OULUN AMMATTIKORKEAKOULU

Jukka Väyrynen

KETTERÄ AUTOMAATIOTESTAUS

Käyttäytymislähtöinen ohjelmistokehitys mobiilisovelluksen testauksessa

KETTERÄ AUTOMAATIOTESTAUS

Käyttäytymislähtöinen ohjelmistokehitys mobiilisovelluksen testauksessa

Jukka Väyrynen
Opinnäytetyö
Syksy 2014
Tietojenkäsittely
Oulun seudun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietojenkäsittely, Web-sovelluskehitys

Tekijä: Jukka Väyrynen

Opinnäytetyön nimi: Käyttäytymislähtöinen ohjelmistokehitys mobiilisovelluksen testauksessa

Työn ohjaaja: Pekka Ojala

Työn valmistumislukukausi- ja vuosi: Syksy 2014 Sivumäärä: 25

Opinnäytetyössä oli tarkoitus tutkia keinoja tehostaa mobiilisovellusten koko ohjelmistokehitystä. Useat laitealustat ja käyttöjärjestelmät ovat koettu haasteellisiksi testauksen automatisoinnille ja automaation puute lisää koko sovelluskehityksen kustannuksia. Onko olemassa toimivaa menetelmää kehittää mobiilisovelluksia kokonaisuutena, unohtamatta testausta ja sen automatisointia? Mitkä ovat mahdollisen menetelmän heikkoudet ja vahvuudet, sekä millaisia reunaehtoja kyseinen menetelmä asettaa koko kehitystyölle?

Työssä tutkittiin ketteriä kehitys- ja testausmenetelmiä. Erityisesti tarkasteltiin Käyttäytymislähtöistä kehitysmenetelmää ja olisiko sillä mahdollista tehostaa mobiilisovellusten kehitystä kokonaisuutena.

Tutkimus pohjaa todelliseen toimeksiantoon, jossa toteutettiin Calabash-testit mobiilisovellukselle, joka tukee kaikkia laitealustoja ja useita rajapintoja.

Calabash, BDD, Käyttäytymislähtöinen ohjelmistokehitys

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Business Information Systems

Author(s): Jukka Väyrynen
Title of Bachelor´s thesis: Agile test automation, mobile application testing with BDD
Supervisor(s): Pekka Ojala
Term and year of completion: Autumn 2014 Number of pages:25

The purpose of this Bachelor's thesis was to study better ways of improving development of mobile applications. One of the major problems of development is software testing and its automation. Automation testing is difficult because of various platforms and several different versions of operating systems

This thesis researched existing issues at mobile development and automation testing, possible solutions to these issues and what as well as the benefits and weakness at this method. One of the main frameworks is the study of agile software development and behavior-driven development as the solutions of the problems.

The study is based on a real-life assignment. Calabash automation tests were conducted on a mobile application that supports the most-used mobile clients

---

Keywords:
Agile, BDD, Calabash

SISÄLLYS

LYHENTEET

BDD	Behavior-driven development. Käyttäytymislähtöinen ohjelmistokehitys
CI	Continuous Integration, jatkuva integraatio
Iteraatio	Toistettava sarja tehtäviä, jotka sisältää projektisuunnittelun, vaatimusanalyysin, ohjelmistosuunnittelun, ohjelmoinnin, testauksen ja dokumentoinnin.
TDD	Test-driven development. Testausvetoinen ohjelmistokehitys

1 JOHDANTO

Ohjelmistotuotannossa on kasvattanut merkitystään testaus. Ohjelmistotestausta käytetään laadunvarmistukseen ja oikein toteutettuna sen avulla on mahdollista nopeuttaa koko kehitysprosessia.

Testattavan tiedon määrä on kasvanut samaa tahtia kuin sovellukset ovat kasvaneet ja saaneet uusia rajapintoja. Yksi sovellus voi hakea tietoa tietokannoista, erillisistä eri yritysten välisistä kerroksista ja esimerkiksi karttasovelluksista. Lisäksi ulkoisten seikkojen ja käytettävyyden on pysyttävä helposti ymmärrettävänä ja toimivana.

Erityisesti mobiililaitteiden testaus on osoittautunut erityisen haastavaksi. Pelkästään jo eri laitealustojen määrä ja laitteiden eri ominaisuudet vaikeuttavat testaamista entisestään. Koska tuettavia alustoja ja versioita on paljon, on sovellukset tehty usein html-pohjaisiksi. Html-pohjaisten sovellusten saumaton integrointi puhelimen omien ominaisuuksien kanssa tuo mukanaan omat haasteensa ja heikkoutensa. Lisäksi useat alustakohtaiset erot ja laitteiden eri resoluutiot tekevät testauksesta haasteellista. Juuri edellä mainituista syistä iso osa mobiililaitteiden testauksesta toteutetaan manuaalitestauksena, joka on kallista ja aikaa vievää. (Base36, Automated vs. Manual Testing: The Pros and Cons of Each, hakupäivä 20.11.2014.)

Web-sovelluksille on ketterän kehityksen mukaisia automaatiotestaustyökaluja jo yleisesti käytössä ja siellä testausta on kehitetty huomattavasti pidemmälle kuin mobiilisovellusten kehityksessä. Tähän tarpeeseen on kehitetty ketterää ohjelmistotestausta tukevia menetelmiä. Yksi niistä on testauslähtöinen ohjelmistokehitysmenetelmä TDD ja siitä jalostunut käyttäytymislähtöinen ohjelmistokehitys BDD (Avustaja, bdd behaviour driven design, hakupäivä 19.7.2014). Suurimpina ongelmina näiden menetelmien yleistymiselle on käyttäytymislähtöistä ohjelmistokehitystä tukevien ohjelmistojen vähäinen määrä ja vielä vähäisempi määrä tutkimustietoa.

