KARELIA-AMMATTIKORKEAKOULU
Tieto- ja viestintätekniikan koulutusohjelma

Antti Kukkonen

ASIANTUNTIJUUDEN EDISTÄMINEN SOVELLUSKEHITTÄJÄNÄ

Opinnäytetyö
Joulukuu 2018

Karelia AMMATTIKORKEAKOULU	**OPINNÄYTETYÖ** **Joulukuu 2018** **Tieto- ja viestintätekniikan koulutus** Karjalankatu 3 80200 JOENSUU +358 13 260 600 (vaihde)

Tekijä
Antti Kukkonen

Nimeke
Asiantuntijuuden edistäminen sovelluskehittäjänä

Toimeksiantaja
Keypro Oy

Tiivistelmä

Opinnäytetyön tarkoituksena oli kehittää tekijän ammatillista osaamista sovelluskehittäjän työtehtävissä Keypro Oy:llä. Opinnäytetyössä tarkasteltiin tekijän ammatillisen osaamisen kehityksen tasoa ja siihen liittyviä kehittämistarpeita. Opinnäytetyö toteutettiin soveltaen Karelia-ammattikorkeakoulun päiväkirjamuotoisen opinnäytetyön alustavaa ohjetta.

Opinnäytetyön toteutusta varten laadittiin aluksi yleiskuvaus tekijän työpaikasta, nykyisistä työtehtävistä, työtehtävissä vaaditusta osaamisesta, osaamisen nykyisestä tasosta sekä kehitystarpeesta. Taitotasojen määrittelyssä sovellettiin Dreyfusin ja Dreyfusin kehittämää viisiportaista taitotason mallia. Päiväkirjamuotoiseen opinnäytetyöhön sisältyi 50 työpäivää kestänyt seurantajakso, jonka aikana raportoitiin päivittäisiä työtehtäviä ja niihin liittyviä haasteita. Lisäksi päiväkirjaraportoinnissa kerrottiin erikseen asetettujen kehittämistehtävien edistymisestä ja niihin liittyvien ongelmien ratkomisesta.

Opinnäytetyön tuloksena syntyi mittava päiväkirjaraportti tekijän suorittamista työtehtävistä. Päiväkirjamuotoisen opinnäytetyön katsottiin kehittävän ammatillista osaamista, sillä työstä syntynyt analyysi auttaa tekijää jäsentämään omaa ajatteluaan, mikä helpottaa tunnistamaan työhön tai osaamiseen liittyviä kehittämistarpeita. Opinnäytetyön teon aikana ilmenneet haasteet osoittivat myös, että päiväkirjamainen menetelmä voi vaatia vielä lisää kehittämistä, jotta työn sisällön rajaaminen helpottuisi.

Kieli suomi	Sivuja 92 Liitteet 2 Liitesivumäärä 2

Asiasanat
asiantuntijuus, ohjelmistokehitys, osaaminen, päiväkirja

THESIS
December 2018
Degree Programme in Information and Communications Technology

Karjalankatu 3
FI 80200 JOENSUU
FINLAND
Tel. +350 13 260 600 (switchboard)

Author
Antti Kukkonen

Title
Improving Expertise as a Software Developer

Commissioned by
Keypro Oy

Abstract

The purpose of this thesis was to improve the author's professional expertise as a software developer at his current workplace, Keypro Oy. The thesis focuses on investigating the current level of expertise and possible development targets. The thesis was diary-based work by structure. A preliminary guide developed by Karelia University of Applied Sciences for diary-based thesis was used as a baseline for thesis planning and implementation.

The implementation of this thesis consisted of the following stages: first a description of the author's workplace, current tasks at work, required skill level, current skill level and development needs were devised. The five-stage model of adult skill acquisition by Dreyfus and Dreyfus was applied for defining the current and desired skill levels. The implementation of diary-based thesis work contained a reporting period which lasted for 50 workdays. During this time, daily work tasks, issues and their solutions were reported. The reporting period included also reporting work for separately set development tasks, which were defined during the planning stage of the thesis and redefined during the middle of the reporting period.

A comprehensive diary from the author's work tasks was produced as a result of this thesis. The results indicate that a dairy-based thesis can be utilized as a method for improving the actor's expertise, as the analysis of the work will allow more clearly to find out the challenges in the work and find out the development needs for the future. The challenges encountered during the thesis work also showed that more work might be needed in developing diary-based guidelines. This would allow the students to define more easily how the content of the thesis can be limited.

Language
Finnish

Pages 92
Appendices 2
Pages of Appendices 2

Keywords
diary, expertise, know-how, software development

Sisältö

Ammattikäsitteet ja lyhenteet

Back-end	Ohjelmistokehityksen osa-alue, joka sisältää ohjelman käyttämän palvelimen, tietokannan ja palvelimessa toimivan ohjelmakoodin kehittämistä ja ylläpitämistä (Wales 2014).
Build	Katso koosteen määritelmä.
CI	Continuous Integration, jatkuva integrointi. Ohjelmakoodiin tehdään pieniä muutoksia, jotka sovitetaan versionhallintaan nopeasti. Metodilla vältetään konflikteja, jos useat kehittäjät ovat työskennelleet saman koodin kanssa. Samalla mahdolliset virheet ohjelmakoodin toiminnassa havaitaan nopeasti. Integraatiopalvelin (staging server) tarkkailee versionhallintaan tehtyjä muutoksia ja luo näistä uuden koosteen (build). Jos koosteen luonti epäonnistuu esimerkiksi testeissä esiintyvien virheiden vuoksi, informoidaan tuotteen kehittäjiä vian korjaamiseksi. (Haikala & Mikkonen 2011, 56 ja 175.)
CSS	Cascade Style Sheet, tyylimäärittely. Määrittää, miten HTML-kielen eri elementit muotoillaan verkkosivustoilla (W3Schools 2018).
Debuggaus	Virheenkorjausprosessi, jossa yritetään selvittää ohjelmakoodin virheellisen tai epätoivotun toiminnan syyt ja korjata ne (McConnell 2015, 535).
Django	Sovelluskehys Python-pohjaisten verkkosovellusten kirjoittamista varten (Django Software Foundation 2018a).
DMAIC	Define, Measure, Analyze, Improve, Control. Lean Six Sigman perusteellinen ongelmanratkaisumenetelmä (Quality Knowhow Karjalainen Oy 2018a).
Eclipse	Eclipse IDE on kattava avoimen lähdekoodin ohjelmointiympäristö, joka on suunnattu erityisesti Java-kehitykseen (Eclipse Foundation 2018). Saatavilla on myös toisia jake-

	luversioita ja lisäosia, jotka lisäävät tuen monille muille ohjelmointikielille, kuten Pythonille PyDevin ja LiClipsen kautta (Brainwy Software Ltda 2018).
Epic	Suomeksi eepos, laajempaa toiminnallisuutta varten luotu kokonaisuus. Helpottaa yhteen kokonaisuuteen kuuluvien tehtävien organisointia (Rehkoph 2018).
ES6	ECMAScript 6, ECMA-262 standardin kuudes versio, joka julkaistiin vuonna 2015. ECMAScript-määritelmä kuvaa yleiskäyttöisen skriptauskielen toteutuksen. JavaScript pohjautuu pääosin ECMAScriptin määritelmään 2017. (Aranda 2017.)
Ext JS	JavaScriptin sovelluskehys, joka sisältää useita valmiita komponentteja käyttöliittymän kehittämiseen verkkosovelluksissa (Sencha Inc 2018).
Front-end	Ohjelmistokehityksen osa-alue, joka sisältää ohjelmiston käyttäjälle näkyvän osan kehittämistä eli käytännössä käyttöliittymän toteuttamista (Wales 2014).
Full-stack	Ohjelmistokehitys, jossa ohjelmoija tekee sekä front-end- että back-end-kehitystä ja ymmärtää näiden nivoutumisen toisiinsa (Wales 2014).
HTML	HyperText Markup Language. Yleisesti käytetty ja standardoitu merkintäkieli, joka määrittää verkkosivuilla ja -sovelluksissa näkyvän sisällön ja sen rakenteen (Mozilla 2018a).
IDE	Integrated Development Environment, suomeksi sovelluskehitin tai (integroitu) ohjelmointiympäristö.
Integraatiopalvelin	Palvelin, joka tarkkailee versionhallintaan tuotuja komponenttien muutoksia ja muodostaa niistä uuden koosteen (Haikala & Mikkonen 2011, 56). Toimivaa koostetta pystytään usein koekäyttämään yrityksen sisäisessä verkossa.
JIRA	Esiintyy myös nimellä Jira, Atlassianin kehittämä projektinhallintaohjelmisto, jolla voidaan hallita projekteihin kuuluvia tehtäviä ja seurata niiden edistymistä (Atlassian 2018).

JSON	JavaScript Object Notation. Kevyt tiedonsiirtoformaatti, jossa tieto esitetään avain-arvopareina. Nimestään huolimatta formaatti ei ole sidottu JavaScriptiin, vaan sitä voidaan käyttää useimmilla eri ohjelmointikielillä. (Ecma International 2017.)
Katselmointi	Ohjelmistokehityksen vaihe, jossa koodin tekijä tai tekijät asettavat koodin muiden ohjelmoijien tarkasteltavaksi. Tavoitteena on löytää ja korjata mahdolliset virheet koodista tai varmistaa muutoin koodin laatua (Radigan 2018).
Kooste	Versionhallinnan komponenteista muodostettu toimiva ohjelma (Haikala & Mikkonen 2011, 56–57). Katso myös CI ja integraatiopalvelin.
Lean	Ajattelumalli, jossa pyritään kehittämään johtamista ja prosessia tarkastelemalla koko toimintaa kokonaisuutena ja eliminoimalla työhön liittyvä hukka (Quality Knowhow Karjalainen Oy 2018b).
Liquibase	Tietokantojen versionhallintaan kehitetty avoimen lähdekoodin ohjelmisto (Datical 2018).
Mercurial	Ilmainen, alustariippumaton ja hajautettu versionhallintatyökalu lähdekoodin hallintaan ja tiimityöskentelyyn (Mercurial 2018).
Mixin	Moniperintätapa ohjelmoinnissa, jolla voi laajentaa luokan toiminnallisuuksia (Esterbook 2001).
Muutoshistoria	Ohjelmistoon kuuluva toiminnallisuus, joka mahdollistaa tallennetun objektin aiempien tietojen tarkastelun ja vanhojen tietojen palauttamisen.
MVC	Model–View–Controller. Ohjelmiston sovellusaluekohtainen arkkitehtuurimalli, jossa tarkoituksena on eriyttää toisistaan ohjelman käyttöliittymä, sovelluslogiikka ja data. (Haikala & Mikkonen 2011, 188.)
Ohjelmistotuotanto	Yleisesti käytössä olevat menetelmät ja teknologiat, joita käytetään ohjelmistojen kehittämiseksi (Haikala & Mikkonen 2011, 11).

PDCA	Plan, Do, Check, Act. Ongelmanratkaisumalli, jota käytetään erityisesti Lean-ajattelussa (Roser 2016).
PyCharm	JetBrainsin kehittämä sovelluskehitin (IDE), joka on suunnattu erityisesti Pythoniin pohjautuvien ohjelmien kehittämiseen (JetBrains s.r.o. 2018).
PyDev	Python-ohjelmointiympäristö, joka pohjautuu Eclipseen (Brainwy Software Ltda 2018).
Python	Yleiskäyttöinen ohjelmointikieli. Voidaan käyttää esimerkiksi verkkosovellusten kehittämisessä, tietokanta- ja järjestelmäohjelmoinnissa, numeerisessa ja tieteellisessä laskennassa, koneoppimisessa ja yleisenä skriptauskielenä (Lutz 2013, 82–93).
Redmine	Jean-Philippe Langin kehittämä avoimen lähdekoodin projektinhallintaohjelmisto (Lang 2018).
Repositorio	Koodivarasto- tai koodisäilö, engl. code repository. Tietovarasto, jossa säilytetään versionhallinnan kautta tehdyt muutokset kehitettävän ohjelmiston komponenteille (Collins-Sussman, Fitzpatrick & Pilato 2013).
REST	Representational State Transfer, ohjelmistoarkkitehtuurimalli, jota sovelletaan web-pohjaisten sovellusten rajapintojen kehittämisessä (World Wide Web Concortium 2004).
Rollback	Tietokantaan tehtyjen muutosten peruuttaminen ja tietokannan palauttaminen tilaan ennen päivitystä. Kertoo tietokantaskripteissä, miten skriptissä suoritetut muutokset peruutetaan (Datical 2018).
RPC	Remote Procedure Call, etäproseduurikutsu. Abstrahoi käytetyn ohjelmointikielen palvelimen ja asiakkaan välisessä tiedonvälityksessä. Tavoitteena helpottaa etäpäätteellä (palvelimella) ajettavan koodin suorittamista. (Dusseau 2018, 9.)
Scrum	Alkujaan jo 80-luvun lopulla kehitetty ketterän menetelmän kehitysprosessi, jonka periaatteita kehitettiin myöhemmin ohjelmistotuotantoon sopiviksi (Haikala & Mikkonen 2011, 46–47).

Sovelluskehitin	Ohjelmisto, joka sisältää kattavan määrän ohjelmistokehityksessä usein käytettyjä toiminnallisuuksia, kuten koodieditorin, debuggerin, versionhallinnan ja koodipohjien käytön (McConnell 2015, 710–711). Kattavat sovelluskehittimet vähentävät tarvetta erillisten ohjelmien käytölle, mutta toisaalta voivat itsessään olla hitaita käyttää. Suomenkielessä käytetään myös termiä ohjelmointiympäristö.
Sovelluskehys	Kokoelma valmiita ohjelmointikielen toiminnallisuuksia, rajapintoja ja komponentteja. Sovelluskehyksiä käyttämällä ohjelmoijien ei tarvitse luoda jokaista perustoiminnallisuutta alusta asti, mikä nopeuttaa ohjelmistokehitystä. (Haikala & Mikkonen 2011, 189.)
Sprintti	Pyrähdys, erityisesti Scrum-prosessissa määritelty vaihe, jonka pituus on 30 kalenteripäivää. Scrum-johdetuissa käytännön projekteissa pituus voi olla lyhyempikin. Sprinttiin on määritelty pieniin osiin pilkotut työtehtävät, jotka pyritään tekemään kyseisellä ajanjaksolla. (Haikala & Mikkonen 2011, 47-50.)
SQL	Structured Query Language, strukturoitu kyselykieli. Käytetään useissa tietokantoihin liittyvissä operaatioissa, kuten kyselyissä, rakenteen määrittelyssä ja muuttamisessa ja tiedon päivittämisessä. (Hovi 2004, 14.)
Stack trace	Ohjelman suorituksen aikana esiintyneen virheen seurauksena näytettävä vikailmoitus, jossa näytetään esiintynyt virhe ja ohjelmassa viimeisimpänä suoritetut funktiokutsut (Sorva 2018).
Template	Sivupohja. Määrittää Djangossa, miten HTML-sivu generoidaan. Sivupohjatiedosto sisältää staattisen ja dynaamisen sisällön luotavalle HTML-sivulle. (Django Software Foundation 2018b.)
Tiketti	Projektin- tai tehtävienhallintaohjelmistossa määritetty, kuvauksellinen tehtävä tai toimeksianto, englanniksi task tai issue. Sprintin suunnittelupalavereissa määritetään, mitkä

	tiketit otetaan työn alle, tiketille tekijä ja tiketin aikatauluarvio (Haikala & Mikkonen 2011, 49–50, 163). Tehtävän tyyppi voidaan määritellä esimerkiksi bugin korjaukseksi, parannusehdotukseksi, tapaamiseksi tai eepokseksi (epic).
TNS	Transparent Network Substrate, TNS-kuvaaja. Määrittelee Oraclen tietokantayhteyksen osoitteet, protokollat ja tunnistetiedot, joiden avulla voidaan muodostaa yhteys tietokantaan. (Oracle 2018.)
Vaatimusmäärittely	Ohjelmistotuotannon kehitysvaihe. Kehityksen alkuvaiheessa laaditaan dokumentti, jossa kerrotaan ohjelman toiminnalliset, ei-toiminnalliset vaatimukset ja reunaehdot. Tällä pyritään vastaamaan, mitä ohjelmisto vaatii toimiakseen ja että ohjelma täyttää asiakkaan asettamat vaatimukset. (Haikala & Mikkonen 2011, 61–66.)
Verbose_name	Djangon tietomallin metaluokan tai attribuutin nimi niin sanotusti ihmisille luettavassa muodossa (Django Software Foundation 2018c).
Versionhallinta	Ylläpitää tietoja ohjelmiston tilasta siihen tehtyine muutoksineen. Ohjelmistosta pidetään numeerista versiopuuta, jota kasvatetaan lineaarisesti siihen kohdistettujen julkaisuversioiden myötä (esimerkiksi 1.0, 1.0.2, 1.0.4, 1.1.). Versionhallinta mahdollistaa eri kehittäjien samanaikaisen työskentelyn sivuhaarojen avulla, vaikka muutokset kohdistuisivat samaan ohjelmakoodiin. (Haikala & Mikkonen 2011, 172.)
XML	Extensible Markup Language, dokumenttien merkintäkieli (World Wide Web Concortium 2008). Liquibasen skriptit voidaan kirjoittaa XML-kielellä.
Yksikkötestaus	Ohjelmoijan kirjoittamia testejä, jotka ajetaan yleensä automaattisesti ohjelman luokkien tai moduulien testaamiseksi (Haikala & Mikkonen 2011, 207).

1 Johdanto

Työskenteleminen ammattimaisena ohjelmistokehittäjänä edellyttää kykyä ja kiinnostusta jatkuvaan osaamisen kehittämiseen. Osaaminen nykyisistä ohjelmistoteknologioista voi käydä lyhyessäkin ajassa vanhaksi, jolloin kehittäjän arvo työmarkkinoilla heikkenee. Kehittämällä osaamista pieninkin askelin on mahdollista saavuttaa vuosien kuluessa suuria tuloksia. (Hunt & Thomas, 12–13; xxi.)

Olen työskennellyt osa-aikaisesti sovelluskehittäjänä Keypro Oy:llä Joensuussa vuodesta 2017 alkaen sen jälkeen, kun suoritin yrityksessä opintoihini kuuluvan harjoittelujakson. Olen keskimäärin työskennellyt noin 7,5–18 tuntia viikossa, mikä on rajoittanut opintoihin käytettävissä olevaa aikaa. Valitsin käytännön syistä opinnäytteeni toteuttamiseksi päiväkirjamaisen menetelmän, koska en ollut saanut koulusta tai työpaikaltani töihini sopivaa opinnäytetyön aihetta. Kuullessani päiväkirjamuotoisesta toteutuksesta ajattelin, että pystyisin sen kautta sovittamaan paremmin työtehtäväni opinnäytetyön toteutukseen.

Päiväkirjamaisen opinnäytetyön avulla tekijä pystyy analysoimaan tekemiään työtehtäviä ja työssä kertynyttä kokemusta, minkä pohjalta pystytään suorittamaan tarvittava opinnäytetyön raportointi (Haaga-Helian Julkaisut 2015). Sovelsin opinnäytetyön rakenteena Karelia-ammattikorkeakoulussa laadittua päiväkirjamuotoisen opinnäytetyön alustavaa ohjetta (Roihuvuo 2017) sekä Haaga-Helian ohjeistusta (Haaga-Helian Julkaisut 2015). Opinnäytetyön toteutusjakso oli 12.6.–12.9.2018. Työpäivien tehtävistä ja huomioista kirjoitin toteutusjaksoon kuuluneen 50 työpäivän aikana muistiinpanot, jotka koostin myöhemmin raporttiin. Raportin päiväkirjaosuudessa kuvaan tekemiäni työtehtäviä, niihin liittyviä huomioita ja työtehtävissä esiintyvien haasteiden ratkomista. Mahdollisuuden mukaan päiväkirjassa on otettu osin huomioon lähdekirjallisuudesta löytyviä ohjelmistokehityksen toimintamalleja.

Opinnäytetyöhön kuuluvat ammattikäsitteet ja lyhenteet on kuvattu lyhyesti johdantoa edeltävässä luvussa. Raportin alussa kuvataan teoreettinen viitekehys,

mikä päiväkirjamaisessa opinnäytetyössä muodostuu lähtötilanteen kuvauksesta, sisältäen tekemäni työtehtävät ja arvioidun osaamistason suhteessa työpaikan vaatimuksiin. Luvussa kolme kuvaan lyhyesti opinnäytetyöhön kuuluneet erilliset kehittymistehtävät. Luvussa neljä esitän toteutusjakson aikana syntyneen päiväkirjan. Raportin lopuksi pohdin opinnäytetyön aikana kertyneen ammatillisen osaamisen kehittymistä, opinnäytetyön teossa esiintyneitä haasteita ja tavoitteita oman osaamisen kehittämiseksi jatkossa.

2 Lähtötilanteen kuvaus

2.1 Yritysesittely

Keypro Oy on ohjelmistoja ja asiantuntijapalveluja tarjoava suomalainen yritys. Yritys tarjoaa paikkatietoon pohjautuvia ratkaisuja, joita voidaan käyttää erilaisten verkostojen, kuten tietoliikenneverkkojen, sähkön jakelun, kaukolämmön, maakaasun, vesijohto- ja viemäriverkostojen dokumentoimiseen digitaalisesti. Yritys operoi Suomessa myös Kaivulupa.fi-johtoselvityspalvelua. (Keypro Oy 2018a) Yrityksen asiantuntijapalvelut kartoittavat myös asiakkaiden olemassa olevia verkkoja asiakkaiden järjestelmiin.

Yrityksen tavoitteena on kehittää verkko-omaisuuden hallintaa, mikä sisältää verkko-omaisuuden suunnittelun, rakentamisen, dokumentoinnin ja ylläpidon. Keypro Oy:llä on yli 200 asiakasta Suomessa. Keypro Oy toimittaa ammattipalveluja ja verkkotietoratkaisuja myös kansainvälisille asiakkaille partneriverkoston kautta (Keypro Oy 2018a). Valtaosa yrityksen työntekijöistä työskentelee Joensuun ja vantaan toimipisteissä. Ohjelmistokehitys on painottunut Joensuuhun.

2.2 Sovelluskehittäjän työtehtävät

Nykyinen ammattinimekkeeni Keypro Oy:llä on sovelluskehittäjä eli ohjelmistokehittäjä. Pääasiallinen työkuvani on työsopimukseni mukaan yrityksen KeyAqua-tuotteen kehittäminen ja tietojärjestelmien käyttöönottaminen. KeyAqua on paikkatietoa hyödyntävä verkkosovellus, joka on tarkoitettu vesi-, viemäri- ja hulevesiverkostojen digitaaliseen dokumentoimiseen (Keypro Oy 2018b). Ohjelmistoon on liitettävissä myös tuotteet kaukolämpö- ja kaasuverkostoille.

Ohjelmistokehitykseen kuuluvat osa-alueet ovat ohjelmakoodin kirjoittaminen, dokumentointi, koodin katselmoinnit ja lyhyet kehityspalaverit. Ohjelmakoodin kirjoittaminen käsittää uusien toiminnallisuuksien luomista, vanhojen toiminnallisuuksien jatkokehittämistä ja ohjelmistosta löytyvien bugien korjaamista. Dokumentointi sisältää suoritettujen työtehtävien raportoimista tehtävienhallintajärjestelmän kautta sekä uusien toiminnallisuuksien teknistä ja toiminnallista kuvausta. Lisäksi kaikki ohjelmistokehittäjät voivat raportoida löydetyistä bugeista ja kehittämistarpeista projektinhallintajärjestelmän avulla, jotta niihin voidaan puuttua jatkossa.

Koodin katselmointeihin osallistun arvioijana ja omien toiminnallisuuksien esittelijänä. Lyhyet kehityspalaverit ovat lähes päivittäisiä, noin 15 minuutin mittaisia tapaamisia, joissa tiimin kehittäjät kertovat nykyisistä tehtävistään ja mahdollisista ongelmakohdista. Päivittäisten palaverien lisäksi työpaikalla pidetään pidempiä kehityspalavereita, joissa voidaan tarkastella esimerkiksi pitkän aikavälin kehityssykliä (roadmap), parannuskohtia osa-alueen toiminnassa, uusien tekniikoiden ja työkalujen käyttöönottoa.

Tietojärjestelmien käyttöönottotehtävät ovat sisältäneet työnkuvassani pääasiassa olemassa olevien ympäristöjen päivittämistä ja niistä löytyvien ongelmien korjaamista. Työ sisältää esimerkiksi asiakasympäristöjen tietokantabugien korjaamista. En ole toistaiseksi osallistunut kokonaan uusien ympäristöjen käyttöönottamiseen, sillä tämä osa-alue on pääasiassa ollut toimituksen ja tuen vastuulla.

2.3 Ohjelmistokehityksen prosessi

Ohjelmistokehityksen prosessi ennen opinnäytetyön toteutusjakson alkamista on esitetty tiivistetysti liitteessä 1. Uusien toiminnallisuuksien ja jatkokehityksen suunnittelu alkaa vaatimusmäärittelystä, jonka laatii usein tuotteen omistaja. Vaatimusmäärittelyssä kerrotaan toiminnalliset, laadulliset ja resurssivaatimukset ohjelman toiminnalle sekä määritetään reunaehdot, joiden rajoissa ohjelman tulee toimia (Haikala & Mikkonen 2015, 61). Toiminnallisuutta voidaan esittää käyttötapausten avulla.

Oma osuuteni vaatimusmäärittelyssä on usein pieni, sillä sen laatiminen kuuluu pääasiassa tuotepäällikön tai tuotteen omistajan vastuulle. Vaatimusmäärittelyn pohjalta ohjelmistokehittäjän olisi kuitenkin kyettävä ohjelmoimaan toiminnallisuuteen liittyvä ohjelmakoodi, mikä edellyttää riittävän yksityiskohtaisia ohjeita. Tarvittaessa teen tarkennuspyyntöjä vaatimuksiin, mikäli niistä ei löydy riittävän tarkkaa kuvausta tehtävienhallintaohjelmistossa. Vaatimusmäärittelyssä säädetään, mihin ohjelmistoversioon muutokset halutaan kohdistaa ja missä pyrähdyksessä ne on tarkoitus saattaa valmiiksi alustavasti.

Vaatimusmäärittelyn jälkeen alkaa varsinainen toteutustyö, joka kuuluu omaan toimenkuvaani. Aluksi kehittäjä hakee tai, jos kyse on uudesta toiminnallisuudesta, kehittäjä luo tehtävienhallinnan kautta vaatimusmääritelmän pohjalta työtehtävät, jotka voivat olla bugikorjauksia, parannusideoita tai uuden toiminnon kehitykseen kuuluvia osa-alueita, esimerkiksi lomakkeen ulkoasun tai tallennusfunktion luominen. Tiketin kuvaukseen on määritelty ominaisuuden tai bugin luonne sekä tiketin tunnistenumero. Tunnistenumeroon perustuen luodaan versionhallinnan kautta omaan käyttöön uusi kehityshaara, johon ohjelmoija tekee tarvittavat muutokset ja testaa koosteen toimimista paikallisessa ohjelmointiympäristössä ja kehitystietokannalla. Tässä vaiheessa suoritetaan halutun toiminnallisuuden ohjelmointi, kehittäjätestaus, kehitysprosessin kuvaus mahdollisine tarkennuksineen ja ongelmineen tehtävienhallinnassa. Lisäksi voidaan kirjoittaa teknisiä ja toiminnallisia dokumentteja, joita käytetään apuna varsinaisen ohjekirjan luomisessa, tuotteen testauksessa ja jatkokehityksessä.

