TAMPEREEN
AMMATTIKORKEAKOULU

MOBIILISOVELLUKSEN TESTAUS-MENETELMÄT

Niko Koli

Opinnäytetyö
Toukokuu 2018
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka

NIKO KOLI:
Mobiilisovelluksen testausmenetelmät

Opinnäytetyö 23 sivua
Toukokuu 2018

Opinnäytetyössä tutkitaan sovellusten testausta osana ohjelmistoprojektia. Työssä keskitytään testauksen vaiheisiin ja menetelmiin, joita voidaan käyttää apuna mobiilisovellusten testauksessa. Työssä tutustutaan myös suosituimpiin testityökaluihin, jotka ovat osittain yhteensopivia React Native sovellusten kanssa.

Asiasanat: ohjelmistoprojekti, testaus, v-malli, testilähtöinen kehitys, tutkiva testaus, automaatiotestaus, testausmenetelmä

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Information Technology Degree Program
Software Engineering

NIKO KOLI:
Mobile Application Testing Methods

Bachelor's thesis 23 pages
May 2018

The thesis covers application testing as part of a software project. The thesis focuses on the steps and methods of testing that can be used to test mobile applications. The study also introduces the most popular test tools that are partially compatible with React Native applications.

Key words: software project, testing, v-model, test-driven development, exploratory testing, automation testing, test method

SISÄLLYS

1 Johdanto

Testaus on suunnitelmallista virheiden etsimistä ohjelmaa tai sen osaa suorittaen. Sovelluksen virheettömyyttä ei voida kuitenkaan täysin varmistaa testaamalla edes yksinkertaisissa tapauksissa, testaukseen kannattaa silti panostaa. Testauksella voidaan osoittaa virheiden olemassaolo ja todennäköisyys niiden esiintymiselle.

Haltulla toteutetaan vuoden aikana useita mobiilisovelluksia erilaisille yrityksille ja yhdistyksille, joista yhtäkään ei testata järjestelmällisesti. Projektit ovat usein verrattain pieniä, eikä testaukselle ole varattu budjettia. Asiakas ei välttämättä arvosta testausta tai sovelluskehitykseen osallistuvat henkilöt eivät osaa vaatia budjettia testaukselle. Suurin osa sovelluksista toteutetaan Android- ja iOS-käyttöjärjestelmille, siksi sovelluskehykseksi valitaan React Native. Työpaikan ilmapiiri on yksi selvimmin ulospäin näkyvistä tekijöistä. Jos ilmapiiri on testausmyönteinen, näkyy se ennen pitkää uusille ja vanhoille asiakkaille.

Opinnäytetyössä tutkitaan eri testausmenetelmiä, jotka voivat sopia mobiilisovellusten testaamiseen. Käytännön osassa testataan eri työkaluja, jotka sopivat hybridi sovellusten testaukseen. Luvussa 2 määritetään virhe ja testauksen vaiheet projektissa. Luku 3 käsittelee V-mallin sisältöä. Luvuissa 4 ja 5 tutustutaan testausmenetelmiin ja työkaluihin ja luvussa 6 pohditaan, mitkä tekijät vaikuttavat hybridisovellusten testaamiseen.

2 Testaus osana projektia

Testaus liittyy projektin kaikkiin vaiheisiin myynnistä käyttöönottoon. Testauksen tarkoitus on löytää virheitä dokumentaatiosta, lähdekoodista, liiketoimintalogiikasta ja jopa -prosessista mahdollisimman aikaisessa vaiheessa. Testaus koostuu pääasiassa virheiden jäljityksestä ja niiden korjaamisesta.

Kappaleen alussa määritetään mikä on virhe, mistä niitä yleensä löytyy ja mitä niiden korjaaminen maksaa missäkin projektin vaiheessa. Jälkimmäisissä kappaleissa käsitellään, miten testaus tulisi ottaa huomioon projektin eri vaiheissa.

2.1 Virhe

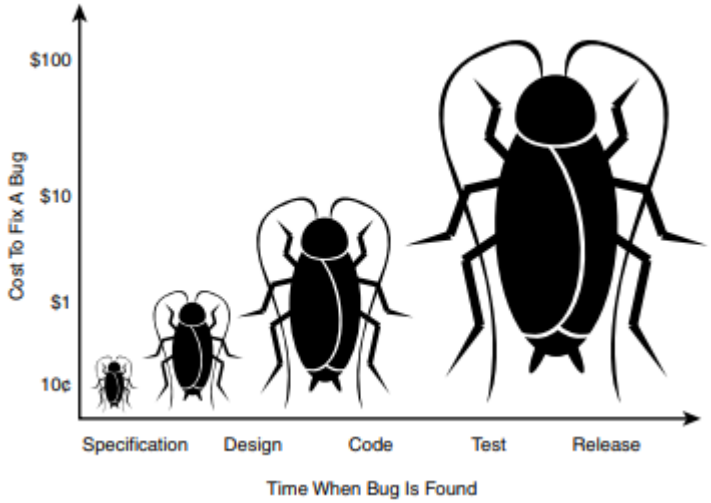
Virhe määritetään tuotteen määrittelyn kautta. Ron Patton listaa kirjassaan (Patton, luku 1) viisi vaihtoehtoa, joista yhdenkin toteutuessa virhe esiintyy.