Tutkimuksen tuloksia on mahdollista hyödyntää tulevissa ketterän ohjelmistokehityksen projekteissa. Tämän avulla on lisäksi mahdollista tehostaa jo olemassa olevan projektin testausta.

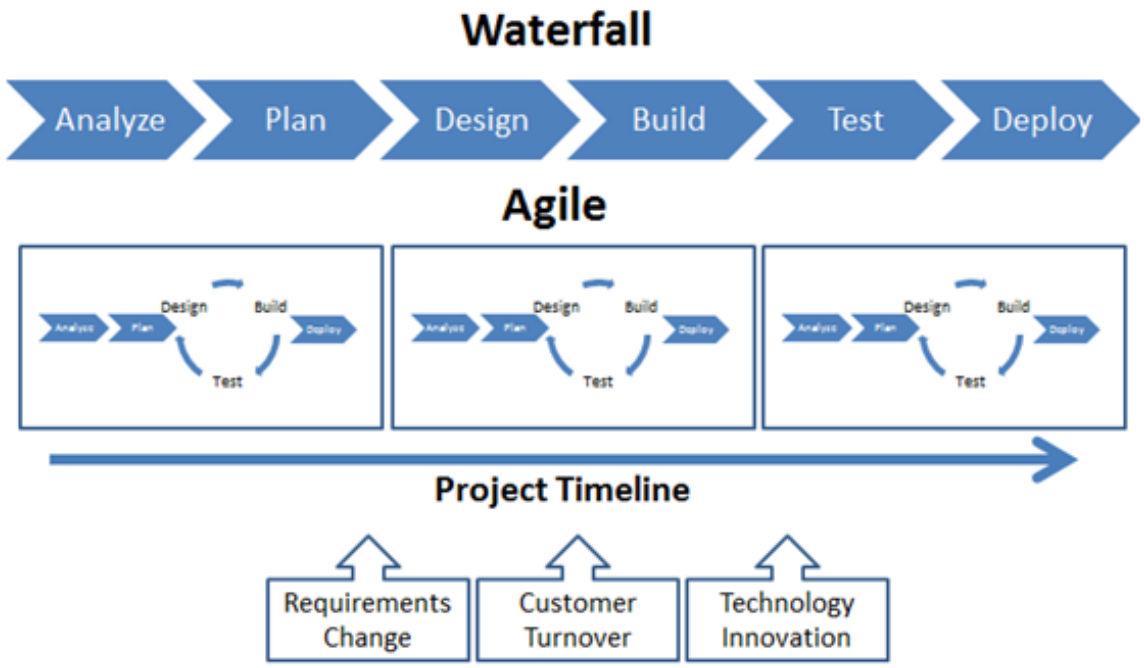
2 AUTOMAATIOTESTAUS

2.1 Ketterä ohjelmistokehitys

Ketterän ohjelmistokehityksen idea näki päivänvalon vuonna 2001 kun merkittävimmät ohjelmistokehityksen uranuurtajat lanseerasivat ketterän ohjelmistokehityksen julistuksen, jonka avulla ketterää ohjelmistokehitystä pyritään määrittämään. Ketterässä kehityksessä keskitytään ohjelmistotuotannossa vain olennaisimpaan eli itse ohjelmistotuotantoon ja pidetäänkin itse ohjelmistoa parhaimpana mittarina kehitykselle. Kehitystyö on pyritty jakamaan lyhyisiin iteraatioihin joiden lopussa pyritään saavuttamaan julkaisukelpoinen ohjelmisto. (Wikipedia, ketterä ohjelmistokehitys, hakupäivä 19.7.2014.)

Vesiputousmallin ja ketterän kehityksen suurin ero on juuri muutoshallinnassa. ”Ketterässä kehityksessä ei käytetä samanlaista muutoshallintaa kuin perinteisessä vesiputousmallissa. Kun toiminnallisuutta toimitetaan osa kerrallaan, tarpeiden muutokset otetaan työlistalle ja toteutetaan soveltuvan prioriteetin mukaan järjestyksessä. Jos muutoksia tulee paljon, kokonaisuuden ennustettavuus vaatii erityistä huomiota”. (Toikkanen, Auer A, Auer L, Heinäsmäki, Hölttä, Kalliala, Laanti, Laine, Lekman, Miinalainen, Naski, Piiparinen, Puhakka, Pyhäjärvi, Pääkkönen, Rosti, Räisänen, Sora, Taipale, Talvio, Tanninen, Toivola, Toro, Valsta, Väyrynen & Weissenberg 2013, 52.)

Ketterässä kehityksessä panostetaan myös suoraan viestintään. Tämän takia ketterät tiimit työskentelevät usein lähellä toisiaan ja tiimiin kuuluukin kaikki, joita tarvitaan ohjelmiston tuottamiseen: ohjelmoijia, testaajia, käyttöliittymäsuunnittelijoita ja päälliköitä. Nämä kaikki vastaavat osaltaan ohjelmiston kehittymisestä ja suunnittelua tehdäänkin läpi koko ohjelmiston kehitystyön. (Wikipedia, ketterä ohjelmistokehitys hakupäivä 19.7.2014.)



