Testien suunnittelu ja toteuttaminen PHP-sovellukselle

Jukka Tuomi

2020 Laurea

Laurea-ammattikorkeakoulu

Testien suunnittelu ja toteuttaminen PHP-sovellukselle

Jukka Tuomi

Tietojenkäsittely

Opinnäytetyö

Marraskuu,2020

Laurea-ammattikorkeakoulu **Tiivistelmä**
Tietojenkäsittely koulutus
Tradenomi (AMK)

Jukka Tuomi

Testien suunnittelu ja toteuttaminen PHP-sovellukselle

Vuosi 2020 Sivumäärä 17

Tässä opinnäytetyössä suunniteltiin ja toteutettiin testejä PHP-sovellukselle. Toimeksiantajana työssä toimi pieni IT-alan konsultointia tarjoava yritys. Työn tarkoitus oli luoda sen toimeksiantajalle kuva siitä, että miten sen asiakkaalleen tuottamaa ohjelmistoa voitaisiin mahdollisesti testata ja toteuttaa alustava toteutus sen tärkeimmäksi koetusta testattavasta ominaisuudesta.

Tätä varten työssä tutustuttiin kyseiselle ohjelmistolle helposti käytettävissä oleviin testausviitekehyksiin. Lisäksi työssä perehdyttiin ohjelmistotestauksen yleiseen teoriaan ja valittiin näitä tietoja hyväksi käyttäen työtä varten parhaaksi nähty lähestymistapa toteutukselle. Opinnäytetyön tutkimusmenetelmänä käytettiin konstruktiivista tutkimusmenetelmää.

Opinnäytetyön tuotoksena ovat testaussuunnitelma, sen pohjalta kirjoitettu yksikkötesti ja toimeksiantajalle testien mukana toimitettu dokumentaatio testeistä. Testaussuunnitelma kirjoitettiin ennen varsinaisten testien toteuttamista ja sen pohjalta aloitettiin varsinainen testien kirjoittaminen. Testien kirjoittamisen yhteydessä kirjoitettiin dokumentaatioon muistiin testien toiminnasta ja lopussa siihen lisättiin yksityiskohtaiset ohjeet testien ajamiseen ja ylläpitoon, sekä jatkokehitysehdotuksia.

Työn tuloksia arvioitiin pohtimalla sen tavoitteiden toteutumista ja sen toimeksiantajalle tuomaa hyötyä. Toteutuksien pohjalta kirjoitettiin myös mieleen tulleista ongelmakohdista ja kehitysehdotuksista.

Asiasanat: Ohjelmistotestaus, testaussuunnitelma, PHP

Laurea University of Applied Sciences **Abstract**
Bachelor of Business Administration (BBA)
Bachelor's Degree Programme in Business Information Technology

Jukka Tuomi
Design and implementation of tests for PHP application
Year 2020 Pages 17

In this Bachelors thesis, tests for a PHP application were designed and implemented. The client was a small IT consulting company. The purpose of the thesis was to give the client an idea of how the software it produced for its client could possibly be tested and to implement a preliminary implementation of its most important testable feature.

To this end, this study introduced familiar testing frameworks for that software. In addition, the general theory of software testing was introduced and the best approach to implementation was chosen using this information. The constructive research method was used as the research method of the thesis.

The output of the thesis was a test plan, a unit test written based on it and the documentation of the tests that was submitted to the client with the tests. The test plan was written before the actual tests were made and the actual writing of the tests was started based on it. The documentation of the functionality of the tests was written down when the test where written, and at the end detailed instructions for running and maintaining the tests were added, as well as suggestions for further development.

Results of the study were evaluated by considering the fulfillment of its objectives and its benefits brought to the client. Based on the implementations, problem areas and suggestions for improvement were also written down

Keywords: Sofware testing, test plan, PHP

Sisällys

1 Johdanto

Tässä opinnäytetyössä suunniteltiin ja toteutettiin asiakasyritykselle alustavat testit sen ohjelmistotestauksen jatkokehittämistä kehittämistä varten. Työssä perehdytään testauksen yleiseen teoriaan pintapuolisesti ja tutustutaan siihen, että millainen työhön liittyvä ohjelmisto on.

Testien toteuttamista ennen perehdyttiin ohjelmalle sopiviin valmiiksi tarjolla oleviin testausviitekehyksiin. Niiden toimintaa ja tarjoamia ominaisuuksia käytiin pintapuolisesti läpi.