1. Ohjelmisto ei toimi tavalla, jolla sen määrittelyn mukaan kuuluisi toimia.
2. Ohjelmisto toimii tavalla, jolla sen määrittelyn mukaan ei kuuluisi toimia.
3. Ohjelmisto toimii tavalla, jota määrittely ei mainitse.
4. Ohjelmisto ei toimi tavalla, jota määrittely ei tunne, vaikka sen pitäisi.
5. Ohjelmisto on hankala ymmärtää, vaikeakäyttöinen, hidas tai käyttäjän mielestä väärin toimiva.

Suurin osa virheistä johtuu määrittelystä. Se on puutteellinen, jatkuvasti muuttuva, huonosti selvitetty kehitystiimille tai sitä ei ole lainakaan. Toiseksi suurin virheen syy on suunnittelu ja vain verrattain pieni osa johtuu lähdekoodista. Määrittely on ehdottoman tärkeää ehjän projektin toteuttamisen kannalta. Ilman määrittelyä ei voida testata tai kehittää oikein toimivaa sovellusta.

Virheen vakavuus riippuu sovelluksesta, mistä virhe löytyi ja siitä, kuinka se ilmenee käyttäjälle. Jotkin virheet estävät sovelluksen käytön, ärsyttävät käyttäjää tai ovat vain kosmeettisia. Virheen voidaan luokitella vakavuuden perusteella esimerkiksi kriittisiin, vakaviin ja vähäisiin virheisiin. Vakavuuden luokittelu riippuu työyhteisöstä.

Mitä myöhemmässä vaiheessa virhe havaitaan, sen kalliimpaa sen korjaaminen on. Pattonin mukaan virheen korjauskustannukset nousevat lähinnä logaritmisesti edetessä projektin seuraavaan vaiheeseen. Järjestelmätestauksen aikana löytynyt virheellinen komponentin korjaus voi vaikuttaa myös muihin osajärjestelmän komponentteihin. Testaus tulee suorittaa uudelleen jokaisella testaustasolla mahdollisten uusien virheiden löytämiseksi.



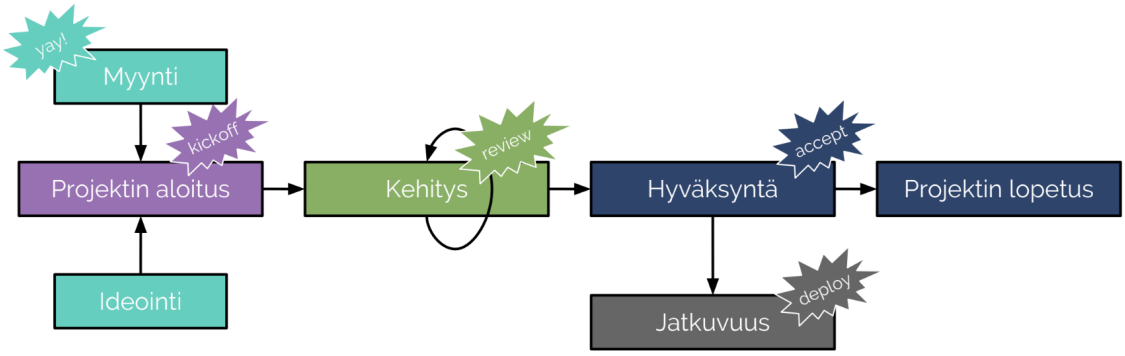
KUVA 1. Virheen hinta projektin edetessä (Patton, luku 1)

Erkki Hietalahti listaa luentomonisteessaan (Ohjelmistotuotanto, testaus) kokemuksiin perustuvia havaintoja virheiden määrästä. Kehityksen aikana havaitaan virhe muutamaa kymmentä riviä kohden ja pitkään käytössä olevissa sovelluksissa on yksi virhe tuhatta riviä kohden eikä 5% ohjelmavirheistä koskaan havaita.

2.2 Sovelluksen elinkaari

Projekti on työ, jolla on aina tavoite, tekijät, budjetti ja aikataulu. Kun tavoitteena on tehdä laadukas sovellus, tarvitaan myös testausta. Sille tulisikin varata tarpeeksi aikaa ja resursseja. Testaus tulee ottaa huomioon jo myynnin aikana esitetyissä alustavissa työmääräarvioissa, jotta se voidaan kirjata sopimukseen.

Projektin vaiheet ovat yleensä määrittely, suunnittelu, toteutus, käyttöönotto ja ylläpito. Haltulla toimitaan ketterän kehityksen mukaan seuraavin askelin (kuva 2).



KUVA 2. Sovelluksen elinkaari Haltulla

2.2.1 Myynti ja ideointi

Myynnin aikana kartoitetaan, mitä asiakas oikeasti haluaa saavuttaa. Lisäksi selvitetään asiakkaan organisaatio ja toimintaympäristö sekä kuka tekee päätökset ja mikä on oikea budjetti. Kokonaisuus voi olla täysin uusi kehitysprojekti tai jatkokehitystä vanhaan projektiin.

Ideoinnissa ajatellaan projektin tavoitteita käyttäjän näkökulmasta. Ratkaiseeko palvelu tavoitetun ongelman? Tuottaako se käyttäjälle lisäarvoa? Haltun tehtävä on auttaa ja ideoita, millä tavalla halutut tavoitteet saavutetaan.

