Antti Karkimo

Yksilöidyn yrityskohtaisen prosessin integroiminen osaksi ohjelmistokehitystyötä pienyrityksessä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

26.4.2018

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Antti Karkimo Yksilöidyn yrityskohtaisen prosessin integroiminen osaksi ohjelmistokehitystyötä pienyrityksessä 29 sivua 26.4.2018
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikan koulutusohjelma
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Lehtori Outi Grotenfelt

Tämän lopputyön tarkoituksena on kuvata pienen ohjelmistoyrityksen matka kohti parempaa kehitysprosessia. Yrityksen käytössä ollut Scrum (tai oikeastaan ScrumBut) ei palvellut sitä toivotulla tavalla. Kuten useimmissa Scrumia käyttävissä kehitystiimeissä, Scrumin osa-alueita käytettiin vain sen mukaan, olivatko ne yritykselle sopivia. Kohdeyrityksessä tyypillinen tehtävä oli liian pitkä, epäselvästi määritelty, ja varsinaisen tuotteen synty ei ollut läheskään niin ketterää, kuin se olisi voinut olla.

Ohjelmistokehitysprosessin mielekkyys nostettiin esille, ja lopulta päätettiin virallisesti tehdä se, jota useimmat ohjelmistokehitystiimit eivät myönnä tekevänsä: kehittää oma malli ottamalla ne osat muista malleista, jotka tiimille toimivat, muokata niitä ja kuvata tämä lopputulos prosessina. Vaikka kaikki ei olekaan mennyt niin kuin alun perin suunniteltiin, on lopputulos silti virkistävä, hyödyllinen sekä edelleen käytössä yrityksessä.

Avainsanat	Scrum, testaus, CI, TDD, Lean

Author(s) Title Number of Pages Date	Antti Karkimo Integrating a company specific process to be a part of software development in a small company 29 pages 26 April 2018
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Outi Grotenfelt, Lecturer

The aim of this thesis is to describe the journey of a small software company towards a better development process. Scrum (or to be more specific, ScrumBut) used by the company wasn't serving it in the way needed. Like most teams using Scrum, parts of Scrum were used only if they seemed to fit. In the target company the typical task was often too long, the specification was unclear and that final product wasn't shaping up in the agile way it might have.

The sensibleness of the process was questioned, and in the end, it was decided to officially do what most software development teams don't admit doing: develop an entire model of their own by taking parts of other models that worked for the team, modify them and describe this end result as a process. Even though everything didn't go as planned, the result is refreshing, useful, and still in use in this company.

Keywords	Scrum, testing, CI, TDD, Lean

Sisällys

Lyhenteet

CI	Continuous integration. Jatkuva integraatio sekä sitä toteuttava koontipalvelin, joka näkee koodimuutokset ja ajaa kaikki testit koko koodipohjalle.
CSS	Cascading Style Sheet. Verkkoselaimen ymmärtämä tyylitiedosto.
DOM	Document object model. Verkkoselaimen sisäisesti käyttämä puurakenne, joka sisältää kaikki verkkosivulla olevat elementit hierarkkisessa järjestyksessä.
IDE	Integrated development environment. Ohjelmisto, joka yhdistää ohjelmakoodin kirjoittamiseen ja ajamiseen tarvittavat työkalut.
PHP	PHP Hypertext Preprocessor. Eräs verkko-ohjelmointikieli.
SSH	Secure shell. Salattu verkkoprotokolla, jolla voidaan ottaa yhteyksiä turvattoman verkon päällä.
TDD	Test driven development. Työtapa, jossa testit määrittelevät toiminnon onnistuneen implementoinnin.
TVUP	Toinen veli unified process. Tämän insinöörityön tuotteena syntynyt ohjelmistokehitysprosessi.
VPN	Virtual private network. Virtuaalinen lähiverkko, joka tunneloidaan minkä tahansa toisen verkon yli, esimerkiksi julkisen internetin yli.
XAMPP	Cross-platform, Apache, MySQL, PHP, Perl. Jakelupaketti, joka yhdistää yleisimmät PHP-verkko-ohjelmointiin käytetyt kielet ja palvelimet yhdeksi asennukseksi useille käyttöjärjestelmille.
XML	Extensible Markup Language. Laajennettava merkintäkieli.

XP	Extreme programming. Ketterän ohjelmistokehityksen metodologia, joka painottaa lyhyitä kehityssyklejä.
Xpath	XML Path Language. Alunperin XML-dokumentin nodejen valintaan kehitetty kyselykieli. Voidaan käyttää myös DOM-puun kuvaamiseen.

1 Johdanto

Ohjelmistoja tekevän yrityksen tärkein työkalu toimivan lopputuloksen saavuttamiseen ei ole käytetty ohjelmointikieli tai laitteet vaan toimiva ohjelmistokehitysprosessi. Ilman prosessia työn edistymisen ja lopputulosten seuranta on mahdotonta. Erilaisia malleja ja prosesseja on ollut niin kauan kuin ohjelmistokehitystäkin, ja näiden välillä valitseminen on epätriviaali haaste. Koska yrityksiä, tiimejä, yksittäisiä kehittäjiä tai tuotteita on hankala mahduttaa samaan muottiin, joudutaan malleista yleensä valitsemaan paras kompromissi.

Tässä työssä kompromissi on tietoinen lähtökohta, ja sillä pyritään yhdistämään olemassa olevista malleista ja työkaluista tälle yritykselle toimivin ratkaisu. Tämän työn tavoitteena oli kehittää ja kuvata erään ohjelmistokehitysyrityksen uusi kehitysprosessi yrityksessä havaittujen epäkohtien pohjalta olemassa olevia malleja hyväksikäyttäen.

Koska prosessi tehtiin puhtaalta pöydältä juuri tämän yrityksen tarpeisiin, oli mahdollista suunnitella se toimimaan myös oikeiden, käytettävissä olevien työkalujen kanssa. Tällaisia työkaluja ovat esimerkiksi koontipalvelin ja versionhallintatyökalut. Työssä esitellään yritys, työtä edeltävä tilanne yrityksessä sekä käytössä olevat mallit. Parhaassa tapauksessa lukija voi saada inspiraation oman työpaikkansa kehitysprosessien parantamiseen, tai voi tahtoessaan implementoida tässä esitellyn mallin sellaisenaan.

2 Lähtötilanne

2.1 Yrityksestä

Tämä lopputyö tehtiin Toinen veli Oy:lle. Toinen veli Oy on helsinkiläinen vuonna 2011 perustettu ohjelmistokehitysyritys. Toinen veli Oy on syntynyt alun perin Metropolian yritysesihautomossa. Yritys työllistää viisi henkeä. Pääliiketoimintaan kuuluu konsultoinnin lisäksi tuoteprojekteja. Tässä insinöörityössä esitetty malli kehitettiin alun perin nimenomaan yhden näistä tuoteprojekteista tarpeisiin.

2.2 Tuote

Yrityksen päätuote on kiinteistöhuoltoyrityksille tehty KiinteistöVELI-järjestelmä. KiinteistöVELI tarjoaa kiinteistöhuoltoa tekeville yrityksille alustan, jossa yritys voi säilyttää tiedot huoneistoista, asukkaista, huoltotöistä, kulukirjauksista ja kaikista asuinkiinteistöissä tapahtuvista oheistoiminnoista, kuten saunavuoroista ja autopaikoista. KiinteistöVELI on internetselaimella käytettävä SaaS-palvelu. Se on kirjoitettu PHP:llä, jQueryllä, CSS3:lla ja HTML5:llä. Sitä on kehitetty aktiivisesti vuodesta 2012 lähtien, ja jotkin asiakkaat ovat olleet käyttäjinä aivan tästä ensimmäisestä versiosta lähtien.

Tuote on jatkuvasti käytössä, joten siihen tehtävät muutokset on tehtävä täydellisellä yhteensopivuudella taaksepäin. Kiinteistöhuoltoyhtiöille tämä järjestelmä mahdollistaa kaiken toimistotyön automatisoinnin, työn ohjauksen ja sen seurannan. On siis ensiarvoisen tärkeää, että mitään tietoa ei katoa ja että katkokset ovat mahdollisimman lyhyitä. Alan peruspiirteistä johtuen kiinteistöhuoltoyhtiöiden on päästävä katsomaan ja muokkaamaan tietoja minä kellonaikana hyvänsä. Esimerkiksi jos asukas on kadottanut avaimensa, on hänen tietonsa pakko päästä tarkistamaan ennen oven avaamista, vaikka keskellä yötä. Lisäksi asukkaat saattavat ilmoittaa havaitsemistaan vioista tai tehdä muuttoilmoituksia viikonloppuisin.

Kaikki tuotantoon vietävät muutokset näkyvät välittömästi kaikilla asiakkailla, sillä he käyttävät samaa tuotantoasennusta. Tämä ei aiheuta lisäsuunnittelua ainoastaan ohjelmiston teknisiin vaatimuksiin, kuten suorituskykyyn, mutta lisäksi ohjelmiston varsinaiseen käytettävyyteen. Jos yksi asiakas toivoo muutosta ohjelmiston käyttölogiikkaan, on se toteutettava sillä tavalla, ettei se sotke täysin käyttölogiikkaa ja aiheuta hämmennystä

muille käyttäjille. Ohjelmiston toimintojen hallinta on suunniteltu sillä tavalla, että Toinen veli Oy:n on mahdollista ottaa käyttöön tai poistaa käytöstä keneltä tahansa asiakkaalta tai asiakkaan käyttäjältä mikä tahansa toiminnallisuus vaikuttamatta muihin käyttäjiin. Koko järjestelmä rakentuu tämän käyttöoikeustoiminnon varaan, ja se on todella granulaarinen aivan yksittäisiin käyttöliittymän nappeihin asti. Tämä sisäänrakennettu ominaisuus tarjoaa mahdollisuuden räätälöidä käyttökokemus tiettyyn rajaan asti.