Ohjelmointityön ollessa valmis suoritetaan koodikatselmointi. Katselmoinnissa muut tiimin kehittäjät arvioivat luotua koodia ja huomauttavat mahdollisista parannuksista, puutteista tai tarkennusta vaativista asioista. Mahdolliset puutteet korjataan katselmoinnin aikana.

Kun katselmointi on hyväksytty, voidaan muutokset tuoda versionhallinnan kautta päähaaraan osaksi jatkuvaa integrointia. Integraatiopalvelin tarkkailee aika-ajoin tuotteen eri versioiden päähaaraan tehtyjä muutoksia. Mikäli muutoksia havaitaan, aloittaa palvelin uuden koosteen (build) luomisen suorittamalla joukon automatisoituja testejä sekä ajamalla tarvittavat tietokantamuutokset. Jos jokin tässä vaiheessa epäonnistuu, lähettää palvelin kehittäjille sähköpostin välityksellä viestin. Tässä vaiheessa kehittäjä tai kehittäjät ovat vastuussa koosteen korjaamisesta tekemällä tarvittavat korjaukset.

Versiomuutosten ollessa päähaarassa on kehittäjä yleensä tässä vaiheessa valmis tekemään muita kyseiseen sprinttiin määriteltyjä töitä. Sprintin lähestyessä loppumistaan alkaa versiotestaus, jossa kyseiseen versioon tehdyt muutokset tarkistetaan virheiltä testausdokumenteissa määriteltyjen ehtojen mukaisesti. Mikäli ohjelma ei toimi toivotulla tavalla, kirjoitetaan testauspöytäkirjaan virhe ja virheestä luodaan uusi tiketti tehtävienhallintaan. Testauksen päätyttyä testauksessa löytyneet bugit korjataan. Kun testauksessa löytyneet bugit on korjattu, tarkistetaan ohjelman käännökset uusien ja muuttuneiden toiminnallisuuksien osalta. Lopuksi tehdään vielä uusi testaus, jossa tarkistetaan, että aiemmin ilmenneet bugit on korjattu ja käännökset ovat kunnossa.

Testauksen päätyttyä oma osuuteni ohjelmistokehityksen prosessissa pääasiallisesti loppuu ja suoritetaan tuotteen julkistustestaus. Mikäli ohjelmiston laatu on todettu hyväksi, merkitään kyseinen ohjelmistoversio julkaisua varten hyväksytyksi. Seuraavaksi toimitus ja tuki toimittavat versiomuutokset niille asiakkaille, joille kyseiset päivitykset tuodaan seuraavaksi käyttöön.

Asiakkaat voivat ilmoittaa ohjelmistosta löytyneistä bugeista ja jatkokehitysajatuksista toimitukseen ja tukeen. Ilmoituksista luodaan uudet tiketit, mitkä johtavat jatkokehityksen vaatimusmäärittelyyn ja sen jälkeen uudelleen kehitykseen.

Bugien osalta korjaukset voidaan tehdä ilmoituksen perusteella ilman tarkempaa määrittelyä, mutta joissakin tapauksissa on syytä tehdä vaatimusmäärittely toiminnalle uudelleen.

2.4 Työtehtävissä tarvittava osaaminen

Työssäni ohjelmistosuunnittelijana vaaditaan laaja-alaista osaamista ohjelmistotuotannossa. Tehtävä on pääosin full-stack-kehitystä, eli työtehtävät sisältävät sekä front-end- että back-end-ohjelmointia. Back-end-osaaminen vaatii tuntemusta palvelimien konfiguroimisesta, tietokantojen määrittelystä ja palvelimessa toimivan Python-ohjelmakoodin kirjoittamisesta Django-ohjelmistokehityksen avulla. Front-end-kehitys puolestaan vaatii osaamista käyttöliittymäohjelmoinnista, mikä sisältää pääosin JavaScript-ohjelmointia sekä HTML-merkintäkielen ja CSS3-tyylimäärittelyn käyttämistä. Full-stack -kehitys itsessään vaatii ymmärrystä siitä, miten nämä kaksi eri kerrosta saadaan välittämään tietoa keskenään erilaisten webrajapintojen avulla ja optimoimaan kokonaisuutta niin, että viiveet toiminnassa pysyisivät pieninä.

Edellä mainittujen, pääasiassa ohjelmakoodin kirjoittamiseen tai määrittelemiseen liittyvän tietämyksen lisäksi työtehtävissä vaaditaan ymmärrystä versionhallinnasta, testausmenetelmistä, tiimityöskentelystä, vaatimusmäärittelystä, dokumentoinnista ja ketterien menetelmien vaikutuksesta ohjelmistotuotantoon. Hyvä englannin kielen osaaminen on myös välttämätöntä, koska yrityksen virallinen työkieli on englanti, eivätkä kaikki työntekijät osaa suomea.

2.5 Kertyneen osaamisen määrittely ja arviointi

Olen suorittanut tällä hetkellä valtaosan koulutusohjelmani opintosuunnitelmaan kuuluvista opintojaksoista, jotka ovat tukeneet työskentelyäni ohjelmistokehityksessä. Nykyisen työni tekemisessä ovat edesauttaneet pääasiassa ohjelmointipainotteiset opintojaksot, kuten ohjelmistojen kehitysmenetelmiin, suunnitteluun ja käytettävyyteen liittyvät kurssit. Koen saaneeni opintojen kautta varsinaisesti

kipinän niihin asioihin, joita tiedän tarvitsevani työelämässä ja joihin voin syventyä omatoimisesti myöhemmässä vaiheessa.

Ennen opinnäytetyön tekemistä suoritin opintoihini kuuluvan 30 opintopisteen harjoittelun työpaikassani, aloittaen harjoittelun tammikuussa 2017. Harjoittelun jälkeen olen ollut osa-aikaisesti töissä kesäkuusta 2017 tähän päivään asti. Työn kautta olen päässyt käytännön tasolla kehittämään lisää omaa osaamistani ja oppimaan kokonaan uusia teknologioita. Työni ansiosta olen päässyt osallistumaan kiitettävän paljon front-end- ja back-end-kehitykseen. Olin ohjelmoinut jonkin verran Pythonilla ja JavaScriptilla ennen yritykseen tuloani ja lukenut alan teoriakirjallisuutta, mutta osaamiseni oli nykytasoon verrattuna paljon matalammalla tasolla. Muiden kollegoiden tuki on ollut erityisessä asemassa oman osaamiseni parantamisessa. Työssäni olen päässyt tutustumaan tarkemmin käyttämiini ohjelmointikieliin sekä ohjelmistokehityksen prosessiin.

Koen harjoittelun ja työni kautta osaamiseni kehittyneen erityisesti full stack -kehityksen, tietokantojen suunnittelun ja versionhallinnan osalta hyvin paljon verrattuna tilanteeseen ennen harjoittelun aloittamista. Ennen kaikkea opin ymmärtämään, minkälaista toimiminen ohjelmistokehityksessä alan yrityksessä tulisi ylipäätänsä olemaan, sillä kyseessä oli ensimmäinen työpaikkani ohjelmistoyrityksessä. Aiemmin olen tehnyt vain ryhmissä projektitöitä, joissa oltiin muutamaan otteeseen tekemisessä alan yritysten kanssa.

2.6 Osaamisen taitotasot

Opinnäytetyössäni käytän Dreyfuksen veljesten kehittämää taitotason mallia. Vaiheet kuvastavat, miten aikuinen kehittää omia taitojaan ohjeistuksen ja käytännön kokemuksen kautta. Mallin viisi vaihetta ovat noviisi, aloitteleva toimija, kyvykäs toimija, taitava toimija ja asiantuntija. (Dreyfus 2004, 177.)

Ensimmäisessä vaiheessa noviisi noudattaa ohjeita täsmällisesti suorittaakseen jonkin työtehtävän. Tarvitaan kuitenkin kokonaiskuvan ymmärtäminen, jotta voidaan päästä hyvään lopputulokseen. Noviisilla edellä mainittua ymmärrystä ei

vielä ole. Aloitteleva toimija -vaiheessa alkaa kehittyä ymmärrys relevantista kontekstista, kun työntekijä saa kokemusta käytännön työstä. Hän kiinnittää huomiota uusiin merkittäviin yksityiskohtiin työtilanteissa. Suoriutuminen ei ole itsenäisestä: työntekijä seuraa esimerkkejä ja ohjeita. Kyvykäs toimija tunnistaa runsaasti työtehtäviin liittyviä elementtejä, mutta henkilölle ei ole vielä kehittynyt täyttä ymmärrystä siitä, mikä on tärkeää tietyissä tilanteissa. Henkilö voikin lannistua kaiken työtehtävään sisältyvän tiedonmäärän alla. Yksityiskohtien asettaminen tärkeysjärjestykseen voidaan oppia joko opastuksen tai kokemuksen kautta. Tällä tavoin henkilö voi selvitä tiedonmäärän taakasta ja saavuttaa kyvykkyyden taitavaan suoriutumiseen. Henkilö alkaa tehdä omia päätöksiä eri tilanteissa tietämättä kuitenkaan, onko valinta ollut oikea. (Dreyfus 2004, 177–178.)

Taitava toimija on sitoutunut ja kokenut. Positiiviset ja negatiiviset tunnekokemukset vahvistavat toimivia näkökulmia ja vähentävät toimimattomia. Henkilön taidon perusta siirtyy vähitellen säännöistä ja periaatteista tilannekeskeiseen ratkaisumalliin. Työskentelyssä henkilö havaitsee tavoitteet ja keskeiset tekijät, muttei tiedä automaattisesti, miten toimia saavuttaakseen tavoitteet. Taitavalla toimijalla ei ole yksinkertaisesti tarpeeksi kokemusta erilaisten reaktioiden lopputuloksista, jotta hän voisi hän voisi erottaa automaattisesti sopivan tavan toimia. Tämän takia hänen pitää päättää kuinka toimia, jolloin hänen pitää turvautua irrallisiin sääntöihin. (Dreyfus 2004, 179.)

Viimeisessä vaiheessa asiantuntija on syventynyt tehtävään ja siihen tarvittavaan toimintaan. Asiantuntija ei pelkästään tiedä, mitä pitää saavuttaa, vaan hän välittömästi ymmärtää, miten on toimittava. Syynä tähän on asiantuntijan laaja tilannetajuisuus. Asiantuntijan erottaa taitavasta toimijasta hänen hienovaraisemmasta tilannetajustaan. (Dreyfys 2004, 179–180.)

2.6.1 Osaamisen arviointi

Dreyfusin määrittelemistä osaamisen tasoista koin olevani opinnäytetyön suunnitteluvaiheessa noviisin ja kyvykkään toimijan välillä. Pääasiallisesti suoritin työ-

tehtäväni hyvin itsenäisesti, usein alusta loppuun asti. Olin myös katselmoinneissa huomannut toisten ohjelmistokehittäjien koodista usein parantamisen arvoisia yksityiskohtia, jotka selkeyttäisivät joko koodin luettavuutta, toimivuutta tai helpottaisivat sen muokkaamista jatkossa. Kuitenkin välillä tarvitsen tarkempaa ohjeistusta esimerkiksi bugien korjaamisen aloittamisessa: ohjelmistot ovat sen verran laajoja, että en ollut välttämättä tietoinen monista toiminnallisuuksista ennen kuin olin ryhtymässä korjaamaan havaittuja bugeja. On hankala lähteä korjaamaan bugia, jos ei ole juuri lainkaan tietoinen siitä, miten ohjelman tietyn toiminnallisuuden tulisi toimia. Tunnistin tässä erityisesti kyvykäs toimija -tasoon kuuluvia piirteitä siitä, miten suoriutumisesta tulee henkisesti kuluttavaa, kun ongelmaan vaikuttavia tekijöitä ja tarjolla olevia toimintamalleja on runsaasti (Dreyfus 2004, 178).

En myöskään kokenut, että suoriutuisin tehtävistä riittävän joustavasti. Uskoin käyttäväni joissakin työtehtävissä paljon enemmän työaikaa, kuin mitä pidempään työskennelleet kollegani. Tarvitsin lisää kokemusta, jotta pystyin hahmottamaan kokonaiskuvan paremmin ja käyttämään tehokkaammin työssä tarvittavia teknologioita.

Olin kokenut työssä usein epävarmuutta ja ahdistusta omista valinnoistani. Onnistumisen kokemukset toivat puolestaan runsaasti hyvää mieltä erityisesti, kun olin saanut palautetta kokeneemmilta työntekijöiltä. Dreyfus kirjoittaa osaamistasoissaan juurikin, että henkilö voi tuntea euforiaa tai huojennusta onnistuneesta ratkaisusta tai pelkoa ja epävarmuutta virheistä (Dreyfus 2004, 178). Bennerin (1984) mukaan jos henkilö ei sitoudu työhönsä tunteellisesti eikä ota vastuuta virheistään hän ei pysty kehittymään taitavammaksi työntekijäksi. Työntekijä kuluu loppuun yrittäessään seurata kaikkia saatavilla olevia toimintamalleja ja ohjeita, ellei hän opi tekemään omia valintoja (Benner 1984, Dreufys 2004, 178–179, mukaan). Lukiessani näitä Dreyfusin määritelmiä koin huojennusta, koska tunnistin näissä juuri piirteitä itsessäni, mikä mahdollistaisi kehittymiseni taitavammaksi osaajaksi.

2.6.2 Kehittämis- ja oppimistarpeet

Tarvetta kehittymisessäni oli oppia paremmaksi virheenkorjaamisessa. Toisinaan koin, että debuggaaminen oli vienyt aikaa tarpeettoman paljon. Helpottavaa oli tosin lukea, että alalla pitkään työskennelleet arvostetut ammattilaisetkin sanovat virheiden korjaamisen olevan yksiä haastavimmista asioista ohjelmistokehityksessä (McConnell 2004, 535). Monet korjaamistani bugeista olivatkin olleet luonteeltaan monimutkaisempia kuin olisi aluksi voinut olettaa. Mikäli opin tutustumaan paremmin järjestelmän eri osa-alueisiin, uskoin tämän ongelman helpottuvan tulevaisuudessa. Ohjelmistossa oli esimerkiksi paljon toiminnallisuuksia, joiden kanssa en ollut koskaan tekemisissä.

Lisäksi halusin kehittää ymmärrystäni käyttämistäni ohjelmointikielistä, sillä niissä oli vielä monia piirteitä, joista en ollut täysin perillä ja jouduin erikseen tarkistamaan dokumentaatiosta. Halusin erityisesti oppia paremmin kirjoittamaan ohjelmistoon liittyviä testejä, kuten yksikkötestejä. Lisäksi perusteellisempi tutustuminen ohjelmointikehyksiin oli paikallaan, sillä saatavillani oli ollut kuitenkin jo pidemmän aikaa näihin tietokirjoja, joissa ohjelmistokehitysten toimintaa selostetaan tarkemmin.

Joskus tutkin myös ongelmia ehkä liian pitkään yksikseni sen sijaan, että hakisin apua ongelmaan aikaisemmin kollegalta. Osaltaan tähän on vaikuttanut työn kiireellisyys, sillä itse kullakin oli alla myös omia työtehtäviä ja koin ainakin itse häiritseväksi sen, mikäli olisin jatkuvasti kysymässä asioita työkavereilta. Joskus oli kuitenkin käynyt niin, että ongelmiin olisi voinut löytyä ratkaisu huomattavasti nopeammin, jos olisin kysynyt asiaan tarkennusta riittävän aikaisessa vaiheessa.

2.7 Työpaikan sidosryhmät

Oma asemani on yrityksen sovelluskehityksessä osana KeyAqua-kehitystiimiä. Sovelluskehitys jakautuu alaryhmien mukaan omiin tuotteisiin, joiden yleisestä kehityssuunnasta on vastuussa tuotepäällikkö. Yhden tuotteen kehitykseen osal-

listuvat siis aina tuotepäällikkö ja ohjelmistokehittäjät, jotka vastaavat tuotepäällikön kanssa määriteltyjen toiminnallisuuksien lopullisesta kehittämisestä. Pääasiassa olen yhteydessä yrityksen sisällä muiden sovelluskehittäjien, toimituksen ja tuen ja ICT:n kanssa.

Sisäisiin sidosryhmiin kuuluvat asiantuntijapalvelut, toimitus ja tuki, ICT, myynti ja markkinointi, sovelluskehitys, taloushallinto, HR ja yrityksen johto. Asiantuntijapalvelut tekevät asiakkaille verkoston suunnittelu- ja kartoitustyötä. Toimitus ja tuki vastaa asiakkaiden järjestelmien päivittämisestä, testauksesta, toimittamisesta, päivittämisestä ja tukipyyntöjen käsittelemisestä. ICT vastaa tietoteknisten ratkaisujen suunnittelusta ja tietoturvasta.

Ulkoisiin sidosryhmiin kuuluvat yhtiön tuotteita käyttävät asiakkaat, tuotteita jälleenmyyvät kumppanit ulkomailla ja erilaiset järjestelmien, aineistojen, palveluiden ja ohjelmistojen toimittamiseen liittyvät kumppanit, jotka liittyvät yrityksen liiketoimintaan. Yhteistyökumppaneihin lukeutuvat muun muassa Econet, Exsane, Comsof, C2 Smartlight, Leica Geosystems, Maanmittauslaitos, Novatron, Oracle, PacketFront Software ja Setics (Keypro Oy 2018a). Yrityksen referensseissä mainittaviin asiakkaisiin kuuluvat muun muassa Telia, Finavia, HSY ja Joensuun kaupunki (Keypro Oy 2018c).

3 Opinnäytetyön kehittymistehtävä

Opinnäytetyön tarkoituksena on raportoida ja analysoida työssä tekemiäni työtehtäviä. Raportissa syntyneen aineiston pohjalta arvioin pystyväni kehittämään ammatillista osaamistani, sillä raportti toimii dokumenttina töissä esiintyvien ongelmien luonteesta, oman osaamisen tasosta ja kehittämistarpeista. Karelia-ammattikorkeakoulun päiväkirjamaisen opinnäytetyön ohjeessa asetetaan tavanomaisten työtehtävien lisäksi toteutettavaksi 2–4 pientä kehittämistehtävää. Kehittämistehtävien tavoitteena on suunnitelmallisesti kehittää ja raportoida henkilön työtä, työyhteisöä tai omaa osaamista (Roihuvuo 2017). Kehittäminen voi

perustua esimerkiksi PDCA- (Plan-Do-Check-Act) tai DMAIC- (Design-Measure-Analyze-Improve-Control) vaiheisiin (Roihuvuo 2017).

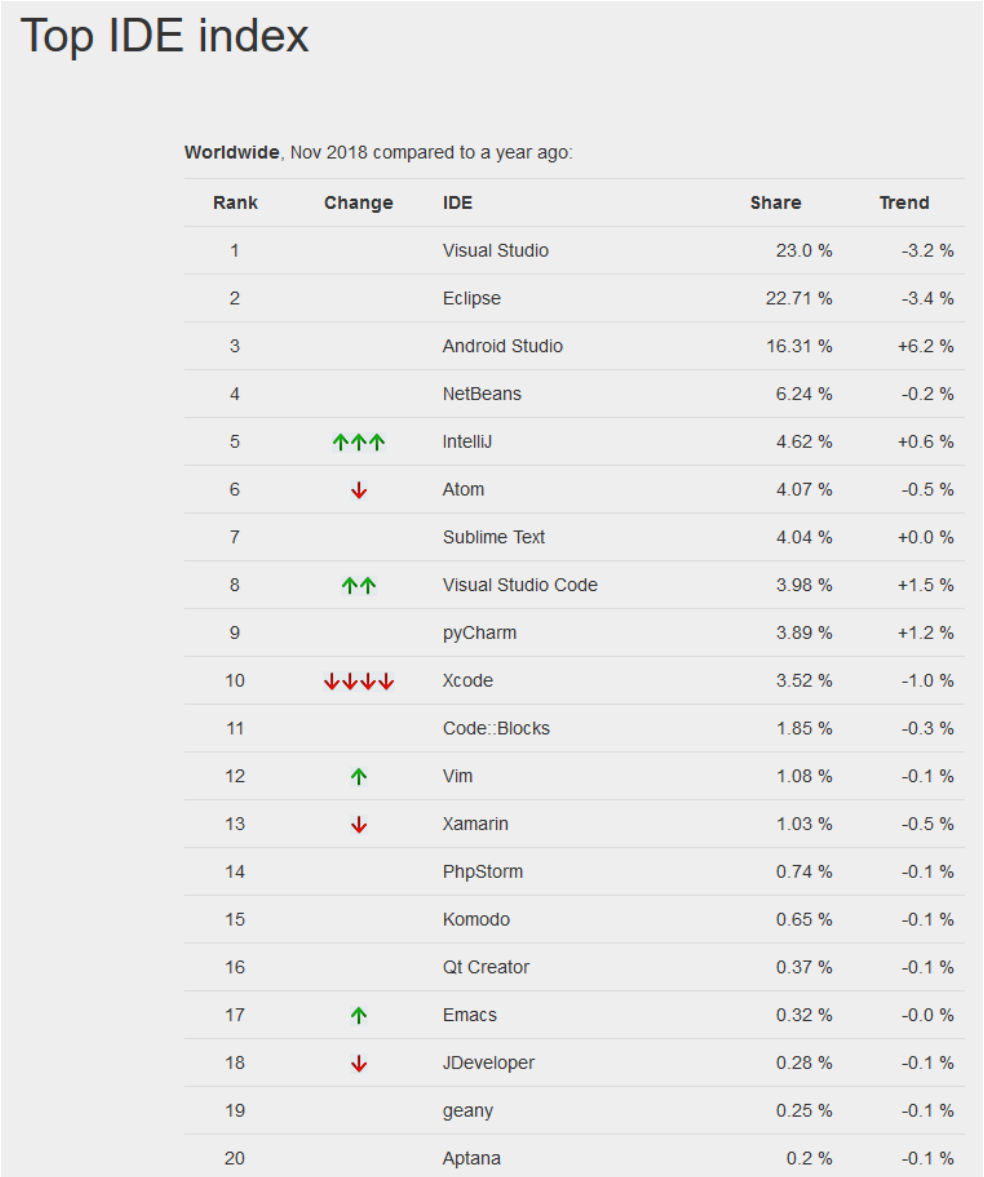
Asetin opinnäytteen ensimmäiseksi kehittymistehtäväksi ohjelmistotyön tehostamisen selvittämällä, miten käyttämäni työkalut ja työasema vaikuttavat työn edistymiseen. Toisena tavoitteena oli kaivotutkimus-toiminnallisuuden kehitys ja raportointi. Näiden kehittymistehtävien edistymisen raportointi on kuvattu muun päiväkirjaraportoinnin ohessa.

3.1 Sovelluskehityksen nopeuttaminen

Sovelluskehitystä tehdessäni olin havainnut työssäni piirteitä, jotka aiheuttivat työsuorituksessani hidasteita. Työasemani itsessään oli jo vanha, mistä aiheutui turhaa odottelua ennen kuin pääsin esimerkiksi aloittamaan työajan seurannan työpäivän alussa. En ollut saamassa hetkeen uutta työasemaa, joten jouduin etsimään muita ratkaisuja kehitystyön nopeuttamiseksi.

Olin aiemmin käyttänyt työssäni pääasiassa Eclipse-sovelluskehitintä, mutta olin kokenut jo pidemmän aikaa, että ohjelma ei enää vastannut täysin tarpeitani. Esimerkiksi Eclipsen koodin automaattinen täydennys ei toiminut erityisen hyvin JavaScript-koodia kirjoittaessa. Samoin Eclipsen debuggeri tuntui verrattavan hitaalta. En myöskään pitänyt siitä, miten käytännössä jokaista kehitysympäristöä varten ohjelmassa piti määritellä erikseen ympäristökohtaiset asetukset, joita ei kuitenkaan usein tarvinnut muuttaa lainkaan.

Moni työpaikallani oli käyttänyt tai siirtynyt käyttämään JetBrainsin PyCharm-sovelluskehitintä, joka on alusta pitäen suunniteltu Python-ohjelmistokehitykseen. Ammattilaisversio sisältää myös integroidun tuen Django-ohjelmistokehykselle. PyCharm vaikutti myös olevan varsin suosittu kehittäjien keskuudessa, kun ottaa huomioon, että kyseessä on pääasiassa Python-kielen kehittämiseen tarkoitettu sovelluskehitin (kuva 1).



Top IDE index

Worldwide, Nov 2018 compared to a year ago:

Rank	Change	IDE	Share	Trend
1		Visual Studio	23.0 %	-3.2 %
2		Eclipse	22.71 %	-3.4 %
3		Android Studio	16.31 %	+6.2 %
4		NetBeans	6.24 %	-0.2 %
5	↑↑↑	IntelliJ	4.62 %	+0.6 %
6	↓	Atom	4.07 %	-0.5 %
7		Sublime Text	4.04 %	+0.0 %
8	↑↑	Visual Studio Code	3.98 %	+1.5 %
9		pyCharm	3.89 %	+1.2 %
10	↓↓↓↓	Xcode	3.52 %	-1.0 %
11		Code::Blocks	1.85 %	-0.3 %
12	↑	Vim	1.08 %	-0.1 %
13	↓	Xamarin	1.03 %	-0.5 %
14		PhpStorm	0.74 %	-0.1 %
15		Komodo	0.65 %	-0.1 %
16		Qt Creator	0.37 %	-0.1 %
17	↑	Emacs	0.32 %	-0.0 %
18	↓	JDeveloper	0.28 %	-0.1 %
19		geany	0.25 %	-0.1 %
20		Aptana	0.2 %	-0.1 %

Kuva 1. Suosituimmat sovelluskehittimet marraskuun alussa 2018 Google Trendsin mukaan (Kuva: Carbonnelle 2018).

PyCharmin lisäksi halusin tutustua Chromen kehitystyökaluihin. Olin käyttänyt pääasiassa verkkosovelluskehityksessä Firefoxin kehitystyökaluja, mutta niiden kehitys tuntui jääneen jälkeen Chromesta. Opinnäytteen raportoinnin aikana pyrin selvittämään, tehostaisivatko uudet työkalut ohjelmistokehityksen prosessia.

Kehittymistehtävään sovelsin löyhästi PDCA-menetelmää. Menetelmään liittyvässä A3-lomakkeessa (liite 2) näkyy yhden laajan PDCA-syklin rakenne. Liitteessä 2 kuvataan ongelman alkutilanne, ongelman juurisyiden tulkinta, ratkaisun etsiminen ja seuranta jatkotoimenpiteitä varten.