Myynnin ja ideoinnin seurauksena syntyy allekirjoitettu sopimus, jonka sisältö on tehty yhteisymmärryksessä ja se on helposti ymmärrettävässä muodossa. Riskin hallinta aloitetaan jo projektin myyntivaiheessa. Kannattaako projekti myydä ja allekirjoittaa, jos sopimuksessa mainituilla ehdoilla sen toteuttaminen ei ole mahdollista ilman tappiota.

2.2.2 Aloituspalaveri

Projektin aloituspalaverissa tehdään kaikille siihen osallistuville selväksi projektin asiakas, budjetti, tavoitteet sekä roolit. Lisäksi selvitetään sovelluksen ajo- ja toimintaympäristö, riippuvuudet sekä aikataulu. Projektin aloituspalaverin tarkoitus vastata kysymykseen miksi projekti on olemassa.

Aloituspalaverin aikana otetaan projektin aikana käytettävät työkalut käyttöön, luodaan asiakas toiminnanohjausjärjestelmään ja luodaan keskustelu kanava sisäiseen ja sidosryhmäviestintään. Toiminnanohjausjärjestelmään kirjataan lisäksi vaatimukset sekä sopimus liitteineen.

2.2.3 Kehitys

Tehtävän taustalla on vaatimus, joka perustuu tavoitteeseen. Kun tehtävällä on työlupa, sille annetaan tekijä ja prioriteetti. Tehtävät suoritetaan prioriteettijärjestyksessä. Vaatimus määrittää milloin tehtävä on valmis. Tehtävä katselmoidaan, kun se on valmis ja siirretään arkistoon, jos se on läpäissyt katselmoinnin. Näin syntyy tehtävän dokumentaatio, jonka avulla voidaan selvittää jälkeenpäin mitä on tehty ja miksi.

Kehityksen aikana keskitytään yksikkö ja integraatiotestaukseen. Projektin lähestyessä loppua testataan myös koko järjestelmä määrittelyä vasten.

2.2.4 Hyväksyntä ja projektin jatko

Projekti päättyy, kun tavoitteet on saavutettu. Projektin päättyminen ei välttämättä tarkoita kehityksen lopettamista, vaan esimerkiksi tietyn osakokonaisuuden valmistumista ja tuotantoon julkaisua.

Projektin hyväksynnässä tarkoitus on selvittää saavutetut tavoitteet, onko asiakas ja kehitystiimi tyytyväinen, miten projekti eteni ja mitä voitaisiin parantaa. Hyväksyntätestaus toteutetaan alussa määriteltyjä tavoitteita vasten. Hyväksynnän aikana syntyneet ideat listataan jatkokehitystä silmällä pitäen.

Jos projekti siirtyy ylläpitoon, kirjoitetaan ylläpitosopimus. Ylläpidon tarkoitus on seurata, että sovellus toimii. Keksitään uusia ideoita ja parannuksia ja tehdään lisämyyntiä.

2.2.5 Dokumentaatio

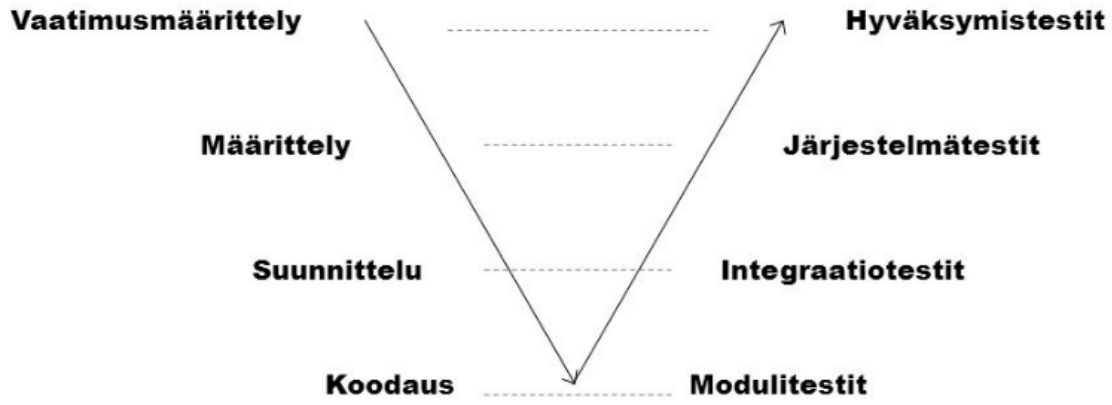
Testauksen suunnittelussa otetaan huomioon määrittelyt, laatujärjestelmä sekä mahdollisuuksien mukaan kokemuksesta muodostunut tarkistuslista ja vanha testaussuunnitelma. Suunnittelun tuloksena syntyy testisuunnitelma, joka vastaa kysymyksiin

- mitä testejä tehdään
- milloin
- missä järjestyksessä
- millaisia lopputuloksia odotetaan

Edellä mainituista kohdista luodaan erilliset testitapaukset, jotka ajetaan järjestyksessä

3 V-malli

V-mallin idea on suunnitella testit niitä vastaavalla suunnittelutasolla (kuva 3). Testit suunnitellaan usein alustavasti ja niitä täydennetään ennen vastaavaa testausvaihetta. Testitulokset todetaan oikeiksi vertaamalla niitä vastaaviin suunnitteludokumentteihin. V-mallia soveltuu projekteihin, jotka kehitetään vesiputousmallin mukaisesti.