Tietoperustan pohjalta testien toteuttamista varten valittiin työhön sopiva testausviitekehys ja sopiva testauksen taso. Työssä luotavaa testausta varten luotiin testaussuunnitelma, jota hyväksi käyttäen testit pyrittiin toteuttamaan.

Suunnitelmaa hyödyntäen ohjelmalle toteutettiin yksikkötesti. Testin toteutuksen kulkua ja sen toimintaperiaatetta kuvailtiin pintapuolisesti. Lopuksi tarkasteltiin työn tuloksia, siinä kohdattuja ongelmakohtia ja sekä työn kautta saavutettuja hyötyjä.

2 Työn lähtökohdat

Tämä opinnäytetyö tehtiin yritykselle X. Kyseessä on konsultointi yritys, joka myy ohjelmistokehitystyötä. Työssä käsitellään yrityksen asiakkaan ohjelmistolle tehtävien ohjelmistotestien suunnittelemista ja toteuttamista.

Työ sai alkunsa, kun lähdin pohtimaan, että mikä olisi aihe, josta olisin itse kiinnostunut ja joka olisi hyödyllinen myös työnantajalleni yritys X:lle. Päädyin siihen, että ohjelmistotestien suunnittelu ja niiden toteuttaminen olisi tällainen aihe.

2.1 Tietoa yrityksestä ja ohjelmistosta

Yritys X on pääasiassa pääkaupunkiseudulla toimiva yritys, joka tarjoaa IT-alan ja maanrakentamisen konsultointipalveluita. Yritys on kooltaan pieni ja se työllistää vain kaksi henkilöä.

Opinnäytetyön kohteena oleva asiakasyrityksen ohjelmisto koostuu tuotannonohjausjärjestelmästä ja sivustosta, jonka kautta luodaan tilauksia kyseiseen järjestelmään. Ohjelmistot on pääosin toteutettu PHP-ohjelmointikielellä käyttäen Codeigniter nimistä viitekehystä. Lisäksi siinä on osittain käytetty hyödyksi myös Vue.js viitekehystä.

2.2 Ratkaistava ongelma

Ohjelmiston testausta oli tarve saada automatisoitua. Kaikki testaus tehtiin silloisella hetkellä manuaalisesti siten, että ominaisuuksia testattiin aina yksitellen, joko kehittämistyön yhteydessä, tai ennen uuden version tuotantoon viemistä.

Yrityksen, sekä asiakkaan puolesta koettiin, että ohjelmistolle olisi tarpeen määrittää testit, jolla voitaisiin vähentää testaukseen kuluvaa aikaa ja parantaa testauksen luotettavuutta. Manuaalisesti tehtävien testien kanssa isoin ongelma oli se, että niitä varten tuli varata aina oma aikansa ja siihen ei yksinkertaisesti aina riittänyt aikaa ja resursseja.

2.3 Aiheen rajaus

Tässä työssä päädyttiin rajaamaan aihe siten, että varsinainen toteutus tehtäisiin käsittelemään vain ohjelmiston tietyn osa-alueen testausta. Tähän päädyttiin siksi, koska oli selvää, ettei tämän työn puitteissa saataisi kirjoitettua testejä, jotka kattaisivat koko ohjelman toiminnan ja näin ollen haluttiin keskittyä vain tärkeimmäksi koettujen osa-alueiden testien toteuttamiseen.

Työssä laaditussa testaussuunnitelmassa pyrittiin kuitenkin käsittelemään pintapuolisesti myös, että miten testejä voidaan mahdollisesti jatkokehittää, sekä sitä, että mitä muita viitekehyksiä ja tekniikoita voitaisiin tällöin käyttää.

2.4 Työn tavoite

Yksi tärkeimmistä tavoitteista oli kartoittaa, että mitä ohjelmistosta tulisi testata. Oli helposti ymmärrettävissä, ettei työn puitteissa saataisi koko ohjelmiston kaikkiin osa-alueisiin tehtyä kattavia testejä. Siksi työssä oli siis tärkeää muodostaa mahdollisimman tarkka kuva siitä mikä olisi ohjelmistossa kaikista tärkein testattava asia, mikä toiseksi tärkein ja niin edelleen. Tämän lisäksi tärkeätä oli selvittää, että oliko näiden osa-alueiden testaaminen kuinka helposti toteutettavissa ja sitä kautta mahdollisesti muuttaa tärkeysjärjestystä

Toinen tärkeä tavoite oli suunnitella testeistä mahdollisimman helposti ylläpidettävät. Käytännössä tämä tarkoitti, että tavoitteena oli luoda testejä varten dokumentaatio, josta kävisi ilmi, että kuinka testit toimivat, kuinka niitä käytetään, ja että kuinka niitä tulisi ylläpitää.