3.2 Kaivon kuntotutkimus

Kaivon kuntotutkimus oli asiakkaalle toteutettava toiminnallisuus, jonka sain tehtäväkseni opinnäytteen päiväkirjaraportoinnin aikana. Valtaosa päiväkirjaraportoinnin työajasta kului lopulta tämän laajan toiminnallisuuden toteuttamiseen, minkä vuoksi valitsin sen yhdeksi opinnäytetyön kehittämistehtäväksi. Toiminnallisuuden tarkoituksena oli kehittää uusi työkalu KeyAqua-ohjelmistoon, jolla pystyttäisiin tekemään viemärikaivoihin kuntotarkistusraportteja. Kattava toiminnallisuus mahdollistaisi kaivoon ja siihen liittyviin putkien erilaisten kunto- ja vikatietojen tallentamisen. Toteutusta ei esitellä työssä yksityiskohtaisesti salassapitovelvollisuuden vuoksi. Raportoinnin tarkoituksena oli tutkia työvaiheita ja niiden onnistumista toiminnallisuuden toteuttamisessa.

Työ lähti liikkeelle tutustumalla aluksi määrittelydokumentteihin, joiden pohjalta luotiin alustava tietomalli ja lomakkeen ulkoasu. Varsinainen toteutus tehtiin pilkkomalla kukin toiminnallisuus mahdollisimman pieniin osiin eri tiketeiksi. Esimerkiksi lomakkeen haku, tallennus, korostus ja liitostoiminnot jaettiin kukin yksittäisiin tiketteihin, jotta nämä tehtävät voitaisiin suorittaa noin viikon mittaisten sprinttien sisällä. Toiminnallisuuteen kuuluvat tiketit lisättiin lisäksi omaan eepokseen (epic), jonka kautta pystyttiin organisoimaan ja tarkastelemaan toiminnallisuuden kehitystyötä.

4 Päiväkirjaraportointi

Toteutin opinnäytetyön pääsääntöisesti Karelia-ammattikorkeakoulun päiväkirjamaisen opinnäytetyön ohjeiden (Roihuvuo 2017) ja Haaga-Helian ohjeistuksen (Haaga-Helian Julkaisut 2015) mukaisesti ottamalla kuitenkin huomioon yleiset Karelia-ammattikorkeakoulun opinnäytetyön ohjeet. Päiväkirjamaisen opinnäytetyön ohjeet löytyvät Opinnäytetyön toteutus -opintojaksolta Moodlesta (Roihuvuo 2017).

Kirjoitin 50 työpäivän eli 10 täyden työviikon aikana ylös työtehtäviin liittyvät muistiinpanot, jotka koostin myöhemmin raportissa nähtyyn päiväkirjamaiseen muotoon. Raportointiin sisältyy tehtyjen työtehtävien kuvausta, ongelmien analyysi a sekä muita huomioita, jotka tulivat esiin työpaikkani ohjelmistokehityksessä, kuten muutokset kehitysprosessissa. Päiväkirjan seurantajakso alkoi 12.6.2018 ja päättyi 12.9.2018.

En sisältänyt erillisiä viikkoanalyyseja päiväkirjan raportointiin, sillä tulin kirjoittaneeksi yksittäisistä päivistä jo alusta alkaen päiväkirjaan huomattavasti enemmän sisältöä kuin mitä esimerkiksi Haaga-Helian ohjeistuksessa (2015, 161) esitetään kirjoitettavaksi tämän tyyppiseen opinnäytetyöhön. Useiden yksittäisten työpäivien kuvaus tuli käytännössä kattamaan jo sen sisällön, mitä viikkoanalyyseihin olisi pääsääntöisesti kuulunut kirjoittaa.

Työni osa-aikaisuuden vuoksi viikkoanalyysien mitoittaminen seurantajaksojen suhteen olisi ollut myös epäkäytännöllistä. Olin joinakin viikkoina vain muutamina päivinä töissä, joten yksittäisten viikkojen kohdalta ei olisi enää kertynyt kuitenkaan tarpeeksi työtehtäviä ja sisältöä erillisiä viikkoanalyysejä varten. Tästä syystä päädyin viikkoanalyysien sijaan kirjoittamaan erikseen lisähuomiot niiltä seurantajaksoilta, joilta ilmeni lisää huomautettavaa päivittäisten raporttien lisäksi. Näiden osioiden sisältö vastaa pääosin päiväkirjamaiseen menetelmään lukeutuvien viikkoanalyysien sisältöä.

4.1 Seurantajakso 1

Tiistai 12.6.2018

Töistä poissaollessani oli toimiston remontti viimein valmistunut ja pääsin aloittamaan työtehtäväni uudessa siivessä. Tavoitteenani oli tänään aluksi tarkistaa, että uudessa työpisteessäni on kaikki kunnossa. Johtojen kiinnittelyn ja uuteen sähköpöytään tutustumisen jälkeen pääsin aloittamaan päivän tehtävien parissa.

Edellisellä kerralla töissä ollessani yhtä aiemmin tekemääni tikettiä oli päivitetty. Olin korjannut toiminnallisuuden venttiileiden tilatietonäkymästä, joka näytti käyttäjälle oikeuksien puuttumisesta kertovan ilmoituksen sijaan stack tracen. Korjasin ongelman lisäämällä palvelinkoodiin Djangon ja tuotteeseen toteutetun oikeuksien tarkistuksen, jolloin ohjelma tarkistaa oikealla tavalla käyttäjän oikeudet ja näyttää käyttäjäystävällisen virheilmoituksen, mikäli käyttäjältä puuttuvat tarvittavat oikeudet. Nyt oli kuitenkin ongelmana, että niin sanotut power user -käyttäjät eivät päässeet enää näkemään tilatietoa. Ilmeisesti testiympäristöstä puuttuivat riittävät oikeudet tälle käyttäjäryhmälle kyseisiin sisältötyyppeihin (content type) liittyen, mikä aiheutti virheilmoituksen.

Tapaus opettaa, että vaikka tarkistin toiminnallisuuden tekemällä käyttäjän ilman perinteisiä oikeuksia, unohdin tarkistaa vielä toimivuuden käyttäjäryhmäkohtaisesti. Olen kuitenkin huomannut, että tähän käyttäjäryhmään liittyviä ongelmia on tullut ilmi testauksessa jo pitkään, erityisesti viimeisen 6 - 7 kuukauden aikana. Ei ole käsittääkseni yleisesti tiedossa, miten erilaisten käyttäjäryhmien pitäisi pystyä näkemään tietoja, eli dokumentointia tulisi parantaa toteutuksen osalta.

Tein tänään loppuun tiketin, jossa lisättiin niin sanotuille solmupisteille kääntämiskulman asettaminen. Olin lisännyt toiminnallisuuden jo aiemmin, mutta en ollut kerennyt vielä tarkistaa sen toimivuutta kehityspalvelimella, ainoastaan omassa kehitysympäristössäni. Tarkistin asian ja kaikki näytti toimivan toivotulla tavalla. Samalla huomasin, että kohteen muutoshistoriasta puuttui kuitenkin käännökset kulmalle ja piirtoskaalaukselle. Tämä johtuu siitä, että kohteelta puuttui Djangon model-määrittelystä verbose_name ja siihen tehtävä käännös. Tämä näytti puuttuvan useammalta muultakin kohteelta, joten tein aiheeseen liittyen tiketin toiminnon korjaamista varten.

Löysin samalla myös bugin järjestelmän muutoshistoriasta. Pistemäisten kohteiden muutoshistorian palautus ei näyttänyt toimivan lainkaan, kun käyttäjä yritti palauttaa kohteen aiemman kääntökulman. Ohjelma ilmoitti, että palautus olisi onnistunut, mutta kun käyttäjä tarkisti kulman kohteen tiedoista, oli kohteella edelleen käytössä aiempi asetus. Tein tästä aiheeseen liittyvän tiketin, jota katsoisin läpi seuraavalla kerralla.

Edellisten tehtävien lisäksi osallistuin tänään yhtiön tiedotustilaisuuteen, jossa käytiin läpi viime aikaisia asioita, kuten toimiston remontoinnin tilannetta, yhtiön arvoja ja sertifikaattien myöntämistä. Kommentoin yhdessä päivän katselmoinnissa Liquibasen kautta määriteltyjä sekvenssejä ja niistä mahdollisesti seuraavia ongelmia, mikäli käytetään automaattista rollbackia. Mikäli tietokannassa taululle on määritelty sekvenssi ja tietokanta palautetaan aikaisempaan tilaan Liquibasen kautta, on vaarana, että sekvenssiin talletettu laskurin arvo poistetaan. Näin on mahdollista, että tietokanta jää osin rikkinäiseen tilaan, mikäli itse taulua ei poistetakaan. Ongelma on mahdollista välttää luomalla sekvenssi ennen taulua tai käyttämällä Liquibasen esiehtoja (preconditions) ja tyhjää rollback-komentoa: näin sekvenssi luodaan vain, jos sitä ei ole olemassa entuudestaan ja rollbackin yhteydessä sekvenssiä ei poisteta tietokannasta.

Perjantai 15.6.2018

Sain tänään uuden maton seisomatyöskentelyä varten. Tänään ei ollut tiedossa erityistä tehtävää. Tarkistin relevantit katselmoinnit, kuten tavallista. Aloitin tänään tarkastamaan tarkemmin muutoshistoriaan liittyvää ongelmia kohteen rotaation määrittelyssä. Havaitsin osasyyksi sen, että rotaatiota ja skaalausta ei nykyisin määritellä lomakkeelle staattisesti HTML-koodissa, vaan ne luodaan dynaamisesti JavaScriptin avulla. Tästä syystä muutoshistoria ei palauttaessa pystynytkään löytämään lomakkeelle kuuluvia kenttiä, eikä palautus siksi tehnyt mitään. Jos sivupohjatiedosto (template) määritti kummatkin arvot, palautus toimi oikein, jos kentät olivat näkyvissä. En kuitenkaan usko, että näitä kenttiä haluttaisiin käyttäjän kannalta näkyviin lomakkeelle. Jos kentät puolestaan piilotettiin, palautus toimisi, mutta mikäli tietoja päivittäisiin, häviäisivät uudet tiedot kokonaan. Eli joko tähän väliin pitää tehdä jotain muuta tai sitten itse KeyCoren-komponenttia tulee muuttaa.

Ongelmana oli, että mikäli sivupohjatiedostoon määriteltiin kentät piilotettuina, ei ohjelma osannut asettaa uusia arvoja, jotka rotaatiotyökalulla saatiin. Korjasin ongelman lopulta muokkaamalla KeyCoren koodia niin, että Dojo-lomakkeen tietojen asettelussa määritetään erikseen arvo rotaatiolle. Tämä ratkaisu näyttäisi

toimivan, kun kohteen tiedot palautetaan muutoshistorian kautta ja kohde tallennetaan palautuksen jälkeen uudelleen.

Ongelmana oli edelleen kohteen skaalaus. Näytti siltä, että kohteen skaalaus asetettiin joka kerta kullekin lomakkeelle latauksen yhteydessä sen mukaan, minkä käyttäjä oli valinnut piirtomittakaava-valitsimesta. Skaalauksen palautus vaikuttaa muutoshistoriassa nykymuodossaan siis turhalta, koska sitä ei pystytä palauttamaan.

Keskiviikko 20.6.2018

Tänään tavoitteena oli aloittaa PyCharmin käyttö ja lisätä Django-modelien (malli) attribuuttien eli kenttien puuttuvat *verbose_name*-arvot ja niiden käännökset. Käännösten puuttuminen aiheuttaa sen, että mikäli ohjelmassa viitataan suoraan kyseisiin attribuutteihin, näytetään käyttäjälle pelkästään kentälle määritelty nimi. Kun attribuutille asetetaan verbose_name ja määritetään verbose_namelle käännös, pystytään kaikki verbose_name-attribuutit näyttämään käyttäjäystävällisessä muodossa niissä yhteyksissä, joissa käyttäjä pystyy kentän jollakin tapaa näkemään (esimerkiksi muutoshistoria tai Djangon admin-näkymä). Django mahdollistaa verbose_namen käyttämisen kahdella tapaa. Monille kentille se voidaan asettaa ensimmäisenä parametrina, mutta tietyille kentille, kuten vierasavaimille, se tulee määritellä ekplisiittisesti erikseen nimellä `verbose_name=”atribuutin_nimi”`.

Eräs päivänaikana tapahtunut ammattilaiseen käytökseen liittyvä asia tuli vastaan työkaverin kanssa keskustellessa. Huomasin, että keskustelumme venyi huomattavan pitkäksi eivätkä asiat liittyneet monella tapaa edes työhön. En itse aloittanut keskustelua, mutta tulin miettineeksi, että voisin sanoa jo ihan ääneen, että tässä tulisi tehdä näitä töitäkin. Parempaa ammattilaisuutta osoittaisikin ajankäytön hallinta keskustelun suhteen. On eri asia jutella kahvitauolla niitä näitä, mutta haluaisin itse käyttää työaikani varsinaisesti tekemiseen. Nyt tuntui käyvän niin, että työnteosta meni huomattava osa ajasta keskustelemiseen esimerkiksi politiikasta, mikä ei mielestäni ole kovin sovelias aihe muutoinkaan työpaikalla.

Jälkeenpäin havahduin, että minun olisi pitänyt ymmärtää ajoissa huomauttaa työkaverille, että haluaisin tehdä nyt jo töitäkin.

PyCharmin käytössä tuli tänään vastaan muutama askarruttava asia. Eclipsessä toimii oletuksena funktioiden parametrien määrittelyssä viittaus Djangon vierasavainten id:n (tietueen sisäinen, yksilöivä tunniste) käyttämällä alaviivalla olevaa ilmaisua. PyCharmissa tämän käyttäminen ei tuonut esiin lainkaan automaattista täydennystä, PyCharm ei siis näyttänyt tunnistavan, että vierasavaimella pystyisi tätä toimintoa käyttämään. En myöskään saanut toimimaan paikallisen kehityspalvelimen käynnistystä tänään vielä Djangon *manage.py*:n avulla. Pitää selvittää seuraavina työpäivinä, mistä tämä johtuu.

Maanantai 25.6.2018 ja tiistai 26.6.2018

Jatkoin näinä päivinä Django-modelien käännösten lisäämistä. Käännöksiin lisättäviä rivejä oli yllättävän paljon, joten tehtävään kului aikaa rutiininomaisista muutoksista huolimatta yllättävän paljon. Helpottaakseni käännösurakkaa otin PyCharmissa käyttöön ensimmäisen uuden itse kirjoitetun koodintäydennyksen. Tein toiminnon siten, että kun kirjoitan PyCharmiin määrittelemäni täydennysfunktion, muodostaa ohjelma automaattisesti antamani parametrin mukaan kyseiselle riville verbose_name-attribuutin. Tein kaksi eri täydennysfunktiota, koska Djangossa verbose_name voidaan määritellä kahdella eri tavalla (kts. 20.6.2018). Esimerkiksi kun kirjoitan `verb1` ja olen kopioinut työpöydälle sanan *`foo_bar`*, lisää PyCharm automaattisesti osoittimen kohtaan verbose_name-kentän käännöksellä tuettuna: `_("foo_bar"),`. Alaviiva sulkujen edessä viittaa Djangon *gettext_lazy*-funktioon, joka on tuotu kyseisen tiedoston tapauksessa alaviiva-notaatiolla. Vastaavasti kun kirjoitan `verb2`, käytetään toista, pidempää esitysmuotoa: `_(verbose_name="foo_bar"),`

Huomasin verbose_name-käännöksiä tehdessä muutamia eroavaisuuksia nimeämiskäytännöissä eri lomakkeiden välillä sekä jonkin verran vanhoja, kommentoituina olevia kenttien jäänteitä Django-modeleissa. Kyseisinä kenttiä ei näkynyt edes tietokannassakaan. Joihinkin malleihin oli jostain syystä myös

määritelty saman nimisiä attribuutteja hieman eri tavoin määritettynä, eli ne ylikirjoitettiin model-luokan latauksen aikana. Tein erikseen tiketit näiden kenttien ja joidenkin vanhojen taulujen siivoamisesta, koska tämä ei kuulunut nykyisen tiketin sisältöön.

Otin PyCharmissa käyttöön tuen Mercurial-versionhallinnalle sekä staattiselle koodintarkistukselle (Flake8, pep8). Kokeilin myös viimein ajaa PyCharmin kautta paikallista kehityspalvelinta, mikä ei aluksi toiminut. Syyksi paljastui yksinkertaisesti, että PyCharm käytti määrityksistä jostain syystä väärän kansion tulkkia. Tulkin asettaminen nykyisen projektin työkansion sisälle korjasi ongelman.

PyCharmin outoutena havaitsin sen, että ainakaan näin alkuun en saanut näkyviin mihinkään järkevään paikaan koodista löytyvien ongelmien näkymää. Eclipsessä näkymän saa pysymään aina esillä, kun se pitää PyCharmissa aukaista erikseen. Tämän lisäksi kaikki viat eivät näy listassa, ellei erikseen aja puutteiden tarkistusta.

Tiistain päätteeksi tarkistin lomakkeiden muutoshistorian toimivuuden verbose_name-lisäysten jälkeen. TNS-palvelimen toiminta aiheutti pieniä ongelmia testaamisessa, koska yhteyttä kehityksessä käytettävään tietokantaan ei pystynyt muodostamaan toimintavarmasti. Lopulta kun TNS-ongelmat menivät ohitse, huomasin, että pari muutoshistoriaa käyttävää lomaketta ei auennut oikein. Ensimmäisessä olin asettanut yhdessä verbose_name-määrittelyn viiteavainkentässä ensimmäiseksi parametriksi ja toisessa tapauksessa muuttanut erehdyksessä tietokantaan viittaavaa parametria. Korjausten jälkeen muutoshistoria näytti lopulta toimivan oikein.

4.2 Seurantajakso 2

Torstai 28.6.2018 ja perjantai 29.6.2018

Nämä päivät olivat tavallista hiljaisempia. Tarkistin kehityshaarat, jotka olisivat menossa mukaan seuraavaan versioon. Laitoin verbose_name-muutokset katselmointiin. Liitin päähaaraan aiemmin tekemäni muutoksen, jossa KeyGasin lomakkeille lisättiin sisäisen tunnisteen näyttäminen, tarkoituksena helpottaa debuggausta ja testausta.

Korjasin aiemmin muokkaamassani kohteen rotaation muutoshistoriassa olevan ongelman, kun kohdetta yritetään palauttaa. Kävi ilmi, että rotaatio ei edelleenkään tallentunut aivan oikein. Jos kääntämiseen liittyvän ikkunan avasi ja kohteen tallensi tämän jälkeen, palautui muutoshistorian kautta oikea arvo. Muutoin arvo saattoi olla pielessä. Ilmeisesti erään funktion parametrissa pitikin käyttää negatiivisia arvoja, koska kääntämiseen liittyvässä koodissa laskettiin muutos itseisarvolla. Tämän jälkeen kääntäminen näytti toimivan oikein ilman, että aukaistiin ensin siihen liittyvää lomaketta.

Korjasin bugin KeyGasin muutoshistoriassa. Eräässä koodin kohdassa oli määritelty vierasavaimen tunnisteeseen viittaus pistenotaatiolla (*objekti.id*), kun siihen olisi pitänyt viitata tässä tapauksessa Djangon alaviivanotaatiolla (*objekti_id*). Huomasin myös, että parin muun lomakkeen muutoshistoria ei toiminut, joten tein niistä erikseen korjaustiketit. Korjasin myös SSL-sertifikaattiongelman yhdestä koontiversiosta. Domain-nimet olivat vaihtuneet, minkä takia vanhat viittaukset repositorioon eivät enää toimineet. Muutos vaati lisäksi vielä integraatiopalvelimen niin sanotun työpöydän resetointia, jotta muutokset astuivat voimaan.

Huomasin, että kaasuvarusteisiin liittyvät symboliviitteet eivät näkyneet oikein uusimmassa versioidussa koonti- ja testiympäristössä. Uusimmassa ympäristössä ne kuitenkin näkyivät. Vika näytti johtuvan yksinkertaisesti puuttuvista karttatiedostoista, jotka lisättyäni viitteet näkyivät oikein.

Keskustelin karttojen päivitykseen liittyvästä ongelmasta työkaverien kanssa, sillä karttojen tilat ovat usein olleet epäselviä: välillä repositoriossa olevat kartat ovat vanhoja ja palvelimella olevat uusia. Toisinaan kumpikaan ei kuvasta nykytilaa, sillä muutokset on tehty erikseen palvelimelle jonkun toimesta. Usein kun karttoihin tehdään muutoksia, unohtuu karttojen päivittäminen testiympäristöihin.

Ratkaisuna näimme automaation ja jatkuvan integroinnin ottamisen käyttöön myös karttojen päivittämisessä. Kun kartat päivitettäisiin repositoriossa, päivitettäisiin ne automaattisesti myös koonti- ja testipalvelimilla. Päätimme, että teen aiheeseen liittyen tiketin ja teemme kesän aikana aiheeseen liittyen uuden toiminnallisuuden, mikä helpottaisi näitä karttojen päivittämiseen liittyviä epäkohtia.

Maanantai 2.7.2018

Tänään tulin töihin iltapäivästä, koska aamulla oli muita menoja. Tavoitteena oli katsoa, että aiemmin tehty toiminto KeyGasiin toimisi integraatiopalvelimella (sisäinen id lomakkeille). Lisäksi tarkastelin, mitä aiemmin luomiini symboliviitteisiin tulisi tehdä, että sen saisi nykyversiossa toimimaan.

KeyGasin sisäinen id näytti toimivan lomakkeilla oikein, joten sain tiketin hoidettua nopeasti suljettuun tilaan. Lähdin tarkastelemaan sitten symboliviitteitä. Koska oli kulunut pitkä aika, kun työstin toimintoa viimeksi, piti kehityshaaraan tuoda osaksi nykyiset muutokset päähaarasta (hg merge 3.0). Testatessani huomasin, että viitetyökalupalkkiin oli tullut lisää muita kohteita, joten palkin leveyttä tuli kasvattaa parilla pikselillä, jotta eri selaimet näyttäisivät viitetyökalut ilman vierityspalkkia.

KeyDH:n symboliviitteet eivät näkyneet aluksi kartalla, mikä johtui käyttämästäni karttapalvelimesta ja sen versiosta. Integraatiopalvelin käyttää viitteiden piirtämiseen kahta eri karttapalvelinta, mutta paikallisessa kehitystyössä pystyn käyttämään vain yhtä palvelinta. Tästä syystä kaikkia karttaan piirrettäviä kuvioita ei pystytä näyttämään välttämättä oikein. Asettaessani karttapalvelimen tukemaan symboliviitteitä, näkyivät ne jälleen oikein.

Asetin ylös tavoitteet tälle ja seuraaville päiville symboliviitteiden loppuunsaattamiseksi:

- Testaa ja muuta toiminnallisuutta, jotta se toimii niin sanottujen jaettavien tasojen kanssa.
- Tarkista testien toimivuus.
- Tarkista TODO/FIXME-tagit toiminnallisuudessa (lähinnä testeissä).
- Lisää uudet kartat integraatiopalvelimille.
- Tee tiketti tarvittavia käännöksiä varten.
- Kysy, mihin versioon toiminnallisuus liitetään, koska pääversio on jo julkaistu ja toiminto sisältää muutamia tietokantamuutoksia.

Päivän lähestyessä loppuaan kollega kysyi apua erääseen ongelmaan. Django ilmoitti paikallista kehitysympäristöä käynnistettäessä, että eräät mallien käännökset ovat jo rekisteröityjä. Ongelma ilmaantui ilmeisesti, koska Djangon modelkäännöksiin oli lisätty uusia rivejä. Rivien pois kommentoiminen näytti poistavan ongelman tilapäisesti. Emme löytäneet ongelmaan lopullista ratkaisua, mutta päätimme, että katsomme sitä yhdessä seuraavana päivänä toisten työkavereiden kanssa aamupalaverissa.

Kokeilin ajaa iltapäivästä aiempiin symboliviitteisiin liittyviä testejä. Sain huomata heti, että ne eivät menneet lävitse yhden Django-sovelluksen ilmeisesti puuttuessa kehitysympäristöstä. Etsin aluksi syytä asetuksista, mutta niiden mukaan Django-sovelluksen olisi pitänyt olla mukana. Läheisempi tarkastelu paljasti, että jostain syystä kehitysympäristön Django-sovellukset (keyaqua, keydh ja keygas) olivatkin asentuneet source-kansion alle sen sijaan, että ne olisivat olleet projektikansioissa. Koska source-kansion sisältö ei päivity silloin, kun versionhallinnasta päivitetään haaraan muutokset, ei paikallinen kehityspalvelin tietenkään pystynyt ottamaan tarvittavaa Django-sovellusta käytettäväksi. En kerennyt ratkaisemaan ongelmaa vielä tänään, vaan päätin jättää sen seuraavalle päivälle.

Tiistai 3.7.2018 ja torstai 5.7.2018

Tein DH:n symboliviitteisiin liittyviä muutoksia testeihin ja poistin jo korjattuja TODO-kommentteja. Kokeilin toimivuutta, jos ympäristössä olisivat käytössä niin

sanotut alitasot. Työkoneeni toimi erittäin hitaasti, kun karttapalvelin oli käytössä, toivottavasti saisin käyttööni pian uuden koneen työntekoa varten.

Autoin työkaveria ongelmassa, joka liittyi Ext JS -sovelluskehyksellä kehitettyjen lomakkeiden otsikossa näkyvän tiedon prosessointiin. Työkaveri oli siinä uskossa, että tämä tapahtuisi Djangon template-tiedostojen avulla. Kerroin hänelle, että tämä tehdään kuitenkin suoraan JavaScript-koodissa. Tällöin ei ole tarvetta erikseen kirjoittaa Djangon kautta toimivia otsikkoprosessoreita.

Huomasin PyCharmissa puutteen XSD-skeeman täydentämisessä, jota Liquibase-skriptien kirjoittamisessa vaaditaan. Automaattinen täydennys ei toiminut laisinkaan. En saanut korjattua tätä vielä tällä viikolla, joten se jäisi selvitettäväksi seuraavalle päivälle. Liitin verbose name-muutokset viimein päähaaraan. Aloitin tekstivakioihin liittyviä käännökset tasonvalitsimelle, jotta ne olisivat käytettävissä englanninkielisessä käyttöliittymässä.