2.3 Ohjelmistokehitysprosessit

Ohjelmistokehitysprosessilla voidaan käsittää jonkinlaisen lopputuotteen syntyyn vaikuttavista tekijöistä kuinka laajaa osaa tahansa. Tämän insinöörityön puitteissa prosessilla tai metodologialla ajatellaan kuitenkin kaikkea sitä, mikä tapahtuu sen jälkeen, kun on havaittu jonkinlainen tarve ohjelmiston kehittämiselle. Yleisemmällä tasolla mukaan voitaisiin lukea myös askeleet ongelmaan perehtymisestä, markkinatutkimuksesta ja vaatimusten keräämisestä. Valitettavasti raja piti vetää johonkin, joten keskityin niihin osa-alueisiin, jossa yritys oli tunnistanut parantamisen mahdollisuuksia.

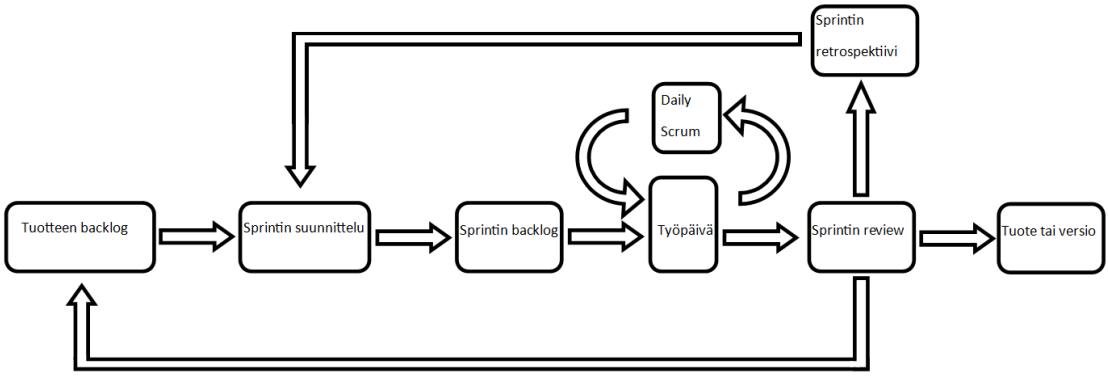
2.4 Scrum

Yrityksessä oli pitkään ollut käytössä Scrum. Scrum on iteratiivinen, ketterä eli agile-ohjelmistokehitysmetodologia. Agile-projektit iteratiivisia, sillä ne luovat edellytykset ohjelmistokehitystoimien toistamiselle tai uudelleenkirjoittamiselle [1]. Scrumissa on kolme tärkeää työroolia.

Työrooleista ensimmäinen on *tuoteomistaja*. Tuoteomistaja voi olla yrityksen sisältä, tai ulkopuolinen asiakkaan edustaja. Hänen tehtävänsä on suomeksi sanottuna vaatia edustamansa organisaation haluamia ominaisuuksia tuotteeseen. Hän ylläpitää nk. product backlogia toivotuista ominaisuuksista. Product backlog sisältää ominaisuudet prioriteettijärjestyksessä, ja sitä tyypillisesti päivitetään jatkuvasti sitä mukaa, kun projekti etenee tai ulkopuoliset tahot sitä edellyttävät. Ominaisuudet ovat tyypillisesti käyttötapauksina tai käyttäjätarinoina. Backlog on tuoteomistajan työkalu varmistaakseen, että kehittäjät laittavat työpanoksensa tuoteomistajan kannalta tärkeimpiin ominaisuuksiin. Hänen tehtävänsä ei ole käyttää aikaansa niinkään teknisten toteutusten miettimiseen, vaan katsoa tuotekehitystä liiketoiminnan kannalta.

Toinen tärkeä työrooli on *Scrum master*. Scrum masterin vastuu on luoda kehittäjille mahdollisimman hyvät edellytykset toimia työssään ja poistaa mahdolliset esteet tai hidastukset. Hänen tehtävänään on myös vastata varsinaisen Scrum-prosessin toiminnasta. Hän pitää huolen siitä, että Scrumiin liittyvät tapahtumat, kuten sprinttipalaverit, katselmoinnit ja päivittäiset nk. stand-upit tapahtuvat. [2.]

Viimeinen rooli on itse *kehitystiimi*. Kehitystiimin vastuulla on toteuttaa varsinainen sprintin tavoite, eli ideaalitilanteessa toimitettava tuote. Toteutukseen ei sisälly pelkästään itse ohjelmakoodin kirjoitus, vaan myös suunnittelu ja testaus.



Kuva 1. Scrumin eteneminen.

Työ jaetaan ajallisesti sprintteihin, jotka kestävät kahdesta neljään viikkoa. Sprintin tavoitteena olisi tarkoitus olla tuotos, joka on valmis käytettäväksi. Sprintin aikana tehdään itse Scrumiin liittyen muutamia erilaisia toimenpiteitä (kuva 1). Jokaisen sprintin alussa on suunnittelusessio, jossa kehitystiimi ottaa product backlogin kärjestä toteutettavia asioita sprintin ajaksi. Tässä määritellään myös sprintin tavoite. Näistä backlogin asioista, esimerkiksi käyttötapaukset, luodaan työtehtävät eli taskit. Yhden sprintin taskit muodostavat tämän sprintin backlogin. [3, s. 9–10.]

Päivittäin, yleensä päivän alussa, on Daily Scrum tai stand-up. Kaikki kehitystiimin jäsenet ovat paikalla dailyssa, ja sen pitäisi tapahtua samaan aikaan joka päivä. Keston olisi syytä olla enintään 15 minuuttia. Dailyssa jokaisen on tarkoitus vastata kolmeen kysymykseen:

- Mitä saavutin eilen?
- Mitä aion saavuttaa tänään?
- Onko tiedossani haasteita, jotka vaikeuttavat sprintin valmistumista?

Dailyn tarkoitus ei ole aikaansaada välitöntä pitempää keskustelua, mutta varsinkin haasteet olisi syytä ottaa ylös myöhempää tarkastelua varten. [3, s. 11.]

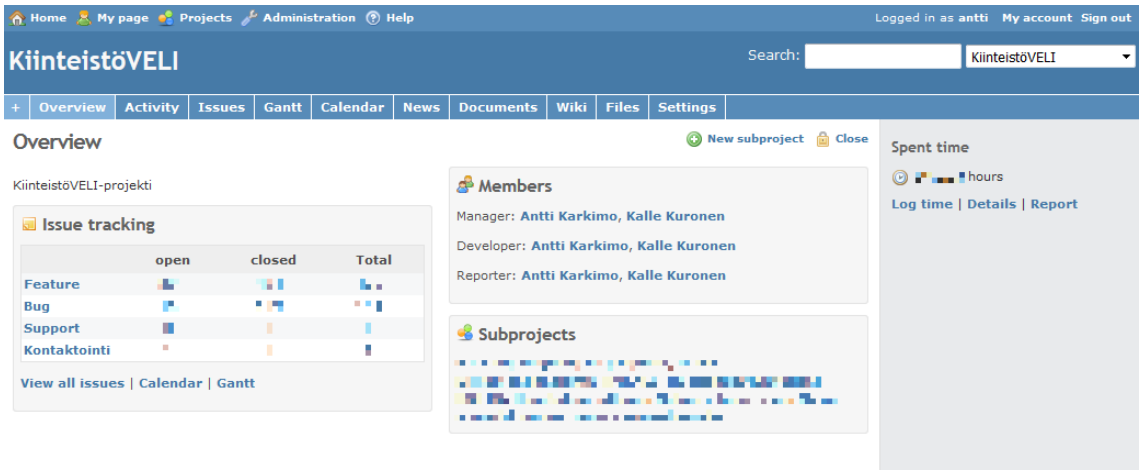
Kun sprint on ohi, nk. review'ssä katsotaan, mitä saatiin aikaiseksi ja mitä jäi tekemättä [4]. Review’ssä on paikalla kehitystiimin lisäksi sekä Scrum master että tuoteomistaja. Kuluneen sprintin saavutetut työtulokset esitellään, ja tuoteomistaja vertaa tuloksia koko tuotteen backlogiin. Kehitystiimi kertoo, mikä sprintissä meni hyvin ja mihin ongelmiin törmättiin, sekä sen, miten ongelmat ratkaistiin. Review’n lopputuloksena pitäisi olla kaikille selkeä kuva siitä, miten projekti etenee, ja antaa pohja seuraavan sprintin suunnittelulle. Koko tuotteen backlogia saatetaan säätää sekä tulosten perusteella, että myös mahdollisesti muuttuneen markkinatilanteen tai muun vastaavan pohjalta tuoteomistajan taholta. [3, s. 12.]

Review’n ja seuraavan sprintin suunnittelun välissä on sprintin retrospektiivi. Tämä sisältää metakeskustelua siitä, miten sprint sujui työnteon sujuvuuden kannalta. Tyypillisiä aiheita voivat olla esimerkiksi ihmisten väliset suhteet, prosessit ja työkalut. Retrospektiivin tarkoitus on saada tiimi toimimaan paremmin. Retrospektiivin jälkeen alkaa seuraavan sprintin suunnittelu.