Näiden pohjalta muodostui käytännön tavoite saada toteutettua yksinkertainen pohja, jonka päälle testausta voitaisiin rakentaa. Tätä varten tuli testausta varten valita myös tarkoitukseen parhaiten sopiva testausviitekehys. Valitulla viitekehyksellä toteutettaisiin yksittäinen, tai muutamia yksinkertaisia testejä ohjelmaan, joista pystyttäisiin jatkossa ottamaan mallia.

3 Testauksen teoriaa

Ohjelmistotestaus jaotellaan useimmiten tapahtuvan erillisillä tasoilla. Tasoja ovat yksikkötestaus, integraatiotestaus, järjestelmätestaus ja hyväksymistestaus. (Kasurinen 2013, 64.)

Tasojen lisäksi puhutaan myös testausmenetelmistä. Testausmenetelmiä, joita käytetään ohjelmiston kehitysvaiheessa ovat mustalaatikko-, lasilaatikko-, ja harmaalaatikkotestaus. (Kasurinen 2013, 64.)

3.1 Testauksen tasot

Testaustasot eroavat toisistaan siinä, että niissä nimensäkin mukaisesti testausta tehdään eri tasoilla. Yksikkötestaus tehdään yhdelle komponentille, integraatiotestaus useammalle kuin yhdelle komponentille ja järjestelmätestauksessa testataan kokonaisvaltaisesti ohjelman eri komponenttien toimintaa yhdessä. Viimeisenä tasona toimii hyväksymistestaus, jossa ohjelman testataan jo joko sen kohde ympäristössä tai sitä tarkasti mallintaen. (Kasurinen 2013, 51.)

Yksikkötestaus on testausmenetelmistä yleisimmin käytetty ja tavallisin. Sen tarkoituksena on varmistaa, että ohjelman pieni toiminnallinen yksikkö eli moduuli, funktio, tai olio toimii ainakin periaatteessa. Siinä esimerkiksi rakennetaan joukko kutsuja, jotka ajetaan testattavalle funktiolle ja joihin testattavan funktion pitää palauttaa oikea vastaus, tai toteuttaa oikea toiminto. Jos yksikkötesti epäonnistuu, niin silloin yksikön vika pystytään korjaamaan ennen kuin kyseistä osaa käytetään osana ohjelman laajempaa osaa. (Kasurinen 2013, 52.)

Integraatiotestauksessa tärkeimpänä tavoitteena on se, että pystytään havaita käytännössä järjestelmän eri osien toimivan myös yhdessä. Se eroaa siis yksikkötestauksesta siinä, että siinä testausta tehdään yhden osan sijasta ohjelmiston useammalle yksittäiselle osalle ja järjestelmä testauksesta siinä, että siinä ei kuitenkaan testata vielä kuitenkaan koko järjestelmää kokonaisuutena. (Kasurinen 2013, 54.)

Järjestelmätestaus on nimensä mukaisesti sitä testaustyötä, jolla testataan kokonaista järjestelmää. Sen tarkoitus on varmistaa, että järjestelmä toimii kokonaisuutena ja toteuttaa kaikki tavoitteet, jotka sille on asetettu. Yleisesti järjestelmätestauksella tarkoitetaan, että suoritetaan testitapauksia käyttäen lasilaatikko- ja mustalaatikkotestausta, mutta sillä voidaan kuitenkin tarkoittaa myös muunlaista testausmenetelmää, jossa tarkastellaan ohjelman toiminnallista kokonaisuutta. (Kasurinen 2013, 57.)

Hyväksymistestaus on testauksen vaihe, jossa järjestelmä tarkastetaan virallisesti. Siinä tavoitteena on osoittaa se, että ohjelma on tarpeeksi korkealaatuinen ja täyttää sille asetetut

vaatimukset. Siinä on tavallista, että testaus siirrettään tapahtumaan sen kohdeympäristössä erillisen testausympäristön, tai muun rajoitetun ympäristön sijasta. (Kasurinen 2013, 57.)

3.2 Testausmenetelmät

Testausta voidaan haluta tehdä usealla eri menetelmällä riippuen esimerkiksi siitä, kuinka valmis ohjelma on. Ohjelman kehitysvaiheessa puhutaan yleisesti mustalaatikko-, lasilaatikko-, ja harmaalaatikkotestauksesta. Näiden lisäksi jo lähes valmiille tai osittain käytössä olevalle ohjelmalle testausmenetelmiä ovat myös esimerkiksi käytettävyystestaus, kuormitustestaus, suorituskykytestaus, sekä alfa- ja betatestaus. (Kasurinen 2013, 70.)