4.3 Seurantajaksojen 1 ja 2 huomiot

Ensimmäisen tarkastelujakson aikana valtaosa ajasta meni käännettävien verbose_name-attribuuttien lisäämisessä vanhoihin Django-modeleihin. Työn määrä yllätti minut, sillä käytännössä työ vaati etenemistä rivi kerrallaan tarkistaen, oliko mallilla jo kyseistä määritystä vai ei. Django mahdollistaa siis verbose_name-määritykset joko käyttämällä niitä modelin kyseisen kentän tyyppimäärityksessä ensimmäisenä parametrina tai erikseen määreellä `verbose_name='nimi'`. Ensimmäinen tapa vähentää kirjoitettavan koodin määrää, mutta ei toimi, mikäli kenttä on tyypiltään vierasavain. Jälkeenpäin ajatellen olisikin ollut ehkä järkevämpää käyttää aina eksplisiittistä viittausta verbose_name-attribuutille (Vincent 2018). Tällöin on helpompi huomata, onko kenttää määritelty vai ei ja koodi on rakenteeltaan yhdenmukaisempaa, koska kaikissa kentissä ei joka tapauksessa voi käyttää ensimmäisen parametrin määritystä.

Analyysijakson aikana aloitin käyttämään uutta sovelluskehitintä, josta minulla ei ollut aiempaa kokemusta. Kestikin jonkin aikaa, että sovelluskehittimen käyttö alkoi tuntua tutulta. Aloin kuitenkin ymmärtämään, että olisi tärkeää sisäistää käytettyjen työkalujen toiminnot, jotta kehitystyö onnistuu tehokkaasti. Osa ohjelmoijista on esittänyt, että ohjelmistokehittäjän työajasta jopa 40 prosenttia voi liittyä jollain tapaa lähdekoodin käsittelyyn, minkä takia hyvään sovelluskehittimeen ja sen käytön opetteluun kannattaisikin panostaa (Parikh 1986, Ratliff 1987, McConnell ym. 2015, 710 mukaan).

Otin tarkastelujaksolla esille aiemmin huomatun ongelman karttoihin liittyvistä ongelmista. Tapaus toimi hyvänä esimerkkinä ongelmien havaitsemista ja korjaamisesta tiimityössä. Viime aikoina ohjelmistokehitystiimimme on parantunut puuttumalla ongelmiin, jotka ovat jo pitkään olleet esillä, mutta joita ei ole korjattu. Keskustelun jälkeen saimme määriteltyä ongelman luonteen ja korjaustarpeet, ja myöhempinä viikkoina toteutimme korjaamiseen liittyvät toimenpiteet. Jos asiasta ei olisi käyty keskustelua, todennäköisesti olisi kestänyt pidempään, ennen kuin tähän ongelmaan olisi reagoitu lainkaan. Tilanne toimi hyvänä esimerkkinä, miten tiimimme on viime aikoina edistynyt kehittämään prosessia myös yleisen kehitystyön parantamiseksi tuomalla esille asioita, joita voisi parantaa.

Djangon id-käytäntöön liittyvä tapaus tuli esiin mielenkiintoisena piirteenä. Django tekee oletuksena vierasavaimille _id-suffiksin, jolloin kohteen vierasavaimeen voidaan viitata tietämättä, millä attribuutilla se on määritetty. Näin esimerkiksi nimellä `foo_bar_id` saataisiin haettua suoraan vierasavaimen `id`, vaikka se olisi määritetty muulla nimellä. Käytännössä piilee kuitenkin sekoittumisen vaara. Jos pääavaimen attribuutti on nimetty esimerkiksi muotoon `pk`, ja kehittäjä yrittää hakea sitä viittauksella `foo_bar.id`, tämä ei tietenkään toimi. Lisäksi ongelmana voi olla ylimääräiset id-suffiksit. Jos attribuutin nimeen on määritelty `id`, tapahtuu viittaus vierasavaimen tunnisteeseen tällöin kömpelösti nimellä `foo_bar_id_id`.

4.4 Seurantajakso 3

Perjantai 6.7.2018

Aloitin tänään tekemään tietokantaan liittyviä käännöksiä niin sanottuihin tekstivakioihin. Tällä tarkoitetaan esimerkiksi lomakkeiden alasvetovalikoita. Valikkojen optioiden käännökset eivät ole osana Djangon binäärikäännöksiä, joten ne tulee määritellä tietokannassa. Käytännössä tämänlaiset käännökset on tehty kirjoittamalla skriptejä, joilla päivitetään halutut tietueet uuden nimisiksi.

Korjasin ongelman liittyen Liquibase-skeeman käyttöön PyCharmissa. Ilmeisesti syynä täydennyksen puutteelle oli se, että skeema pitää ottaa jostain syystä manuaalisesti käyttöön, vaikka se olisikin määritelty XML-tiedostossa. Ilman tätä PyCharm ei jostain syystä pystynyt täydentämään skeemaan liittyviä kenttiä. Ilman tukea täydennykselle olisi skriptien kirjoittaminen hyvin kömpelöä, koska kaikki pitäisi kirjoittaa ulkomuistista.

Pidimme tänään sprinttipalaverin, jossa puhuttiin seuraavan kahden kuukauden aikana toteutettavista asioista. Sain palaverissa kuulla tulevasta toimeksiannosta. Ensi viikolla olisi tapaaminen asiakkaalle tulevasta toiminnallisuudesta, joka mahdollistaa kuntotutkimusten tekemisen kaivoille. Olisin mukana yhtenä tämän toiminnallisuuden toteuttajista.

Maanantai 9.7.2018

Tänään pidettiin aamulla palaveri JIRAan siirtymisestä, mikä toisi muutoksia verrattuna nykyiseen kehitysprosessiin. Määrittelimme palaverissa muun muassa ”story points”-pisteet, mitkä ketterissä menetelmissä kuvastavat tehtävien vaativuustasoa ja suorittamiseen kuluvaa aikaa. Pisteet määritellään Fibonachin lukujen mukaisesti: 1, 2, 3, 5, 8, 13, 20 ja niin edelleen. Lukujen tarkoitus on antaa suuntaa antava arvio, miten paljon tehtävän tekemiseen vaaditaan panostusta. Ketteriä menetelmiä kehitettäessä on tullut esille, että perinteisesti annetut aikaarviot tehtävän suorittamiseen (esimerkiksi alle päivä, kaksi päivää, viikko, kuukausi ja niin edelleen) eivät usein toimi koviin hyvin, koska ohjelmistokehitys on

luonteeltaan hyvin monimutkainen prosessi, jossa on hankalaa arvioida tarkkaa tehtävien tekemiseen kuluvaa kestoa erityisesti, kun kyse on isommista tehtävistä. Abstraktimpien aika-arvioiden antaminen on yleensä helpompaa erityisesti, kun ratkaistavat ongelmat on pilkottu pienempiin osiin. Jatkossa tekisimme yhden viikon mittaisia sprinttejä, jossa kullakin työntekijällä on käytettävissä 20 pistettä täyttä työviikkoa kohti. Ennen tehtävien lisäämistä sprinttiin, tulisi kullekin tehtävälle määritellä sen vaativuustaso.

Toinen muutos JIRAn käytössä Redmineen verrattuna olisi niin sanottu verifiointitila. Kun tiketit on katselmoitu, tehdään niille jonkin toisen henkilön, yleensä kehittäjän tai alkuperäisen tiketin kirjoittajan puolesta lyhyt tarkistus tai testaus, että tiketissä tehdyt seikat ovat kunnossa. Tällä on tarkoitus välttää sitä, että jo tehdyt asiat eivät olekaan rikki, kun ne on lisätty esimerkiksi päähaaraan ja sitä myöten versiokohtaiselle integraatiopalvelimelle.

Muita tähän päivään kuuluvia asioitani olivat tietokantaan kuuluvien käännösten jatkaminen. Tein myös JIRAAn valmiiksi loppuihin tietokantakäännöksiin liittyvät tiketit. Olin tehnyt joihinkin tekstivakioita käyttäviin tauluihin liittyvät käännökset, jäljellä oli vielä muitakin saman tyyppisiä tauluja. Joissakin tauluissa käännettäviä termejä oli huomattava määrä enemmän kuin aiemmin kääntämissäni, joten niiden kääntäminen oli järkevämpi pilkkoa erillisiin tiketteihin suuren aikavaativuuden vuoksi. Käännösten tekemisen yhteydessä mieleeni tuli hakemistorakenteen eroavaisuudet. JIRAan siirtyminen muuttaisi kehityshaarojen ja skriptien nimeämistapaa, joten halusin ottaa puheeksi, mikä voisi olla tiimimme uusi yhteinen käytäntö. Päädyimme nimeämistapaan, jossa Redminesta tuodut tiketit nimettäisiin molemmilla tunnisteilla, kun taas JIRAssa luodut tiketit nimettäisiin ainoastaan JIRAn tunnuksilla.

Tiistai 10.7.2018

Tänään tein aluksi loppuun kesken olleen tietokantakäännös-tiketin. Perehdyin tulevaan toiminnallisuuteen eli kaivotutkimukseen liittyviin dokumentteihin, jotta olisin paremmin perillä palaverin alussa, missä uudessa toiminnallisuudessa olisi oikein kyse. Palaverissa esiteltiin kaivotutkimuksen periaatteet, vaatimukset ja

toteutuksen lähtökohdat. Ensimmäisenä toimintoon liittyväksi kysymykseksi tuli lomakkeen käyttötarkoituksen yleistäminen. Suunniteltu toteutus olisi hyvin suppeaan alueeseen liittyvä. Olisiko minkäänlaista tarvetta luoda yleiskäyttöisempää tutkimustyökalua, jonka osana kaivotutkimus olisi? Eräänä huolenaiheena olivat myös päällekkäiset työkalut. Nykyisessä KeyAqua-tuotteessa oli jo useampi työkalu, jotka osaltaan tarjosivat tutkimukseen liittyviä työkaluja. Ne eivät kuitenkaan olleet läheskään niin kattavia sisällöltään. Nykyinen yleiskäyttöinen ”tutkimus”-työkalu oli toiminnaltaan aika hidas.

Toteutukseen liittyvä ongelma oli myös erittäin pitkissä alasvetovalikkojen optioissa, jotka asettaisivat haasteita lomakkeen asettelulle. Pisimmät valikkotekstit olisivat yli 100 merkkiä pitkiä. Miten nämä siis kannattaisi esittää lomakkeella, jotta lomake olisi selkeän näköinen, mutta kuitenkin helposti käytettävissä?

Palaverin päätteeksi täsmensimme, että kehitys alkaisi vakioiden lisäämisestä kehityskantaan ja lomakkeen ulkoasun määrittämisestä. Kaivotutkimuksen valmistelun lisäksi yritin selvittää ongelmaa symboliviitteissä. Symboliviitteet eivät näkyneet joissain testiympäristöissä enää lainkaan. Sain selville sen, että kartat olivat integraatiopalvelimella kunnossa, mutta jostain syystä symboliviitteet näyttivät käyttävän eri versiota karttapalvelimesta kuin mitä on ollut tapana symboliviitteillä. Reititys näytti kuitenkin olevan kunnossa. Katsoisin tätä läpi tarkemmin seuraavina päivinä.

Keskiviikko 11.7.2018

Päätehtäväni tänään oli määrittää kaivotutkimuksen toiminnallisuuden toteuttamista. Määrittelin, miten lomake olisi käytettävissä. Tarvittaisiin ainakin kaksi eri lomaketta. Toisen kautta muokattaisiin tutkimukseen liittyviä tietoja ja toisen kautta nähtäisiin kaikki tiettyyn kaivoon liittyvät tutkimukset.

Vertasin olemassa olevia lomakkeita, mikäli ne olisivat vielä käytettävissä uutta toiminnallisuutta varten. Alustavasti mietin myös, mikäli olemassa olevan työkalun korkotietojen päivittämistä varten olisi pystynyt jotenkin liittämään kaivotutki-

mukseen. Ideana toteutuksesta tuli ennen kaikkea mieleen kaivotutkimuksen tietojen esitäyttö. Koska tutkimukseen liitetään olemassa oleva kaivo, on osa tiedoista mahdollisesti jo olemassa järjestelmässä. Näinpä tulisi olla mahdollista liittää jollakin tapaa olemassa olevat tiedot kaivoista suoraan lomakkeelle. Eräänä pienennä ongelmana tässä toteutuksessa voi olla tietojen poikkeavuus lomakkeiden välillä: osa tiedoista on kaivolomakkeella valintalomakkeilla, kun tutkimuslomakkeella ne olisivatkin vapaina tekstikenttinä. Lisäksi valintalomakkeiden käyttämät tyypit olisivat erilaisia (eri tietokantatauluissa ja eri muodossa). Tieto jouduttaisiin siis muuttamaan joissain tapauksissa kokonaan toiseen muotoon.

Autoin tänään työkaveriani tietokantaskriptien teossa. Hänellä ei ollut tällä hetkellä käytettävissä kehityskantaa, joten annoin tunnukseni omaan kantaani skriptien kirjoittamista varten. Autoin työkaveria myös SQL-skriptin kirjoittamisessa, joka tarkistaisi haluttuihin tauluihin liittyvät rajoittimet. Tätä voitaisiin käyttää osana esiehtoa ennen kuin suoritetaan tiettyjä tietokantaskriptejä.

Selvitin tänään lisää symboliviitteisiin liittyvää ongelmaa. Kokeilin päivittää välityspalvelimen asetuksia niin, että välityspalvelin ohjaisi pyynnöt oikealle karttapalvelimelle. Tästä ei kuitenkaan jostain syystä ollut lainkaan apua. Näytti yhä siltä, että pyynnöt ohjattiinkin väärälle karttapalvelimelle.

Torstai 12.7.2018

Suunnittelin tänään kaivotutkimuksen tietokantamallia määrittelydokumenteissa olevien hahmottelujen pohjalta. Aloitin kaivotutkimukseen liittyvien tikettien kirjoittamisen. Koska työtä toiminnallisuuden tehtäväksi olisi paljon, jaoin tehtävät mahdollisimman pieniin osa-alueisiin, jotta ne olisivat tehtävissä yhden tai kahden viikon mittaisissa sprinteissä. Täydensin tiketteihin liittyen lomakkeen toiminnallisuuteen liittyviä huomautuksia, joita oli tullut esiin tuotepäällikön ja tuotteen omistajan kanssa.

Selvitin työkaverin kanssa tänään tarkemmin symboliviitteisiin liittyvää ongelmaa. Kävi ilmi, että jotkin ympäristöt käyttävät useampaa välityspalvelinta. Näinpä te-

kemäni muutokset välityspalvelimelle eivät välittyneet, koska karttapalvelimen liikenne ohjattiin vielä toiselle välityspalvelimelle. Minulla ei ollut aluksi edes pääsyä tämän välityspalvelimen asetuksiin. Kun pääsin niitä tarkastelemaan, huomasin, että karttapalvelimen liikenne tosiaan ohjattiin menemään aina samalle palvelimelle, joka ei tukenut symboliviitteitä. Näin tapahtui kuitenkin vain silloin, kun käytössä oli ohjelmistoon kuuluva uusi toiminnallisuus, aluepohjainen oikeuksien hallinta. Ilmeisesti tätä ei oltu otettu huomioon, kun toiminnallisuutta oli alun perin suunniteltu. Laitoin muistiinpanot ylös nykyiseen tikettiin, jotta voisin laatia seuraavana työpäivänä tiketin huomiota varten. Jatkossa tulisi tarkastella, voidaanko oikeuksien hallintaa muuttaa siten, että se toimisi eri karttapalvelinten versioiden kanssa.

Autoin työkaveria myös tietokannan näkymään (view) liittyvässä ongelmassa. Erästä skriptiä ei jostain syystä voitu ajaa, koska sen niminen näkymä oli tietokannan järjestelmän mukaan jo olemassa. Kävi kuitenkin lopulta ilmi, että kyseinen näkymä olikin jostain syystä luotu tauluna. Kun kyseisen taulun poisti, toimivat skriptitkin oikein. Syytä taulun luomiseen ei tässä yhteydessä pystytty selvittämään.

4.5 Seurantajakso 4

Perjantai 13.7.2018

Tämä päivä oli varsin hiljainen. Jatkoin kaivotutkimuksen toteutuksen suunnittelua. Kirjoitin kaivotutkimukseen liittyvät tiketit riittävän valmiiksi, jotta ensi viikolla pystyisin aloittamaan kaivotutkimuksen varsinaisen toteutuksen. Tein tiketin liittyen Liquibasen/SQL-kehityksen parantamiseen ohjelmistokehityksen kannalta. Lisäksi tein tiketin eilen havaitusta symboliviitteisiin liittyvästä ongelmasta, kun käytössä on aluepohjainen oikeuksien hallinta, minkä käyttö ei tue toista karttapalvelinta.

Liitin tänään tekemäni tekstivakiokäännökset osaksi päähaaraa ja tarkistin, että muutokset näkyisivät integraatiopalvelimella. Selvitin, mihin tilaan ratkaisemattomat tiketit tulisi laittaa. Sain tietää, että JIRAssa tiketit voidaan asettaa lopulta ratkaistuiksi, minkä yhteydessä voidaan kertoa, jos ongelmaa ei pystytty korjaamaan jostakin syystä. Asetin alkuperäisen symboliviitetiketin ratkaistuksi tiedolla, että asiaa ei pystytty tässä yhteydessä korjaamaan, koska toiminnon muuttaminen vaatii lähempää tarkastelua ja useampien eri tiimien osallistumista kehitykseen.

Maanantai 16.7.2018

Tänään tarkastelimme ensimmäisen uuden sprintin suoriutumista sen jälkeen, kun siirryimme JIRAn käyttöön. Kokonaisuutena olimme varsin tyytyväisiä JIRAn käyttöön. Työkalu on vielä uutuudeltaan hieman sekava käyttää, mutta mahdollistaa muutoin huomattavasti monipuolisemmat työkalut tikettien luomiseen ja ylläpitämiseen. Lisäksi JIRAn Scrum ja Kanban-tauluista on helppo nähdä, mikä on nykyisen sprintin tikettien vaihe. Keskustelimme palaverissa lisäksi vielä uusien haarojen nimeämiskäytännöstä. Päädyimme ratkaisuun, jossa kaikki kirjaimet olisivat pieniä ja väliviiva korvattaisiin alaviivalla (esimerkiksi TUOTE-1 -> tuote_1_uuden_kehityshaaran_nimi).

Aloitin tänään kaivotutkimuksen kehitystyön. Aluksi ajoin kaikki uudet tekstivakiot kehityksessä käyttämääni tietokantaan ja kasvatin taulun kokoa, koska uusien tekstien pituus ylitti aiemmin sallitun pituuden. Laadin Djangon modelit tutkimuksen käyttämälle tietomallille. Tein hakemistorakenteen uudelle Django-sovellukselle, johon kaivotutkimukseen sisältyvä koodi pääasiallisesti kirjoitettaisiin. Lisäsin kuvakkeen ja tyylimäärittelyn kaivotutkimusta varten. Asetin kaivotutkimuksen avaavan painikkeen osaksi sovelluksen työkalupalkkia. Pieni erehdys tapahtui työkalupalkin valinnassa. Näytti siltä, että käytössä olisi ollut edelleen kaksi työkalupalkkia, mutta joista vain toista käytetään aktiivisesti. Asetin aluksi uuden painikkeen väärään työkalupalkkiin, josta syystä painike ei aluksi näkynyt.

Uusi, toistaiseksi tyhjä lomake ei tänään vielä jostain syystä auennut, selvittäisin asiaa tarkemmin seuraavana päivänä. Päivän aikana ehdin laatimaan lisäksi

PyCharmiin skeeman, jossa uusiin tiedostoihin lisätään automaattisesti ohjelmakoodin lisenssitiedot sisältävä koodilohko. PyCharmissa on mahdollista käyttää ohjelman sisäisiä muuttujia, joiden avulla on helppo määrittää esimerkiksi tiedoston luoja ja luontipäivämäärä automaattisesti tiedostojen alkuun sijoitettavissa koodilohkoissa.

Tiistai 17.7.2018

Selvitin tänään, miksi kaivotutkimuslomake ei auennut. Olin ilmeisesti jättänyt määrittämättä Ext JS -koodiin sisällytyksen uuden lomakkeen polulle. Koska uuden lomakkeen sisällytys (koodin importoiminen) ei ollut käytössä, ei sovellus pystynyt lataamaan lomaketta oikein napin painalluksen jälkeen. Lisättyäni sisällytyksen tyhjä lomake aukesikin oikein.

Tein lomakkeelle välilehdet, joihin sisällytetään tutkimuksen eri tyyppiset vikatiedot. Sijoitin lomakkeen kenttien koodin erilliseen tiedostoon, jotta lomakkeen ulkoasun muuttaminen olisi helpompaa. Yhdessä tiedostossa otetaan siis vain käyttöön toisessa tiedostossa määritelty lomakkeen kenttien ja muiden elementtien sijoittelu. Lisäsin kentät alkuperäisten hahmottelujen mukaan karkealla asettelulla, jota tulisi päivittää työn edetessä.

Hain tänään käyttööni IT:n puolelta toisen monitorin, koska vaikutti vahvasti siltä, että en pystynyt toimimaan kovin tehokkaasti työskennellessä yhden ruudun varassa. Käyttäisin päämonitoria koodin kirjoittamiseen ja toista monitoria sisältämään luodun näkymän, jota olisi näin helppo päivittää, kun koodiin on tehty muutoksia. Samalla välttyy tarpeelta vaihtaa jatkuvasti sen hetkistä näkymää eri ikkunoiden välillä, kuten yhdellä monitorilla usein joutuu tekemään. Haittapuolena on toki ergonomian huomioiminen: miten näytöt ja niiden esittämä sisältö tulisi asettaa, jotta työskentelyasento pysyisi luonnollisena?

Keskiviikko 18.8.2018

Tänään tein kaivolomakkeelle alasvetovalikot ja lisäsin niiden kentissä määritetyt tunnisteet Ext JS:n tietomallitiedostoihin. Valikoissa näkyvät optiot oli varsin

helppo ottaa käyttöön, koska olemme jo aiemmin luoneet toiminnallisuuden, joka lataa automaattisesti Ext JS -lomakkeissa alasvetovalikkoihin määritetyt optiot muistiin ohjelman käynnistyksen aikana. Riittää, että vakiot löytyvät tekstikannasta ja että niiden tunnus on määritelty alasvetovalikon määrityksiin. Ulkoasun laatiminen helpottuukin tästä syystä huomattavasti, koska lomakkeen toiminnallisuuteen vaikuttavaa koodia ei tarvitse tässä vaiheessa juuri kirjoittaa.

Päivän aikana pohdin muutamaa lomakkeeseen liittyvää määritysseikkaa. Lomakkeelle halutiin tuoda mukaan kaivon sijaintitiedot. Käytettävissä olisi kuitenkin myös valmis ja monipuolisempi osoitetiedot-komponentti. Kysyin tuotteen omistajalta, haluttaisiinko tässä käyttää osoitetieto-komponenttia. Ilmeisesti rajoitetumpi tekstikenttä riittäisi kuitenkin tähän tarkoitukseen. Kannen koordinaattikentät pohdittivat myös: haluttaisiinko kaikki koordinaatit esittää erikseen vai kaikki samassa kentässä? Päädyin tässä yhteen kenttään, koska se olisi helpompi tietomallin toteuttamisen kannalta. Viimeisin pohdittava asia koski lomakkeen pakollisia kenttiä, joita ei oltu esitetty määrittelydokumenteissa. Ilman pakollisia kenttiä käyttäjä pystyisi luomaan lomakkeen ilman, että lomakkeelle olisi kirjoitettu mitään. Päätin lisätä muutaman pakollisen kentän lomakkeelle ja lähetin tuotteen omistajalle vielä kysymyksen aiheeseen liittyen, ovatko kentät sopivia ja tulisiko muita kenttiä ottaa käyttöön.

Muutin PyCharmista väriteeman. Näytti siltä, että en huomannut joitakin sovelluskehittimen ilmoittamia virheitä koodista siitä syystä, että nykyinen väriteema hankaloitti niiden näyttämistä. Tein myös PyCharmissa uuden Live Template -ryhmän alasvetovalikoiden kirjoittamista varten. Live Templatet ovat joko valmiiksi määriteltyjä tai käyttäjän tekemiä pieniä pohjia, jotka mahdollistavat koodin generoimisen automaattisesti tiettyjen parametrien mukaan. Niiden käyttäminen voi helpottaa huomattavasti rutiininomaisen koodin kirjoittamista, koska kaikkea ei tarvitse enää kirjoittaa käsin alusta loppuun.

Perjantai 20.7.2018

Jatkoin jälleen kaivotutkimuslomakkeen ulkoasun työstämistä. Uskoin saavani alun perin lomakkeen ulkoasun vielä tämän päivänä aikana valmiiksi. Jatkoin rakenteellisten vikojen työstämisestä, minkä lisäksi jäljelle jäivät vielä toimintaan ja kaivoon liittyvien putkien vikatietojen lisääminen (yksi välilehti kummallekin). Löysin muutamaan tekstivakioon liittyvän outouden: osa tekstivakioista oli nimetty samaan ryhmään, vaikka niiden sisällöt olivat täysin erilaiset ja skriptin kommenttikenttä vihjasi myös samaan. Todennäköisesti skriptin tekijältä oli käynyt pieni erehdys.

Korjasin puutteen muokkaamalla näiden tekstivakioiden ryhmänimeä, joiden uskoin kuuluvan toiseen ryhmään. Lisäksi näytti siltä, että tehdasviat ja asennusviat kuuluivat samaan ryhmään, mutta tekstivakiossa tämä ei tullut esille, oliko kyseessä virhe vai kuuluuko näin ollakin. Osasta kentistä näytti puuttuvan kokonaan myös lisätiedot-määritelmä lomakkeen ulkoasun hahmotelmasta, mutta tekstivakiot oli kuitenkin määritelty skriptissä. Tarkistan asian vielä tuotteen omistajalta erikseen ensi viikolla, todennäköisesti nämä ovat jääneet vain puuttumaan hahmotelmasta.

Jälkimmäiseltä putket välilehdeltä jäi vielä puuttumaan taulukko, mikä näyttäisi kaivoon liittyvät putket ja niiden mahdolliset vikatiedot. Jouduin lisäämään manuaalisesti putkien tekstivakiot, koska näitä ei oltu vielä kirjoitettu tässä vaiheessa järjestelmään. Sain tehtyä skriptit ja näytettävät teksti- ja valikkokentät, mutta en ehtinyt tekemään vielä taulukkoa putkille.

Pidimme tänään toisen sprintin päätöspalaverin, jossa tarkastelimme tehtyjä vikoja. Tällä viikolla onnistuimme paremmin tavoitteessamme, eli sprintin käyrä muistutti rakenteeltaan enemmän lineaarista, laskusuuntaista, porrastettua viivaa. Muutama tiimin jäsen olikin onnistunut erityisen hyvin tuottamaan loppuun useamman tiketin tämän sprintin aikana. Itse olin aliarvioinut ajankäytön tarpeen kaivotutkimuslomakkeen suhteen. Lomake vei selvästi odotettua enemmän aikaa, vaikka en työskennellyt juuri muiden asioiden parissa tällä viikolla.