KUVA 3. V-malli

Verifiointi tapahtuu V-mallin vasemmassa sarakkeessa. Sen tarkoitus on varmistaa, että sovellus on kehitetty oikein. V-mallin oikeassa sarakkeessa validoidaan, että sovellus vastaa määrittelyä. Validointi vastaa kysymykseen, onko kehitetty oikea sovellus.

Testauksen määrä on kompromissi, joka riippuu käytettävissä olevista resursseista, luotettavuuden varmuudesta, jäljellä olevien vikojen kustannuksista ja projektin myöhästymisestä aiheutuva tuoton menetyksestä. Testauksen määrä on vaikea arvioida etukäteen, eikä se ole sama testauksen tehokkuuden kanssa.

3.1 Yksikkötestaus

Yksikkötestauksen aikana testataan yksittäisten komponenttien, esimerkiksi luokan tai funktion, sisäistä toimintaa ja käyttäytymistä teknistä määrittelyä vasten. Testaajana toimii kyseisen komponentin kehittäjä. Yksikkötestauksessa käytetään testiajuria, joka kutsuu komponenttia annetuilla parametreillä ja vertaa tulosta odotettuun arvoon. Testitulokset kirjataan testiraporttiin hyväksyttynä tai hylättynä.

Yksikkötestaus on kehitystä tukevaa työtä ja sen tarkoitus on havaita virheet implementaatiosta.

3.2 Integraatiotestaus

Integraatiotestaus aloitetaan, kun osajärjestelmään vaadittavat komponentit ovat valmiit. Komponentit koostetaan yleensä kokoavasti tai jäsentävästi osajärjestelmiksi, joita testataan integraatiotestauksessa arkkitehtuurisuunnittelua vasten. Testauksen aikana tavoitteena on löytää virheitä komponenttien rajapinnoista ja keskinäisestä vuorovaikutuksesta.

3.3 Järjestelmätestaus

Järjestelmätestauksessa koko sovellus testataan vaatimusmäärittelyjen suhteen tuotantoa vastaavassa ympäristössä. Järjestelmätestauksen toteuttajien tulee olla kehitystyöstä riippumattomia testaajia, joiden tavoitteena on selvittää sovelluksen ja siihen liittyvien järjestelmien keskinäinen kommunikointi käytön aikana.

Järjestelmätestaus on laajin testaustaso. Se jaetaan eri osa-alueisiin, joille asetetaan omat tavoitteensa, joita vasten sovellusta testataan. Järjestelmätestauksen osa-alueita ovat esimerkiksi rasituksen sieto, tietoturva, käytettävyys, suorituskyky ja luotettavuus.

3.4 Hyväksyntätestaus

Asiakas varmistaa, että sovellus vastaa vaatimusmäärittelyä eikä siinä ole puutteita. Toimittajan pitäisi olla testannut ehtojen täyttyminen jo järjestelmätestauksen aikana. On tärkeää, että asiakas on itse määritellyt hyväksymistestauksen testitapaukset tai ainakin ne huolellisesti katselmoinut. Hyväksyntätestaus tehdään asiakkaan tuotantoympäristössä.

Yleensä hyväksymistestauksessa on kaksi vaihetta: alfa- ja betatestaus. Alfatestaus suoritetaan toimittajan tiloissa ja betatestaus asiakkaan johdolla sen omissa tiloissaan. Mobiilisovellukset betatestataan yleensä avoimella tai suljetulla betatestillä, jossa käyttäjät koekäyttävät sovellusta oikeassa ympäristössä.

4 Testausmenetelmät

Testausmenetelmä määrittää tavan testata ja se ohjaa koko testausprosessin läpi. Se määrittää testauksen suunnittelun ja ajoituksen, antaa tarkoituksen ja päämäärän testaamiselle sekä tiedot, jotka tarvitaan koko testausprosessiin.

Testausmenetelmät jaetaan yleensä kahteen päästrategiaan: lasilaatikko- ja mustalaatikkomenetelmiin. Testausmenetelmä määräytyy yleensä testauksen tason mukaisesti. Menetelmät voidaan jakaa myös staattisiin ja dynaamisiin menetelmiin. Staattisessa testauksessa kiinnitetään huomio suunnitelmien ja lähdekoodin katselmointiin, kun taas dynaamisessa testauksessa suoritetaan testitapauksia.

4.1 Lasilaatikkotestaus

Lasilaatikkotestauksessa testataan sovelluksen sisäistä rakennetta lähdekoodin avulla. Lasilaatikkotestausta voidaan tehdä yksikkö, integraatio ja järjestelmätasoilla, mutta sitä yleensä käytetään vain yksikköjen testaamiseen.

Lasilaatikkotestauksen suorittaa usein yksikön kehittäjä. Ulkoisen testaajan käyttäminen ei ole kustannustehokasta, sillä lasilaatikkotestaus vaatii usein hyvää osaamista käytetystä kielestä. Kehittäjä tuntee koodinsa parhaiten, siksi hänen on helpoin tehdä testitapaukset. Testitapauksia ei kuitenkaan saa tehdä koodin perusteella, vaan teknistä suunnittelua vasten.