Musta laatikko on testausmenetelmä, jota käytetään yleisimmin ohjelmistoprojektin testitapauksissa. Se eroaa lasilaatikko- ja harmaalaatikkotestauksesta siinä, että siinä ei olla kiinnostuneita ohjelman sisäisestä toiminnasta. Toisin sanoen siinä ei siis seurata tai tutkita, että mitä ohjelman kooditasolla tapahtuu. Siinä periaatteena on, että ohjelmalle annetaan testitapauksien mukaisia syötteitä ja tarkastellaan vastaavatko saadut tulokset niille määriteltyjä odotettuja tuloksia. (Juvonen 2018, 29.)

Lasi laatikko tai valkoinen laatikko on testausmenetelmä, jossa testaus tehdään siten, että tarkastellaan myös ohjelman sisäistä toimintaa. Se on työläämpi testausmenetelmä, kuin mustalaatikko, koska siinä testaajan tulee olla perehtynyt ohjelmiston sisäiseen rakenteeseen ja mahdollisesti sen lähdekoodiin. Sen etuna verrattaessa mustaan laatikkoon, voidaan nähdä se, että siinä testaajalle kehittyy parempi kuva ohjelmiston toiminnasta, josta voidaan myöhemmin hyötyä. (Juvonen 2018, 29.)

Harmaalaatikko on testausmenetelmä, joka pitää sisällään mustanlaatikon ja valkoisenlaatikon testausmenetelmät. Jos esimerkiksi tiedetään tietyn ohjelman osan olevan erityisen kriittinen ja altis virheille, niin silloin sitä voidaan tutkia tarkemmin valkoisenlaatikon menetelmällä, kun taas ohjelman muiden osien testaukseen voidaan käyttää menetelmänä mustaa laatikkoa. (Juvonen 2018, 29.)

4 PHP-ohjelman testaus viitekehykset

PHP-sovelluksen testaukseen on olemassa useita valmiita viitekehyksiä, jotka on suunniteltu helpottamaan sovelluksen testausta. Työssä tutustuttiin pääpiirteittäin kolmeen mielestäni tässä tapauksessa parhaaseen vaihtoehtoon ja käytin läpi niiden ominaisuuksia. Näitä olivat PHPunit, Selenium ja Codeigniterin yksikkötestaus luokka.

PHPunit tarjoaa monipuolisia työkaluja ohjelmiston kattavaan testaukseen. Sitä käyttämällä voidaan tehdä ohjelmalle yksikkötestejä, joissa testataan ohjelman toiminnallisuutta

yksittäisten osien pohjalta. PHPunit ei kuitenkaan sisällä toiminnallisuutta, jolla pystyttäisiin luomaan simulaatiota, jossa selaimessa testataan ohjelman käyttöliittymää ja sen toimintaa, toisin kuin Seleniumissa. (Software Testing Help 2020.)

Seleniumin avulla pystytään testaamaan ohjelman käyttöliittymää simuloimalla käyttäjän toimintaa sivustolla. Sen käyttäminen on mahdollista useilla eri ohjelmointikielillä vain pelkän PHP:n sijaan. Seleniumin etuna on PHPunitiin verrattaessa se, että sillä pystytään testaamaan käyttöliittymää ja se että sen ohjelmointi on mahdollista useilla eri ohjelmointikielillä. Sillä ei kuitenkaan pysty kuitenkaan testaamaan yhtä helposti ohjelman yksittäisiä osia. (Software Testing Help 2020.)

Ohjelmistossa on valmiina käytössä Codeigniter viitekehys, josta löytyy yksikkötestausluokka. Sen toiminta on hyvin pelkistetty verrattuna PHPunit ja Selenium viitekehyksiin. Se tarjoaa käytännössä vain hyvin pelkistetyn pohjan, jonka päälle on mahdollista rakentaa monimutkaisempia testejä. Se toimii käytännössä siten, että siinä verrataan luodun testin tulosta sille ennalta asetettuun oikean tulokseen ja sen perusteella testi näytetään läpäistynä, tai pieleen menneenä. Sen etuna kuitenkin on se, että se löytyy jo valmiiksi ohjelmasta ja se, että sen toiminnan ymmärtäminen on helpompaa sen yksinkertaisuuden vuoksi. (Codeigniter 2020.)