Yksittäisinä syinä tähän pitkään kestäneeseen toteutukseen olivat pitkään kestänyt taukoni Ext JS -lomakkeiden kehityksessä, tenttipäivän vaatima poissaolo ja yksinkertaisesti työn laajuus. Työ oli pitkälti lähes toinen toistaan vastaavan rakenteen lisäämistä ohjelmakoodissa, mutta lomakkeen laajuuden vuoksi tässä kesti yllättävän paljon aikaa. Eri kenttiä varten tuli etsiä oikeankielisiä termejä englanniksi, koska määrittelydokumentit olivat kaikki suomeksi. Lomakkeen kokoa tuli muuttaa, sillä lomakkeen valittavissa olevissa tekstikentissä ei näkynyt riittävän paljon tietoja. Samoin kyseisiä tekstivakioita tuli editoida hieman, sillä osa niistä ei asentunut tietokantaan liian pitkien ryhmänimien vuoksi.

Osan ajasta varasti myös tikettien kirjoitus ja ohjeistus. Kirjoitin viikon aikana muutaman tiketin riittävän kattavasti, mikä vie aina jonkin verran aikaa. Eräs työkaverini tekee osan tietokantaskripteistä valmiiksi puolestani, joten tein aiheeseen liittyen tiketin, jonka pohjalta hän pystyisi tekemään työtä eteenpäin.

4.6 Seurantajaksojen 3 ja 4 huomiot

Näiden seurantajaksojen aikana työpaikalla otettiin käyttöön JIRA. Redmineen verrattuna kyseessä on huomattavasti näyttävämpi ja monipuolisempi kokonaisuus, mikä ei sinällään yllätä, kun kyseessä on kaupallinen ohjelmisto. Toisaalta ohjelmiston käyttö tuntuu välillä hieman kömpelöltä. Esimerkiksi tiketti avautuu Scrumin sprinttinäkymästä vain puoliksi näkyviin. Tästä näkymästä ei voi muokata kaikkea tikettiin liittyvää tietoa, vaan siitä pitää erikseen avata tiketti vielä muokkausta varten. Muilta osin JIRA vaikuttaa mielenkiintoiselta työkalulta, joka luultavasti jatkossa tulee helpottamaan ohjelmistokehitystä työpaikallani.

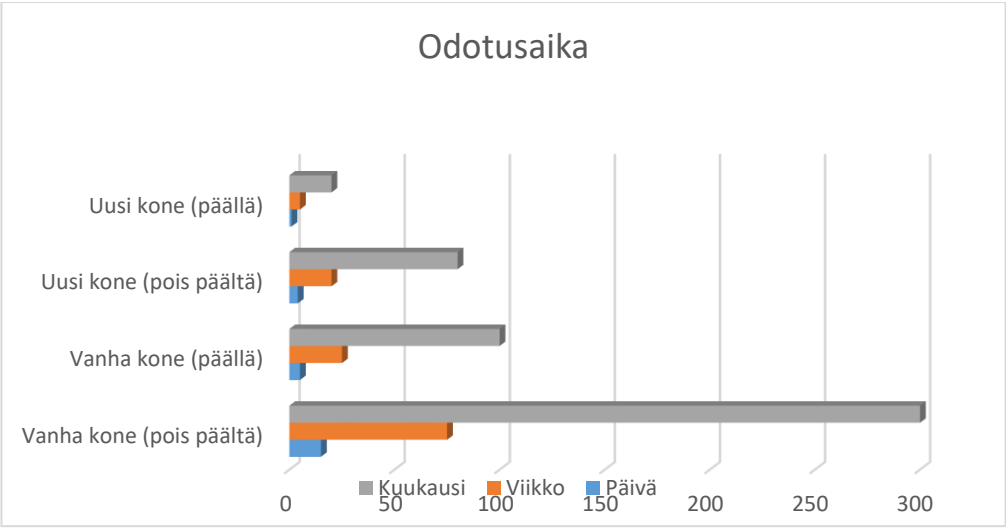
Määritimme yksikössämme JIRAssa uusien työtehtävien aika-arviot Fibonachin numeroiden perusteella, mitä myös Haikala ja Mikkonen esittävät yhdeksi arviointimenetelmäksi (2011, 163). Emme kuitenkaan ottaneet käyttöön niin sanottua suunnittelupokeria, joka etenisi seuraavasti:

- Jokainen Scrum-palaveriin osallistuja esittää omalla kortillaan työaika-arvion, joka kuvaa heidän mielestään sopivaa työmäärää tehtävälle.
- Muista arvioista selvästi poikkeavat henkilöt voivat puolustaa arviotaan.

- Prosessia toistetaan, kunnes osallistujat saavuttavat konsensuksen. (Haikala & Mikkonen 2011, 163.)

Kyseinen menetelmä ei luultavasti toimisi yksikössämme jatkuvassa käytössä pelkästään jo tikettien suuren määrän vuoksi, jos jokaisen tiimin jäsenen tulisi ehdottaa kaikkiin sprintissä tehtäviin tiketteihin työaika-arviota. Tällä hetkellä tiketin tekijät esittävät oman arvionsa työajasta, jos sitä ei ole tikettiin ennalta määritetty. Uskon kuitenkin, että suunnittelupokeria voitaisiin rajoittuneessa mittakaavassa soveltaa niihin tiketteihin, joiden uskotaan olevan kooltaan tavallista suurempia ja siten hankalammin työajan suhteen ennustettavia.

Tein osana ohjelmistokehityksen tehostamiseen liittyvää kehittymistehtävää laskelman työajasta (kuvio 1), jota menee ylimääräiseen odotteluun työaseman hitauden vuoksi. Kuviossa 1 yksi viikko sisältää viisi työpäivää ja kuukausi 20 työpäivää. Vanhan koneen käynnistyminen ja sammuminen veisi erityisen paljon työaikaa, mikäli kone sammutettaisiin ja käynnistettäisiin jokaisena työpäivänä. Laskelmaan on päivitetty myös opinnäytetyön toteutusjakson jälkeen saamani uuden työaseman odotusajat. Yllätyin suuresti, miten näennäisesti lyhyet odotusajat muuttuvat pitkällä aikavälillä hyvinkin suureksi. Olin aiemmin pitänyt työasemani päällä arkisin, mutta sammuttanut sen perjantaisin. Maanantaisin kuluikin aika odottaessa koneen avautumista. Tästä oppineena en enää sammuttanut vanhaa työasemaani turhan takia, vaikka sähkönkulutus näin hieman kasvaisikin.



Kuvio 1. Odotusaika-arvio minuuteissa.

Huntin ja Thomasin (2015, 82–83) mukaan ohjelmistokehityksessä kannattaisi perehtyä pääasiassa yhden editorin käyttöön. Etuna tässä on, että käyttäjä oppii luonnostaan käyttämään työkaluja työssään ilman, että hänen täytyy muistella niiden toimintoja. Käytettävän editorin tulisi olla laajasti konfiguroitavissa, laajennettavissa ja ohjelmoitavissa. Ohjelman tulisi löytyä kaikille käytössä oleville kehitysalustoille. Lisäksi olisi hyödyllistä, että ohjelmisto tukisi syntaksin korostamista, automaattista täydentämistä ja sisennystä, koodipohjia ja ohjelmointiympäristön kaltaisia toiminnallisuuksia (koodin kompilointi, debuggaus).

Allekirjoitan nämä huomiot jo itsekin, vaikka olen käyttänyt PyCharmia vasta vähän aikaa. Työ tuntuu jo nyt tehokkaammalta, kun olen oppinut tuntemaan sovelluskehittimen näppäinoikoteitä, käyttämään parempaa tukea koodin täydennykselle ja kehittämään kustomoituja koodipohjia. Olen käyttänyt myös muita editoreita taustalla esimerkiksi liittäessäni tekstiä tai koodia muistiinpanoja varten, mutta pääasiassa tulen käyttämään vain PyCharmia versionhallintaan asetettavan koodin kirjoittamiseen.

4.7 Seurantajakso 5

Tiistai 24.7.2018 ja keskiviikko 25.7.2018

Tein tänä aikana kaivotutkimuksen putkitiedoille ruudukon valmiiksi. Ruudukolla näytetään jatkossa ne putket, jotka ovat kytkettyinä tutkimuskohteena olevaan kaivoon. Lisäsin alasvetovalikoille määreen, joka estää niissä näkyvän tekstin muuttamisen. Tekstin muuttamiselle ei ole tarvetta, koska valikoissa käytetään ainoastaan vakioksi määriteltyjä optioita.

Pohdin myös, miten lomakkeella olevia, erittäin pitkiä tekstivakiokenttiä pystyisi muuttamaan paremmin toimiviksi. Mielessäni oli lyhenteiden käyttäminen lomakkeelle valitussa kentässä ja tekstin esittäminen koko pituudessaan työkaluvinkin näkymässä. Nykyinen ohjelmistokehys ei kuitenkaan näyttänyt taipuvan tähän

menetelmään. Toinen vaihtoehto, mitä mietin, olisi ollut erillisen ikkunan käyttämistä näiden pitkien tekstien näyttämiseen. Koin kuitenkin, että tämänkaltainen menettely olisi ollut huomattavan hidasta käyttäjän kannalta.

Päätin siis käyttää perinteistä tapaa, eli muutin lomakkeiden kenttien leveyttä hyvin suurille alasvetovalikkojen tekstikentille siten, että ne näkyisivät kokonaan. Menettely asetti kuitenkin haasteita lomakkeen selkeälle asettelulle. Käytännössä järkevintä olikin sijoittaa kentät siten, että niitä on vierekkäin kaksi, jos kenttien tekstit ovat lyhyitä ja vain yksi, mikäli siinä näkyvä teksti voi olla hyvinkin pitkä. Jouduin samalla hieman muuttamaan ikkunan kokoa, jotta kaikki tiedot mahtuisivat hyvin tämänkaltaiselle asettelulle.

Keskiviikon aikana sain luotua katselmoinnin lomakkeen ulkoasuun liittyvästä koodista. Samalla aloitin tekemään Django-modeleita, jotka määrittävät tietokantamallin toiminnallisuudelle. Koska Django-modelit sisältävät tämän toiminnallisuuden yhteydessä lukemattoman määrän eri kenttiä eli attribuutteja, pyrin sijoittamaan kentät mallissa samaan tapaan kuin ne olisivat varsinaisella lomakkeella. Tämä helpottaisi mielestäni hahmottamaan Django-modelin perusteella, mihin paikkoihin lomaketta mikäkin attribuutti sijoittuu.

Torstai 26.7.2018

Tein tänään loppuun Django-modelit, jotka määrittävät tietomallin uudelle toiminnallisuudelle. Kolme model-luokkaa määrittävät tutkimuksen tiedot, tutkimukseen liittyvät kaivot ja tutkimukseen liittyvät putket. Liitostaulut eivät sisällä viiteavaimia putkiin tai kaivoihin – tällä pyritään välttämään sidonnaisuus pelkästään tietyn tyyppisiin kaivoihin tai putkiin. On mahdollista, että tutkimusta voitaisiin jatkossa laajentaa ja lisätä siihen tuki muillekin kuin viemärikaivojen ja viemäriputkien kuntotarkistusta varten.

Sain katselmoinnissa palautetta muutamista käyttämistäni sanamuodoista tutkimuslomakkeella. Korjasin nämä puutteet, koska esitetyt ehdotukset olivat parempia. Tein tänän myös muutaman uuden Live Templaten PyCharmissa. Komennoilla *char50*, *char256* ja *txtconst* pystyin nyt tekemään rivi kerrallaan valmiiksi

Django-modeliin lisättävän kentän. Char50 ja char256 määrittävät tekstikentän pituudeksi 50 ja 256 merkkiä. Txtconst luo puolestaan uuden vierasavainkentän, jossa tietotyyppi perustuu tekstivakioon. Nämä kolme templatea toimivat siten, että kopioin halutun kentän leikepöydälle ja kirjoittan sitten halutun template-koodin lyhenteen. Sen jälkeen ohjelmalla voi luoda automaattisesti rivin, jotka sisältävät kyseisen templaten mukaisen määrittelyn leikepöydän mukaisella nimellä. Tämä nopeuttaa toistuvien rakenteiden luomista, sillä nämä kenttätyypit toistuivat hyvin usein kaivotutkimuksen Django-modeleissa.

Huomasin modeleita tehdessäni muutamia puutteita alkuperäisessä määrittelyssä. Putken sijainti kellon mukaan oli määritelty kokonaisluvuksi, mutta kaivotutkimuksen ohjepaperissa sijainti voidaan määrittää myös viittaamalla siihen kahdella eri kellonsuunnalla (esimerkiksi 03-04). Parempi olisikin määritellä kyseinen kenttä tässä tapauksessa merkkijonoksi. Saneerausmateriaalin tietotyyppinä oli jostakin syystä NUMBER, vaikka kyseessä pitäisi olla merkkijono tai tekstivakio. Lisäksi pari muuta putkiin liittyvää kenttää näyttivät puuttuvan kokonaan lomakkeen hahmotelmasta. Otan nämä esiin seuraavassa palaverissa, kun lomakkeen toiminnoista puhutaan.

Modelien ollessa valmiit kokeilin ohjelmiston käynnistämistä. Sain huomata nopeasti, että paikallinen kehityspalvelin ilmoitti virheestä liittyen vierasavainten *related_name*-kentttin: koska tätä ei määritelty erikseen, tuli joissakin tapauksissa joillekin kentille yhtenevät related_name-arvot. Korjasin määrittämällä nämä arvot tarvittavissa kohdissa manuaalisesti. Lopuksi nimesin vielä tietokannan taulut uudelleen käyttämällä niissä KA-määrettä. Tämän tein siitä syystä, että kaivotutkimuksen tauluit erottuisivat muista tuotteista: kaivotutkimus ei ole yleiskäyttöinen muiden tuotteiden kaivotyyppien, kuten kaasu- ja kaukolämpökaivojen kohdalla. Laitoin lopuksi model-koodit ja skriptit katselmointiin.

Päivän aikana autoin myös kollegaa karttatiedostojen parissa. Ilmeisesti eräissä DH:n karttatiedostoissa oli määritelty pistemäiset kohteet tasolle, joista niiden ei tiedettävästi tulisi löytyä – pistemäiset kohteet kun oli jo määritelty joillakin muilla tasoilla. Ymmärtääkseni kyse oli vanhentuneesta määrittelystä, sillä joskus aiemmin nykyiset tuotteet eivät olleet yhdessä asennettuja, joten karttatiedostotkin

erosivat enemmän toisistaan. Kerroin tämän kollegalle ja pyysin häntä lisäämään tämän tiedon tikettiin tai luomaan asiasta uuden tiketin, jotta karttatiedostojen päivitys tulisi dokumentoitua.

Perjantai 27.7.2018

Tänään loin aamulla osittain uusiksi tiketit kaivotutkimukseen liittyen. Olin aikaisemmin laatinut lähinnä isot koostetiketit back-end- ja front-end-toiminnallisuuksien työstämistä varten. Loin nyt erikseen tiketit, jotka liittyvät perustoiminnallisuuksien osa-alueiden työstämiseen: tutkimusten haun, luomisen ja tallentamisen sekä poistamistoimintojen lisäämiseen. Lisäksi tein tiketit muiden toiminnallisuuksien, kuten putkien/kaivojen kytkemiseen lomakkeelle, listanäkymän lisäämiseen, raporttien luomiseen sekä tutkimuslomakkeen aukaisemiseen kaivolomakkeen kautta (lista tutkimuksista, jotka kyseiselle kaivolle on tehty). Työn osa-alueitten jakaminen pienempiin osiin helpottaa työskentelyä sprintissä ja työn keston arviointia, sillä yksittäisen tehtävän arviointi on helpompaa kuin mitä suuren kokonaisuuden määrittely. Lisäksi se voi tehostaa työskentelyä, sillä keskittyminen kohdistuu pienempään osa-alueeseen kerrallaan suuren kokonaisuuden sijasta.

Toisena tavoitteena oli aloittaa haun tekeminen kaivotutkimuksen kohteille. Tätä varten olin syöttänyt eilen tietokantaan testidataa, jonka noutamisen tulisi onnistua lomaketta käsiteltäessä. Mietin myös, miten tutkimuslomakkeen saisi toimimaan paremmin MVC-mallin mukaisesti. Nyt näkymäkoodiin oli joutunut kirjoittamaan huomattavan paljon niin sanottua businesslogiikkaa, koska MVC-mallin mukaista ohjainta ei pystynyt ottamaan käyttöön nykytoteutuksessa.

Ennen haun suorittamista yritin kuitenkin vielä liittää Ext JS -lomakkeen käsittelijään (controller), siinä kuitenkaan onnistumatta. Sain käsittelijän toimimaan kunnolla lomakkeen luomisessa ja sulkemisessa, mutta lomakkeen sisällä olevien toimintojen liittäminen käsittelijään ei toiminut. Liittämisen jälkeen käsittelijä ei tunnistanut näkymässä suoritettuja komentoja (esimerkiksi tallennus tai haku).

Kokeilin suorittaa käsittelijän liittämistä Ext JS API-dokumentaation ohjeiden mukaisesti (Sencha 2016) sekä ottamalla mallia muualta KeyComin reitinseurannasta. Lomakkeelle periytyvät painikkeet olivat vain yhteydessä näkymän toimintoihin ja kun käytin omia painikkeita, ei niihinkään liitos tuntunut toimivan. Debuggauksessa ilmeni, että koodi ei koskaan edes yrittänyt suorittaa painikkeelle määriteltyä funktiota. Kyseessä saattaa olla ohjelman muun arkkitehtuurin rajoitus, sillä käsittelijää on ohjelmakoodin haun perusteella harvoin käytetty laajojen Ext JS -lomakkeiden yhteydessä – yleensä niillä on määritelty vain lomakkeen aukaisu tai sulkeminen. Lomakkeiden niin sanottu. businesskoodi on sisällytetty usein suoraan näkymiin, mutta näkymän varsinaiset sarakkeet on kuitenkin sisällytetty erillisiin näkymätiedostoihin.

Käytin käsittelijän toimintaan saattamiseen paljon aikaa, mutta se ei kuitenkaan tuottanut tulosta. Seuraavana työpäivänä toteutin haun ilman MVC-mallin noudattamista, sillä aikaa oli mennyt hukkaan yrittäessäni noudattaa parempia arkkitehtuurisia ratkaisuja. Seuraavan aamun palaverissa kävikin ilmi, että ongelma johtui puuttuvasta ohjainkoodin latauskoodista. Ext JS -sovelluskehyksessä ohjainlogiikan koodit tulisi sijoittaa ladattaviksi sovelluksen alussa app.js-tiedostoon, mitä ei nykyisin voida tehdä yksittäisissä tuotteissa. Ongelman korjaaminen vaatisi jonkin verran muutoksia, mikä ei ole toteutettavissa pelkästään tämän toiminnallisuuden nimissä. Tästä syystä päätin käyttää perinteistä tapaa kehityksessä. Sijoitin kuitenkin valtaosan lomakkeen ulkoasun määrittelevästä JavaScript-koodista erilliseen näkymätiedostoon, jotta ainakin jollain tapaa pystyttäisiin eriyttämään ulkoasuun liittyvä koodi sovelluksen businesskoodista.

Tiistai 31.7.2018

Aloitin tänään toteuttamaan hakua tutkimuslomakkeelle. Haun toteuttamista varten olin lisännyt tietokantaan jo aiemmin testidataa, mikä päivittyisi näkyväksi lomakkeelle haun ollessa valmis. Haun toteuttamisen yhteydessä lisäisin useat Djangon käyttämät komponentit, kuten resurssit (resources.py), näkymät (views.py) ja osoitteet (urls.py) käytettäväksi.

Käytin haun pohjana aikaisemmin toteuttamassani liittymälomakkeessa olevaa hakukoodia, koska toiminnallisuus on periaatteessa vastaavanlainen. Toteutukseen ei tarvittu kovinkaan paljoa erikseen kirjoitettu koodia, vaan varsin suuri osa toiminnallisuudesta onnistuu muutamilla koodiriveillä, koska toteutuksen runkona käytetään valmista REST-arkkitehtuuria ja sitä varten luotuja Mixin-näkymiä, jotka hoitavat palvelimen puolella kutsun käsittelyn. Haun yhteydessä lähinnä vain määritellään JavaScript-koodissa, mitä tietuetta (record) haku koskee ja mitä urlia haussa käytetään. Palvelin palauttaa asiakaskoneelle oikean urlin, jonka jälkeen kannasta tehdään haku lomakkeessa määritellyille kohteille.

Hakua tehdessä huomasin pieniä ongelmia, kuten *AttributeError*, kun hakua suoritettiin. Ongelma johtui siitä, että olin nimennyt virheellisesti näkymissä niihin viittaavan resurssin. Viittaus kohdistui resurssitiedostoissa suoraan modeliin näkymän sijaan. Tämän korjattuani huomasin, että moni lomakkeen kohde ei näyttänyt päivittyvän lainkaan. Syynä tähän oli se, että ilmeisesti nykyinen arkkitehtuuri vaatii täsmälleen saman nimeämislogiikan eri komponenteille: Ext JS -sovelluskehyksen käyttämän näkymän ja modelin sekä Djangon resurssien että Django-modelien kenttien nimien tulee vastata toinen toisiaan. Mikäli ne poikkeavat joltakin osin toisistaan, ei sovellus osaa yhdistää tietoja toisiinsa.

Paikoitellen olin nimennyt kenttiä hieman eri tavalla, joten joudun lähipäivinä käymään koodia lävitse puuttuvien tai eroavien kenttien osalta. Todennäköisesti teen tämän silloin, kun kirjoitan koodin tallennukselle. On helpompi testata toiminnallisuutta, kun koko lomakkeen tiedot on mahdollista täyttää ja sen jälkeen testata tallentamalla, että tiedot löytyvät uudelleenhaun jälkeen lomakkeelta.

Ongelmana samassa nimeämislogiikassa on myös se, että myös vierasavainkenttien nimien tulee vastata toinen toisiaan. Tässä tulee vastaan ristiriita JavaScriptin ja Pythonin kenttien sekä muuttujien nimeämiskäytäntöjen osalta: JavaScriptissa suositellaan kamelinselän (*fooBar)* käyttämistä nimeämisessä (Crockford 2018), kun taas Pythonissa käärmeenselkä (*foo_bar)* on standardi (Python Software Foundation 2018). Lisäksi vierasavainkentät tarvitsevat taas oman funktionsa resurssitiedostoihin, sillä kenttiin ei viitata suoraan. Päätimmekin tiimin sisällä, että voisimme käyttää vierasavainten nimeämiskäytäntönä isoa

alkukirjainta ja alaviivaa, jota on käytetty myös toisinaan aiemminkin vastaavissa tapauksissa. Tavalliset kentät nimettäisiin pienin kirjaimin.

Päivän aikana esiintyi ongelmana myös Mercurialin tilapäinen toimimattomuus. En pystynyt tekemään lainkaan lisäyksiä tai koodin kommitointia, vaan sain ainoastaan kryptisen virheilmoituksen *abort: Allekirjoitus ei kelpaa.* Internetistä ei ollut apua ongelman ratkaisemiseen, eikä Mercurialia saanut käynnistettyä koneella uudelleen. Käynnistin koneen uudelleen ja Mercurial toimi nyt oikein, mutta virheen pohjimmainen syy jäi vielä mysteeriksi.

4.8 Seurantajakso 6

Keskiviikko 1.8.2018

Jatkoin tänään haun työstämistä. Muokkasin model-kenttien nimiä vastaamaan toisiaan ja lisäsin puuttuvia resursseja, koska en ollut lisännyt kaikkia vielä aiemmin, sillä niitä ei aiemmin tarvittu pelkkää ulkoasun suunnittelua varten. Tein tänään uuden Live Templaten PyCharmissa. Kyseinen koodinpätkä lisää automaattisesti uuden resurssin pohjautuen annettuun nimeen:

```
def $NAME$ (self, instance):
    return instance.$VARNAME$ and instance.$VARNAME$_id
```

Lisäksi opin käyttämään multikursoria PyCharmissa: painamalla alas alt-painiketta, pystyy käyttäjä asettamaan kursorin useaan eri kohtaan ja eri riveille samanaikaisesti. Näin käyttäjä voi muokata useaa riviä samasta kohtaa kerralla, mikä on näppärää esimerkiksi silloin, jos käyttäjä ei voi hyödyntää merkkien korvaus -toimintoa.

Aloitin tekemään tänään myös lomakkeen tallennusta ja poistamista. Lisäsin uuden painikkeen työkalupalkkiin, koska aiempien lomakkeiden luomisen yhteydessä testaajalta oli tullut palautetta, että tallentamiseen ja uuden kohteen luomiseen tulisi käyttää eri painikkeita. Lisäsin tarvittavat koodit JavaScriptin ja

Djangon puolelle. Edelleen uuden koodin määrä pysyi pienenä. Eilen ilmennyt ongelma nimeämiskäytännöistä tuli esille myös tämän päivän työskentelyssä. Huomasin, että tiettyjen toiminnallisuuksien osalta ei vaadittu täsmälleen samaa määrittelyä front-endissa ja back-endissa. Syynä tähän on luultavasti eroavaisuudet nykyisessä haku- ja tallennuslogiikan arkkitehtuurissa. Saattoi esimerkiksi käydä niin, että lomakkeen tiedot tallentuivat järjestelmään oikein. Tallennettuja tietoja ei kuitenkaan pystytty näyttämään silloin, kun lomakkeen avulla haettiin aiemmin luotuja tutkimuksia.

Torstai 2.8.2018

Jatkoin uusien tutkimusten luomista ja päivittämistä, sillä kaikkien kenttien tiedot eivät vielä tallentuneet järjestelmään. Työstin lomaketta rivi kerrallaan läpi, kunnes kaikki lomakkeen kentät tallentuisivat. Huomasin erityisesti sen, että joillakin desimaalikentillä tuli vastaan *virheilmoitus ValidationError: [u-arvo tulee olla desimaali.]*, jos kentille ei ollut asettanut arvoa. Tämä johtui siitä, että en ollut määrittänyt Ext JS -malleissa kentille tarkkaa määrittelyä tiedon tyypistä. Näin selain lähettikin tyhjän kentän sisältämän tiedon palvelimelle tyhjänä merkkijonona, mitä ei pystytä implisiittisesti muuttamaan desimaaliluvuksi. Ongelma korjaantui, kun Ext JS -modelissa määrittelee tiedon tyypin ja null-arvojen käytön:

```
{name: 'variable_name', type: 'float', useNull: true}
```

Sain huomata työstäessäni lomaketta, että kellonaika ei päivittynyt oikein tietokantaan. Kyseessä näytti olevan rajoitus Djangossa käytettyjen mixinien arkkitehtuurissa, sillä ne muuttivat automaattisesti päivämäärän ja ajan pelkäksi päivämääräksi. Huomasin kuitenkin, että järjestelmässä oli jo olemassa pieni koodinpätkä, joka muuttaa Ext JS -sovelluskehyksen näyttämän päivämäärän ja kellonajan Pythonille sopivaan muotoon, sitä vain ei oltu vielä käytetty yhdessäkään mixinissä. Ratkaisuna olisi joko luoda kokonaan uusi mixin, joka käyttäisi kyseistä kellonaikaa tai ottaa mixinissä esimerkiksi huomioon uusi keyword-argumentti, jonka ollessa käytössä otettaisiin koko aikatieto huomioon lomaketta tallennettaessa.