KUVIO 1. Ketterän kehityksen työjärjestys (Agile Enterprises. Advantages of Agile Development hakupäivä 19.7.2014)

2.2 Testivetoinen ohjelmistokehitys

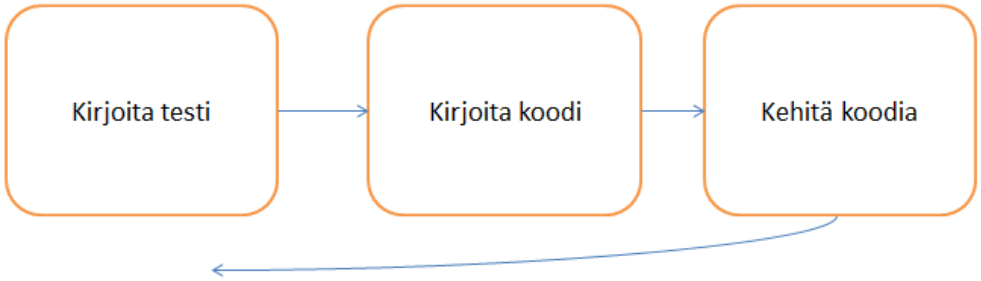
Ketterän kehityksen ohessa alettiin käyttää TDD-tekniikkaa eli testivetoista ohjelmistokehitystä. Tämä ei itsessään ole testausmenetelmä, vaikka nimi siihen viittaakin, vaan lähinnä suunnittelu tai määritysmenetelmä, jonka sivutuotteena syntyy joukko toimivia testejä. (Nieminen 2014, 3.)

TDD:ssä sovellusta tarkastellaan testauksen näkökulmasta, ja sovelluksen toteuttaminen aloitetaan aina suunnittelemalla kyseiselle ominaisuudelle testi, joka kuvaa kyseistä tapausta ja jonka tulee mennä läpi valmiissa sovelluksessa.

```
4
5
6  Feature: User login
7      User wants to login to the service
8
9
10
```

KUVIO 2. Esimerkki testivetoisen kehityksen ehdosta

TDD-menetelmässä on hyvä miettiä jokaiselle ominaisuudelle sitä kuvaava ehto. Ehdon ja siinä samalla testin valmistuttua ohjelmoidaan itse sovellukseen kyseinen ominaisuus, jonka jälkeen on vielä mahdollista parannella ohjelmakoodia vastaamaan käyttötarkoitustaan. (Agiledata.org TDD hakupäivä 4.10.2014.)



KUVIO 3. Testivetoisen kehityksen toiminta

Tämä menetelmä luo yksinkertaista ohjelmakoodia ja ohjelmasta on helppo todentaa varhaisessa vaiheessa sen virheet. Tämä on huomattavasti kustannustehokkaampaa mitä virheiden löytäminen ja korjaaminen myöhemmin (Agiledata.org TDD 4.10.2014.)

TDD on otettu käyttöön monessa yrityksessä perehtymättä asiaan kunnolla ja sen takia se on saanut osakseen paljon kritiikkiä. TDD vaatii perehtymistä ja usein sen on odotettu korvaavaan aiemmat ohjelmistokehitysmenetelmät. Parhaiten menetelmä toimiikin vain laajamittaisen suunnittelun ja vahvojen testaustaitojen myötä. (Karjalainen 2012, 6.)

2.3 Käyttäytymislähtöinen ohjelmistokehitys

BDD-menetelmän eli käyttäytymislähtöisen ohjelmistokehityksen voidaan sanoa kehittyneen juuri TDD-menetelmästä, vaikka menetelmissä onkin joitakin eroavaisuuksia. Suurimpana erona TDD-menetelmään voidaan pitää itse testien luettavuutta. TDD-menetelmässä testit tekee yleensä sama henkilö, joka kirjoittaa itse ohjelmakoodinkin. BDD:ssä testit ovat tavallisesti luettavassa muodossa mikä mahdollistaa muiden kuin ohjelmoijan kehittävän testejä. BDD-testauksessa keskitytään tarkastelemaan sovelluksen toimintaa käyttäjän näkökulmasta, eli itse menetelmä ei ota kantaa taustalla toimivaan koodiin vaan asioihin, jotka näytetään käyttöliittymän kautta. Tarkoitus on, että jos testit menevät läpi, niin silloin sovelluksen oletetaan toimivan niin kuin sen on tarkoituskin. (Kärkkäinen 2013, 9.)

BDD:ssä pääajatuksena on, että määrittelijä, ohjelmoija ja testaaja suunnittelevat yhdessä käyttötapaukset selkokielisesti. Tämän etuna on välttää tilannetta, jossa ohjelmoija on mahdollisesti ymmärtänyt ohjelmoitavan ominaisuuden väärin. Lisäksi tämä vähentää testaustyötä, joka kohdistuu alati muuttuvaan sovellukseen. Sovelluksen muuttuessa ja päivittyessä itse käyttötapaukset eivät muutu. Lisäksi BDD ei edellytä testien kehittäjältä tai ajajalta samanlaista ohjelmointiosaamista, kuin perinteinen automaatiotestaus. (Kärkkäinen 2013, 9.)

Koska BDD-testit suunnitellaan kehitystyön alkuvaiheessa, se vaatii samalla koko kehitystiimin sitoutumista menetelmään. Lisäksi menetelmän käyttö vaatii käyttöönottoa mahdollisimman aikaisessa vaiheessa. Mitä pidemmällä projekti on sitä haastavampaa ja kalliimpaa BDD:n käyttöönotto on.

Suurin hyöty BDD:stä on testien ylläpidon helppous. Testien ylläpitäjän ei välttämättä tarvitse olla sama henkilö, joka testit kirjoittaa (Kärkkäinen 2013, 9). Selkokielisten testien kirjoittaminen helpottaa muutenkin myöhemmin tehtävien muutosten tekoa.