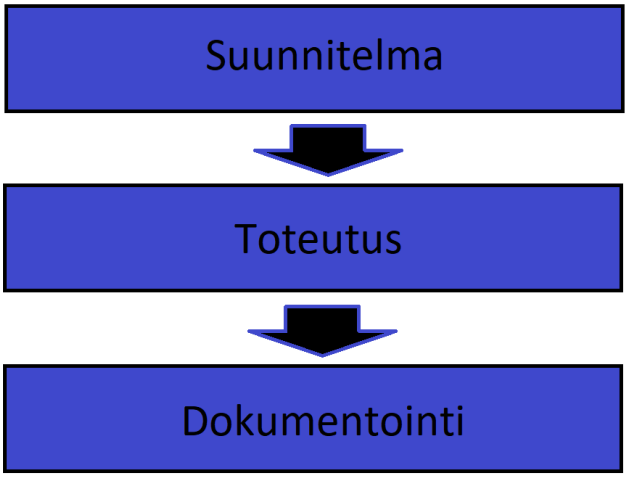
5 Tutkimusmenetelmä

Tämän opinnäytetyön tutkimusmenetelmä on konstruktiivinen tutkimusmenetelmä. Siinä aikaisemmin olemassa olevan tietojen pohjalta, pyritään rakentamaan uusia innovaatioita, sekä tutkitaan, että miten ne voitaisiin rakentaa. Nämä innovaatiot tarkoittavat tavoiteltuja uudistuksia, joiden on kautta, tuotetaan jotain hyötyä. Tutkimus jaetaan kolmeen erilliseen osaan: lähtötila, toteuttaminen ja tavoitetila. Aluksi lähtötilanne selvitetään havaitsemalla ongelmakohdat ja asettamalla tavoitteet. Toteuttamisvaihe pitää sisällään suurimman osan ja se pitää sisällään projektin käytännön toteuttamisen. Viimeisenä tulee tavoitetila, johon tutkimus päätetään. Siinä arvioidaan, että miten innovaatiot toteutettiin ja saavutettiinko niillä tavoiteltu lopputulos. (Järvinen & Järvinen 2004, 104–107.)

Reliabiliteetti ja validiteetti ovat käsitteitä, joiden avulla voidaan arvioida luotettavuutta. Ne perustuvat korrespondenssiteoriaan. Validiteetilla tarkoitetaan sitä, että vastaavatko tutkimustulos ja asian todellinen tila toisiaan, kuinka hyvin. Reliabiliteetti kertoo, että kuinka hyvin tutkimustulos pysyy samana, jos tutkimus toistetaan uudestaan. (Kakkori 2009, 53.)

6 Toteutus

Opinnäytetyön toteutus jakautui testaussuunnitelman laatimiseen ja testin varsinaiseen toteuttamiseen ja dokumentaation laadintaan (Kuvio 1). Testaussuunnitelma laadittiin ensin. Siinä vaiheessa käytiin läpi tarkemmin se, että mitä toteutettavien testien tulisi testata, mitä niiden toteuttamisessa tulisi ottaa huomioon ja se, että miten ne tulisivat toimimaan.



Kuvio 1: Testien toteutuksen kulku vaiheittain

Testien toteutusta kuvaavassa osiossa käydään pintapuolisesti läpi se, miten niiden toteutus käytännössä eteni. Lisäksi osiossa kuvataan niiden toimintaperiaatetta. Toteutuksen viimeisessä vaiheessa kerrotaan lyhyesti dokumentaatiosta

6.1 Testaussuunnitelma

Ohjelmiston tärkeimpiä testattavia asioita selvitettiin miettimällä yrityksen työntekijöiden kesken ja kysymällä suoraan asiakkaalta asiasta. Tärkeimmäksi testattavaksi koettiin ohjelmistossa tapahtuvan laskennan testaus. Toisaalta pidettiin tärkeänä myös, että ohjelman monia muita osia testattaisiin automaattisesti, koska koettiin, että aikaa ei yleisesti ole riittävästi testata aina uusia toiminnallisuuksia manuaalisesti.

Jotta testit toimisivat mahdollisimman pitkään ja olisivat luotettavat, niin tulisi niitä varten olla selkeä dokumentaatio. Tässä vaiheessa suunniteltiin, että mitä tuon dokumentaation tulisi pitää sisällään, jotta sitä voitaisiin rakentaa työssä samalla kun kirjoitetaan testejä. Dokumentaatio kirjoitettaisiin testit sisällään pitävään kansioon erilliseen tekstitiedostoon. Siinä kerrottaisiin mahdollisimman selkeästi, että miten testit toimivat. Tiedostossa tulisi olla ohjeistus siitä, että miten testejä ajetaan ja miten niitä voitaisiin muokata. Opinnäytetyön

puitteissa tavoite olisi saada ohjelmiston laskentaa varten luotua testejä ja laajentaa vasta tulevaisuudessa testausta käsittämään myös integraatiotestejä.