KiinteistöVELIn kehityksessä käytettiin kahden viikon sprinttejä. Työ oli sujunut hyvin ensimmäisen kahden vuoden aikana, mutta asiakkaiden määrän lisääntyessä alkoi myös ominaisuuspyyntöjen ja havaittujen bugien määrä kasvaa. Lisäksi tulevista ominaisuuksista ja niiden aikataulutuksesta alkoi tulla asiakkailta ja muilta sidosryhmiltä entistä enemmän kysymyksiä. Alun perin hyvin ketteränä alkanut kehitystyö oli alkanut lipsua

kohti perinteistä vesiputousmallia, sillä lupauksia tulevasta piti tehdä niin tiukasti tulevaisuuteen. Ketterän ohjelmistokehityksen julistus määrittelee arvostettavammaksi asiaksi muutokseen vastaamisen suunnitelmassa pitäytymisen sijaan [5], mutta tilanne ajautui koko ajan entistä vahvemmin toiseen suuntaan työn mielekkyyden kustannuksella.

Yrityksessä Scrumia käytettiin Redmine-sovelluksella (kuva 2). Redmine mahdollistaa koko projektin hallinnan, sprintien ajoituksen ja issue-järjestelmän, johon taskit luodaan. Taskeille on mahdollista luoda erilaisia tiloja, joissa ne liikkuvat elinkaarensa aikana. Tilat voivat olla esimerkiksi ’suunnittelussa’, ’työn alla’, ’testauksessa’ ja ’valmis’.



Kuva 2. Projekti Redmine-sovelluksessa. Yksityiskohtia poistettu tietosuojasyistä.

2.5 ScrumBut

ScrumBut eli vapaasti suomennettuna ScrumMutta viittaa erilaisiin syihin, joita ohjelmistokehitystiimit tarjoavat sille, miksi eivät toteuta kaikkia Scrumin osasia. Jokainen Scrumin rooli, sääntö ja aikalaatikko on suunniteltu tarjoamaan halutut edut ja ratkaisemaan ennustettavia ongelmia [6]. Yleensä ScrumButit kertovat jostain organisaatiossa piilevästä kipupisteestä eivätkä niinkään itse mallin mahdottomuudesta. Käytännössä yksi ScrumBut voisi olla esimerkiksi: ”Käytämme Scrumia, mutta emme järjestä retrospektiivejä, koska niissä sanotaan ikäviä asioita.” Tässä tapauksessa ScrumBut ei oikeasti johdu siitä, että retrospektiivien pitäminen olisi mahdotonta, vaan esimerkiksi siitä, että työyhteisössä esiintyisi henkilöiden välisiä negatiivisia jännitteitä. Oikeasti henkilöiden kokemat negatiiviset tuntemukset olisivat hyvä aihe käsiteltäväksi retrospektiivissä, niin

niihin voisi puuttua. [6.] Ainakin jonkin ScrumButin esiintyminen vaikuttaa olevan melko yleistä, varsinkin pitempään Scrumia käyttäneissä organisaatioissa [7].

Tämän insinöörityön kohdeyrityksessä lähtökohtiin vaikuttanut suurin ScrumBut oli roolituksen puute. Tiimin pienestä koosta johtuen ei ollut mahdollista saada erillistä tuoteomistajaa, Scrum masteria sekä ohjelmistokehitystiimiä. Koska tämän ScrumButin oireet eivät olleet vielä aiemmin osoittautunut ongelmaksi, ei rooleja myöskään jaettu kellekään, vaan kaikki tekivät vähän kaikkea. Kun tuoteomistajaa ei ollut, ei kukaan ollut myöskään vastuussa siitä, että backlogia käytäisiin läpi. Näin ollen tuotteen myöhästyminen ja epähyödyllisten asioiden tekeminen ei aiheuttanut toimenpiteitä.

2.6 Testaus

Ohjelmiston testaus oli toteutettu toimivasti, mutta äärimmäisen raskaasti: käytännössä ainoa testausmenetelmä oli hyväksytystestaus uuden version valmistuttua ennen käyttöönottoa. Testaussuunnitelma (kuva 3) kaikista testitapauksista tehtiin manuaalisesti taulukkolaskentaohjelmalla, määriteltiin tarvittavat selaimet (esimerkiksi Mozilla Firefox, Google Chrome, Internet Explorer 9 / 10 / 11, Apple Safari) ja toteutettiin testit manuaalisesti jokaisella testi/selain-yhdistelmällä. Vaikka testaus oli hyvin perinpohjaista, se oli silti hyvin hidasta – kaksi ihmistä testasi pientäkin versiomuutosta tyypillisesti ainakin pari päivää mahdollisine bugikorjauksineen.

Kuva 3. Ote tyypillisen version tai iteraation testaussuunnitelmasta, näkyvissä noin yksi neljäsosa. Testin kohde, oletettu tulos, ja sarake jokaista selainta kohti siitä , mitä oikeasti tapahtui.

2.7 Tilanteen kärjistyminen

KiinteistöVELIn käyttöliittymälle tehtiin täydellinen uudelleenkirjoitus alkuvuodesta 2014. Tässä uudistuksessa tuotteen näkymät suunniteltiin graafisesti uudelleen modernimmaksi. Samalla säilytettiin kuitenkin alkuperäisestä versiosta tuttu helppo käytettävyys ja yksinkertaisuus. Projekti oli ajallisesti pitkä ja työmäärältään valtava, isoin yksittäinen projekti koko tuotteen siihenastisen elinkaaren aikana. Projekti kesti noin neljä kuukautta koko tiimin työtä. Koska näin mittavaa projektia ei ollut aiemmin tehty, ajautuivat työmetodit koetukselle. Lisäksi uudelleenkirjoitus koki niin sanotun scope creepin, eli määritellyissä muutoksissa ei pysytty, vaan mukaan tuli tavaraa, joka projektiin ei olisi oikeasti kuulunut. Tämä on asia, jonka dedikoitu tuoteomistaja olisi huomannut ennen kuin siitä tulee ongelma.

Osa työtehtävistä eli taskeista oli verrattain valtavia. Yrityksessä oli havaittu, että yhden työtehtävän pituus tuntui parhaalta ollessaan alle yhden työpäivän mittainen. Parasta olisi, jos työtehtävät olisivat atomäärisiä, eli niitä ei voi jakaa enää pienempiin osiin. Tällöin tehtävän suoritus muuttuu binääriseksi – on mahdollista kertoa selkeästi, että tehtävä joko on tehty tai ei ole tehty. [8, s. 62–63.] Jos yksi kehittäjä tekee taskiaan puolet sprintistä, on hyvin haasteellista sanoa, mikä siinä kesti niin pitkään ja missä olisi parantamisen varaa. Tämä vaikeuttaa merkittävästi läpimenoajan seuraamista. Ideaalitilanteessa jokainen työtehtävä olisi samanmittainen, vaikka käytännössä se ei ole kovin realistista.

Isot työtehtävät aiheuttivat myös osallaan sen, että niiden tarkastelu review'ssä oli hankalaa. Kuinka esitellä tehty task, jos kehittäjä ei itse edes muista kaikkea, mitä se sisälsi? Puutteellisen tarkastelun myötä nämä tehdyt taskit sisälsivät enemmän bugeja kuin tyypillistä.

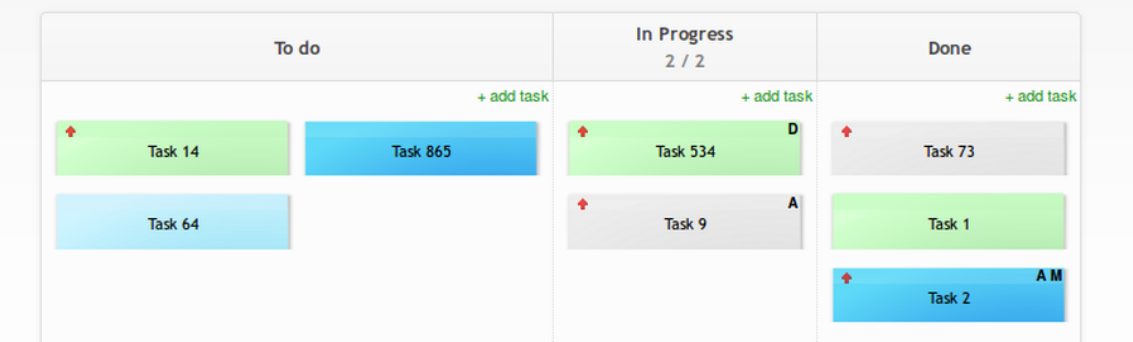
Testauskäytäntöjen takia ajettavien testien määrä oli suunnaton. Projektin lähestyessä loppuaan odotti edessä murskaavan tuntuinen seinä hyväksytystestausta. Kehittäjien piti ajaa vaadituilla käyttöjärjestelmillä ja selaimilla yhteensä tuhansia testitapauksia, ja ottaa huomioon sekä oikeat, että myös virheelliset käyttötapaukset. Testauksen ollessa käsityötä, myös tulosten virheellisen tulkinnan määrä kasvaa, kun yksi kehittäjä toistaa näennäisesti samankaltaisia tilanteita selaimellaan tuntien ajan.