Kärkkäinen (2013, 35) toteaa lisäksi, että testien automatisointi lisää kustannuksia yleensä hetkellisesti juuri testien tekovaiheessa, mutta ajan kuluessa ja eteenkin ylläpitovaiheessa BDD-testit maksavat itsensä takaisin. Varsinkin jos testien ylläpitäjänä toimii joku muu henkilö kuin itse testien kirjoittaja, on testien helpon päivitettävyyden oltava oikeasti toimivaa.

Menetelmän työmäärää voidaan pitää myös aluksi suurena juuri testitapausten luomisen ja testien automatisoinnin takia, mutta tämä samalla vapauttaa testaajat haastavampiin ja manuaalitestausta vaativiin tehtäviin. Lisäksi ei ole testaajan motivaation ja tuloksen kannalta järkevää pitää ammattilaisia testaamassa kaikkia mahdollisia syötevariaatioita. (Kärkkäinen 2013, 35.)

Toimiessaan BDD mahdollistaisi tehokkaan mobiililaitteiden testauksen, koska itse testien määrittäminen ei ole laitteistoriippuvaista vaan samat käyttäjätarinat pätevät kaikissa laitteissa. Lisäksi käyttöliittymämuutokset eivät yleensä vaikuta käyttäjätarinoihin.

3 MOBIILISOVELLUKSEN TESTAUS

3.1 Tutkimusongelma

Opinnäytetyössä keskityttiin ongelmaan, joka liittyy mobiilitestauksen automatisointiin. Useat laitealustat ja käyttöjärjestelmät ovat koettu haasteellisiksi testauksen automatisoinnille ja automaation puute lisää koko sovelluskehityksen kustannuksia. Onko olemassa toimivaa menetelmää kehittää mobiilisovelluksia kokonaisuutena, unohtamatta testausta ja sen automatisointia? Mitkä ovat mahdollisen menetelmän heikkoudet ja vahvuudet, sekä millaisia reunaehtoja kyseinen menetelmä asettaa koko kehitystyölle?

3.2 Ratkaisuehdotus

Tutkimuksen alkuvaiheessa tuli esille mahdollisuus kokeilla BDD-menetelmän soveltuvuutta mobiilisovelluksen testaukseen ja tehostamiseen. Lisäksi olisi mahdollista tutkia menetelmää myös koko sovelluskehityksen näkökulmasta.

Menetelmä soveltuisi vallalla olevaan ketterään ohjelmistotuotantoon ja vastaisi tarpeeseen, jossa saataisiin tehostettua testausta ja testejä voitaisiin suunnitella jo varhaisessa vaiheessa ohjelmistokehitystä.

Sovelluksen valinnassa huomio kiinnittyi muutamaan seikkaan. Sen tulisi soveltua vallalla olevaan nykyiseen ohjelmistokehitysmalliin, eli toimittava osana ketterää kehitystä. Lisäksi sen käytön ei tulisi olla liian haastavaa vähemmän ohjelmistokokemusta omaavallekkaan testaajalle. Tämä on siinä mielessä tärkeä seikka, koska sovelluksilla on tarve päivittyä, on myös testejä päivitettävä samaa tahtia. Valitettavana tosiasiana voidaan pitää myös mobiilialustojen ja versioiden määrää, joka asettaa oman haasteensa itse testaukselle. Testauksen on siis keskityttävä käyttäjäkokemukseen ja siihen, että asiat näkyvät käyttäjälle juuri niin kuin on suunniteltu, ottamatta kantaa siihen miten ohjelmakoodi taustalla toimii.

Sovellukseksi valikoitui Calabash-testaustyökalu. Calabash perustuu paljon käytössä olevaan Cucumberiin, sen hallitseminen tai edes sen aikaisempi käyttö ei ole vaadittavaa Calabash-testauksessa. (LessPainful 2012, hakupäivä 23.7.2014.) Lisäksi Calabash-sovelluksen käyttöönottoa helpottaa testien ohjelmoinnissa käytettävä Gherkin-ohjelmointikieli. Kieli on täysin selkokielistä ja ymmärrettävää, joten sen ohjelmointi ja lukeminen on helppoa.

Calabash on avoimeen lähdekoodiin perustuva, ilmainen ohjelma. Sen valintaan vaikutti myös sovellusta kehittävä aktiivinen verkkoyhteisö, joka oli tehnyt kattavan joukon ohjeita ilmaiseksi internetistä haettaviksi. Ohjeita löytyi asennuksesta lähtien. Lisäksi tältä verkkoyhteisöltä olisi mahdollista saada apua ongelmatilanteissa hyvinkin nopeasti.

3.3 Rajaukset

Testit rajattiin käsittämään vain Android-laitteet. Osaksi sen takia, että sovelluksen on tarkoitus toimia kaikilla alustoilla aivan samoin. Calabash-testit on kyllä mahdollista laajentaa toimimaan jälkikäteenkin iOS-laitteiden kanssa. Yksi testeihin merkittävästi vaikuttava rajaus oli kielivalintojen huomioiminen testejä tehdessä.

Tutkimuksessa ei myöskään käsitellä testien kytkemisestä johonkin jatkuvan integraation työkaluun, jolla olisi mahdollista automatisoida testien ajaminen. Työkalun avulla pystyy ajastamaan testit suoritettavaksi esimerkiksi kerran vuorokaudessa tai useammin. Tämä mahdollistaisi myös testien liittämisen suoraan versionhallintaan jolloin testit suoritetaan aina uuden ohjelmistoversion päivityksen myötä. Tästä ei kuitenkaan tulla kertomaan tarkemmin, koska tutkimuksen laajuus olisi kasvanut valtavasti ja sen myötä olisi ajauduttu kauas itse BDD-testaamisesta.