Tein mixiniin-liittyvästä aiheesta KeyCoren tehtävienhallintaan tiketin, esitin ratkaisumallin ja lähetin vielä pyynnön aiheen tarkistamisesta tuotepäällikölle. Päivän aikana puhuin lisäksi lomalle jäävän tuotteen omistajan kanssa työn edistymisestä. Määrittelimme lomakkeen ulkoasun pienistä muutoksista sekä tietojen lisäämisestä joillekin kentille, jotka puuttuivat aiemmista määrittelyistä.

Perjantai 3.8.2018

Tänään tulin töihin tavallista aiemmin, koska olin lähdössä samana päivänä Espooseen matkalle. Tarkoitukseni oli tänään saattaa loppuun uusien tutkimusten luonti ja päivitys. Huomasin kuitenkin heti päivän aluksi, että jostain syystä tutkimuksen aikatieto ei näyttänytkään päivittyvän oikein, vaikka olin eilen saanut sen toimimaan. Kesti jonkin aikaa ennen kuin huomasin, että olin kommentoinut pois päältä Djangon näkymäkoodista uuden toiminnallisuuden, joka ei vielä käytetty muualla. Tapaus opettaa, että olisikin ensiarvoisen tärkeää jättää koodi aina viimeksi toimivaan tilaan ennen töistä lähtemistä, koska nyt en ollut muistanut, että olisin ottanut uuden toiminnon osittain pois käytöstä.

Ennätin päivän aikana tarkistamaan ja muokkaamaan vielä loput kentät toimivaan muotoon. Nyt kaikki paitsi koordinaattitiedot päivittyvät lomakkeelle oikein. Sain ideaksi myös lisätä tooltipit eli työkaluvinkit joillekin kentille, jossa ne saattaisivat tuoda lisäarvoa käyttäjälle. Huomasin myös, että uutta tutkimusta tehdessä tuli tänään esiin myös teksti ”Tuloksia ei löytynyt”, vaikka tutkimus luotiinkin onnistuneesti. Pitää selvittää myöhemmin, mistä on kysymys. Tutkimuslomake oli uuden tutkimuksen luomisen ja päivittämisen suhteen näiltä osin lähes valmis, ensi kerralla aloitan tutkimuksen poistamisen lisäämisen.

Tiistai 7.8.2018

Tavoitteena tälle päivälle oli toteuttaa lomakkeen tallennus loppuun. Korjasin eilen löytyneen ongelman uuden kohteen luomisessa, jolloin tuli vastaan ilmoitus, että kohdetta ei löytynyt. Lomakkeen hakua tuli hieman muuttaa, jotta kohde löytyi oikein. Haku ei nimittäin etsinytkään juuri luotua kohdetta, sillä hakuun ei oltu

määritetty kohteen tietoja. Lisäsin lomakkeelle niin sanotun *getById*-haun, eli uuden kohteen luomisen yhteydessä lomakkeelle haetaan suoraan kyseisellä id:llä kaikki tiedot näkyviin – tällä myös varmistetaan, että kohde tosiaan on tallennettu tietokantaan ja käyttäjä pystyy halutessaan muokkaamaan kohteen tietoja. Lisäksi kyseistä hakua pystyttäisiin käyttämään, jos haluttaisiin muutenkin etsiä juuri tietyllä tunnisteella oleva kohde suoraan lomakkeelle ilman varsinaisia hakuparametreja.

Lisäsin tänään myös työkaluvinkit lomakkeen kentille, jotka saattavat kaivata selitystä. Lisääminen oli varsin helppoa jo olemassa olevien liitännäiskomponenttien vuoksi: riittää, että lomakkeen kentälle määriteltiin liitännäinen ja työkaluvinkille ohjeteksti. Päivitin lomakkeen lataustekstit – nyt haettaessa, tallennettaessa ja päivitettäessä tietoa tulisi näkyä kuhunkin tilanteeseen viittaava teksti sen sijaan, että näytettäisiin pelkkä geneerinen latausteksti.

Tein lisäksi toiminnallisuuden kohteen poistamiselle. Tehtävä sujui hyvin, sillä toteutusta varten ei tarvinnut laatia kovin paljoa lisää koodia olemassa olevan rungon pohjalle. Lomakkeen tehtäväpalkin koodia tuli hieman päivittää, että kohdetta poistettaessa lomakkeelle ladattujen tietueiden määrä näkyisi oikein. Olin myös nimennyt epähuomiossa yhden tietokannan kentän väärin käyttäen monikkomuotoa, mikä aiheutti aluksi virheilmoituksen kohteita poistaessa.
Löysin päivän aikana vielä bugin kohteen hausta, joissa käytettiin desimaalikenttiä: haku ei näillä kentillä toiminut aivan odotetusti. Joko tuloksia ei näytetty lainkaan tai arvo tuli määritellä hyvin tarkasti, että hakutuloksia palautettiin. Päätin, että lisään bugin ylös tikettiin ja ratkon sen myöhemmin muiden mahdollisten pienten ongelmien kanssa.

Keskiviikko 8.8.2018

Tämän päivän aikana eräs toinen työntekijä liittyi mukaan kaivotutkimuksen kehittämistyöhön. Opastin häntä tehtävän aloituksen suhteen kertomalla, mistä lomakkeessa on kysymys ja mitä tikettejä hän pystyisi tekemään. Hän aloittaisi työn tekemällä lomakkeen aukaisun kaivolomakkeen kautta listaamalla kaikki löydetyt tutkimukset. Lisäksi tehtäisiin kaivotutkimuksen liittäminen varsinaiseen kaivoon.

Omalle vastuulle jäi tänään listojen kehittäminen. Toiminto näyttäisi listanäkymässä kaikki lomakkeelle haetut tutkimukset. Sain tehtyä toiminnon pääpiirteittäin aika nopeasti valmiiksi, mutta muutamia ongelmana tuli vastaan. Listanäkymä toimii eri tavalla kuin päälomake, joten tietojen näyttäminen samalla tavalla kuin päälomakkeella ei toimisi oikein. Niinpä jouduin määrittelemään uuden resurssitiedoston, jota käytettäisiin pelkästään näiden listalla näkyvien resurssien näyttämiseen. Erona on muun muassa se, että päälomakkeen resursseissa pystyin käyttämään suoraan vierasavaimen id:tä, kun taas listanäkymässä tulisi käyttää tekstiä – muutoin lista näyttäisi pelkästään vakiokenttien id:t, mikä näyttäisi hyvin hölmöltä. Aikakenttä vaati myös uuden määrittelyn, sillä lomake ei osannut suoraa näyttää sitä oikein. Jouduin tekemään aikaa varten uuden funktion resursseihin.

Haluttuja kenttiä listanäkymään ei oltu myöskään määritetty missään. Lomakkeella on olemassa todella paljon erilaisia kenttiä, joten näitä tuskin voitaisiin näyttää mitenkään järkevästi yhdessä näkymässä kerrallaan. Asetin lomakkeelle mielestäni järkevän oloiset peruskentät ja otan jatkossa yhteyttä tuotepäällikköön, miten hän haluaa toiminnon toteuttaa.

Päivän aikana katselmointiin tuli parannusehdotus kellonaika-mixiniin: koska järjestelmässä ja Djangossa on erikseen kenttäattribuutit *DateTimeField* ja *DateField*, kannattaisi funktion tarkistuksessa käyttää DateTimeFieldiä. Koska DateTimeField kuitenkin perii DateFieldissa, tulee DateTimeFieldin sijaita ohjelman suorituksessa kyseisessä järjestyksessä. Muutoin kun kohteen tietoja tallennetaan, kentän tyyppimäärityksen tarkistus ohittaisi aina DateTimeField-kentän.

4.9 Seurantajakso 7

Torstai 9.8.2018 ja perjantai 10.8.2018

Jatkoin näinä päivinä listan työstämistä. Eilen oli ilmennyt ongelma, että tekstivakiot eivät järjestyneet oikein listalla. Syynä on se, että lista järjestää ilmeisesti oletuksena tiedot id-kentän mukaan. Näin siis kentät järjestyivät sisäisesti oikein,

mutta käyttäjän näkökulmasta olisi varmastikin suotavaa, että kentät järjestäytyisivät niissä näkyvän tekstin perusteella. Näyttikin siltä, että listan järjestäminen saattaa vaatia kustomoidun järjestyskoodin kirjoittamista.

Ongelman ratkaisua varten tutustuin muihin lomakkeisiin, joissa lista oli otettu käyttöön. Yllätyin siihen, että missään niissä ei listalle oltu juuri määritelty kustomoitua järjestyskoodia, mutta tekstivakiot näkyivät silti oikein. Toiset lomakkeet eivät kuitenkaan käyttäneet vastaavia tekstivakioita, mikä saattoi vaikuttaa asiaan.

Tein listaa varten funktion, joka järjestää tekstivakiokentät niissä olevan tekstin perusteella. Käytin tässä Pythonin lambda-koodia, joka hakee funktiona listan järjestyksessä käytetyn vakion tekstiarvon. Ongelmana tässä kuitenkin oli se, että ilmeisesti listan järjestyksellä voi olla ainakin kaksi järjestystä. Koodin pitäisi siis pystyä etsimään tekstiarvot useissa eri tapauksissa. Nykyinen koodi ei toimisi silloin, kun käytössä on kaksi eri järjestysarvoa: kentät järjesteltäisiin vain yhden arvon mukaan, toisella arvolla ei olisi mitään väliä. Djangon order_by järjestystäkään ei pystyisi käyttämään, koska tämä ei osaisi tehdä järjestystä tekstikentän mukaan.

Väliaikaiseksi ratkaisuksi laitoin järjestyksen toimimaan niin, että yhtä järjestystä käytettäessä tehdään lajittelu tekstivakioiden tekstin mukaan ja useampaa järjestystä käytettäessä alkuperäisen lajittelun mukaan. Näin lopputulos näyttää siistiltä, vaikkakin jälkimmäisessä tapauksessa järjestys ei tapahdu tekstin mukaan. Aion tutkia vielä jatkossa, mikäli tätä saisi muutettua paremmaksi.

Muistin päivän jälkeen tehneeni virheen tämän koodin asettelussa. Djangon kyselyssä olevien alkioiden pituus tulisi selvittää aina *count()*-funktiolla, mutta käytin koodissa sen selvittämiseen *len(qs)*-funktiota. Len-funktion käyttäminen suorittaa kuitenkin kyselyn tietokantaan. Mikäli kyselyä ei tarvitse suorittaa, on tehokkaampi käyttää count()-funktiota sen pituuden selvittämiseen.

Perjantaina minulla oli toisen tuotepäällikön kanssa katselmus lomakkeen tilanteesta, sillä alkuperäinen lomakkeesta vastannut päällikkö oli nyt lomalla. Pääosin olimme hyvässä tilanteessa, mutta mobiilipuolen toimivuus ja vaatimukset olivat kovin epäselviä. Raportit ovat kuulemma usein aiheuttaneet ongelmia versiopäivitysten yhteydessä, joten niiden tekeminen jätetään ulos nykyisestä aikataulusta. Tuotteen toimitus siirrettäisiin myös seuraavaan sprinttiin, jotta kaikki tarvittava toiminnallisuus saataisiin tehtyä vielä valmiiksi.

Maanantai 13.8.2018 ja tiistai 14.8.2018

Tarkistin aluksi viime perjantain oman huomion listoihin liittyen. Olin ilmeisesti muistanut väärin asian laidan, sillä tarkisteltava kohde ei ollutkaan kyselyjoukko, vaan tavallinen lista. Näin ollen pituuden selvittämiseen tulikin käyttää len-funktiota.

Päivän aikana korjasin muutamia asioita, joista huomautettiin aiemmassa katselmoinnissa. Kohteen poistamisen *doDelete*-funktiossa en ollut käyttänyt tällä kertaa puolustavan ohjelmoinnin tyyliä, jossa funktion suoritus lopetetaan jo heti alussa, mikäli saadut parametrit tai näkyvyysalueen muuttujat ovat puutteelliset. Yhdessä funktiossa olin käyttänyt myös tätä tyyliä, mutta olin tehnyt ”turhaan” *if-else* haaran: koska funktiossa lopetetaan suoritus etuajassa, ei else-haaran määritys ole tarpeellista.

Sain tarkistettavaksi työkaverini tekemän toiminnallisuuden kaivotutkimukseen. Hän oli tehnyt lomakkeelle kaivojen kytkemistyökalun, jolla pystyy valitsemaan kaivot kartalta ja liittämään ne osaksi tutkimusta. Kokeillessani työkalu ei toiminut aluksi lainkaan. Syynä oli ilmeisesti työkaverin tekemät muutokset, joita hän oli tehnyt katselmoinnin aikana. Ottamalla muutokset pois käytöstä, toiminnallisuus toimi oikein. Huomautin vain yhdestä virheilmoitukseen liittyvästä tekstistä, sillä sen kuvaus oli käyttäjän näkökulmasta hieman epäselvä.

Aloitin tänään tekemään kaivotutkimukselle liitteitä. Liitteet ovat eri lomakkeille määritelty toiminnallisuus, jossa kohteelle voidaan lisätä yhteyksiä erilaisiin tiedostoihin, kuten kuviin tai dokumentteihin. Kuntotarkistuksen yhteydessä nämä

voisivat olla esimerkiksi valokuvia tarkastetun kaivon kunnosta. Toiminnallisuus on jo olemassa usealla eri lomakkeella, mutta yllätyin siitä, ettei toiminnolle ollut määritelty minkäänlaista toteutusdokumentaatiota ohjelmistokehityksen wikiin. Jouduin tekemään oman toteutuksen käytännössä seuraamalla olemassa olevaa koodia ja muuttamalla sitä tarpeen mukaan sopivaksi tutkimuslomakkeelle sopivaksi.

Sainkin huomata, että liitteiden toteutus olikin monimutkaisempi kuin alkuun kuvittelin. Toimintoa varten tuli määrittää koodia moneen eri paikkaan ja toiminnallisuus vaati myös uuden tietokantataulun luomista, mitä ei otettu alkuperäisessä dokumentaatiossa huomioon. Aluksi liittäminen ei toiminut, sillä olin erehdyksessä määritellyt liitteen lisäys- ja poistokoodiin liittyvät osoitteet ja näkymät ristiin.

Liitteiden lisääminen ja lataaminen toimivat pääosin ongelmitta, mutta lomakkeen latauskoodia joutui muuttamaan. Arkkitehtuurin runkokoodia käyttämällä liitteet eivät lainkaan latautuneet olemassa oleville kohteille, mutta näkyivät silloin, kun kohteelle luotiin uusi liite. Alkuperäisissä ratkaisuissa latausongelma oli ratkaistu ylikirjoittamalla kokonaan lomakkeelle tietoja lataava funktio. Päätin itse tehdä näistä muutoksista oman funktion, koska funktion ylikirjoitukselle ei ollut varsinaista tarvetta. Tein erikseen pienen funktion, joka hoitaa liitetietojen latauksen ja toisen funktion, jota käytetään liitetietojen tyhjentämiseen.

Liitteissä huomasin myös, että niiden ulkoasu oli määritelty hieman eri tavalla riippuen lomakkeen tyypistä. Dojo-lomakkeet näyttivät päivämäärän oikein, mutta niissä liitteen lisääjänä näytettiin kirjautuneen käyttäjän sijaan tietokannan skeemanimeä. Ext JS -lomakkeissa aikaa ei puolestaan aseteltu oikein. Dojo-lomakkeet käyttivät taustalla RPC-lomakkeita, joissa aika- ja päivämäärä kenttien määrittelyssä käytettiin sellaista kenttäfunktiota, joka automaattisesti asettelee tiedon näkymään oikeassa muodossa. Ext JS -lomakkeen käyttämästä REST-näkymästä tämä näytti puuttuvan. Jouduin muuttamaankin liitteiden resurssitiedostoa siten, että aikaa ja päivämäärää haettaessa se muotoillaan oikean tyyppiseen muotoon, esimerkiksi *19.08.2018 16:28*.

Tiistaipäivän työskentely loppui hieman lyhyeen, sillä olin lähdössä käymään vanhempien luona viikon välissä. Tätä ennen törmäsin lisäksi salasanan vaihto-ongelmaan: tänään oli pakotettu salasanan vaihto, mutta salasanan vaihtamisen jälkeen en pystynyt lisäämään koodimuutoksiani ollenkaan versionhallintaan. Kaikkialla muualla tunnukseni toimivat kuitenkin oikein. Yritin koodin lisäämistä useampaan kertaan, mutta tunnukseni lukkiutuivat kahteen eri kertaan, jolloin jouduin ottamaan yhteyttä IT-tukeen niiden avaamiseksi.

Käynnistin yhdessä vaiheessa tietokoneen uudelleenkin, koska epäilin syyksi välimuistiongelmaa. Uudelleenkäynnistyksestä ei kuitenkaan ollut apua. Koska tunnukseni toimivat muualla kuin Mercurialissa, ajattelin, että vaihtaisin salasanan toiseen. Uudessa salasanassani oli skandinaavisia merkkejä, joiden epäilin aiheuttavan virheitä versionhallintatyökalun kanssa. Yritin vaihtaa salasanan joksikin toiseksi, mutta tämäkään ei onnistunut. Huolimatta käyttämästäni salasanayhdistelmästä, Windows kertoi uuden salasanan olevan joka kerta joko liian samanlainen edellisen kanssa tai että se ei täyttänyt salasanalle asetettuja turvallisuusehtoja. Päätin jättää asian sikseen ja selvittää ongelman ratkomista seuraavalla kerralla, kun olen töissä.

17.8.2018 pe

Viimeksi työskennellessäni salasanan vaihto oli aiheuttanut ongelmia. Kokeilin heti töihin päästyäni aiempien muutoksien lisäämistä versionhallintaan, eikä se edelleenkään toiminut. Päätin vaihtaa salasanaa saman tien ja tällä kertaa vaihto onnistui ilman ongelmia. Heti salasanan vaihdettuani kokeilin koodin lisäämistä uudelleen versionhallintaan, mikä onnistui ilman ongelmia. Ilmeisesti siis skandinaaviset merkit voivat aiheuttaa ongelmia, jos Mercurialia käyttää komentorivin kautta.

Salasanaongelman selvittyä jatkoin liitteiden muokkaamista. Tein muutamia pieniä parannuksia koodiin. Korvasin useassa kohdassa käytetyn komennon viestiikkunan pyyhkimiseen erillisellä funktiolla, jota on helpompi kutsua kuin Ext JS -

sovelluskehyksen *defer*-metodia (lykkää komennon suorittamista tietylle määräajalle). Korjasin myös pari pientä bugia koodista, sillä koodista oli unohtunut muutamassa kohdassa `var me = this;` määrittelyt.

Laitoin tänään aiemman mixiniin tekemäni muokkauksen odottamaan tarkistusta, sillä katselmoinnin aikaraja oli jo mennyt umpeen pari päivää sitten. Tarkistin työkaverini tekemän lomakkeen esitäytön. Toiminto oli hyvin tehty ja ainoastaan parista kohdasta oli huomauttamista: esitäytön painike ei aktivoitunut uuden kohteen luomisen jälkeen (jos kohde oli kytketty kaivoon) ja mikäli kaivolta puuttuivat koordinaatit, näkyivät lomakkeen koordinaattikentässä tiedot väärin. Lisäksi toiminto ei tässä vaiheessa tehnyt mitään kohteen putkitiedoille – toisaalta putkille näytettävien tietojen taulukkoa ei oltu muutenkaan vielä tässä vaiheessa määritelty tarkemmin.

Päivän aikana käytin varsin suuren osan aikaa osallistumalla työkaverini tekemään katselmointiin viimeisten kääntämättömien tekstivakioiden osalta. Käännöstyö oli varsin iso, sillä skriptitiedostolla oli pituutta noin 5000 riviä. Sain käytyä päivän aikana tästä läpi noin 3600 riviä. Laitoin omia käännösehdotuksia mukaan niiltä osin, missä näin niille tarvetta. Moni muu oli jo huomauttanut puutteista itse skripteihin liittyen, joten en käyttänyt tähän enää aikaa katselmoinnissa.

Iltapäivän päätteeksi laitoin sähköpostia tekstivakioiden tilasta henkilöille, jotka vastaavat tuotteen esittelystä ulkomaisille asiakkaille. Ohjelmistokehittäjillä ei ikävästi ole niin tarkkaa kuvaa siitä, mitä loppupelissä käyttäjät haluavat käännöksiltä. Käännösten teknisyyden takia on hankala sanoa, mitkä käännökset olisivat sopivia – kaikille termeille ei välttämättä edes ole suoraa käännöstä englanninkielellä. Tein aiheesta ehdotuksen katselmointia tai muuta tarkastelua varten, jossa arvioitaisiin toteutumisen onnistumista.

Seuraavana työpäivänä on tavoitteena aloittaa lomakkeen putkitietojen tekeminen, muilta osin lomakkeen pääasialliset toiminnallisuudet ovat valmiina.

4.10 Seurantajakso 8

Maanantai 20.8.2018

Aloitin tänään kaivotutkimuksen putkitietojen työstämisen. Putkitietojen ulkoasu oli jo määritelty, mutta varsinainen toiminnallisuus puuttui. Tarkoitus oli listata putket ruudukkoon. Putken valinta ruudukosta johtaisi lomakkeen kenttien päivittymiseen, minkä kautta käyttäjä pystyy päivittämään yksittäisen putken tietoja.

Ensimmäiseksi siirsin putkitiedostojen sijainnin, kuten olin sopinut tuotteen omistajan kanssa. Olisi käyttäjän kannalta selkeämpää, jos putket esiteltäisiin lomakkeella ennen varsinaisia kaivon vikatietoja. Tämän jälkeen aloin tekemään putkitiedoille tarvittavia modeleita – ne puuttuivat aiemmasta toteutuksesta kokonaan. Uusissa modeleissa määriteltiin kentät, joita halutaan putkelle tallettaa. Asetin putkitiedoille pari aiemmin puuttuvaa kenttiä, jotka eivät olleet alkuperäisessä ulkoasuhahmotelmassa mukana, mutta jotka mainittiin kuitenkin tietokannan toteutuksessa.

Lisäsin putkitietojen näyttämistä varten Djangoon urlit, näkymät ja Django-modelit. Putkitietojen testaamista varten lisäsin tietokantaan testidataa yhdelle tutkimukselle, jotta voisin kokeilla putkitietojen näyttämistä lomakkeella. Tein haun pohjan valmiiksi. Haun oli tarkoitus toimia niin, että kun tutkimus haetaan lomakkeelle, haetaan sen jälkeen tutkimukseen liittyvät putket. Haku ei kuitenkaan vielä toiminut, vaan näytti jäävän jostain syystä ikuiseen silmukkaan. Pitää selvittää huomenna, mistä tämä oikein johtuu.

Sain tänään tiedoksi kätevän työkalun dokumentoimista varten. *ScreenToGif* on ilmainen ohjelma, jolla pystyy tallettamaan gif-animaatioita ruudulta valitun alueen sisältä. Työkalu on kätevä erityisesti toimintojen esittelyä ja bugien toistettavuuden näyttämistä varten. Päivän muina tehtävinä katselmoin loppuun työkaverin tekemää suuren tekstivakio-päivitystä, jossa oli yhteensä yli 5000 riviä tarkistettavaa riviä tietokantamuutoksiin.

Tiistai 21.8.2018

Etsin tänään syytä siihen, miksi tekemäni putkien haku ei toiminut. Syynä näytti olevan pari pientä sekaannusta käytetyssä haussa. En ollut lisännyt tutkimuksen tunnistetta hakuun, joten putkia ei haettu oikealle kohteelle, vaan se kohdistettiin kaikkiin kannassa oleviin putkiin. Lisäksi putkien haku kohdistui putkivaraston sijaan lomakkeen käyttämään varastoon. Sekaannus aiheutti lomakkeen tietueen muutoksia tarkastelevan kuuntelijan aktivoitumisen, joka johti ikuiseen silmukkaan tietojen päivittyessä aina haun loputtua.

Tein alustavan tavoitteen joidenkin tekstivakioiden muuttamisesta saman nimisiksi. Ei ole juuri syytä käyttää uusia vakioita arkipäiväisille asioille, kuten ”Kyllä/Ei” tyyppisille valikkoruuduille. Samoja vakioryhmiä on mahdollista käyttää taatusti useammassa paikassa.

Laitoin tuotteen omistajalle viestiä putkitietojen kenttien tarpeellisuudesta ja käytetyistä mittayksiköistä, joita ei pystynyt suoraan ohjelmasta tai ohjelmakoodista näkemään. Eräs kenttä oli mukana lomakkeella, vaikka lähes vastaava kenttä oli myös olemassa lomakkeella. Toisena asiana pohdin putkitietojen päivittämistä. Periaatteessa listaa olisi ollut mahdollista päivittää suoraan ruudukosta sen sijaan, että tiedot ladataan ensin kenttiin. Ongelmana tässä on kuitenkin se, että kyseinen menettely ei mahdollista pudotusvalikoilla toteutettujen kenttien päivittämistä. Tästä syystä päädyin toteuttamaan vain tämän päivitysmahdollisuuden. Ei ole järkeä juuri mahdollistaa kahta eri tapaa, jos vain toisella tavalla on mahdollista päivittää kaikki tiedot.

Keskiviikko 22.8.2018 ja torstai 23.8.2018

Jatkoin näinä päivinä taas putkitietojen työstämistä. Tavoitteena oli toteuttaa tietojen tallennus valituille putkille. Tämän tekemiseen menikin odotettua enemmän aikaa. Ensin piti miettiä, miten ruudukosta valitut kentät saadaan päivitetyksi lomakkeen kenttiin muokkaamista varten. Toteutin aluksi ratkaisun, jossa kentät päivitetään suoraan lomakkeella olevien kenttien tietojen perusteella. Tässä ratkaisussa huonona puolena oli kömpelyys ja tarvittavan koodin määrä. Jokainen

kenttä piti päivittää yksitellen ja mikäli niiden nimeä joutuisi muuttamaan, lakkaisi päivitys heti toimimasta.

Paremmaksi ratkaisuksi päädyin määrittelemään putkiin liittyvälle tiedolle oman Ext JS mallin ja tietovaraston, mihin ladattaisiin uusi tietue, kun käyttäjä valitsee ruudukosta putken. Tämä ratkaisu toimikin erinomaisesti, vähentäen huomattavasti tarvittavan koodin määrää ja ollen muutenkin joustavampi. Ratkaisuissa piti vain muistaa käyttää Ext JS -sovelluskehyksen *updateRecor*d-komentoa ennen tallentamisen kutsumista, mikä saattaa helposti unohtua. Jos näin ei tehdä, eivät muutokset lomakkeen kentistä tallennu lainkaan muokattavaan tietueeseen.