Tämän projektin aikana heräsi ajatus kehitysprosessin uudistamisesta vastaamaan paremmin yrityksen tarpeita. Parannettavaa olisi ainakin testauksessa, mutta myös muissa työskentelytavoissa aivan koodin kirjoittamisesta asti. Lisäksi ScrumButeista olisi syytä päästä eroon määrittelemällä tuleville projekteille tuoteomistaja ja Scrum master.

3 Mallin muotoutumisen vaikutteet

3.1 Kanban

Samaan aikaan KiinteistöVELIn uudelleenkirjoituksen kanssa yrityksellemme tehtiin lopputyötä Kanbanin käytöstä ohjelmistokehityksessä [9]. Testasimme Kanbania eräässä sivuprojektissa, ja se vaikutti järkeenkäyvältä. Testiprojektissa toteutimme KiinteistöVELIn blogisivuston käyttäen Kanbania. Kanban juontaa juurensa alun perin Toyotan tuotantolinjoilleen kehittämästä Toyota Production System -työnhallintametodista, jossa työn suorituksen kaikki vaiheet kerätään Kanban-tauluun. Kanban-taulussa on määritelty eri vaiheita organisaation tarpeeseen, yksinkertaisimmillaan esimerkiksi 'ei aloitettu - työn alla – valmis (kuva 4)'. Kanban-taulu voi olla fyysinen taulu, jossa paperilaput vastaavat työtehtäviä, tai se voi olla verkossa toimiva palvelu. Aina kun työtehtävä saapuu yhteen vaiheeseen, tämä ajankohta merkitään ylös. Näin pystytään seuraamaan yksittäisen työn läpimenoaikaa. Jokaisessa vaiheessa olevien työtehtävien maksimimäärä on myös rajoitettu, yleensä yksi jokaista työntekijää kohden. Kanban-taulu vastaa visualisoituna sitä, mitä yrityksessä tehtiin jo Redmine-sovelluksen taskien tiloilla.



Kuva 4. Yksinkertainen Kanban-taulu. [Lähde: https://commons.wikimedia.org/wiki/File:Basic-kanban-board.png]

3.2 Lean

Myös Lean juontaa juurensa Toyota Production Systemiin. Lean software development terminä tulee Mary ja Tom Poppendieckin kirjasta, jossa leanin määritteet tuotiin ohjelmistokehityksen maailmaan. Leanissa on seitsemän periaatetta [10, s. 13–15]:

- Eliminoi ’waste’ eli turha työ.
- Vahvista oppimista.
- Päätä niin myöhään, kuin mahdollista.
- Toimita niin nopeasti, kuin mahdollista.
- Voimista tiimiä.
- Sisäänrakenna yhtenäisyys.
- Näe kokonaisuus.

Lean-mallissa turhan työn (waste) määrä minimoidaan. Kaikki, joka ei tuo lisäarvoa asiakkaalle, on wastea. Käytännössä pyritään tekemään vain sellaiset työt, joista joku asiakas on valmis maksamaan jotain. Kaikki muu työ on määritelmän mukaan wastea eli hukkaan heitettyä työtä. Tähän sisältyy esimerkiksi osittain tehty työ, ominaisuudet , joita ei ole pyydetty, ja odottelu. Yleinen ajatus Leanista ajatusmallista vaikutti järkeenkäyvältä, varsinkin reflektoiden aiempaan projektiin, johon tuli mukaan siihen kuulumattomia ominaisuuksia, sekä käsin tehtyyn testaukseen, jonka voisi myös automatisoida tai rakentaa sisään prosessiin.

3.3 TDD

Samoihin aikoihin, kun muutostarpeet prosessissa alkoivat kirkastua, näin myös hyvän esityksen TDD:stä. TDD eli Test Driven Development on ohjelmistokehityksen periaate, jossa ensin kirjoitetaan kullekin taskille testit, ja sen jälkeen vasta koodi, joka toteuttaa kyseiset testit. TDD on osa Extreme Programming (XP) -konsepteja. Kuten XP muutenkin, TDD prosessina painottaa lyhyen kehityssyklin toistamista. TDD:ssä toistetaan viittä vaihetta:

- Lisää testi.
- Aja kaikki testit ja katso, epäonnistuuko testi.
- Kirjoita varsinainen koodi.
- Aja kaikki testit.
- Refaktoroi koodi.

Kaikki alkaa testien kirjoituksella. Tärkeää on tarkistaa testien mahdollinen läpimeno ennen koodin kirjoitusta, sillä näin varmistutaan siitä, että testiympäristö itsessään toimii, ja myös siitä, että jokin aiemmin tehty toiminnallisuus ei jo toteuta testiä. Testi ei siis saa mennä läpi toisessa vaiheessa. Painoarvo on nimenomaan testin läpimenemisellä vasta neljännessä vaiheessa. Vasta viimeisessä refaktorointivaiheessa puututaan sellaisiin asioihin, kuten kovakoodattujen arvojen poistoon, jotka eivät voi olla koodissa sen mennessä tuotantoon.

KiinteistöVELIn tapauksessa TDD:stä tulisi olemaan hyötyä ainoastaan yksikkötestauksessa. Aivan koko hyväksytystestausta tällä en pystynyt poistamaan.

4 Prosessin kehitys

4.1 Kehityssuunnan valinta

Näistä lähtökohdista päätin lähteä tuottamaan Toinen veli Oy:lle toimivampaa uutta prosessia, joka poistaisi ongelmakohdat ja tarjoaisi kokonaisvaltaisen ratkaisun sisällyttäen Scrumin oikean käytön ilman ScrumButeja, mutta myös pakottaisi testauksen tapahtumaan loogisemmalla tavalla. Lisäksi halusin yhtenäistää kaikki käytössä olevat työkalut kehittäjille. Tämä tekisi prosessin inherenttien ongelmien ratkomisesta varsinkin alkuvaiheessa paljon yksinkertaisempaa.

Tutkittuani erilaisia olemassa olevia malleja läpi, en löytänyt yhtä sellaista, joka olisi vastannut kaikkiin tarpeisiimme. Pitkän harkinnan jälkeen totesin, että olisi mahdollista ottaa muista prosesseista se, mikä tuntuu parhaalta, ja laatia näistä oma kuvaus. Näin ohjelmistotuotantotyö olisi varmasti mielekkäintä toteuttaa. Lähdin liikkeelle siitä lähtökohdasta, että olisimme ehdottomasti iteratiivinen ja aiempaa ketterämpi kehityksessä. Listasin asiat, joissa kehityksessä oli parannettavaa: testaus, taskien mitta ja seuranta sekä turhan työn välttäminen.

Malli jalostui Scrumin pohjalta, sillä se oli toiminut hyvin niiltä osin, kun olimme sitä toteuttaneet.

4.2 Prosessin kuvaus

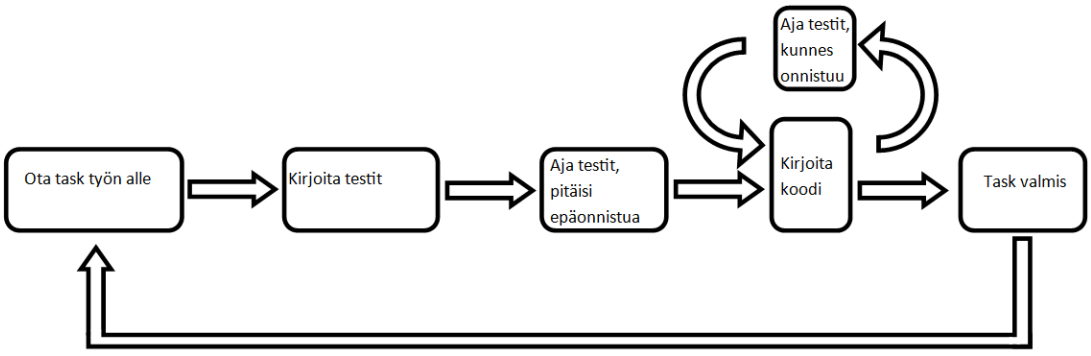
Prosessimallin nimeksi tuli TVUP, eli Toinen veli unified process. Toinen veli siksi, että se tehtiin Toinen veli Oy:lle. Unified process eli yhtenäinen prosessi viittaa siihen, että prosessi on tarkoitettu toteutettavaksi missä tahansa tämän yrityksen ohjelmistotuotteessa. Yrityksen pääkompetenssi on nimenomaan käytetyissä kielissä, joten työkaluketjussakaan ei pitäisi olla suurempaa varianssia. Jos kuitenkin kävisi niin, että kieliä vaihdettaisiin, kohdistuisivat muutokset lähinnä IDE:n vaihtumiseen johonkin toiseen IntelliJ-perheen tuotteeseen. Yksikkötestit tehtäisiin edelleen xUnit-frameworkilla, johon PHPunitin lisäksi sisältyy lukuisia muitakin kieliä. Koontipalvelin Jenkins on tässä tapauksessa kieliagnostinen, eli se ei ota kantaa koostettavan ohjelmiston kirjoituskieleen.

Prosessi noudattaa Scrumin pohjaa. Sprintin pituus on tarvittaessa vaihteleva, mutta oletuksena käytetään kahta viikkoa. Jokaisen sprintin alussa tehdään mahdolliset muutokset edellisen sprintin retrospektiivin pohjalta työskentelytapoihin, sekä tarvittaessa tehdään muutoksia itse TVUP:iin näiden pohjalta. Tuoteomistaja reflektoi mahdollisesti muuttunutta tilannetta asiakkaiden toiveiden pohjalta, ja varmistuu siitä, että kehitys on menossa oikeaan suuntaan. Sprinttiä suunnitellessa taskeille määritellään arvioitu kesto reaalisina tunteina, ei story pointeina tai vastaavina taskien keskinäistä mittaa vertailevina pisteinä.