Toteutusta varten kirjoitettiin pääpiirteittäin ylös työn testien rakenne ja toimintaperiaate. Ohjelmassa tapahtuvaa laskentaa varten tulisi kirjoittaa yksikkötestit. Nämä testit toteutettaisiin käyttämällä Codeigniter viitekehyksen yksikkötestausluokkaa. Laskennan testejä voitaisiin käyttää myös jatkossa pohjana, jos ohjelmiston muille osille haluttaisiin toteuttaa yksikkötestejä.

Testejä varten tallennettaisiin valmiisiin pohjiin tiedot siitä, mitä tietoja laskentaa tekevien yksikköjen tulisi ottaa ja mitä tietoja niiden tulisi palauttaa. Tätä varten tulisi kerätä varmat tiedot siitä, että mitkä ovat missäkin tapauksessa oikeat arvot, jotka ohjelman tulisi palauttaa. Laskennan testien oikeat vastaukset tulisi saada suoraan asiakkaalta, jolloin he pystyisivät parhaiten varmentamaan, että laskenta todella toimii halutulla tavalla.

Laskennan testien oikeiden vastausten lisäksi tulisi tallentaa samaan tapaan erillisiin pohjiin virheellistä sisään otettavaa tietoa ja virheellisiä vastauksia, joilla voitaisiin testata, että testi todella palauttaa oikeanlaisen virheen, jos laskennan arvot eivät vastaa oikeita arvoja. Virheellisiä arvoja tulisi testata mahdollisimman monipuolisesti siten, että laskentaa tekevälle yksikölle annettaisiin sekä vääriä arvoja, että väärässä muodossa olevaa dataa.

Laskennan testaukseen käytettävien valmiiden tietojen tallentaminen tulisi tehdä selkeästi erillisiin kansioihin ja tiedostoihin ohjelman sisälle. Testien tulisi olla mahdollisimman helppolukuisia ja niissä pitäisi olla selkeästi kommentoituna niiden toiminta periaate.

6.2 Testien kirjoittaminen

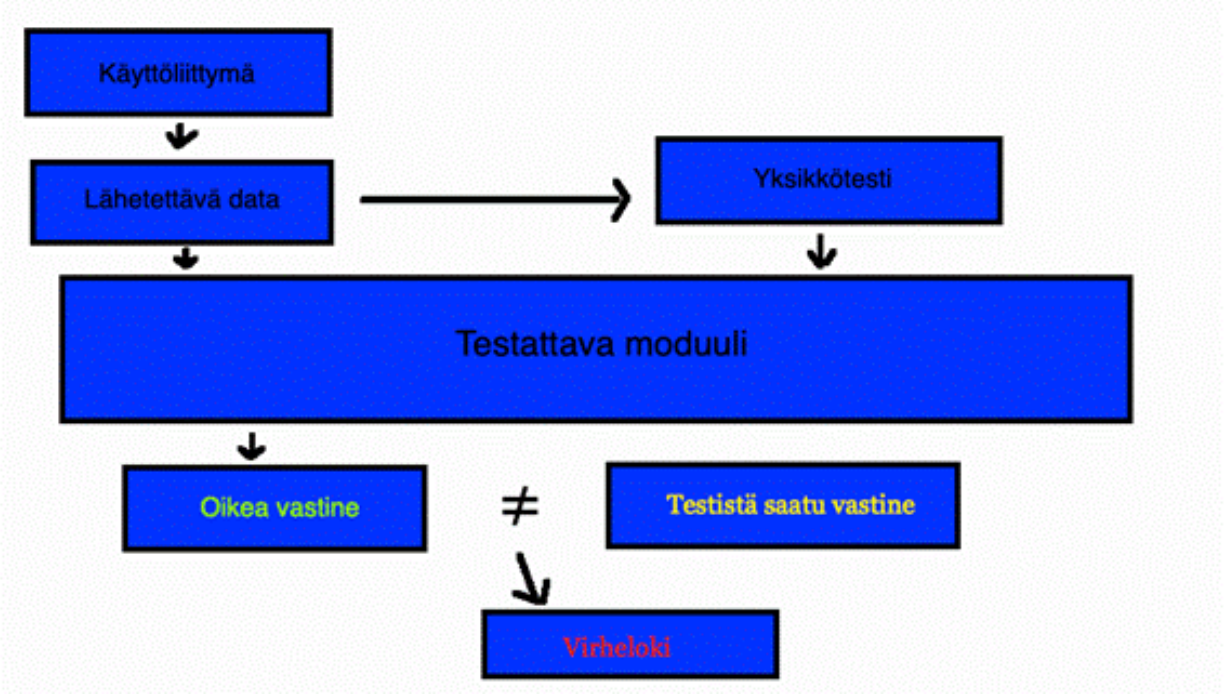
Testien kirjoittaminen aloitettiin luomalla omat kansiot testeille. Kansioiden sisään luotiin uusi tekstitiedosto, johon tultaisiin tekemään testeihin liittyvä dokumentaatio. Tämän jälkeen tutustuttiin Codeigniterin yksikkötesti luokkaan, sen dokumentaatioon ja siihen, että miten sitä käytetään.