4.2 Mustalaatikkotestaus

Mustalaatikkotestaus on testausmenetelmä, jossa testaaja ei tunne sovelluksen sisäistä toimintaa, toisin sanoen testaajalla ei ole pääsyä lähdekoodiin. Mustalaatikkotestausta kutsutaan myös funktionaaliseksi testaukseksi. Testauksen aikana verrattaan sovelluksen käyttäytymistä määrittelyihin verrattuna.

Mustalaatikkotestaus jää usein projektin loppupuolelle. Sen aikana löydetyt virheet voivat aiheuttaa lisäkustannuksia

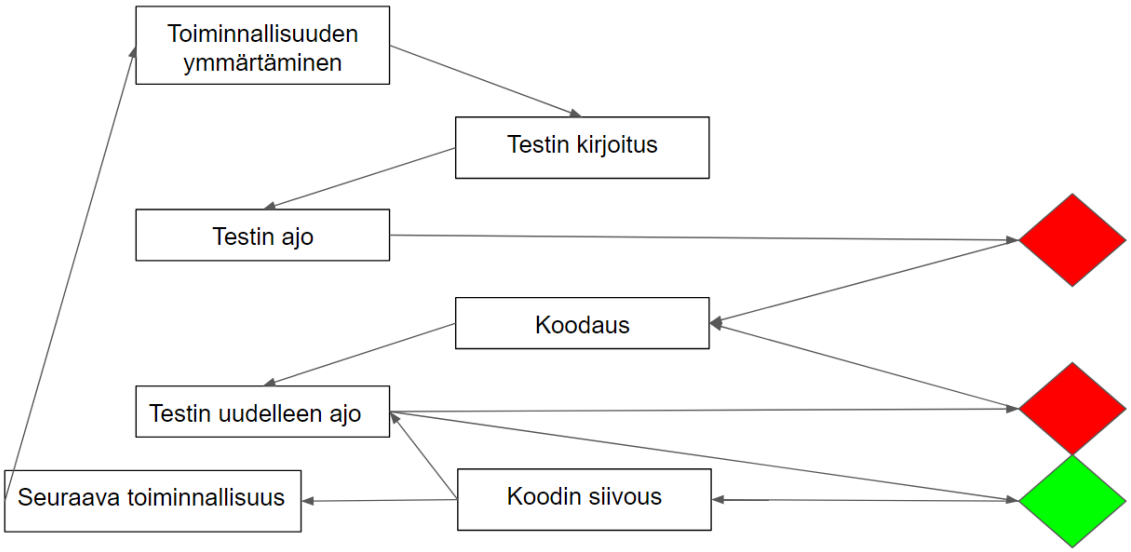
4.3 Tutkiva testaaminen

Kun testataan vain etukäteen suunniteltuja testitapauksia, kaikkia mahdollisia tapauksia ei tule vastaan. Testaajat ovat aina suorittaneet suunnitellun testauksen ohella epäviral-lista testausta. Tämä testauksen muoto on ajan saatossa vakiintunut, ja sitä on alettu kut-sumaan tutkivaksi testaamiseksi (Testaus ketterissä menetelmissä).

Tutkivassa testauksessa testaaja ohjaa toimintaansa testien tulosten perusteella. Testita-pauksia ei suositella etukäteen. Tutkiva testaus on saman aikaista oppimista, testien suun-nittelua ja ajoa (James Bach, Exploratory Testing Explained). Täysin satunnaista se ei kuitenkaan ole, vaan testaussessiolle annetaan jokin tavoite. Esimerkiksi testataan käyn-nissä olevan sprintin tärkeimpiä ominaisuuksia.

4.4 Testilähtöinen kehitys

Testilähtöinen kehitys on ohjelmointia tukeva tekniikka, jota käytetään yleisesti ketterien kehitysmenetelmien kanssa. Testilähtöinen kehitys ei tarkoita sitä, että testaamiseen orientoitunut henkilö näyttää tien tekniselle toteutukselle, vaan testauksen suunnittelu ja toteutus kuuluu yksikön kehittäjälle.



KUVA 1. Testilähtöisen kehityksen vaiheet

Testilähtöisessä kehityksessä luodaan testi, joka varmistaa yksikön toiminnallisuuden. Yksikön toteutusta implementoidaan, kunnes testi menee läpi. Onnistuneen testin jälkeen ohjelmakoodi siivotaan ja varmistetaan, että testi menee edelleen läpi.

Testilähtöisen kehityksen käyttöönotto on vaativaa, jos toimintamalli ei ole kehittäjälle tuttu. Kommentit tuplatyöstä yleistyvät, projektien aikataulut kiristyvät ja paluu vanhaan totuttuun tapaan tuntuu hyvältä ajatukselta. Testilähtöisestä kehityksestä on kuitenkin paljon hyötyä. Hietalahden (Ohjelmistotuotanto, testaus3) mukaan tuottavuus ominaisuuksien määrällä mitattuna on pysynyt samana tai jopa kohonnut, sovelluksien laatu on parantunut, toteutus on monimutkaistunut vain joissain tapauksissa sekä sivutuotteena on tullut erittäin kattava joukko testitapauksia.

