantes ja G. Gallardo, ”Mobile Web Apps,” *IEEE Software,* osa/vuosik. 30, nro 5, pp. 22-27, 2013.

[14] C. Mullins, ”Responsive, mobile app, mobile first: untangling the UX design web in practical experience,” tekijä: *SIGDOC '15 Proceedings of the 33rd*

Annual International Conference on the Design of Communication, pp. 1-6, Limerick, 2015.

[15] H. Desruelle ja F. Gielen, ”Context-driven Progressive Enhancement of Mobile Web Applications: a Multicriteria Decision Making Approach,” Computer Journal, pp. 1732-1746, 2015.

[16] L. F. Gonçalves, L. G. Vasconcelos, E. V. Munson ja L. A. Baldochi, ”Supporting Adaptation of Web Applications to the Mobile Environment with Automated Usability Evaluation,” tekijä: *SAC '16 Proceedings of the 31st Annual ACM Symposium on Applied Computing, pp. 787-794*, Pisa, 2016.

[17] B. O. Turan ja K. Şahin, ”RESPONSIVE WEB DESIGN AND COMPARATIVE ANALYSIS OF DEVELOPMENT FRAMEWORKS,” *The Turkish Online Journal of Design Art and Communication,* osa/vuosik. 7, nro 1, pp. 110-121, 2017.

[18] E. Marcotte, Responsive Web Design, New York: A Book Apart, 2011.

[19] S. L. Henry, S. Abou-Zahra ja J. Brewer, ”The Role of Accessibility in a Universal Web,” tekijä: *W4A '14 Proceedings of the 11th Web for All Conference, pp. 1-4*, Soul, 2014.

[20] O. C. Novac, M. Novac, C. Gordan, T. Berczes ja G. Bujdosó, ”Comparative study of Google Android, Apple iOS and Microsoft Windows Phone mobile operating systems,” tekijä: *2017 14th International Conference on Engineering of Modern Electric Systems (EMES), pp. 154-159*, Oradea, 2017.

[21] R. Coppola, M. Morisio ja M. Torchiano, ”Mobile GUI Testing Fragility: A Study on Open-Source Android Applications,” *IEEE Transactions on Reliability,* osa/vuosik. 68, nro 1, pp. 67-90, 01 lokakuu 2018.

[22] Y. Li, W. Cui, D. Li ja R. Zhang, ”Research based on OSI model,” tekijä: *2011 IEEE 3rd International Conference on Communication Software and Networks, pp. 554-557*, Xi'an, 2011.

[23] A. Memon, I. Banerjee ja A. Nagarajan, ”GUI Ripping: Reverse Engineering of Graphical User Interfaces for Testing,” tekijä: *Proceedings 10th Working Conference On Reverse Engineering, pp. 260-269*, Victoria, 2003.

[24] R. Coppola, M. Morisio ja M. Torchiano, ”Scripted GUI Testing of Android Apps: A Study on Diffusion, Evolution and Fragility,” tekijä: *PROMISE: Predictive Modelling in Software Engineering, pp. 22-32*, Toronto, 2017.

[25] M. Linares-Vasquez, ”Enabling Testing of Android Apps,” tekijä: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, pp. 763-765*, Firenze, 2015.

[26] A. Memon, I. Banerjee, N. B. N. ja B. Robbins, ”The first decade of GUI ripping: Extensions, applications, and broader impacts,” tekijä: *2013 20th Working Conference on Reverse Engineering (WCRE), pp. 11-20*, Koblenz, 2013.

[27] J. Gao, X. Bai, W.-T. Tsai ja T. Uehara, ”Mobile Application Testing: A Tutorial,” *Computer,* osa/vuosik. 47, nro 2, pp. 46-55, 06 January 2014.

[28] Meiliana, I. Septian, R. S. Alianto ja Daniel, ”Comparison Analysis of Android GUI Testing Frameworks by Using an Experimental Study,” *Procedia Computer Science,* osa/vuosik. 135, pp. 736-748, 2018.

[29] L. Cruz ja R. Abreu, ”Measuring the energy footprint of mobile testing frameworks,” tekijä: *ICSE '18: Proceedings of the 40th International*

Conference on Software Engineering: Companion Proceeedings, pp. 400-401, Göteborg, 2018.

[30] A. A. Tirodkar ja S. S. Khandpur, ”EarlGrey: iOS UI Automation Testing Framework,” tekijä: *2019 IEEE/ACM 6th International Conference on Mobile Software Engineering and Systems (MOBILESoft), pp. 12-15*, Montreal, 2019.

[31] M. Soeken, R. Wille ja R. Drechsler, ”Assisted Behavior Driven Development Using Natural Language Processing,” tekijä: *Objects, Models, Components, Patterns, pp. 269-287*, Praha, 2012.

[32] D. R. Mohammad, S. Al-Momani, Y. M. Tashtoush ja M. Alsmirat, ”A Comparative Analysis of Quality Assurance Automated Testing Tools for Windows Mobile Applications,” tekijä: *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC), pp. 414-419*, Las Vegas, 2019.

[33] I. C. Morgado ja A. C. R. Paiva, ”Mobile GUI testing,” *Software Quality Journal 2018,* osa/vuosik. 26, nro 4, pp. 1553-1570, 2017.

[34] M. Linares-Vásquez ja K. P. D. Moran, ”Continuous, Evolutionary and Large-Scale: A New Perspective for Automated Mobile App Testing,” tekijä: *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 399-410*, Shanghai, 2017.