Yksittäisen taskin sisällä suoritusjärjestys määräytyy niin, että jos kyseessä on PHP:llä backendissa tapahtuva funktio:

- Kirjoitetaan testit.
- Varmistetaan testin epäonnistuminen.
- Kirjoitetaan koodi.
- Varmistetaan testin onnistuminen.

Tätä toistetaan, kunnes koodi on refaktoroitu järkeväksi (kuva 5). Testit suoritetaan sekä niin, että ne varmistavat oikean toiminnallisuuden, sekä niin, että ne varmistavat virheilmoitukset tiedetyillä väärillä syötteillä. Tämän jälkeen task on valmis.



Kuva 5. Backend-funktion TDD-tyyppinen eteneminen TVUP:ssa.

Jos kyseessä on frontendissa tapahtuva muutos:

- Kirjoitetaan koodi (HTML-rakenne sekä JavaScript).
- Luodaan tyylit.
- Kirjoitetaan WebDriver-testit.

WebDriver-testien on myös syytä testata käyttäjän oletettavat oikeat toimenpiteet, jotka toteuttavat käyttötapauksen. Mutta lisäksi testataan kaikki mahdolliset väärät toiminnot, kuten virheelliset syötteet, jotka validaattorin pitäisi napata. WebDriver-testit ajetaan TestingBotia käyttäen ja TestingBotin tuloksesta nähdään testien onnistuminen.

Kun kaikki sprintit on ajettu loppuun, tehdään hyväksytystestaus. Tämä hyväksytystestaus pitää sisällään ulkoasun tarkastelun ja käyttötapausten toiminnan varmistaminen.

5 Työkalujen valinta ja käyttöönotto

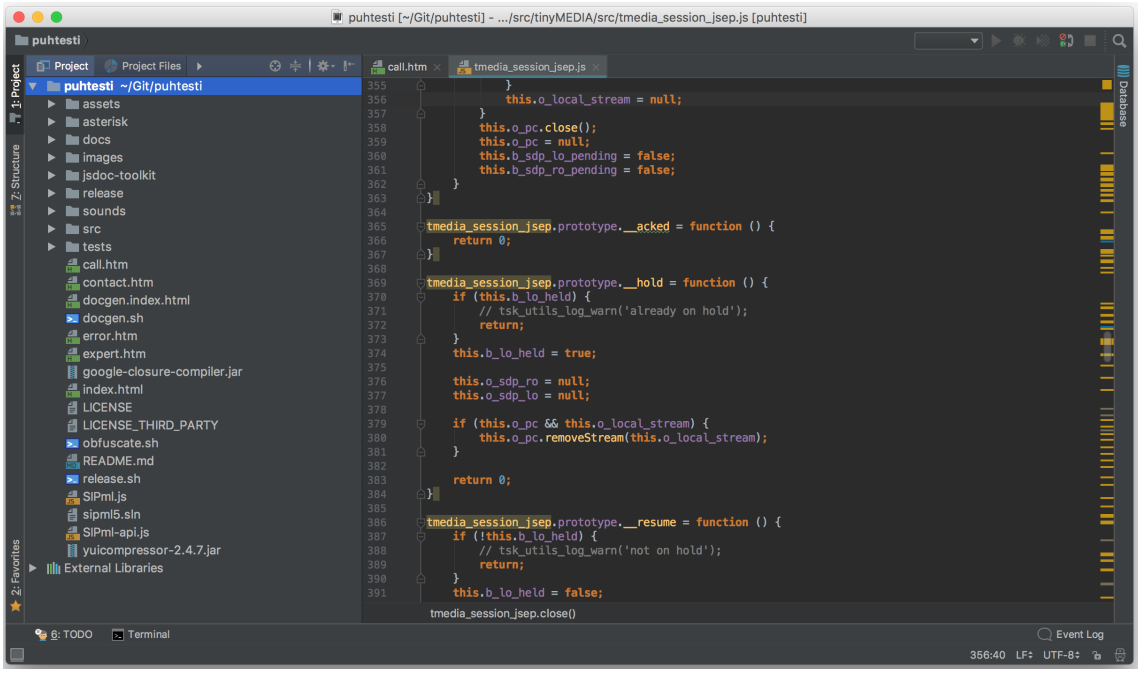
5.1 Työkalujen valinta

Yhtenäistetty työkaluketju oli integraalinen osa prosessin sujuvaa käyttöönottoa. Lähtökohtina oli testauksen automatisointi sekä yksikkötestauksen puolesta että funktionaalisen testauksen puolesta, ja jonkinlainen oikea IDE, joka yhdistäisi koodia tuottavan ohjelmoijan suoraan 1) Git-versionhallintaan, 2) kehityspalvelimeen sekä 3) koontipalvelimeen. Koontipalvelin ajaisi testit jokaiselle iteraatiolle.

5.2 IDE

Aiemmin kehittäjät olivat käyttäneet koodin muokkaamiseen itselleen mieluista tekstieditoria, yleensä vimiä. Versionhallintaan käytettyä Gitiä käytettiin suoraan komentoriviltä. Vaikka se onkin tyypillisesti tehokasta, monimutkaisemmat operaatiot, kuten kolmisuuntaiset merget tai mergekonfliktien ratkominen, voi käydä pitemmän päälle uuvuttavaksi. Lisäksi lähdekoodin siirtäminen käyttäjän omalle paikalliselle kehityspalvelimelle oli turha työvaihe, tai käyttäjän itse omalle ympäristölleen muokkaaman Git-hookin takana. Yleisesti ottaen projektin koodipohja alkoi käydä niin laajaksi, että sen visualisointi graafisella käyttöliittymällä ei voisi haitata työtehoa ja kokonaisuuden hahmottamista.

Pitkällisen kartoituksen jälkeen IDE:ksi valitsin työn tekemishetkellä kovassa nosteessa olleen Jetbrainsin PHPstormin. PHPstorm kuuluu Jetbrainsin IntelliJ-IDE-perheeseen. Tämä oli siksi kätevää, että IntelliJ-tuoteperhe pitää sisällään IDE:t myös monien muiden suosittujen ohjelmointikielien tuottamiseen. Näin ollen käyttöliittymä (kuva 6) on valmiiksi tuttu, jos kehittäjä on käyttänyt jotakin muista IntelliJ-tuotteista, kuten suosittua Java-ympäristöä, IntelliJ IDEAa. PHPstorm tuki lisäksi kaikkia vaadittuja ominaisuuksia, ja se sisältää mahdollisuuden lisätä ominaisuuksia erilaisten liitännäisten kautta. Esimerkiksi yrityksessä käytössä olleen versionhallinnan Gitin käyttöön löytyi suoraan liitännäinen. Tämän jälkeen Git löytyy vaihtoehtona kaikista PHPstormin VCS-valikoista, ja osaa näyttää esimerkiksi merge-konfliktit graafisesti ratkottavana.



Kuva 6. PHPstorm ja projektin tiedostoja.

Ohjelmiston tuki vaikutti myös olevan erinomaisella tasolla, ja netistä löytyi monenlaisia ulkopuolisiakin resursseja sen käytön avuksi. Ominaisuuksien, tuen ja yleisyyden lisäksi houkuttelevaa PHPstormissa oli myös sen startup-yrityksille muita yrityksiä edullisempi hinnoittelu. PHPstormin käyttöönotto oli suoraviivaista.

5.3 Kehityspalvelin

Kehityspalvelimeksi valittiin paikallinen Apache + MySQL + PHP -stackin yhdistävä XAMPP-palvelin aiemmista hyvistä kokemuksista tunnettuna. Yritimme myös käyttää sen tarjoamaa paikallista MySQL-palvelinta. Valitettavasti yhden tavoitteeksi asetetuista kehitysympäristöistä, Windows 8.1:n, tiedostojärjestelmä ei tee eroa isojen ja pienien kirjaimien välillä [11]. Tästä johtuen myöskään MySQL:n taulujen nimissä olleet isot ja pienet kirjaimet eivät olleet mahdollisia. Parhaaksi ratkaisuksi koettiin Windowsin tapauksessa siirtää MySQL toimimaan yrityksen yhteiselle kehityspalvelimelle aluksi SSH-tunnelin yli, ja myöhemmin OpenVPN:n kautta SSH-tunnelin suorituskykyongelmista johtuen. PHPstormista asetettiin muuttuneet tiedostot siirtymään suoraan XAMPP-palvelimelle Deploy-toiminnon kautta.