3.4 Sovelluskohde

Sovelluskohteeksi valikoitui pankkisovellus Pivo. Kyseinen sovellus on eräänlainen mobiililompakko, joka mahdollistaa käyttäjälle omien tilitietojen seuraamisen. Lisäksi sovelluksesta löytyy erilaisten yritysten tarjouksia ja kanta-asiakasohjelmia. Sovelluksen testaamisen tekee haastavaksi nopeasti päivittyvä sovellus ja tuettujen mobiililaitteiden valtava määrä. Eli haasteiksi koet-

tiin juuri samat asiat mitkä ovat yleisesti tiedossa mobiilisovellusten testauksessa. Lisäksi oman haasteensa testaukseen tuovat sovelluksen useat rajapinnat. Sovellus käyttää paikkatietoja, dataa erilaisista yrityksistä, yhteistyökumppaneista ja lisäksi tietenkin pankin omista järjestelmistä haettuja tietoja.

Pankin omat järjestelmät vaativat lisäksi tarkkuutta testauksessa. Testiympäristöt on tarkkaan määrättyjä ja valvottuja. Lisäksi valmiin sovelluksen tulee täyttää tarkat ehdot niin käytettävyyden kuin tietoturvankin osalta.

3.5 Testien toteutus

Tärkeä osa tutkimustyötä oli itse varsinainen testaustyö. Vain käytännössä toteutetuilla testeillä voitaisiin varmistaa testeihin liittyvät haasteet ja mahdollisuudet. Testattavien ominaisuuksien tulisi keskittyä mobiilisovelluksen käyttöliittymään ja testien tulisi mukailla käyttäjän toimia sovelluksessa.

Testien kirjoittaminen ei vaadi Calabash-sovelluksen osalta suuria. Tavallinen tekstieditori riittää, mutta jos ohjelmointia haluaa helpottaa ja nopeuttaa, ovat erityisesti ohjelmointiin tarkoitetut editorit suositeltavia. Tärkein ominaisuus editorissa on, että se tunnistaa Ruby-syntaksin. Muutettavia tiedostoja ei ole montaa ja niiden hahmottaminen on helppoa. Calabash luo itse alla kuvatun kansiorakenteen josta löytyvät tärkeimmät tiedostot helposti (kuvio 4).

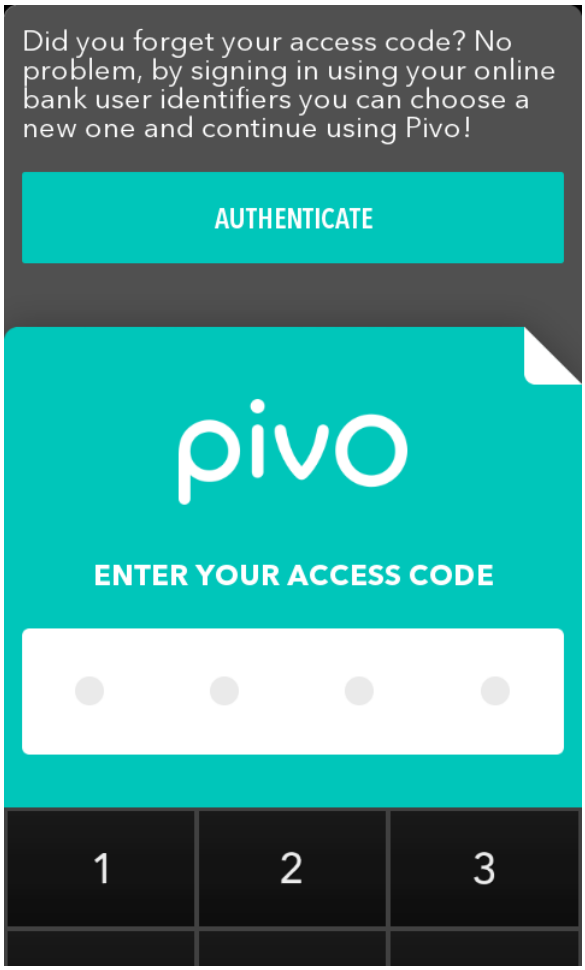
```
features
|_support
| |_app_installation_hooks.rb
| |_app_life_cycle_hooks.rb
| |_env.rb
| |_hooks.rb
|_step_definitions
| |_calabash_steps.rb
|_my_first.feature
```

KUVIO 4. Calabash-sovelluksen oletushakemistorakenne (GitHub.com Calabash-Android, hakupäivä 19.7.2014)

Projektissa johon testit toteutettiin oli jo valmiina testitapaukset, joten niiden täysipainoiseen suunnitteluun ei ollut tarvetta. Lähinnä piti miettiä mitkä tapaukset olisi mahdollista toteuttaa automaatiolla ja mitkä olisi pakko testata manuaalisesti. Esimerkiksi erilaisten salasanavariaatioiden käsin syöttäminen on aikaa vievää ja tämä olisi mahdollista toteuttaa helposti automaatiolla. Oman haasteensa testauksessa aiheuttaa sovelluksen useat rajapinnat ja esimerkiksi paikkatietojen käyttö. Tämä asia vaatii tarkkaa harkintaa siitä tullaanko testit ajamaan aina tietyssä sijainnissa, jolloin paikkatieto ei muutu vai onko jokin syy miksi paikkatieto voisi vaihdella. Tällä kertaa testit toteutettiin toimimaan vain yhdessä sijainnissa ja jos paikka muuttuu, se vaatii tämän testitapauksen muuttamista.