Kun yksikkötestien luomiseen käytettävän kirjaston toiminnallisuus oli selvillä, aloitettiin varsinaisten testien kirjoittaminen. Testit pyrittiin rakentamaan mahdollisimman helpoiksi ymmärtää ja pitää huolta siitä, että niiden ylläpitäminen olisi mahdollisimman helppoa. Testit päädyttiin kirjoittamaan vain yhdelle ohjelman laskentaa tekevää luokkaa varten, joka oli aikaisempien kyselyiden pohjalta todettu kaikista tärkeimmäksi testattavaksi asiaksi koko ohjelmassa.

Yksikkötesti tehtiin siis testaamaan vain yhtä ohjelmiston laskentaan liittyvää luokkaa. Tämän luokan toiminta periaate oli sellainen, että se otti sisäänsä dataa JSON-muodossa, joka

ohjelmassa muodostui lomakkeista, joita käyttäjä täytti. Sisään otetun datan perusteella suoritettiin laskentaa ja lopuksi palautettiin dataa JSON-muodossa.

Testiä varten luotiin jokaiselle testille omat tiedostot, joissa oli määritelty, että mitä dataa testi lähettää ja mitä sen tulisi vastaanottaa testattavasta luokasta. Lähetettävä data kopioitiin suoraan ohjelman käyttöliittymässä luoduista lomakkeista ja tallennettiin JSON-tiedostoihin. Näihin JSON tiedostoihin tallennettiin, myös tieto siitä, että mikä oli odotettu data, jota testattavan luokan tulisi palauttaa. Sitä varten testiin tehtiin toiminnallisuus, jossa testi kirjoitti aina erillisiin lokitiedostoihin, että mitä se oli palauttanut. Ajamalla testi ensimmäistä kertaa ei ollut siis vielä tiedossa mikä olisi oikea palautettava data, niin siksi se käytiin kopioimassa lokitiedostosta ja tallennettiin kyseistä testiä varten luotuun JSON pohjaan testin oikeaksi vastaukseksi. Kuviossa 2 on esitetty testin toimintaperiaate pääpiirteittäin



Kuvio 2: Testin toimintaperiaate

Luotujen yksikkötestien ajaminen tehtiin menemällä sitä varten luotuun selaimen näkymään, johon tulostui kaikista testeistä niiden tulokset. Testit kirjoitettiin siten, että niitä olisi mahdollista jatkossa liittää osaksi laajempaa testauksen kokonaisuutta.

6.3 Dokumentointi

Testien kirjoitusvaiheessa kirjoitettiin samalla dokumentaatiota luoduista testeistä. Dokumentaatio sijoitettiin sijaitsemaan testien kanssa samaan tiedostoon. Kirjoitusvaiheessa

dokumentaatioon kirjoitettiin muistiin testien toimintaperiaate, mitä testauksella voitaisiin testata ja mitä testi ei testannut.

Toteutuksen valmistuttua dokumentaatiota täydennettiin vielä lisää. Siihen kirjoitettiin ohje, jossa vaihe kerrallaan kerrottiin, kuinka testi ajettiin ja kuinka sitä muokattaisiin. Lisäksi siihen kirjattiin ylös mahdollisia jatkokehitysehdotuksia.

7 Tulokset ja johtopäätökset

Työn perusteella saatiin luotua suunnitelma ohjelmiston testausta varten ja luotiin yksinkertaiset ensimmäiset testit, joiden päälle testausta voitaisiin jatkossa kehittää. Samalla myös saatiin parempi ymmärrys siitä, että miten tämän kaltaista ohjelmistoa voitaisiin testata.