5.4 Koontipalvelin

Koontipalvelimen tehtävänä on saada tieto versionhallinnassa tapahtuvista muutoksista. Tämän tiedon saatuaan koontipalvelin ajaa kaikki projektiin määritellyt testit ja raportoi niiden onnistumisen tai epäonnistumisen. Sen jälkeen tämä tieto on keskitetysti saatavilla yhdestä paikasta koko projektin olemassaolon ajan. Myös tässä työssä kuvatun kaltaisessa prosessissa koontipalvelin tuo ylimääräisen edun siinä mielessä, että prosessissa ajetaan useammanlaisia testejä: yksikkötestejä sekä funktionaalisia testejä. Näistä kaikista tulee yhdistetysti raportti. Jos yksikin testityyppi epäonnistuu, koko build epäonnistuu. Koska koontipalvelin ajaa testit joka muutoksen jälkeen, ei pääse syntymään tilannetta, jossa muutos rikkoo jonkin aiemmin toimivaksi todetun ominaisuuden. Koontipalvelimeksi valitsin Jenkinsin, sillä minulla oli siitä aiempaa kokemusta. Myös Jenkins ja sen esi-isä Hudson ovat hyvin edustettuina erilaisissa esimerkeissä, joten ongelmatilanteisiin sen käytössä olisi helppo löytää apua internetistä. Jenkinsille on mahdollista löytää myös liitännäisiä kaikenlaisiin muihin järjestelmiin ja työkaluihin.

5.5 Yksikkötestaus

Yksikkötestaus tarkoittaa yksittäisten ohjelman osasten toiminnan testaamista irti kontekstista mahdollisimman yksinkertaisina palasina. Näin toimitaan siksi, että pystytään varmistumaan juuri kyseisen osan toiminnasta monimutkaistamatta asioita turhaan. Mahdollisen virhetilanteen sattuessa tiedetään tarkalleen, mistä kohtaa ohjelmakoodia virhe on syntynyt. Kun esimerkiksi kokeillaan, ettei käyttäjä voi syöttää virheellistä tietoa järjestelmään, pystytään näkemään, että vaikka kaikki muut tarkistukset epäonnistuisivat, jokaisen osan oma virheentarkistus nappaa epäkelvot syötteet eikä anna niiden tallentua.

Yksikkötestaus PHP:ssä tämän tuotteen kohdalla tarkoittaa sitä, että jokaiselle tietoa käsittelevien luokkien funktioille tehdään testi, jossa kokeillaan syöttää oikeita arvoja, ja verrataan eli assertoidaan, että paluuarvo oli oikea. Sen lisäksi annetaan epäkelpoja syötteitä ja assertoidaan saadut virheet niiksi, jotka kyseisestä epäkelvosta syötteestä tulee tulla.

Suosituin yksikkötestauskehys PHP:lle on PHPUnit [12], ja myös sen käyttöön päädyin. PHPUnitin käyttöönotto oli suoraviivaista. Sen pystyi asentamaan suoraan PEAR-paketinhallinnalla, ja tuki Jenkinsiin löytyi suoraan. Myöskään kehittäjien työasemissa asennus sujui ongelmitta. Työasemiin asennus oli ensiarvoisen tärkeää, sillä ilman sitä TVUP:n TDD-osio ei olisi ollut mahdollinen.

5.6 Funktionaalinen testaus

Funktionaalinen testaus oli prosessia suunnitellessa suurin kysymysmerkeistä. Tiesin, että sen käyttöönotto tulisi olemaan yksi isoimmista aikasäästöistä testauksen osalta pitemmällä aikavälillä. Yhden testin kirjoitukseen menevällä työmäärällä testaa samalla yhden selaimen, tai sata käyttöjärjestelmä/selain-yhdistelmää. Myös aiemmassa mallissa oli yleisesti testattu vain ne toiminnot, joihin kulloinkin oli tehty muutoksia. Jos muutokset olisivat rikkoneet jotain toimintoja muualla järjestelmässä, ei niitä välttämättä olisi havaittu välittömästi. Koontipalvelimen jatkuvasti tekemät testit kävisivät nyt kokeilemassa kaikki sivut ja toiminnot joka kerta, kun jotain muutetaan. Funktionaalisen testauksen keinoja tutkiessani kuulin paljon hyvää Seleniumista ja sen Selenium Grid -moniajoympäristöstä. Selenium toimii käytännössä PHPUnitin laajennuksena, eli toisen testausmenetelmän hyvästä osaamisesta seuraa synergiaetuja myös toiseen.

Seleniumia käskytetään Webdriver-rajapinnan läpi. Koska kirjoitamme niin ohjelmiston kuin sen testitkin PHP:llä, meille luonnollinen valinta oli Facebookin kirjoittama PHP-Webdriver. Käytössä PHP-Webdriver on hyvin luotettavatoiminen, mutta luokka- ja metodilistausten [13] lisäksi ainoa olemassa oleva dokumentaatio oli noin yhden A4-liuskan kokoinen pikastartti. Kyseinen ohjelma on tässä työssä esitellyistä ainoa, jossa ongelmatilanteiden kanssa olin ns. omillani – apua ja esimerkkejä oli melko vaikea löytää Stackoverflow’n kaltaisilta sivustoilta tai yhtään mistään.

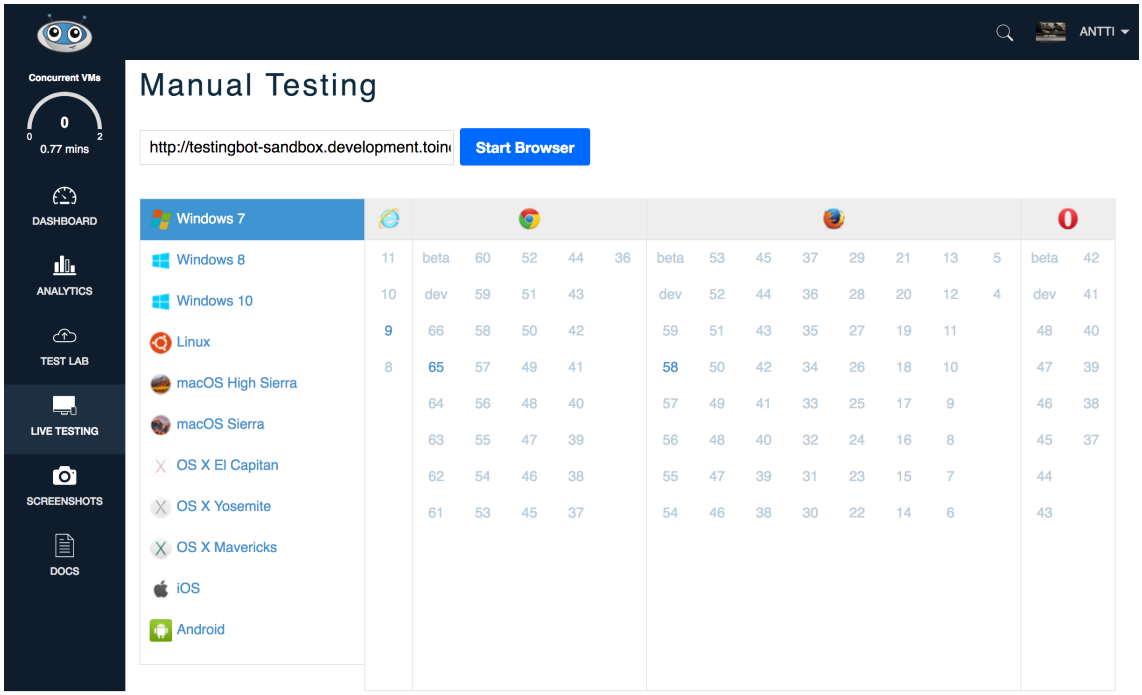
Webdriver osaa omien selainliitännäistensä kautta käskyttää selainta suoraan, kuin ihmiskäyttäjä sitä käyttäisi. Suurin osa käytetyistä funktioista liittyy siihen, mitä käyttäjäkin tekisi selaimellaan – katsotaan mitä ruudulla näkyy, etsitään elementti ja vuorovaikutetaan sen kanssa jollain tavalla. Käytännössä rakennetaan haluttu selector, joka yksilöi jonkun elementin sivulla. Tämän jälkeen voidaan saada siitä arvo, kuten teksti, tila tai väri, ja assertoidaan, että se on odotettu. Vaihtoehtoisesti yksilöityä elementtiä voidaan klikata tai kirjoittaa siihen jotain.

5.7 Selenium

Selenium on virtualisoitujen selaimien ajamiseen käytetty alusta. Se mahdollistaa Webdriverille ympäristön, jossa se käynnistää selaimiaan. Selenium-ympäristöä selvittäessäni minulle alkoi valjeta kokonaisen Selenium gridin ajamisen raskaus ja monimutkaisuus valitsemiemme alustojen kanssa, joten ennen käyttöönottoa suunnitelmia piti jo revisioida. Selenium grid olisi vaatinut yhden virtuaalikoneen jokaista alustaa varten, ja tulevan mobiilitarpeen tietäen minulla ei ollut mahdollisuuksia pystyttää esimerkiksi Apple iOS -virtuaaliympäristöä tarpeeksi luotettavasti. Myös erilaisten selainten käyttöön vaadittavat ajurit ovat levällään ympäri internetiä, ja niiden tukiaste ja kypsyys vaihtelevat merkittävästi. Onneksi en ollut ainoa joka painii kyseisen ongelman kanssa. Erilaiset yritykset tarjoavat Selenium-testausta myös palveluna. Käytännössä Selenium palveluna tarkoittaa sitä, että funktionaalisessa testitiedostossa valitaan vain halutut alustat ja selaimet, ja palvelu hoitaa koneiden ja selainten käynnistyksen, Webdriverin rajapinnan sekä mahdollisesti tallentaa kuvankaappauksia tai videota testin aikana.

Varteenotettavimmat vaihtoehdot projektin tekohetkellä olivat BrowserStack [14], Testingbot sekä Sauce Labs. Kaikki tarjoavat suunnilleen samanlaisen peruspaketin satoja käyttöjärjestelmä-selain-yhdistelmiä, mutta jokaisessa on omia erityispiirteitään, kuten esimerkiksi Sauce Labsin laaja dokumenttivaranto [15].