4.5 Automaatiotestaus lyhyesti

Testitapaus kannattaa automatisoida, jos se on manuaalisesti aikaa vievää tai vaikeasti toteutettava. Automaatiotestauksen sovelluskohteita ovat yksikkö- ja integraatiotestaus, savutestaus, regressiotestaus ja joltain osin myös hyväksyntätestaus.

Automaatiotestaus on käytännössä ilmaista ja nopeampaa kun manuaalinen testaus kaikissa tilanteissa. Kun testit automatisoidaan, niitä voidaan ajaa niin usein kuin halutaan. Ihmisten tietotaito voidaan keskittää automaatiotestauksen ansiosta muihin tehtäviin. Tuotekehitystä voidaan seurata luotettavasti reaaliajassa, kun testit ajetaan jokaisen muutokset jälkeen.

Toisin kuin ihmiset, automaatio ei koskaan väsy pitkien ja uuvuttavien testitapausten suorittamiseen. Kun testitapaukset on automatisoitu tarkoituksenmukaisesti, ne voidaan suorittaa kerralla useassa ajoympäristössä. Automaatiotestaus ei kuitenkaan koskaan voi korvata täysin manuaalista testausta.

5 Testityökalut

Testityökaluja on paljon ja niitä julkaistaan koko ajan lisää. Testityökalua valittaessa tulee ottaa huomioon testauksen tavoitteet, kehitysympäristö, käyttöönotto prosessi, testien automatisointi, testiraportit, tuetut käyttöjärjestelmät ja kuinka hyvin työkalu sopii tiimille.

Testauksen tavoite on ehkä kriittisin vaatimus testityökalulle. Jos sovelluksen suurimmat virheet ovat esimerkiksi graafisessa käyttöliittymässä, yksikkötestaukseen sopiva työkalu ei tuota haluttua tulosta.

Testityökalujen käytön ja käyttöönoton tulisi olla mahdollisimman helppoa, jotta testityökalua käytettäisiin ohjelmoinnin tukena. Testityökalut voivat olla erillisiä ohjelmia tai käyttäjän kehitysympäristöön asennettavia liitännäisiä. Huomioitava on myös pitääkö olemassa olevaa ympäristöä muuttaa, jotta testityökalu voidaan ottaa käyttöön, tai pitääkö lähdekoodia muuttaa testien takia.

Testityökalut ovat usein joko kaupallisia tuotteita tai avoimeen lähdekoodiin perustuvia ratkaisuja, joista molemmissa on hyviä ja huonoja puolia. Kaupalliset työkalut ovat usein maksullisia ja niissä on kattava dokumentaatio ja käyttötuki. Päivitykset riippuvat usein siitä, missä vaiheessa työkalu on elinkaartansa. Aktiivisessakaan kehityksessä olevat työkalut eivät välttämättä saa pyydettyjä ominaisuuksia. Avoimen lähdekoodin työkaluissa on taas helpompi vaikuttaa ominaisuuksiin käyttäjäyhteisön voimin tai toteuttamalla haluamansa toiminnallisuus itse. Onkin tärkeää, että työkalulla on aktiivinen käyttäjäyhteisö. Se yleensä merkitsee jatkuvaa testityökalun kehittämistä. Avoimen lähdekoodin testityökaluissa on usein puutteellisempi dokumentaatio, mutta apua käyttöönottoon ja integrointiin saa yhteisöltä.

Usein testityökalut yhdistetään myös CI-prosessiin, jolloin testityökalun tulee tukea sitä. CI-prosessi automatisoi sovelluskehitystä. Konfiguraatiosta riippuen se voi esimerkiksi rakentaa sovelluksen, ajaa ja raportoida testit ja julkaista sovelluksen testikäyttäjille tai sovelluskauppaan. Kun testityökalu toimii CI -ympäristössä, kehittäjä saa suoraan tuloksen testistä. Tällöin kehittäjän on helpompi etsiä virheitä ja korjata koodi.

Suosituimmat työkalut React Native -sovellusten testaamiseen ovat Appium, Cavy, Detox sekä Facebookin kehittämä Jest.

5.1 Appium

Appium on avoimeen lähdekoodiin perustuva testiautomaatio kehys, joka tukee natiivi, hybridi ja web-sovellusten testaamista sekä Android että iOS käyttöjärjestelmissä. Appium -testejä voi kirjoittaa lähes millä tahansa ohjelmointikielellä.

Appiumin filosofian mukaan sovellusta ei tuli muokata millään tavalla, jotta sitä voisi testata. Appium käyttääkin toimittajan tarjoamia testiautomaatio kehyksiä, jotta sovellus ei tarvitse Appium spesifisiä tai kolmannen osapuolen tarjoamia kirjastoja.

Appiumin valitsijat, joilla voidaan valita tietty komponentti ovat:

- Accessibility ID
- Class name
- ID
- Name
- XPath

React Nativen tapauksessa ID on käyttökelvoton. ID viittaa elementin tunnisteeseen: Android järjestelmän *resource-id* ja iOS järjestelmän *name*. React Native kehys ei anna mahdollisuutta asettaa *id* tunnistetta komponenteille.