Helpoimmin automatisoitavat kohteet ovat erilaiset kentät, joissa käyttäjä antaa jonkin syötteen (kuvio 5). Erilaisia numero- ja kirjainvariaatiota on useita joita voidaan yksinkertaiseen kenttään syöttää, mutta BDD:ssä on helppo määrittää, että salasanan on sisällettävä vain numeroita ja oltava esimerkiksi neljä merkkiä pitkä. Testit alkavat aina ominaisuuskuvauksella (feature).

Tässä kuvauksessa kerrotaan selkeästi millaista asiaa tai ominaisuutta halutaan tutkia. Tapauksessa kerrotaan tarkemmin nimi ja askeleet millä keinoin testi toteutetaan ja millainen vastaus askeleista tulisi saada. Calabashin käyttämät askeleet ovat: Given, When ja Then. Lisäksi And-avainsanalla on mahdollista lisätä askelia testiin (kuvio 6).



KUVIO 5. Salasanan syöttökenttä

```
4
5   Feature: Input password
6
7   As a user I want to input my password
8
9
10
11  Scenario: User inputs too long password
12
13  When I input too long Password
14
15  And Press submit
16
17  Then I should see "Error"
18
19
20
```

KUVIO 6. Esimerkki BDD-menetelmän käyttäjätarinasta ja askeleista

Lisäksi testeihin toteutettiin erillinen @smoke-tagilla merkitty testijoukko. Tageilla on helppo merkitä erilaiset tapaukset, jotka mahdollisesti halutaan ajaa eri kerroilla. Koska kokonaisen testijoukon ajaminen saattaa kestää puolesta tunnista useisiin tunteihin, tämä smoke-testijoukko tehtiin, jotta kaikkia testejä ei tarvitsisi ajaa joka kerta vaan olisi nopeaa ja helppoa ajaa pelkästään tärkeimmät tapaukset.

Calabash tuntee jo valmiiksi tiettyjä askelia, joiden käyttö helpottaa ja nopeuttaa testien tekoa alkuvaiheessa. Lisää askelia on silti tarve tehdä, jotta askelista ja samalla testeistä saataisiin helpommin luettavia ja muokattavia. Nämä yksityiskohtaisemmat askeleet tehdään calabash_steps.rb-tiedostoon. Askeliin on myös mahdollista saada lisää ohjelmallisuutta. Esimerkiksi salasanan syöttöön voidaan ohjelmoida funktio, jolloin saman salasanan käyttö myöhemmin testissä on mahdollista.

3.6 Testien ylläpito

Tärkeä osa testausta ja toimivia testejä on niiden elinkaari. Jossain vaiheessa tullaan tilanteeseen, että testauksen kohteena olevaa tapausta on täytynyt syystä tai toisesta muuttaa ja, jotta testeistä voitaisiin saada luotettavaa tulosta vielä tulevaisuudessakin, on testejä pystyttävä muokkaamaan ja päivittämään.

Tämä ei välttämättä ole ongelma jos testejä on päivittämässä sama henkilö, joka ne on alun perin ollut tekemässä. Jos taas testien ylläpito vastuu on siirtynyt toiselle henkilölle, jolla ei ole välttämättä ole ohjelmointiosaamista, olisi tärkeää, että testit tehtäisiin mahdollisimman yksinkertaisiksi ylläpitää. Tämä on eräs BDD-testauksen parhaita ominaisuuksia ja asioita mihin erityisesti kiinnitettiin huomioita. Testeistä pyrittiin tekemään aina mahdollisimman luettavia ja ymmärrettäviä, jolloin niiden ylläpito helpottuu.