Putkien tallentamiseen oli nyt käytössä oma painike, koska putkien päivittämisen yhteydessä ei tarvita päivittää mitään muuta tietoa. Päivitys on lisäksi järkevää kohdistaa vain yhteen putkeen kerrallaan. Huomasin tallennuksen yhteydessä, että jostain syystä väärän putket haettiin näkymään, kun tallennuksen jälkeen tehtiin uusi haku ruudukkoon.

Torstai-aamuna sain havaita, että lomake ei auennut ollenkaan. Sain ihmetellä asiaa jonkin aikaa, kunnes huomasin, että kyseessä oli yksinkertainen syntaksivirhe. Tapaus kertookin tutun opetuksen, että olisi aina syytä jättää koodi virheettömään tilaan töitä lopetettaessa, jotta syytä ei tarvitse etsiä myöhemmin.

Torstain aikana kantautui myös ikävä vastausviesti sähköpostiini. Olin kuullut jo pidemmän aikaa kyselyitä etenemiselle ja sain tänään kuulla siihen syyn: toiminnallisuuden olisi pitänyt olla valmiina jo kesäkuun lopussa asiakkaalle. Asia tuli minulle yllätyksenä, koska ensimmäinen kehityspalaverini toiminnallisuuden toteuttamisesta oli ollut vasta heinäkuussa. Jotain oli siis mennyt aikataulutuksessa pieleen, koska kehitystyö aloitettiin vasta huomattavan paljon myöhemmin.

Rupesin pohtimaan torstaina myös tarvetta yksiköiden näyttämiseen kentissä. Vanhemmilla lomakkeilla joidenkin kenttien kohdalla yksiköt ja niiden asettelu näkyvät kentissä, kun taas joissakin ne puuttuvat. Uudemmilla lomakkeilla olisi kuitenkin mahdollista käyttää työkaluvinkkejä, joissa yksiköt voitaisiin näyttää ja samalla poistaa ne näkyvistä lomakkeen kenttäteksteistä. Keskustelin asiasta

työkaverin kanssa ja hän näki tässä kaksi eri näkökulmaa. Tottuneemmat käyttäjät todennäköisesti tietävät jo entuudestaan, missä muodossa ja yksiköissä annetut arvot ovat.

Perjantai 24.8.2018

Tarkoitus oli tälle päivälle lisätä tutkimukseen toiminnallisuus, jossa kaivoon kytketyt putket tuodaan automaattisesti osaksi kaivotutkimusta. Liitin tämän toiminnon osaksi kaivoon liittämistä. Kun käyttäjä kytkee kaivon tutkimukseen, haetaan palvelimen päässä kaivoon liittyvät putket. Mikäli putkia löytyy, tehdään niistä uudet objektit tutkimusta varten. Näiden niin sanottujen tutkimusputkien lisäyksen yhteydessä tehdään putkille tietojen esitäyttö alkuperäisten tietojen mukaan, mikäli ne putkelta löytyvät (oletuksena käyttäjän ei pitäisi edes pystyä lisäämään järjestelmään putkia ilman näitä tietoja, koska ne on määritelty pakollisiksi).

Samalla toteutukseen lisättiin putkien päivitys, mikäli käyttäjä vaihtaa tutkimukseen liitettyä putkea esimerkiksi virheellisen kytkennän vuoksi. Tällöin olemassa olevat tutkimusputket poistetaan ja korvataan uuden kaivon putkilla. Toteutuksissa tuli ottaa huomioon myös se, että putkeen liittyvät tiedot haetaan oikein. Yksittäinen putki voi olla yhteydessä kahteen eri kaivoon, minkä takia esimerkiksi korkotiedot ovat putken alkupäässä erilaiset kuin mitä lopussa. Oikeiden tietojen päivitystä varten koodissa verrattiin putkeen liittyvän kaivon tunnistetta kaivoon, mikä tutkimuslomakkeeseen on liitetty.

Olin varsin tyytyväinen tämän päivän aikaansaannoksiin. Seuraavalle päivälle jäi tehtäväksi lähinnä koodin siivoaminen ja katselmoinnin tekeminen. Eräänä haasteena tämän päivän toteutuksessa oli tekstivakioiden erilaisuudet eri lomakkeiden välillä. Putkilomakkeen ja tutkimuslomakkeen käyttämät tekstivakiot erosivat arkkitehtuuriltaan toisistaan. Jotta tiedot saataisiin päivittymään oikein tutkimuslomakkeen putkille, olisi tutkimuslomakkeelle käytännössä lisättävä uudet tekstivakiot, jotka vastaisivat aiempien vakioiden muotoilua. Näin tutkimuslomakkeen tiedot voitaisiin päivittää siten, että käyttäjälle näkyisivät samat valikkojen optiot, vaikka ne järjestelmässä sijaitsivatkin eri tietokantatauluissa. Jos tutkimuslomak-

keella olisi käytetty vanhantyylisiä tekstivakioita, olisi lomakkeelle joutunut lisäämään erillisen latauskoodin alasvetovalikkojen optioille. Päädyin siksi lisäämään aiemmat vakiot uuteen tauluun, koska uusi tapa alasvetovalikkojen koodille ei vaadi erillisen latauskoodin kirjoittamista. Tällöin ohjelman ylläpitäminen ei vaadi yhtä paljoa muokkauksia ohjelmakoodin toimintaan.

Toinen ongelma liittyi putkitietojen yksikköihin. Nämä yksiköt eivät näkyneet lomakkeella joiltain osin lainkaan, eivätkä toiset kehittäjätkään olleet näistä tietoisia. Sain lopulta kuitenkin ohjekirjan avulla tiedoin oikeista yksiköistä. Kolmas pieni ongelma toteutuksessa oli tulkita, mitkä putkitietojen lomakkeista vastasivat mitäkin Django-modelien attribuutteja. Eri vierasavainkentät oli nimetty hyvin samankaltaisesti, joten lomakkeilla näkyviä kenttiä ei helposti tunnistanut Django-modelien käyttämistä kentistä pelkän nimen perusteella.

4.11 Seurantajakso 9

Maanantai 27.8.2018

Tavoitteena tälle päivälle oli siivota hieman putkitietoihin liittyvää koodia. Olin aiemmin tehnyt lähinnä toimivan prototyypin toteutukselle, jossa toteutus oli käytännössä yhden funktion sisällä. Tarkoitus oli muuttaa koodia erillisiin funktioihin, jotka toteuttavat kukin yhden tietyn tehtävän, selkokielellä esimerkiksi ”Hae varusteen putket”, ”Hae varuste”, ”Täytä putken tiedot” jne. Tein tätä varten uuden tiedoston, johon sijoitin putkitietoihin liittyvät uudet funktiot, joita kutsutaan Djangon näkymistä. Näin näkymän koodi pysyy puhtaampana ja sisältää lähinnä palvelimen REST-arkkitehtuuriin liittyvät komennot.

Muutin putkitietojen päivityksen siten, että mikäli käyttäjä valitsee kytköstä tehdessä kaivotietojen päivityksen, päivitetään samalla myös putkitiedot. Olettaisin, että käyttäjä haluaa samalla päivittää kummatkin tiedot. Mikäli käyttäjä ei tee esitäyttöä kytkennän aikana, voi esitäytön tehdä edelleen manuaalisti putkitietojen välilehdeltä.

Huomasin pienen logiikkavirheen putkitietojen päivityksessä. Jos tutkimukseen liitetään entisen kaivon tilalle uusi kaivo, käytännössä edelliset putket poistetaan ja korvataan uuden kaivon tiedoilla. Jos sattui käymään niin, että uudella kaivolla ei ollutkaan putkia, saattoivat vanhat putket jäädä poistamatta. Syynä tähän oli yksinkertaisesti se, että putkien poistokoodia ei suoriteta, mikäli uudella kaivolla ei ollut putkia. Siirsin poistokoodin ylemmäs, jolloin putket poistettiin aina ennen kuin uuden kaivon putkien olemassaoloa tarkistettiin.

Sain tehtyä päivän tavoitteet varsin hyvin loppuun asti. Huomenna tehtäväksi jää lähinnä putkitietojen viimeistely siten, että saan koodin katselmoitavaksi.

Tiistai 28.8.2018

Tänään päiväkirjaan tuli tavallista vähemmän merkintöjä. Määrällisesti huomioitavia asioita oli tavallista työpäivää vähemmän, enkä tehnyt mainittavasti monia asioita. Lisäsin putkitietojen koodiin funktioiden kommentit ja poistin tarpeettomat mixinit, joita oli käytössä joissakin kaivotutkimuksen selainkoodin tiedostoissa. Poistin koodista myös monia jo korjattuja TODO-kommentteja. Lähetin tuotteen omistajalle sähköpostia askarruttavista asioista lomakkeen toiminnallisuuteen liittyen. Mietin esimerkiksi, oliko putkitietojen korkokenttä enää tarpeellinen. Lisäksi pyysin tarkennusta mobiilinäkymän toimivuudesta ja raporttiin sisällytettävistä asioista.

Tein lopulta katselmoinnin putkitietoihin liittyvästä koodista. Aloitin samalla korjaamaan kaivotutkimukseen liittyviä bugeja ja huomautuksia, joita olin yhdessä työkaverini kanssa löytänyt. Korjattavia kohtia oli kertynyt yllättävän paljon, sillä olimme päättäneet siirtää verifiointivaiheessa ilmenneet ongelmat korjattavaksi tähän yhteen päätikettiin. Jatkan näiden parissa työskentelyä huomisesta alkaen.

Keskiviikko 29.8.2018 ja torstai 30.8.2018

Korjasin tänään haun kentille, joissa käytettiin desimaali- tai liukulukuja. Ongelmana oli ollut, että hakuehdoissa ei pystynyt käyttämään niin sanottua ”osittaishakua”. Jos esimerkiksi hakisi tietuetta, jolla on määritelty numeraaliseen kenttään

arvo 1337 hakukomennolla 13%, ei kyseistä tietuetta löytyisi lainkaan. Ainoastaan täsmälleen samalla luvulla oleva arvo löytyisi. Ongelma johtui ilmeisesti siitä, että nykyisessä arkkitehtuurissa pitää myös numeraalisilla kentillä käyttää Ext JS -malleissa tekstikenttiä, mikäli halutaan mahdollistaa näille kentille haku lomakkeelta käsin. Tässä tavassa on ongelmana kuitenkin se, että oletuksena tyhjät arvot näissä kentissä tallennetaan tyhjänä merkkijonona, mikä aiheuttaa ongelmia, kun tekstikentän arvo muutettaisiin palvelimella liukuluvuksi. Näinpä mallissa pitää erikseen määritellä vielä muunnosfunktio, mikä asettaa tyhjän tekstikentän arvon null-tilaan (Pythonissa None).

Lisäsin lomakkeen eri painikkeille tilat, joiden perusteella painikkeet ovat joko käytettävissä tai pois päältä. Aktiiviset painikkeet perustuvat lomakkeella olevan tutkimuksen tilaan. Ilman tutkimusta käytettävissä on vain luontipainikkeet, ilman kaivoa olevalla tutkimuksella ei ole esitäyttöä ja täysi kytketyllä tutkimuksella on kaikki painikkeet käytettävissä. Vastaavasti mikäli tutkimukseen kytketyllä kaivolla ei ole putkia, putkitietojen painikkeet eivät ole käytössä.

Korjasin yhden pakollisen kentän määrityksen Djangon malleissa. Lomakkeelle oli asetettu pakollinen kenttä, jota ei kuitenkaan pakotettu Djangon mallissa. Asetin Djangon mallissa kentälle arvon blank=False, mikä tarkoittaa, että Django ei hyväksi tallentamista, mikäli kentälle ei ole määritelty arvoa.

Korjasin tänään myös listan ja päivämääräkentän haun. Listanäkymä ei auennut yksinkertaisesti siksi, että olin aiemmin korjannut näkymien osoitteet ja epähuomiossa listan osoite oli muuttunut vääräksi. Päivämääräkentässä oli ongelmana, ettei haku toiminut, kun käyttäjä määritti haun tietylle välille (esimerkiksi päivien 15.7.2017–20.7.2017 välille). Syynä oli ilmeisesti puuttuvat määrittely. Päivämääräkenttien hakua varten tuli määritellä malliin parametri, joka mahdollisti haun käyttäjän määrittämälle välille.

Olin aiemmin huomannut, että poimi-toiminto ei mene pois päältä, kun käyttäjä painaa hiiren kakkospainiketta. Tämä on ollut oletuksena käytössä monilla Dojo-lomakkeilla. En saanut tätä kuitenkaan toimimaan. Tarkempi tutkimus osoitti, ettei

tämä ollut käytössä missään muissakaan Ext JS -lomakkeissa, kun poimi-toiminto oli lomakkeella olemassa. Pitää ottaa jatkossa esille, miten tämä olisi mahdollista toteuttaa.

Huomasin tänään, että kaivon listanäkymässä oleva tutkimus on hivenen harhaanjohtava. Lomakkeella on olemassa painike uuden tutkimuksen luontia varten. Painikkeen napsauttaminen aukaisee tutkimuslomakkeen ja esitäyttää kaivon tiedot lomakkeelle, mutta ei kuitenkaan tee kytköstä kaivoon. Olettaisin, että useimmat käyttäjät uskoisivat tämän toiminnon automaattisesti yhdistävän tutkimuksen kaivoon, koska lomakkeella näkyvät jo esitäytetyt tiedot. Muutin koodia siten, että mikäli käyttäjä aikoo luoda uuden tutkimuksen tätä kautta, luodaan samalla myös uusi yhteys kaivoon. Samalla esitäyttöä ei myöskään pyydetä käyttäjältä enää uudelleen, koska lomakkeella on joka tapauksessa jo tiedot kaivosta.

Edellistä toimintoa tehdessäni huomasin myös, että tutkimus käytti poimi-työkalun kanssa sisältötyyppinä (Djangon content type) viemärivarustetta, kun taas kaivolomakkeen sisältötyyppi on viemärikaivo (niin sanottu proxy model). Kysyin tästä toiselta työntekijältä mielipidettä asiaan. Hän tuli samaan tulokseen kuin minä, että parasta olisi käyttää tässä yhteydessä suoraan itse kohteeseen viittaavaa sisältötyyppiä, mikäli se on toiminnan kannalta mahdollista. Kysyin sen jälkeen syytä tähän valintaan päätymiseen ominaisuuden tehneeltä työntekijältä. Ilmeisesti syynä olivat jotkin valintatyökaluun liittyvät ongelmat, jotka vaativat, että sisältötyyppinä käytettäisiin varustetta proxy modelin sijaan (huom. myöhemmin tämä vaihdettiin viittaamaan kuitenkin proxy modeliin, koska mobiilinäkymässä käytettiin sittenkin viittausta siihen).

Perjantai 31.8.2018

Korjasin tänään kaivotutkimukseen työkaverini tekemän listanäkymän, joka näyttää yksittäiselle kaivolle tehdyt tutkimukset. Näkymässä eivät toimineet lainkaan raportit eikä myöskään käyttäjän valitsemien kenttien mukaisesti tehty järjestäminen. Uudelleenjärjestämisen syyksi paljastui se, että listan resurssit käyttivät eri tiedostoa kuin mitä listassa todellisuudessa näytetään. Tällöin lista näyttää kyllä sisällön, kun ikkuna avataan, mutta uudelleenjärjestyksen yhteydessä järjestelmä

ei osaa hakea kenttien tietoja. Vaihdoin resurssitiedoston samaan kuin mitä päälomakkeen lista käyttää, jolloin listan uudelleenjärjestys toimi heti ongelmitta.

Listan raporttinäkymän aukaisu aiheutti Djangon template-virheilmoituksen. Syyksi paljastui listalta puuttunut alkuperäisen kyselyn tiiviste, mikä tarvitaan listaraporttien tekemiseen. Tein koodiin muutokset siten, että alkuperäisen kyselyn tiiviste tallennettaisiin välimuistiin, jos sitä ei ole olemassa listan luontihetkellä. Mikäli tiiviste löytyy, käyttää listaraportti sitä raporttitiedoston laatimiseen.

Muita tänään tekemiä asioita olivat tutkimusten liittyvien kohteiden poistaminen määrittämällä vierasavaimille CASCADE-ehto. Toteutuksen alussa olin käyttänyt PROTECT-ehtoa toiminnallisuuden yleistä suunnittelua ja testausta varten. Kun olin saattanut tietomallin lopulliseen muotoon, muutin tämän määrittelyn CASCADE-muotoon. Näin varmistetaan, että kun tutkimus poistetaan järjestelmästä, poistetaan myös tutkimukseen liittyvät kohteet tietokannasta. Toteutuksen aikana kävi myös ilmi, että kaivon poistamisen jälkeen poistettaisiin myös kaivoon liittyvät tutkimukset. Koska toiminnallisuuden mahdollinen laajentaminen muihin kaivotyyppeihin oli pidetty alusta asti mielessä, käytettiin liitostaulussa viittauksessa vierasavainten sijaan alkuperäisen kohteen id- ja content type -määrittelyä. Tästä syystä kaivon poistologiikkaan tuli lisätä tarkistus, jossa kaivon poiston yhteydessä poistetaan kaivoon liittyvät mahdolliset tutkimukset.

Korjasin vielä putken esitäyttöön liittyvät ongelmat. On mahdollista, että käyttäjä poistaa tai kytkee irti kaivosta siihen aiemmin liitetyn putken, minkä vuoksi esitäyttö lakkaa toimimasta. Korjasin tämän lisäämällä esitäytön yhteyteen huomautuksen, mikäli järjestelmä ei pysty enää löytämään alkuperäistä putkea.

Seuraavalle viikolle jää tehtäväksi koordinaattikenttiin liittyvät muutokset. Huomasin tänään, että koordinaatit eivät ilmeisesti vielä tallennukaan järjestelmään, koska kentät on yhä kommentoitu pois koodista. Mikäli kentät otetaan käyttöön, tulee asiasta heti tallennuksen yhteydessä virheilmoitus.

4.12 Seurantajakso 10

Keskiviikko 5.9.2018

Tänään tein lyhyemmän päivän, sillä menen joka toinen viikko koululle ryhmäohjaukseen. Tavoitteeni oli lisätä koordinaattien tallennus, joka oli jäänyt aiemmilta kerroilta uupumaan. Kollegani oli myös löytänyt lisää huomauttamisen aiheita lomakkeesta: yhden sarakkeen koko olikin puolet sallittavasta tilasta, joten kasvatin sen koon suuremmaksi. Toinen huomautus koski Djangon sisältötyyppejä. Aiemmin ehdottamani sisältötyyppi, jota käytetään viemärilomakkeella, otettiin nyt kuitenkin käyttöön, koska ilmeisesti mobiilinäkymä vaatii sen toimiakseen. Pitää varmistaa seuraavalla kerralla, että kohteet tallennetaan oikein sisältötyypin mukaan.

Kolmas huomautus koski jo viime perjantaina huomaamaani bugia: on toisinaan mahdollista, että putkilomakkeen ruudukko jää näennäisesti latautumaan eikä lomakkeen nollaus tyhjennä sitä. En saanut tätä toistettua perjantaina, joten tulee tutkia, milloin bugi esiintyy ja miten se tulisi korjata.

Korjasin lomakkeen leveyden muuttamalla yksinkertaisesti lomakkeen oletusleveyden täydeksi, jolloin erilliset määrittelyt pystyttiin poistamaan kokonaan lomakkeen yksittäisiltä kentiltä. Lisäsin lomakkeen tallentamiseen Ext JS -sovelluskehyksen oman validoinnin, joten tallennus keskeytyy jo ennen palvelimelle menevää käskyä, mikäli muutamia lomakkeelle määriteltyjä, vaadittuja kenttiä ei ole täytetty.

Tein koordinaateille toimivan tallennuksen. Tämä vaati hieman opettelua OpenLayersin ja säännöllisten lausekkeiden käytöstä. Ensiksi kohteelle esitäytetty koordinaattitieto tuli käsitellä esimerkiksi säännöllisillä lausekkeilla oikeaan muotoon. Poistin säännöllisillä lausekkeilla koordinaattikenttään tulevasta tiedosta sulut, tyhjät välimerkit sekä lisäsin erottimeksi pilkun X-, Y- ja Z-koordinaattien väliin, jotta tiedoista pystyi muodostamaan OpenLayersin pistemäinen kohde (OpenLayers.Geometry.Point). Tämän jälkeen muodostetaan uusi OpenLayersin GeoJSON-objekti, jolle annetaan parametriksi aiemmin luotu piste. Lopuksi

GeoJSON-objekti sijoitetaan Ext JS -tietueen koordinaattien tilalle. Nyt käytettävissä on uusi JSON-muotoa oleva objekti, joka voidaan tallentaa Djangon kautta tietokantaan geometriatietona.

Ongelmia edellisessä aiheutti säännöllisten lausekkeiden oikea käyttö sekä se, että tallensin tietueeseen vahingossa väärällä id:llä tiedon (tietueen id on eri kuin mitä kentän itemID). Hain tiedon käsittelyä varten aluksi kentästä, kun se olisi kannattanut hakea suoraan Ext JS:n tietueesta (record). Tästä syystä tietoa tallennettaessa tuli traceback, koska tietueella oli kaksi koordinaattitietoa, joista toisesta ei pystytty muodostaan oikein sijaintitietoa. Muutettuani tiedon haun ja tallennuksen tietueeseen kohdistuvaksi tallennus toimi oikein.

Seuraavana päivänä pitää jatkaa toisen koordinaattikentän käsittelyllä ja määritellä vielä tiedon esitystapa: nyt tieto näkyy kentässä GeoJSON-tietona, mikä ei ole käyttäjän näkökulmasta kovin järkevä esitystapa.

Torstai 6.9.2018

Tavoitteena oli tänään saada koordinaattikenttien tallennus ja näyttäminen täysin kuntoon. Eilen olin saanut toisen koordinaattikentistä toimivaksi, mutta tiedon esitystapa muuttui tallennuksen jälkeen lomakkeella GeoJSON-muotoon. Tämä johtui ilmeisesti siitä, että Ext JS -lomakkeelle oli määritelty tietueen lataaminen, kun kohde on tallennettu. Tietue ladattiin palvelimelta saadusta palautteesta, joten sen muotoilu ei ollut oikeanmukainen. Poistin tämän latauksen käytöstä, sillä sinällään sille ei ole tarvetta. Mikäli sitä haluttaisiin käyttää, tulisi lomakkeelle määritellä tarkemmin Ext JS -malliin konvertointi toiseen muotoon.

Konvertointifunktion käyttäminen tuntuu siinä mielessä tarpeettomalta, että tallennuksen yhteydessä koordinaattikentät tulee joka tapauksessa käsitellä säännöllisillä lausekkeilla. Konvertointifunktio tuli kuitenkin ottaa käyttöön koordinaattikentille, koska muutoin kenttien merkkijonot olisivat tyhjiä, mikä aiheuttaa ongelman, kun tietoja tallennetaan. Nyt konvertointifunktio yksinkertaisesti asettaa tyhjän merkkijonon vain null-arvoon.

Tein toisen koordinaattikentän toimivaksi. Jouduin samalla muuttamaan koodia niin, että käsittely tehdään paremmin erillisissä funktiossa saman tallennusfunktion sisällä – näin yksittäinen funktio on selkeämpi lukea. Huomasin myös, että tietyssä muodossa olevat koordinaatit aiheuttivat yhä ongelmia, joten muutin koordinaattien säännöllisen lausekkeen käsittelyä siten, että vain tietyt merkit sallitaan (numerot, pisteet, sulut ja välimerkit). Muussa tapauksessa annettu koordinaatti hylätään ja tallennus keskeytetään ilmoitukseen, että koordinaatit tulee antaa oikeassa muodossa ”x, y, z”.

Keskustelin tänään tuotteen omistajan kanssa toiminnallisuuden tilasta ja esittelin toimintaa paikallisen ympäristöni kautta. Esille tuli muutamia parannusehdotuksia muun muassa lomakkeen asettelun suhteen ja väärin määritellyn täydennyksen suhteen kannen korkeusaseman kanssa. Huomasin samalla pari bugia, mutta pääosin nykytilaan oltiin tyytyväisiä. Tavoite olisi saattaa toiminnon työpöytätila ensi viikolla käyttöön integraatiopalvelimen kautta koostettuun ympäristöön. Mukaan tuli myös pari uutta toimintoa yhä tehtäväksi. Kaivot pitäisi pystyä paikantamaan suoraan, kuten monissa muissa lomakkeissa on mahdollista (nyt tarjolla on vain korostustoiminto). Lisäksi tutkimukseen tulisi lisätä nappi, josta alkuperäisen kaivolomakkeen pystyy aukaisemaan. Eräs toivomus oli määritellä myös värit sen mukaan, miten vakavia eri viat ovat, mutta tämän toteutus jää myöhempään ajankohtaan.

Keskustelin yhden työkaverin kanssa katselmoinnissa havaitusta asiasta liittyen Django-modelien `null=True` ja `blank=False` määrittelyihin. Olin käyttänyt näitä muutamille kentille, jotka on määritelty pakollisiksi, jotta lomake voidaan täydentää. Kuitenkin olisi hyvä olla sallimatta null-arvoja, jos oletuksena halutaan, että käyttäjän tulisi syöttää nämä arvot. Keskustelin myös toisen työkaverin kanssa opinnäytetyön päiväkirjasta ja kerroin hänelle oman prosessini kulusta.

Perjantai 7.9.2018

Tänään oli tarkoitus korjata loput kaivotutkimuksesta löydetyt bugit ja viimeistellä toiminnallisuutta niin, että toiminto olisi mahdollista ottaa ensi viikolla osaksi pää-

haaraa. Teinkin lukuisia pieniä korjauksia, joista toiminnon omistaja oli eilen huomauttanut. Siirsin korkeusasema-kentän lähemmäksi siihen viittaavaa kenttää, jotta käyttäjä ei sekoittaisi sitä samankaltaiseen kaivon korkotietoon. Poistin automaattisen täytön kyseiseltä kentältä, koska se oli tarpeeton ja ilmeisesti viittasi juuri väärään tietoonkin. Asetin koordinaattikentille pyöristyksen kahden desimaalin tarkkuuteen, kun tietue haetaan järjestelmästä. Data siis tallennetaan yhä tarkemmassa mittakaavassa kuin mitä se esitetään käyttäjälle lomakkeella.