Ainoa vaihtoehto, mitä ylläolevasta valitsijajoukosta voidaan käyttää React Native sovellusten testaamiseen, on *Accessibility ID*. Se on helppokäyttötoiminnon tunniste, joka on Androidilla elementin kuvaus ja iOSilla elementin id. Tunniste luetaan esimerkiksi sokealle käyttäjälle. Tämän valitsimen käyttö johtaa vääjäämättä suuntaan, jossa kehitetään rinnakkain tuotanto ja testisovellusta.

5.2 Cavy

Cavy on järjestelmäriippumaton integraatiotestikehys React Nativelle. Cavy käyttää hyväkseen *ref*-attribuuttia komponentin valitsemiseen. Cavy toimii applikaation kanssa, eikä se simuloi renderöintiä.

Ref-attribuutin käyttöä ei kuitenkaan suositella. Sen avulla voidaan manipuloida suoraan tiettyä komponenttia, joka ei ole React Nativen ideologian mukaista. Kaikkien muutosten tulisi tapahtua *state* ja *prop* attribuuttien avulla, jolloin koko komponenttipuu piirretään uudelleen. Tämän vuoksi Cavyn käyttöönotto aiheuttaa muutoksia olemassa olevaan lähdekoodiin, eikä sitä voi ottaa suoraan käyttöön vanhaan projektiin.

5.3 Detox

Detox on dokumentaationsa mukaan päästä-päähän testikehys, joka käyttää harmaalaatikko testimenetelmää. Se ajaa sovellusta simulaattorissa ja vuorovaikuttaa sovellukseen, kuten käyttäjä. Detox testi suorittaa itse mobiilisovellusta ja testikehystä rinnakkain eri prosessissa.

Detoxin pääominaisuus on kyky automaattisesti synkronoida testisuoritus sovelluksen kanssa. Detoxin kehittäjien mukaan päästä-päähän testauksen ärsyttävin piirre on epäluotettavuus, jonka Detox poistaa seuraamalla verkkopyyntöjä, animaatioita, ajastimia, React Native Bridgen suorittamia asynkronisia tehtäviä, asynkronisia React Native asettelua sekä Java Script tapahtumasilmukkaa. Detox monitoroi sovellusta, ja odottaa edellä mainittujen valmistumista ennen, kuin testiä jatketaan.

Detox ei ole kirjoitushetkellä Android yhteensopiva. Se toimii vain iOS simulaattorilla. Detoxissa käytään komponentin valitsimena *testID* attribuuttia.

5.4 Jest

Jest on Facebookin kehittämä ja suosittelema testikehys, jolla voi testata mitä tahansa JavaScript koodia ja se kasvattaa suosiotaan jatkuvasti. Jest on alun perin kehitetty Jaminen päälle, mutta nykyään suurin osa sen ominaisuuksista on korvattu ja sen päälle on lisätty useita muita ominaisuuksia.

Jest perustuu jäljitettyjen komponenttien testaamiseen. Jest tulee uusissa sovelluksissa esi asennettuna ja vanhoihin sovelluksiin se on hyvin helppo integroida. Jestin tärkeimmät ominaisuudet ovat Snapshot, sisäänrakennettu koodikattavuus työkalu sekä nopeus.

6 Pohdinta

Sovellusten testaamisen pääasiallinen on laadunvarmistus, mitä se kulloinkin tarkoittaa riippuu tilaajasta, toimittajasta, sidosryhmistä ja ilmapiiristä. Testausta ei useinkaan arvosteta, varsinkaan pienissä tilaaja organisaatioissa, koska toimittajan oletetaan tekevän virheetöntä koodia. Tämä harhaluulo johtunee monen asian summasta, joista esimerkkeinä kerrottakoon testauksen hintalappu, tehokkuus, vaikutus projektin kestoon sekä käytettävissä olevat resurssit. Testauksen tulisi aina tuottaa helposti ymmärrettävissä olevaa lisäarvoa asiakkaalle.

Ideaalitilanteessa testaus alkaa jo ennen projektin alkua. Myynnin aikana aletaan tehdä sisäistä testausta riskien hallitsemiseksi. Minkälainen asiakas on kyseessä, mikä on projektin aikataulu ja tarkoitus ja missä toimintaympäristössä toimitettava palvelu toimi. Näitä testataan sisäisesti aikaisempaan kokemukseen verraten. Projektia ei kannata myydä sellaisilla ehdoilla, jotka mahdollistavat todennäköisesti projektin epäonnistumisen.

Projektin alussa luodaan projektin dokumentaatio, joka pitää sisällään sopimuksen, vaatimus-, arkkitehtuuri- ja teknisenmäärittelyn. Näiden dokumenttien pohjalta luodaan testaussuunnitelma, jossa määritellään testauksen tavoite, testattavat pääkohteet ja osa-alueet, testityökalut ja -ympäristöt, milloin ja millä tavalla testaus suoritetaan, miten testit raportoidaan ja mitkä ovat hyväksyntä kriteerit. Tämä siis ideaalitilanteessa.

Ratkaiseeko yllämainitut dokumentit kaiken testaukseen liittyvän? Haltulla mobiiliprojektit ovat verrattain pieniä, ja jo yllämainittujen dokumenttien laatiminen kattavasti kuluttaisi (merkittävän) osan budjetista. Yksi Haltun arvoista on auttaa vilpittömästi asiakkaita. Tämä pitää sisällään mielestäni asiakkaan sovelluksen laadunvarmistuksen. Testauksen tulee kuitenkin olla sovittua ja testausprosessien läpinäkyviä, jotta asiakas näkee mitä tehdään, että hän kokee saavansa lisäarvoa koko projektin ajan.