Näihin tutustumisen jälkeen päädyin valitsemaan lähinnä hinnoittelunsa perusteella Testingbotin [16]. Testingbotissa on ilmainen koekäyttö, jolla saa 60 ilmaista automatisoitua testausminuuttia (kuva 7). Yksi testi, vaikka kuinka yksinkertainen yhden UI-elementin klikkaus, vie noin 30 sekuntia testausaikaa. Tämä johtuu siitä, että testausaikaan lasketaan myös testin ajavan virtuaalikoneen käynnistys ja sammutus. Automatisoidun testauksen lisäksi Selenium-palveluista löytyy yleensä myös manuaalista testausta, jolla käytännössä voi käynnistellä haluamiansa virtuaalikoneita ja käyttää niitä selaimen kautta. Tällainen testaus ei kuitenkaan TVUP-prosessissamme auta mitenkään.



Kuva 7. Testingbotin käyttöjärjestelmiä ja selaimia valmiina testaukseen.

Koska Webdriverin pitää päästä tekemään testejään jossain osoitteessa, tein Jenkinsin Git-liitännäistä käyttäen koukun, joka aina uuden koontiversion syntyessä siirtää koko KiinteistöVELIn ohjelmakoodin erilliselle testipalvelimelle. Tälle testipalvelimelle Testingbotin Webdriverilla on pääsy ja omat tunnukset. Jokainen funktionaalinen testi alkaa sillä, että Webdriver kirjoittaa KiinteistöVELIn sisäänkirjautumissivulle omat tunnuksensa ja salasanansa, aivan kuten tavallinenkin käyttäjä tekisi.

6 Ensimmäinen projekti prosessia käyttäen

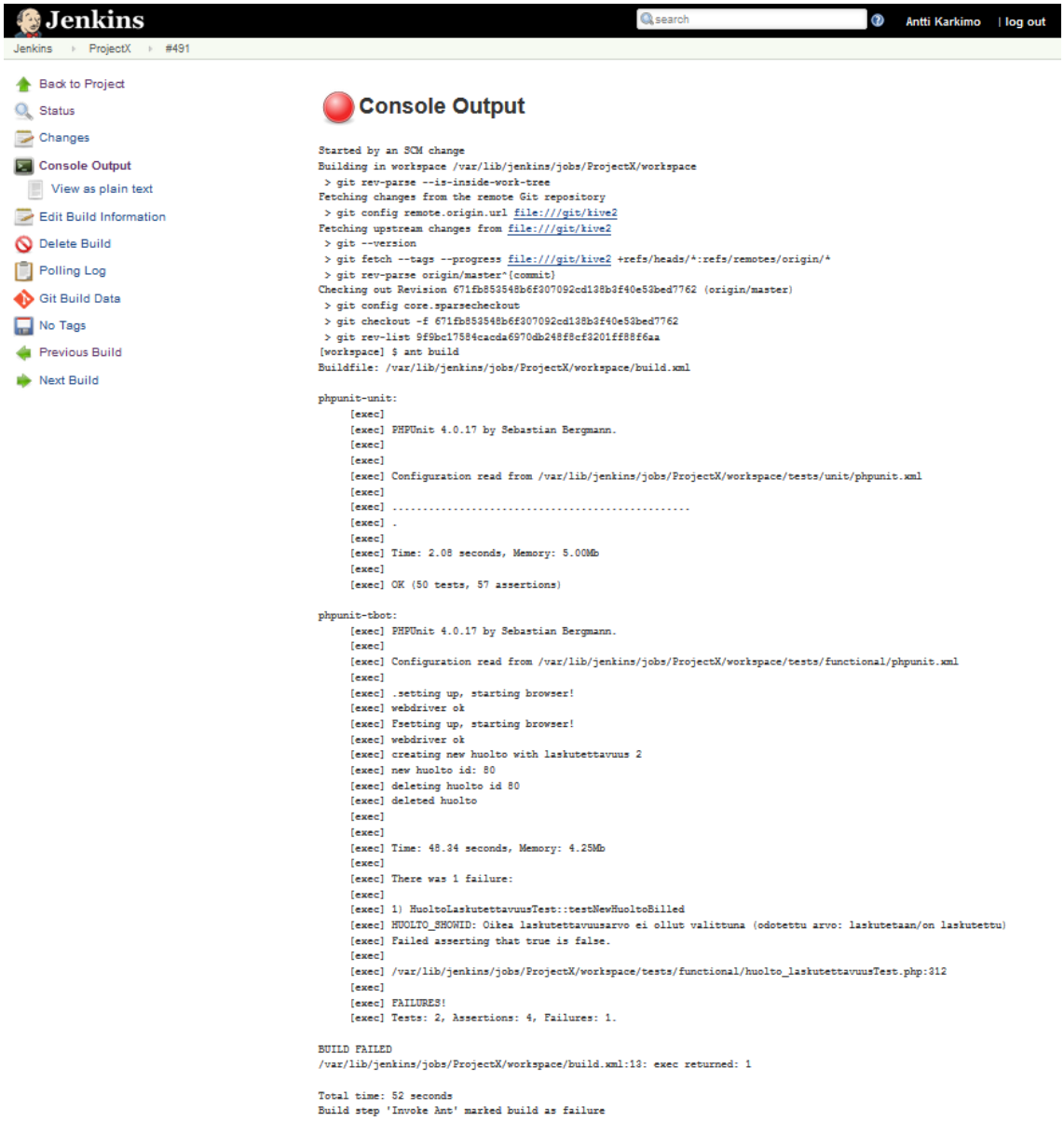
Prosessin käyttöönotto tapahtui ilman sen suurempia seremonioita. Neitsytprojektina oli KiinteistöVELIn mobiilikäyttöliittymä jQuery Mobilella. Tavoitteena oli siis kirjoittaa olemassa olevan KiinteistöVELI-sovelluksen rinnalle toinen, kevyempi käyttöliittymä, joka käyttäisi hyväkseen jQuery Mobilen tuomia erityisominaisuuksia mobiilikäyttöä varten. Tällaisia ovat esimerkiksi useamman sivun pitäminen selaimen saatavilla yhtä aikaa, jotta selaimen taaksepäin-nappia painaessa ei edellistä sivua tarvitse ladata uudelleen internetyhteyden yli. Sen lisäksi itse käyttöliittymä soveltuu hyvin erikokoisten, myös todella pienien laitteiden näyttöjen näytettäväksi - fonttien ja elementtien koot ovat tarpeeksi isoja ja palstoittuvat sekä rivittyvät hyvin kaikissa tilanteissa. Vaikka olisimme nämä asiat voineet itsekin tehdä, tahdoimme kokeilla kirjastoa, jossa ne ovat valmiina.

Työasemiin olin asentanut valmiiksi PHPstormin ja XAMPP-palvelimen. Koontipalvelimen Jenkins-asennus oli jo jonkin verran mutkikkaampi, mutta PHPunit sekä Selenium-liitos onnistui Jenkinsiin kohtalaisella vaivalla.

Huomasimme mobiilikäyttöliittymää kehittäessämme, että testien kirjoitus etukäteen vaatii sekä harjaantumista että kurinalaisuutta. Alun perin ajatuksena oli kokeilla, olisivatko funktionaaliset Webdriver-testit mahdollista tehdä myös TDD-periaatteella. Valitettavasti Testingbot-ympäristön overhead-ajat palvelinten käynnistyksen suhteen, eikä yleinen kankeus luonut tälle edellytyksiä. Lean-periaatteen mukaisesti eliminoimme wasten eli tässä tapauksessa TDD:n funktionaaliselta puolelta, sillä se sisälsi liikaa turhaa odottelua.

Niiltä määrin, mitä kerkesimme TDD:tä funktionaalisissa testeissä kokeilla, havaitsimme, että se ei välttämättä olisi kovin mielekästä ollutkaan. Testit on huomattavasti vaikeampi kirjoittaa etukäteen, sillä erilaisten HTML-elementtien tietoja sekä suhteita toisiinsa on hankala arvioida, ennen kuin elementit ovat olemassa oikeilla paikoillaan. Hyvän testisuiten kirjoittaminen voi viedä enemmän aikaa kuin varsinaisen toiminnon kirjoittaminen. Tämä oli varsinkin totta tilanteissa, joissa oli loppujen lopuksi pakko käyttää nk. Xpath-selectoria elementin sisältämän tekstin tai elementin id:n sijasta – Xpath kun rakentuu suoraan elementin DOM-puun sijainnin mukaan. Ennen kuin näkymä tai sen osa on olemassa, on mahdotonta ennustaa, minkälaisen puurakenteen sisään elementti lopulta päätyy.

Funktionaaliset testit päätettiin rakentaa jälkikäteen. Niiden olemassaolo on silti äärimmäisen hyödyllistä, sillä testi tarvitsee kirjoittaa vain kerran, ja Selenium huolehtii sen ajamisesta kaikilla selaimilla. Lisäksi testit jäävät olemaan, ja kun koontipalvelin ajaa ne joka kerta uuden koontiversion valmistuttua (kuva 8), havaitaan mahdolliset regressioongelmat välittömästi koontiversion hajotessa epäonnistuneisiin funktionaalisiin testeihin.



Kuva 8. Testingbot raportoi epäonnistuneista funktionaaisista testeistä Jenkins-koontipalvelimelle.