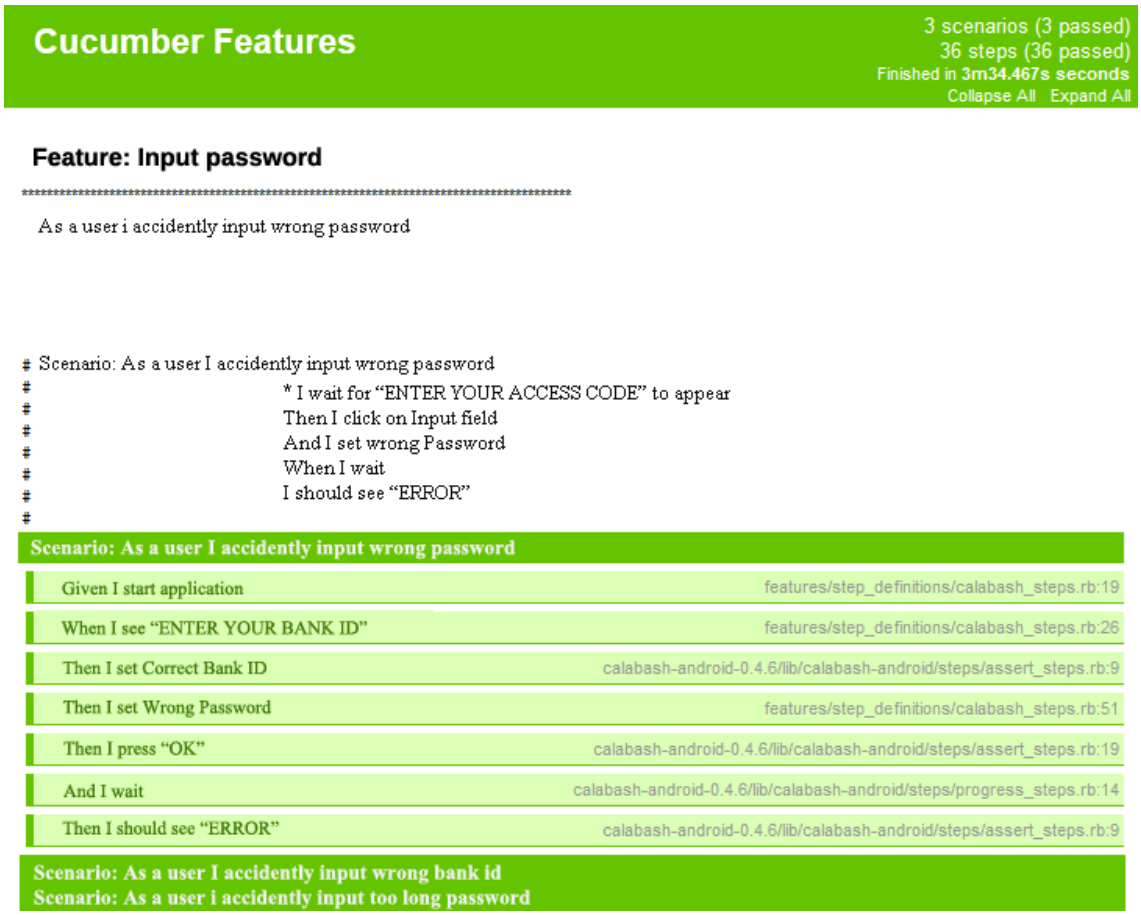
3.7 Testien ajaminen

Testien ajaminen on tärkeä osa automaatiotestausta. Lähinnä mahdollisia toteutusvaihtoehtoja on joko käynnistää testit käsin aina tiettynä ajankohtana tai asettaa testit esimerkiksi käynnistymään automaattisesti jonkin jatkuvan integraation työkalun avulla tiettynä kellonaikana tai esimerkiksi versionhallintaan tehdyn muutoksen jälkeen. Varsinkin jos halutaan välttyä inhimillisen virheen mahdollisuudelta on jälkimmäinen eli testiajon automatisointi järkevin ratkaisu. Toki tämä vaatii oman osaamisen henkilöltä, joka ylläpitää ja huolehtii mahdollisista jatkuvan integraation laitealustoista.

KUVIO 7. Käsin ajettaessa testit käynnistetään Rubyn komentoriviltä

Hyväkään testi ei ole riittävä jos siitä ei saada selkeää ja helppolukuista raporttia. Raportti on väline, jolla saadaan helposti palautetta virheiden määrästä. Calabash on mahdollista käskeä antamaan html-muotoinen raportti aina testiajon jälkeen (kuvio 8). Raportista nähdään helposti jos jokin testitapaus ei mene läpi. Lisäksi Calabash-sovellus ottaa kuvankaappauksen virheen aiheuttavasta sovelluksen kohdasta.

Usein raporttia voidaan käyttää myös havainnollistamisvälineenä asiakkaalle tai johdolle tiimin edistymisestä, jolloin erityisesti korostuu sen merkitys. Lisäksi tallennetuista raporteista on mahdollista seurata pidemmällä aikavälillä sovelluksen laatua.



KUVIO 8. Calabash-raportti

4 JOHTOPÄÄTÖKSET

Tutkimuksen kohteena oli selvittää millä keinoilla ja työkaluilla olisi mahdollista tehostaa mobiilisovellusten testausta. Aluksi käytiin läpi aikaisempia menetelmiä ja miten niiden kautta on päädytty tämänhetkisiin menetelmiin. Tarkempi tutkiminen keskittyi BDD-menetelmään ja sen mukanaan tuomiin ominaisuuksiin. Käytännön osuudessa selvitettiin miten BDD-menetelmää hyväksikäyttäen voidaan luoda testejä ja testitapauksia. Tämän tutkimuksen perusteella voidaan todeta kyseisen kaltaisen testauksen säästävän huomattavasti resursseja ja vapauttavan testaajat tärkeämpiin ja vaikeammin automatisoitaviin kohteisiin.

Vaikka BDD-testaus helpottaa tietyiltä osin testaajan kuormaa, se ei kuitenkaan sovellu ihan kaikkeen. Testattaessa esimerkiksi paikkatietoja on tarkkaan harkittava tehdäänkö testit toimimaan vain tietyssä sijainnissa vai jätetäänkö tämä kenties kokonaan manuaali testattavaksi.

Tutkimukseen valikoitui Calabash-testaussovellus, jonka avulla kehitin Pivo-sovellukseen testit. Calabash oli positiivinen kokemus testisovelluksien rintamalla. Se täytti täysin sille asetetut odotukset ja vaikka minulla ei aiempaa kokemusta kyseisestä ohjelmasta ollutkaan, sen käyttöönotto oli helppoa. Testaussovellus ymmärsi suoraan kehitystyössä käytetyt käyttäjätarinat, jolloin testeistä saatiin helposti luettavia ja ymmärrettäviä.

5 POHDINTA

Tarkoituksenani oli tutustua BDD-menetelmään ja sitä kautta miettiä tehokasta keinoa jolla voidaan ratkaista mobiilisovellusten kehitykseen ja testaukseen liittyvät ongelmat. Vaikka raportissa on keskitytty tutkimusongelmaan mikä mobiilisovellusten kehitykseen liittyy, oli opinnäytetyöni suurimmaksi osaksi kehitystehtävä, jossa pääsin hyödyntämään uusia oppimiani tietoja ja kehittämään niiden pohjalta testit Pivo-mobiilisovellukselle.