Työn pohjalta tuli selkeämmäksi se, että kuinka ohjelmistotestauksella pystyttäisiin mahdollisesti vähentämään oleellisesti testaukseen käytettävää aikaa ja parantamaan sen luotettavuutta. Samalla saatiin myös parempi ymmärrys siitä, että mitä asioita automatisoituja testejä varten tulisi selvittää, mitkä ovat sen heikkoudet ja se, kuinka haastavaa kaiken kattavien testien tekeminen käytännössä olisi.

Tämän tyyppisessä työssä validiteetin ja reliabiliteetin arviointi on hankalaa, koska työn pohjalta ei saada selkeää dataa sen tuomista tuloksista ja niiden laadusta, vaan sen tuoma lisäarvo on nähtävissä vain mahdollisesti toimeksiantajan prosesseissa. Kuitenkin mielestäni työn validiteettia voitaisiin pitää hyvänä, koska kyseiset testit testaavat juuri sitä ohjelman osa-aluetta, joita sen oli tarkoitus testata. Samalla tavoin sen reliabiliteettia voidaan mielestäni pitää työssä hyvänä, koska testien toiminta on samanlainen, riippumatta siitä kuinka usein, tai milloin niitä ajetaan.

Suurimpana ongelma työn tuloksena syntyneissä testeissä näen niiden käyttöönoton ja niiden ylläpitämisen. Ohjelman testaukseen ei ole käytetty aikaa muussa, kuin tarpeellisiksi koetuissa tilanteissa ja tämän takia koen haastavaksi sen, että jatkossa testien kehittämiseen ja parantamiseen tultaisiin käyttämään tulevaisuudessa tarpeeksi aikaa. Automatisoitujen testien yhtenä suurimpana haasteena koen sen, että niitä tulee ylläpitää ja päivittää osana kehitystyötä, koska muutoin ne eivät ole luotettavia ja voivat pahimmillaan jopa hidastaa testausta, verrattaessa manuaaliseen testaukseen.

Toinen ongelmakohta on se, että testaavatko ne tarpeellisia ominaisuuksia ja voidaanko niiden läpäisyn oikeasti kokea kertovan ohjelman virheettömästä toiminnasta. Onko siis mahdollista, että testit nimittäin näennäisesti testaavat jonkun luokan toimintaa, mutta eivät kuitenkaan havaitse siinä ilmeneviä bugeja, joita luullaan voivan havaita testillä.

Työn tuloksena saatuja testien toimintaa ei ole niiden valmistumisen jälkeen otettu käyttöön kokonaisvaltaisesti ja siksi niistä saatavaa hyötyä ei voida vielä arvioida tarkasti. Kuitenkin tätä varten tekemästäni opiskelusta ja käytännön ohjelmoinnista on ollut paljon apua päivittäisessä työssäni

Luotujen testien käytännön hyöty on siis tullut ilmi toistaiseksi vain omassa työssäni. Olen käyttänyt hyödyksi testejä, joiden avulla olen voinut saada nopeutettua testaamistani ja varmistuttua paremmin siitä, että koodiin tekemäni muutokset eivät riko testattavan komponentin toimintaa. Toisaalta hyötynä voidaan myös nähdä yleisesti testauksesta oppimani asiat, jotka valmistavat minua mahdollisesti jatkokehittämään ohjelmiston testejä tulevaisuudessa.

Työn pohjalta ohjelmistoon kirjoitettiin ohjeet siinä luotujen testien toiminnasta ja ohjeet niiden päivittämistä varten. Niiden pohjalta voivat muut ohjelman kehitykseen osallistuvat ihmiset tutustua helpommin kyseisissä testeissä käytettyihin tekniikoihin ja oppia käyttämään testejä jatkossa myös omassa toiminnassaan.

Lähteet

Painetut

Järvinen, A. & Järvinen, P. 2004. Tutkimustyönmetodeista. Tampere: Opinpajan kirja

Juvonen, R. 2018. Ohjelmistoprojektin sudenkuopat ja miten ne vältetään. Helsinki:Books on Demand.

Kasurinen, J. 2013. Ohjelmistotestauksen käsikirja. Jyväskylä: Docendo.

Kaakkori, L. 2009. Martin Heideggerin olemisen kysyminen. Tampere: Tampereen yliopiston paino Oy-Juvenes Print.

Sähköiset

Software Testing Help 2020. Top 10 Popular PHP Testing Frameworks And Tools. Viitattu 01.11.2020. https://www.softwaretestinghelp.com/php-testing-framework-tools/

Code Igniter 2020. Unit Testing. Viitattu 01.11.2020. https://codeigniter-id.github.io/user-guide/libraries/unit_testing.html

Kuviot