Varsinaisten PHPunit-yksikkötestien kirjoitus etukäteen sen sijaan toimi mitä mainioimmin. Kun ajatusmalliin pääsi sisälle, oli testien kirjoittaminen ennen varsinaista ohjelmakoodia helppoa ja järkevän tuntuista. Varsinkin yksittäisen taskin suoritetuksi merkitseminen oli kiistatonta, jos task sisälsi vain luokkatason toiminnallisuuksia. Kun testit menevät läpi eikä koodia voi refaktoroida yksinkertaisemmaksi, task on valmis.

PHPstorm kehitysympäristönä toimi mutkattomasti kaikilla tarvittavilla käyttöjärjestelmillä. Lisättynä etuna projektiin pystyi määrittelemään koodin ulkoasulle oman määrittelytiedoston, joka loi entisestään yhtenäisyyttä kehityskäytännöille. Tämä ulkoasutiedosto määrittelee esimerkiksi tabulaattorin tai välilyöntien käytön koodin sisennyksessä, nimeämiskäytäntöjä, välilyöntien määrän sekä sijainnin if-lausekkeissa ja muita vastaavia asioita, joiden olisi hyvä olla yhtenäiset.

7 Lopputulos ja päätelmät

Ensimmäinen TVUP-prosessilla suoritettu projekti sujui hyvin. Uusien työkalujen opettelussa oli hieman alkuvaikeuksia, jonka takia ensimmäisessä sprintissä ei aivan päästy tavoitteeseen. Suurin yksittäinen haaste oli ehdottomasti funktionaaliset testit. PHP-Webdriveriin ei löytynyt apua juuri mistään, ja työ oli pakko aloittaa yritys-erehdys-menetelmän kautta. Tietokantayhteyksiin käytetty SSH-tunneli koki suorituskykyongelmia, kun tietokantayhteyksiä availtiin useampia kerrallaan saman tunnelin yli. Yritin vaihtaa tunnelin salaustapaa tehokkaampaan, mutta tällä oli vain näennäinen etu, ja se oli silti liian hidas oikeaan käyttöön. Tästä syystä asensin työasemille ja palvelimelle OpenVPN-tunnelin, jonka kautta oli mahdollista yhdistää suoraan virtuaalisen sisäverkon kautta.

Työkalujen opettelun ja alkukankeuksien jälkeen prosessia voitiin toteuttaa onnistuneesti. Varsinkin atomääristen taskien luonti helpotti sprinttien toteuttamiskelpoisen laajuuden ennakointia hyvällä tavalla. Pisin varsinainen suunniteltu task oli mitaltaan kolme tuntia ja lyhin oli puoli tuntia. Ainoastaan loppuvaiheen bugienkorjaustaskit olivat hieman pitempiä. Läpimenoajassa ei ollut suuria variansseja suunniteltuun nähden, sillä lyhyisiin taskeihin ei jäänyt suuria yllätyksiä. Sen sijaan varsinaiseen sprintin suunnitteluun aikaa meni lähes tuplamäärä aiempaan nähden. Mielikuvaksi jäi kuitenkin, että tämä oli kannattava aikasijoitus läpimenoajan ennakoinnin paranemisen kannalta. Projektin ajaksi myös valittiin tuoteomistaja ja Scrum master. Tiimin pienestä koosta johtuen molemmat olivat myös kehittäjiä kehitystiimissä, mutta vastuu oli joka tapauksessa jaettu. Myös se oli tärkeää, ettei tuoteomistaja ja Scrum master ollut sama henkilö. Tuoteomistaja osasi vaatia projektin lopputuloksen kannalta oikeita asioita.

Webdriver-testit osoittautuivat todella isoksi aikasäästöksi, kun testit tarvitsi kirjoittaa vain kerran, ja valita haluttu joukko selaimia ja käyttöjärjestelmiä, jolla ne ajettiin. Aiemman mallista hyväksytystestausta suoritettiin pieni määrä, mutta tyypillinen testitapaus oli lähinnä mallia ”piirtyykö sivu oikein?”, eikä työlästä syötteiden täyttämistä kuten aiemmin. Suurin osa testaukseen käytetystä aikamäärästä oli nyt sisällytetty taskien työmäärään, ja sen viemä aika sekä väheni reaalisesti mitattuna että varsinkin tuntui lyhyemmältä, kun se oli jaettu pitkin sprinttejä. Näin saimme objektiivisesti katsottuna vähennettyä wastea, joka oli yksi prosessin tavoitteista. Tuotantoon viedystä projektista ei löytynyt käyttäjien toimesta yhtään bugia, joka on omiaan kasvattamaan asiakastyytyväisyyttä. Tämä kertoo Lean-ajattelumallin kuudennen kohdan, sisäänrakenna yhtenäisyys hyvästä sisäistämisestä prosessiin.

Prosessia on toteutettu tämän ensimmäisen projektin jälkeen jatkossakin Toinen veli Oy:ssä hyvällä menestyksellä. Parhaimmillaan se on tapauksissa, jossa tuoteomistaja on vapaa kaikista muista rooleista kyseisessä projektissa – esimerkiksi jonkin toisen projektin kehittäjä. Suoraviivaistamalla ja prosessimallin spesifikaatioksi kirjoittamalla työstä on tullut mieluisampaa, kun tarvitsee toteuttaa vain samaa iteratiivista sykliä lopputuotteen saavuttamiseksi. Eräänlainen hämmennys ja erillisten tuotantovaiheiden välillä siirtyminen, kuten lopputestauksen aloittaminen, on jäänyt paljon vähemmälle. Aikaa jää enemmän varsinaisen työn tekemiselle.

Prosessin laatiminen on iso työtaakka yritykselle, ja näen, että sitä ei olisi syytä yrittää, ennen kuin on tiettyä pohjatietoa sekä käytännön kokemusta muista prosesseista, joista hakea pohjaa työlle. Scrum oli kuitenkin ollut käytössä Toinen veli Oy:ssä jo kolme vuotta, ennen kuin parantamisen mahdollisuudet olivat selkeinä mielessä. Lisäksi työkaluketjun integroiminen osaksi prosessia vaatii sekä rautaisen otteen että yhteisen ymmärryksen yrityksen sisällä siitä, mitä tullaan ottamaan käyttöön. Pitkään työtä tehneillä kehittäjillä on usein vahva näkemys siitä, minkälaisia työkaluja he käyttävät mieluiten tai tehokkaimmin. Jos prosessia ei saada mielekkääksi kaikille kehittäjille, ei sitä voida täysin toteuttaa, jolloin ollaan taas ScrumBut-tilanteessa, mutta oman ’täydellisen’ prosessin kanssa. Siinä tapauksessa joku voisi väittää, että koko työ on ollut turhaa.

Toinen veli Oy:lle laadittu TVUP-prosessi on ollut onnistunut sekä suunnittelunsa että käyttöönottonsa kannalta, ja se täytti tarpeen, johon sitä alettiin tekemään. Prosessiin on jouduttu tekemään tämän työn työosuuden valmistumisen jälkeen parannuksia ja päivityksiä työkaluketjuun, mutta itse prosessi on edelleen käytössä.

Lähteet

1 Agile Alliance. Iterative Development – Definition. [https://www.agilealliance.org/glossary/iterative-development]. Luettu 3.2.2018.

2 Scrum Alliance. Scrum Roles. [https://www.scrumalliance.org/learn-about-scrum/scrum-elearning-series/scrum-roles]. Katsottu 10.4.2018.

3 Schwaber, Ken; Sutherland, Jeff. The Scrum Guide. [http://www.scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-US.pdf]. Luettu 16.12.2017.

4 Scrum Alliance. Learn About Scrum. [https://www.scrumalliance.org/learn-about-scrum]. Luettu 10.4.2018.

5 Beck, Kent; Beedle, Mike ym. Ketterän ohjelmistokehityksen julistus [http://agilemanifesto.org/iso/fi/manifesto.html]. Luettu 3.2.2018.

6 Scrum.org. What is ScrumBut? [https://www.scrum.org/resources/what-scrum-but]. Luettu 11.4.2018.

7 Eloranta, Veli-Pekka. Exploring ScrumBut — An empirical study of Scrum anti-patterns. Information and Software Technology volume 74, s. 194-203.

8 Pries, Kim H.; Quigley, Jon M. Scrum Project Management

9 Kuronen, Kalle. Lean software development - Kanban-prosessimalli ohjelmistokehitystyössä.

10 Poppendieck, Mary; Poppendieck, Tom. Lean Software Development: An Agile Toolkit.

11 Oracle Corporation. MySQL 8.0 Reference Manual – Identifier Case Sensitivity [https://dev.mysql.com/doc/refman/8.0/en/identifier-case-sensitivity.html]. Luettu 21.4.2018.

12 Sitepoint. Introduction to unit testing in PHP with PHPunit [https://www.sitepoint.com/tutorial-introduction-to-unit-testing-in-php-with-phpunit/]. Luettu 21.4.2018.

13 Facebook. PHP-WebDriver API. [http://facebook.github.io/php-webdriver/master/Facebook/WebDriver.html]. Luettu 24.4.2018.

14 BrowserStack. Features. [https://www.browserstack.com/features]. Luettu 21.4.2018.

15 The Sauce Labs Cookbook. [https://wiki.saucelabs.com]. Luettu 21.4.2018.

16 TestingBot. Pricing. [https://testingbot.com/pricing]. Luettu 21.4.2018.