Vaikka Calabash helpottaakin testaamista, on sen asennukseen varattava aikaa. Varsinkin ensimmäisellä kerralla kaikkien liitännäisten asennukseen voi kulua aikaa. Eteenkin jos asennukset tehdään yrityksen sisäverkosta, voi tietoturva aiheuttaa ongelmia. Lisäksi alussa testien kirjoittaminen voi olla hidas. Myöhemmässä vaiheessa hyvin suunnitellut testiaskeleet nopeuttavat ohjelmoitia.

BDD-menetelmä voidaan kiteyttää, että se vaatii koko tiimin panostamista kyseiseen kehitysmalliin, jos halutaan saavuttaa parhaat mahdolliset tulokset. Lisäksi varsinkin alkuvaiheessa testien kehitystyö voi olla aikaa vievää ja kuormittaa muutenkin testausta ja yleistä kehitystyötä, mutta kun BDD-menetelmä saadaan toimimaan tiimin sisällä, se mahdollistaa kokonaisvaltaisen nopean kehitystyön. Käyttäytymislähtöisen menetelmän käyttöönotto vaatii hiukan enempi aikaa johonkin muuhun verrattuna. Lisäksi menetelmän täysi ymmärtäminen voi muodostua ongelmaksi varsinkin alkuvaiheessa. Tämä ehkä sen takia, että menetelmästä ei löydy paljoa täysin aukotonta ja yksiselitteistä tietoa, koska menetelmä on enempi oma näkökanta katsella ohjelmistokehitystä.

Testitapauksia miettiessä kannattaa kiinnittää huomiota myös kielivalintoihin. Onko jokin syy käyttää jotain muuta kuin oletuksena olevaa englantia? Lisäksi testitapauksista kannattaa tehdä mahdollisimman selkokielisiä ja helposti luettavia. Testien luettaviksi tekeminen hidastaa testitapausten kehittämistä huomattavasti, mutta jos testejä on tarkoitus muokata myöhemmin esimerkiksi sovelluksen käyttöliittymän uudelleenohjelmoinnin vuoksi, tämä helpottaa sitä huomattavasti. Lisäksi seuraavien testien ylläpitäjien on helppoa lukea testikoodia.

Lisäksi on hyvä miettiä valitseeko testin osoittamaan esimerkiksi tiettyyn painikkeeseen tai kenttään ID:n vai kentän sijainnin mukaan. Käyttöliittymän muuttuessa ID yleensä pysyy samana,

joten tämä ei vaadi uudelleenohjelmointia, mutta jos testejä on tarkoitus käyttää esimerkiksi toisen käyttöjärjestelmän vastaavaan sovellukseen, voi sijainnin mukaan määritelty kenttä olla hyvä vaihtoehto.

Tulevia projekteja silmällä pitäen tulisi jo projektin suunnitteluvaiheessa miettiä käytettävää testausmenetelmää ja kehittää muutakin kehitystyötä tukemaan tätä. Lisäksi on huomioitavaa, että käyttäytymispohjainen ohjelmistokehitys vaatii alussa tarkempaa suunnittelua ja koko kehitystiimin panostusta kuin esimerkiksi testivetoinen kehitys.

LÄHTEET

Agiledata.org Introduction to Test Driven Development. Hakupäivä 4.10.2014
http://www.agiledata.org/essays/tdd.html

Agile Enterprises.com Advantages of Agile Development. Hakupäivä 19.7.2014
http://www.agileenterprises.com/agile-development/advantages-of-agile-development

Avustaja.blogspot.fi BDD behavior driven design Hakupäivä 19.7.2014
http://avustaja.blogspot.fi/2008/11/bdd-behaviour-driven-design.html

Base36.com Automated vs. Manual Testing: The Pros and Cons of Each. Hakupäivä 20.11.2014
http://www.base36.com/2013/03/automated-vs-manual-testing-the-pros-and-cons-of-each/

GitHub.com Calabash-Android Hakupäivä 19.7.2014
https://github.com/calabash/calabash-android

Karjalainen, V 2012 Testivetoinen web-sovelluskehitys. Helsingin Yliopisto,
Tietojenkäsittelytiede. Pro gradu.

Kärkkäinen, S. 2013 Käyttäytymislähtöinen ohjelmistokehitys. Metropolia Ammattikorkeakoulu, Tietotekniikan koulutusohjelma. Opinnäytetyö.

LessPainful Calabash. Hakupäivä 23.7.2014
http://blog.lesspainful.com/2012/03/07/Calabash/

Nieminen, A 2014 Käyttäytymislähtöisen kehityksen työkalut. Metropolia Ammattikorkeakoulu, Tietotekniikan koulutusohjelma. Opinnäytetyö.

Toikkanen T. Auer A. Auer L. Heinäsmäki M. Hölttä J. Kalliala E. Laanti M. Laine K. Lekman L. Miinalainen P. Naski H. Piiparinen T. Puhakka H. Pyhäjärvi M. Pääkkönen T. Rosti J. Räisänen S. Sora H. Taipale M. Talvio J. Tanninen A. Toivola T. Toro K. Valsta A. Väyrynen

V&Weissenberg M 2013. 398 Vuotta ketteriä kokemuksia. S52. Oulu: Systeemityöyhdistys Sytyke ry

Wikipedia.fi Ketterä ohjelmointikehitys. Hakupäivä 19.7.2014
http://fi.wikipedia.org/wiki/Ketterä_ohjelmistokehitys