V-mallin käyttäminen projekteissa ei ole nykyään kovin yleistä. Sen käyttö soveltuu vesiputousmalliin, jossa kehitysprosessi etenee vaihe vaiheelta isoista kuvista kohti teknistä määrittelyä. Projekteissa käytetään usein ketterän kehityksen malleja, joissa kehityssyklit ovat nopeita ja asiakasviestintä suoraa. Tämä mahdollistaa reagoimaan nopeasti tuleviin

muutoksiin. V-mallin mukaiset testaustasot ovat kuitenkin mielestäni hyvä tapa tehdä testausta. On järkevää aloittaa pienistä kokonaisuuksista edeten kohti isompaa järjestelmää. Tällöin virheet huomataan mahdollisimman aikaisessa vaiheessa ja niiden korjaaminen on yksinkertaisempaa.

Testilähtöinen kehitys on mielestäni askel kohti testausmyönteistä työkulttuuria. Testilähtöinen kehitys yhdistettynä helposti ymmärrettävään ja käytettävään työkaluun saadaan aikaiseksi nopeasti lähtökohdat syvemmille testauksen tasoille. Lisäksi järjestelmän ymmärrys paranee, kun testit suunnitellaan ensin. Vaikka testien suunnittelu ennen toiminnallisuuden toteuttamista onkin alussa aikaa vievää, on siten saavutettu sama tai jopa parempi työtehokkuus.

Miellän testaamisen usein, osittain työkokemuksen perusteella, käyttöliittymätestityökalulla tehtäväksi sovelluksen ”klikkailuksi”. Olin työharjoittelussa yrityksessä, jossa kehitettiin testikehystä, jolla voi tehdä yksinkertaisia testitapauksia, vain sovellusta käyttämällä. Työkalu generoi valmiiksi suoritettavan koodin, jota voi ajaa valitsemassaan CI-ympäristössä. Etuudet tämänkaltaisen työkalun käytössä on sen helppous, testi voidaan suunnitella, kun tarvittavat komponentit ovat valmiita. Myös asiakkaat voitaisiin ohjata kokeilemaan GUI työkalun käyttöä. Arvioin, että se sitouttaisi myös tilaajan organisaation paremmin projektiin ja varsinkin testien tekemisestä suoraan näkyvä hyöty voisi muuttaa käsitystä aikaa ja rahaa vievästä testauksesta. Tämän kaltaisen työkalun käyttö React Native sovelluksissa on kuitenkin mahdotonta toteuttaa järkevästi tämän työn kirjoitushetkellä. Komponenteille ei voi asettaa resurssi id:tä, joka on muuttumaton komponentin nimestä tai sijainnista näytöllä huolimatta. Vaihtoehtona on käyttää hyväksi helppokäyttötoiminnon kuvausta, mutta kaikkien komponenttien kuvauksen ääneen lukeminen on mielestäni väärin, eikä kahta eri sovellusta ole järkevää ylläpitää. Sovellus testiattribuuteilla ei ole sama sovellus, kuin julkaistava, jossa ei niitä ole.

React Nativelle suunnattuja testityökaluja on verrattain vähän ja osa niistäkään ei ole vielä valmiita. Kaikissa tässä työssä esitetyissä työkaluissa on puutteita tai ne väärinkäyttävät järjestelmää. Jest on käyttäjäyhteisön hyväksymä, koko ajan kasvava ja kehittyvä työkalu, mutta ajatuksena komponentin kopion testaaminen puistattaa. En luottaisi alkuperäiseen moottoriin, jos sitä on testattu vain kopion avulla. Kopio ei koskaan ole identtinen alkuperäisen kanssa.

Opinnäytetyön teoriaosan ja työkalututkimuksen perusteella mielestäni tehokkain tapa testata React Native sovelluksia on dokumentoida sovelluksen toiminallisuudet, joiden perusteella sovellus voidaan julkaisu testata järjestelmällisesti. Tähän yhdistettynä testilähtöinen kehitys yksikkötesteineen sekä tutkiva testaus sprinttien lopussa takaavat mielestäni riittävän laadun.

7 LÄHTEET

Ron Patton, Software Testing. Luettu 4.5.2018.
https://shekharsk.files.wordpress.com/2016/01/ron-patton-software-testing.pdf

Erkki Hietalahti. Ohjelmistotuotanto, testaus 1-3. Luentomateriaali. Luettu 16.5.2018

James Bach, Exploratory Testing Explained. Luettu 21.5.2018
http://www.satisfice.com/articles/et-article.pdf

Testaus ketterissä menetelmissä, luentomateriaali. Luettu 21.5.
https://www.cs.helsinki.fi/group/java/k12-ohtu/luento6.pdf

Appium. Johdanto. Luettu 26.5.2018.
http://appium.io/docs/en/about-appium/intro/

Cavy. Johdanto. Luettu 26.5.2018.
https://github.com/pixielabs/cavy

Detox. Johdanto. Luettu 26.5.2018.
https://github.com/wix/detox