Korjasin bugin, joka liittyi tutkimuksen tekemiseen suoraan tutkimuslistauksesta. Ongelma johtui yksinkertaisesti siitä, että uuden tutkimuksen luontikoodista puuttui aiemmin tekemäni korjaus koordinaattitietojen käsittelyyn, minkä olin tehnyt jo aiemmin osaksi aiemman tutkimuksen tallentamista. Koordinaattien esitys asetti myös yhden ongelman, sillä käyttäjän olisi periaatteessa mahdollista etsiä tallennettuja tutkimuksia koordinaattien perusteella. Tässä ei kuitenkaan olisi käytännössä järkeä, sillä tutkimukset eivät itsessään sijaitse missään tietyssä pisteessä, vaikka sisältävätkin tiedon kaivojen sijainnista. Koordinaattihaku tuli siis ottaa pois käytöstä, mihin löytyi ratkaisu Djangorestframeworkista. Näkymille on mahdollista asettaa attribuutti, mikä sulkee pois kyseisen kentän lomakkeen hakuarvoista. Asetin koordinaattikentät osaksi tätä poissulkemista ja nyt haku ei enää huomioinut koordinaatteja haun yhteydessä. Samalla poistin kaivon kytkemisen tilaan liittyvän attribuutin pois käytöstä, sillä se vaikutti joissakin harvoissa tapauksissa aiheuttavan myös virheilmoituksia hakua tehdessä.

Kaivon kartalta kytkemiseen liittyvä bugi vaikuttikin johtuvan yksinkertaisesti siitä, että tietty karttataso ei ollut valintahetkellä aktiivisena. Käyttäjän tulisi siis manuaalisesti asettaa karttataso aina aktiiviseksi, ennen kuin kytkemistä on mahdollista tehdä. Lähetin tuotteen omistajalle kysymyksen siitä, tulisiko nämä tarvittavat tasot aktivoida saman tien, kun käyttäjä valitsee työkalun kytkemistä varten vai jätetäänkö nykyinen tila yhä käyttöön.

Putkien latauksessa esiintyi harvakseltaan bugi, jota en pystynyt toistamaan. Harvinaisessa tilanteessa saattoi käydä niin, että putkitietojen lataamista kuvaava maski jäi päälle eikä lähtenyt pois edes lomaketta nollaamalla. Ongelma esiintyi minulle kerran, kun työpaikalla oli ollut joko ongelmia tietokantayhteyksien

kanssa tai olin samaan aikaan muuttanut paikallisesti koodia, mikä oli keskeyttänyt ohjelman suorituksen. Työkaverini oli kerran raportoinut saman, mutta hänkään ei pystynyt ongelmaa enää toistamaan. Kyseinen bugi jäi siis tällä kertaa korjaamatta, mutta on tiedossa jatkoa varten.

Asetin lopulta bugikorjaustiketin katselmointiin. Tikettiin oli kasaantunut aluksi pienistä vioista sangen kattava kokonaisuus. Jälkeenpäin ajatellen olisikin ollut jo järkevämpää jakaa osa tiketissä esiintyneistä vioista erillisiin tiketteihin. Pieniä parannusehdotuksia oli kasaantunut mukaan lisää sitä mukaa, kun olin jatkanut alkuperäisen tiketin työstämistä. Bugitiketin jälkeen pyysin työkaveria verifioimaan aiemmin tekemäni putkitietojen tiketin loppuun, sillä katselmointi oli sen osalta viimein valmis. Verifiointi menikin läpi nopeasti, koska työkaverini oli jo tehnyt mobiilinäkymää, jossa oli käytetty samaa toiminnallisuutta hyväksi.

Seuraavaksi tehtäväkseni aloitin viimeistelemään toiminnallisuutta, jotta pystyn tuomaan muutokset käytettäväksi nykyiseen päähaaraan ja sitä myöten integraatiopalvelimelle. Tätä varten kyselin työkaverilta, miten nykyisin kannattaisi ottaa huomioon toiminnot, jotka eivät näy kaikille asiakkaille. Nykyisin tähän oli kuulemma olemassa kolme eri tapaa, joissa yhdessä käytettiin ympäristökohtaisia asetuksia ja toisessa tarkistettiin vakion tila tietokannasta. Kolmas tapa oli uusi, josta en saanut tietoon tässä vaiheessa tarkempaa kuvausta työkaveriltani.

Päätin käyttää perinteistä menetelmää toiminnallisuuden tilan tarkistamiseen, sillä kaivotutkimus ei tule aluksi kuin vasta yhden asiakkaan käyttöön. Toiminto myös vaatii aktivoimisen tarkistamista useammassa paikassa ohjelmakoodia, koska tutkimukseen pääsee käsiksi kahdesta eri paikasta: suoraan päävalikosta ja aktiivisen viemärikaivon kautta. Näin ympäristökohtaisten asetusten käytön paremmaksi siitä syystä, että tämän tarkistuksen käyttö ei vaadi ylimääräistä tietokantakyselyä, koska asetusten tieto sijaitsee ohjelman suoritukseen sisältyvässä muistissa. Haittapuolena on se, että asetusten tarkistuksia täytyy lisätä useampiin paikkoihin ohjelmakoodissa kuin jos käytettäisiin tietokantaan pohjautuvaa kyselyä.

Opinnäytteen toteutuspuoli lähestyy loppuaan, joten ensi viikolla tavoitteeksi jää saattaa kaivotutkimus toimintaan integraatiopalvelimen koostamassa ympäristössä ja viedä siten toiminnallisuuden yksi vaihe päätökseen. Opinnäytteen toteutuksen ulkopuolelle jää toiminnon kehittymisen seuranta ja ylläpito.

Tiistai 11.9.2018

Tiistaipäivälle kertyi paljon tekemistä. Lisäsin tänään kaivotutkimuksen päätikettiin linkin tulevalle wikisivulle, johon kuvataan yleisesti toiminnallisuuden kuvaus. Korjasin kaivotutkimuksen katselmoinnissa havaittuja asioita. Yhdessä resurssitiedoston funktiossa oli määritelty palautus niin, että periaatteessa kentälle annettu arvo 0 tai 0.0 ei välttämättä palautunut oikein. Alkuperäinen ohjelmakoodi muistutti seuraavaa:

```
return instance.foo_bar if instance.foo_bar else None
```

Muutos oli helppo korjata käyttämällä seuraavantapaista funktiota:

```
return instance.foo_bar if instance.foo_bar is not None else None
```

Katselmoinnissa huomattiin myös, että Ext JS -mallien konvertointifunktiot saattaisivat olla globaaleja. Vastaavia funktioita on käytetty monilla muillakin lomakkeilla, joten päätin testata asian integraatiopalvelimen koostamassa ympäristössä. Totesin väittämän pitävän paikkansa. Lomakkeiden convert-funktiot olivat käytettävissä globaalisti ilman näkyvyysalueen määrittelyä, eli käyttäjän ei tarvinnut edes tietää, missä moduulissa funktiot sijaitsisivat.

En löytänyt tähän kovin hyviä korjausratkaisuja, jotka toimisivat nykyisessä toteutuksessa. Muutin koodia siten, että siirsin mallien muunnosfunktiot kokonaan omaan tiedostoon, joita kutsuttiin sitten mallitiedostossa erikseen kussakin tarvittavassa tapauksessa. Haittapuolena tässä on se, että koodi näyttää hieman tökeröltä, koska jokaisen funktion joutui tallentamaan muunnoksen yhteydessä väliaikaiseen muuttujaan, ennen kuin niitä pystyy käyttämään, esimerkiksi seuraavaan tapaan:

```
var converter = Ext.utils.converter.TypeConverter.stringToFloat;
return converter(value);
```

Kömpelyydestä pääsisi eroon käyttämällä ES6:n esittelemää import-lauseketta (Mozilla 2018b), jolla pystytään ottamaan toisista moduuleista funktioita käyttöön näppärämmin. Ikävä kyllä monet vanhemmat selaimet eivät tätä tue, joten sitä ei pystynyt tässä yhteydessä käyttämään.

Kommentoin päivän aikana työkaverin katselmointia, jossa oli esitetty muutokset mobiilinäkymän toimivuutta varten. Lisäsin maininnat puolustavan ohjelmointitavan käyttämisestä ja varhaisesta lopetuksesta (if-else niin, ettei else-haaraa välttämättä tarvita), muutamista kirjoitusvirheistä funktioiden nimissä sekä ympäristön asetusten tarkistamisesta. Jälkimmäisessä tapauksessa tuli esiin hyvä huomio, että alkuperäinen tapa oli parempi kuin esittelemäni tapa, jossa tarkistetaan suoraan asetusmuuttujien arvo. Syynä tähän oli se, että mobiilinäkymä on käytössä kaikissa tuotteissa. Mikäli mobiilinäkymässä vaadittaisiin suoraa viittausta asetuksiin, tarkoittaisi tämä sitä, että kaikissa mobiilinäkymää käyttävissä tuotteissa tulisi ottaa käyttöön asetuksen määrittely, vaikka tuote ei edes sisältäisi KeyAquaa. Näin ollen alkuperäinen menettely olikin parempi.

Päivän aikana kokeilin myös työkaverieni tekemiä tietokantaskriptejä. Sain kuitenkin huomata, että ne eivät toimineet täysin kaivotutkimuksen kanssa. Joidenkin kenttien tyyppimäärittely oli ollut väärin tietokantaskripteissä, mikä aiheutti virheilmoituksia kohteita tallennettaessa. Lisäksi useamman tekstivakion nimi oli jossakin vaiheessa muuttunut, mikä aiheutti virheilmoituksen lomaketta avattaessa. Toiminnallisuuden testaamiseksi päätin kokeilla ottaa pois käytöstä virheelliset kentät. Mahdollisesti tämän takia moni lomakkeen toiminnallisuus näytti kuitenkin jostain syystä hajoavan, toisin kuin yleensä tulisi käydä. Lomakkeen aukaisu saattoi jäädyttää lomakkeen käyttökelvottomaksi tai muuttaa asettelun eriskummalliseksi. Päätin, että kokeilen seuraavan kerran toimintoa, kun olen tehnyt tarvittavat muutokset.

Aloitin korjaavien tietokantaskriptien kirjoittamisen. Tein skriptit kokonaan puuttuville tekstivakioille, joita käytettiin putkitiedoissa. Olemassa oleville tekstivakioille

korjasin tarvittaessa nimet skripteissä sellaisiksi, kuin ne olivat olleet kehityksen loppuvaiheessa. Osasta tekstivakioista muutin nimet koodista, koska tietokantaan asennetut vakiot olivat näissä tapauksessa järkevämmän nimisiä. Muutin yhden tekstivakion yleiskäyttöiseksi, koska sitä käytettiin vain vaihtoehtojen ”Kyllä” tai ”Ei” osoittamiseen. Kyseistä vakioryhmää pystyisi varmasti hyödyntämään useammassakin tapauksessa. Päivän lopuksi laitoin nämä tietokantamuutokset vielä uuteen katselmointiin, jossa arvioitaisiin muutoksia, mitä tulee tehdä ennen kuin kaivotutkimuksen voi liittää päähaaraan. Päivittäisin kyseistä katselmointia vielä huomenna, koska kaikkia muutoksia en ehtinyt vielä tänään tekemään.

Keskiviikko 12.9.2018

Opinnäytetyön viimeisenä raportointipäivänä oli tavoitteena saattaa kaivotutkimus päähaaraan vietäväksi. Tässä en vielä tänään onnistunut, sillä vastaan tuli vielä muutamia uusia ongelmia.

Päivän aikana eräässä katselmoinnissa oli esitetty sama tilanne, josta eilen huomautettiin minulle: JavaScript-funktioista tulee globaaleja, mikäli niitä ei määritellä minkään toisen funktion tai luokan sisälle. Ilmoitin tästä katselmoinnin tekijälle ja ehdottamastani korjauksesta.

Lisäsin tänään koontipalvelimen asetuksiin attribuutin, jotta kaivotutkimus aktivoidaan automaattisesti päälle, kun ympäristö on päivitetty. Muutin nykyisestä ohjelmakoodista tuotteen aktivoinnin takaisin alkuperäiseen paikkaan, eli ennen kuin template-tiedostossa lisätään HTML-koodi päävalikkoon. Tein muutoksen siksi, koska aiemmin olisi ollut yhä mahdollista käyttää toimintoa selaimen konsolin kautta, vaikka toiminto olikin suljettu piiloon käyttöliittymästä. Nyt koko toimintoa ei ladata lainkaan käyttöön, mikäli asetusmuuttujaa ei ole määritelty.

Vertasin omaa tietokantaani integraatiopalvelimen kantaan tekemällä seuraavan kaltaisen kyselyn ja vertaamalla sen tuloksia:

```
select column_name, data_type, data_length
  from user_tab_columns
```

```
where table_name = 'TABLE_NAME'
order by column_name;
```

Kysely paljasti, että yhteen kenttään oli lipsahtanut Liquibase-skripteissä nurinpäin koordinaattikentän ja korkokentän tietotyypit, minkä takia uusien tietueiden luominen ei ollut onnistunut. Otin nämä rikkinäiset kentät tilapäisesti pois käytöstä, jotta pystyin testaamaan kaivotutkimusta koontipalvelimen kannalla. Sain samalla huomata useampia pienempiä ongelmia. Lomakkeen kytkentäpainike kaivoon ei toiminut, jostain syystä koodi ei koskaan näyttänyt käyttäjälle vahvistusta, mutta mitään virheilmoitustakaan ei tullut esiin. Samalla tämä johti lomakkeen toimimattomuuteen ennen kuin sovellus käynnistettiin uudelleen. Myös viemärilomakkeen tutkimuslistanäkymän lataaminen rikkoi heti näkyvät Ext JS -lomakkeet.

Uuden tutkimuksen luonti, haku ja tallentaminen kuitenkin toimivat oikein. Tutkimuksen sai liitettyä kaivoon kartalta valintatyökalulla, mutta näytti siltä, että tämä asetti kaivoon väärän sisältötyypin (content type). Huomasin ongelman siitä, että kaivon esitäyttö ilmoitti, ettei kytkettyä kaivoa löytynyt, vaikka yhteys oli juuri luotu. Ilmeisesti työkalun valitsemiskoodissa oli jäänyt väälle viittaus viemärivarusteisiin, joka oli muualla muutettu viemärikaivoksi. Tämän muuttaminen näytti korjaavan ongelman valintatyökalussa.

Iltapäivän päätteeksi olin ajamassa Liquibase-muutoksia omaan kantaani. Jouduin aluksi poistamaan aiemmin luodut tietokantaobjektit kaivotutkimusta varten, jotta pystyin ajamaan Liquibase-muutokset. Ongelmana näytti tulevan vastaan siinä, että valtaosa tekstivakioista piti poistaa vielä erikseen. Onneksi tämä onnistui suhteellisen helposti käyttämällä poistossa LIKE-määrettä, sillä kaikki toiminnallisuuteen liittyvät vakiot käyttivät samaa alkutermiä.

Kun olin poistanut kaikki aiemmin luodut objektit ja tekstivakiot, näytti Liquibasen päivitys kuitenkin kestävän huomattavasti tavallista pidempään ja lopulta jumiutuvan tiettyyn pisteeseen. Lokitiedosto ei kasvanut ja mitään virheilmoitusta ei tullut. Kesti useampi tovi, ennen kuin huomasin syyn: tietokannan lukot olivat jääneet osittain päälle tehdessäni muutoksia kantaan SQL Developerin kautta. Tästä syystä Liquibase ei pystynyt tekemään muutoksia. En huomannut tätä

aluksi lokitiedostosta, vaan vasta sitten, kun yritin lisätä yksittäisiä muutoksia. Lukkojen aukaisun jälkeen Liquibase suorittikin päivityksen nopeasti. Liquibase-päivityksen hitauden vuoksi meni paljon työaikaa, joten en ennättänyt vielä tänäänkään liittämään kaivotutkimusta päähaaraan. Liittäminen jäänee siis tällä kertaa opinnäytetyön toteutuksen seurannan ulkopuolelle.

4.13 Seurantajaksojen 9 ja 10 huomiot

Varteenotettava huomio viimeiseltä seurantajaksolta tuli vastaan SQL- ja Liquibase-skriptien kirjoitusprosessissa. Aluksi suunnittelin kaivotutkimuksen tietomallin perustan SQL-skriptien avulla. Käytössäni oli alkuvaiheessa myös valmiina skriptit alasvetovalikkojen tietojen lisäämiseksi. SQL-skripteissä etuna on se, että ne ovat nopeasti kirjoitettavissa ja muokattavissa. Liquibasen sopii mielestäni paremmin lopullisen tietomallin luomiseen, kun tiedossa ei enää ole varsinaisia muutoksia. Syy on yksinkertaisesti Liquibasen versionhallinnassa: jos skripteihin tehdään muutoksia ja kehittäjän tietokanta halutaan päivittämään, joutuu kehittäjä poistamaan jo ajetut muutokset ja sen jälkeen tekemään ne uudelleen. Varsinkin protoilussa tämä ei toimisi ja ainoastaan hidastaisi työtä.

Haittapuolena tässä kehitystavassa on kuitenkin tarve kirjoittaa skriptit kahteen kertaan ja varmistaa, että Liquibase-skriptit vastaavat SQL-skriptejä. Kaivotutkimuksen toteutuksessa osa Liquibase-skripteistä ei vastannut SQL-skriptejä. Skripteissä oli virheitä myös siitä syystä, että SQL-skripteihin ja ohjelman toimintaan oli tehty vielä pieniä muutoksia sen jälkeen, kun alkuperäiset Liquibase-skriptit tekstivakioiden lisäämiseksi oli jo kirjoitettu. Alkuperäisissä Liquibase-skripteissä osa tekstivakioista käyttikin väärää ryhmänimeä, mikä korjattiin toteutuksen aikana.

Kehitystyötä ajatellen järkevintä olisikin tehdä ehkä kaksi tietokantaa isompia toiminnallisuuksia varten. Toiseen kantaan muutokset tehdään perinteisesti SQL-skripteinä, jolla varmistetaan ratkaisun toimivuus sovelluksessa. Sen jälkeen kirjoitetaan Liquibase-skriptit SQL-skriptien pohjalta ja ajetaan ne toiseen kantaan. Näin vältytään mahdollisilta päällekkäisyyksiltä, jos osa SQL-skriptien koodista on edelleen jäänyt kehittäjän kantaan.

Kaivotutkimuksen tekoprosessi jättikin lopulta mieleen monia huomioita. Osa työtehtävistä olisi pitänyt määritellä alusta asti tarkemmin, koska työkaverini toimintatapa erosi omastani. Osa tehtäviin liittyvistä tarkistettavista asioista jäi huomioimatta, mikä johti puuttuviin piirteisiin ja uusiin bugikorjauksiin.

Kaivotutkimuksen tarkempi määrittely olisi helpottanut kehitystä. Jatkuvaa integrointia olisi voinut hyödyntää enemmän, jolloin tuotteen omistaja olisi nähnyt jo aiemmin, miten toteutus toimii. Silloin olisi syntynyt vähemmän tarvetta olemassa olevien toimintojen uudelleenkirjoittamiselle. Toteutus muistuttikin loppujen lopuksi perinteistä vesiputousmallia ja johti siihen, että ohjelmaan tuli loppuvaiheessa vielä paljon lisää muutoksia, mikä viivästytti entisestään testaustyön aloittamista. Olisikin tärkeää myös muistaa, milloin ohjelmisto on tarpeeksi hyvä: missä vaiheessa uusien toimintojen tekeminen lopetetaan ja todetaan nykyisen ratkaisun olevan jo riittävän hyvä (Hunt & Thomas 2015, 9–10).

5 Pohdinta

5.1 Osaamisen kehittyminen tarkastelujakson aikana

Ennen opinnäytetyön tarkastelujakson alkamista olin laatinut kuvauksen työtehtävieni vaatimustasosta ja omasta osaamisestani luvuissa 2.5 ja 2.6. Lisäksi määrittelin luvussa 2.7 taitotasoni Dreyfusin taitotason mallin perusteella. Koin tuolloin olevani Dreyfusin määrittelemistä taitotasoista noviisin ja kyvykkään toimijan välillä. (Dreyfus 2004, 77.) Kirjoitin tuolloin haluavani kehittää yleisesti osaamistani ohjelmointikielissä (sovelluskehyksistä) ja virheenkorjauksessa. Ennen tarkastelujaksoa olin epävarma omasta osaamisestani ja kyvyistäni hallita näin isoa kokonaisuutta. Olin työskennellyt aiemmin aikavaatimuksiltaan pienempien työtehtävien parissa.

Tarkastelujakson aikana sain runsaasti lisää varmuutta omaan osaamiseeni ohjelmistokehittäjänä. Minulta kysyttiin neuvoja erilaisiin ongelmiin ja pystyin antamaan katselmoinneissa paljon palautetta toisten ohjelmistokehittäjien tekemiin

ratkaisuihin. Tarkastelujakson aikana en juurikaan ollut mukana tekemässä virheenkorjauksia, minkä olin määritellyt yhdeksi tarkasteltavaksi kehittymiskohteeksi. Valtaosa työajastani kului opinnäytetyön toteutuksen aikana kaivotutkimuksen kehittämiseen. Tähän yksittäiseen toiminnallisuuteen paneutuminen kehitti osaamistani erityisesti JavaScriptin osalta. Opin lisää Ext JS -sovelluskehyksen piirteistä, mikä helpotti uusien lomakkeiden työstämistä.

Toteutettu toiminnallisuus oli varsin laaja ja olin pääasiallisesti vastuussa toiminnallisuuden kehittämisestä. Kysyin tarvittaessa neuvoja ja apua muilta työntekijöiltä. Olin melko tyytyväinen lopputulokseen ja asiakastapaamisessa sain positiivista palautetta myös asiakkaalta, vaikka toteutukseen kuluikin aikaa enemmän kuin olin aluksi ajatellut. Asiakastapaaminen oli 2.10., eli tarkastelujakson jälkeen.

Tarkastelujakson aikana opin erityisesti tehtävien pienempiin osiin pilkkomisesta, ketterästä kehityksestä, koodin täydennyksestä ja uuden sovelluskehittimen käyttöönotosta. Kesällä siirryimme JIRAn käyttöön ja viikon mittaisiin sprintteihin, mikä pakotti minut tarkastelemaan työtehtäviin liittyvän työpanoksen haastavuutta uudella tavalla ja pilkkomaan työtehtäviä pienempiin osiin. Yksittäinen työtehtävä on toteutettava viikossa uudessa projektimallissa. Jakson aikana yhdessä kehittämistehtävässäni asetin tavoitteeksi ottaa käyttöön PyCharmin ohjelmistokehityksessä. Opin käyttämään PyCharmissa paremmin oikoteitä, mikä nopeuttaa omaa työskentelyäni. Omaksuin PyCharmin myötä Live Template -mallien käytön, joiden avulla ohjelmoija voi luoda halutun mukaisen koodirungon. Koodin täydennys toimi varsinkin paremmin PyCharmissa kuin Eclipsessä, jota käytin aikaisemmin.

Koen olevani tällä hetkellä kolmannella taitotasolla, eli kyvykkäänä toimijana. Osaan nykyään hahmottaa melko hyvin, mitkä osa-alueet työtehtävissä ovat tärkeämpiä kuin muut. Toisin kuin aloitteleva toimija -vaiheessa, en työskentele enää täysin pelkkien esimerkkien ja ohjausten mukaisesti, vaan teen päätöksiä omaan harkintaani pohjautuen. Päätösten lopputulemasta olen kuitenkin vielä päätöksen tekohetkellä epävarma ja päätöksien lopputulokset vaihtelevat.

Osasta saan onnistumisen kokemuksia ja osan päätösten epäonnistuminen aiheuttaa minulle turhautumista. Kuitenkin nämä epäonnistumiset kehittävät minua toimimaan paremmin tulevaisuudessa. Kyvykkään toimijan taitotasoon kuuluvat erilaiset tunnekokemukset työskentelystä ja itse koinkin tunteita laidasta laitaan. Tunteiden sietäminen ja päätösten tekeminen ovat välttämättömiä, jotta ihminen voi kehittyä edelleen. (Dreyfus 2004, 177–178.)

5.2 Eettisyys ja luotettavuus

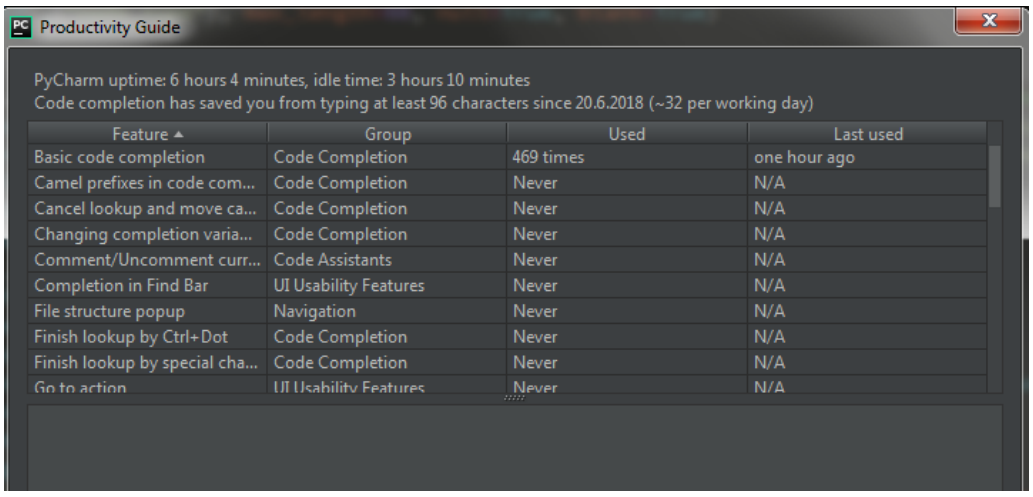
Eettiseen toimintaan kuuluvat monet eri seikat kuten rehellisyys, yleinen huolellisuus, tarkkuus ja ihmisarvon kunnioittaminen (Hirsjärvi, Remes & Sajavaara 2009, 24–25). Opinnäytteen toteuttamisesta laadittiin työnantajan Keypro Oy kanssa toimeksiantosuunnitelma, jossa otettiin huomioon Karelia-ammattikorkeakoulun yleiset opinnäytetyön käytänteet. Sopimukseen kirjattiin työnantajan toiveet, että työnantajan aineetonta omaisuutta (lähdekoodi) ei esitetä opinnäytetyössä. Esimerkiksi mikäli analyysin perusteella löytyneet paremmat toimintatavat ja menetelmät liittyvät suoraan työnantajan ohjelmistoon, pyritään nämä ratkaisut abstrahoimaan raportissa. Olen kunnioittanut prosessissa toimeksiantosuunnitelmassa esitettyjä toiveita ja asiakkaani anonymiteettia. Työnantajani on myös tarkistanut opinnäytetyöni sisällön ja hyväksynyt sen.

Tutkimuksen luotettavuutta parantaa tutkijan tarkka kuvaus hänen toiminnastaan ja jokaisesta tutkimuksen vaiheesta. Toiminnan olosuhteet tulisi kirjata raporttiin totuudenmukaisesti ja selkeästi. Virhetulkinnat, käytetty aika ja tekijän itsearviointi omasta työskentelystä on kerrottava raportissa. (Hirsjärvi ym. 2009, 232.) Kirjoitin tarkastelujakson aikana vihkoon jokaisena päivänä tavoitteeni työskentelylleni ja kokemukseni kyseisestä päivästä. Myöhemmin kokosin raporttiin yhteenvedon ja analyysin kokemuksistani. Pyrin toimimaan koko prosessin ajan mahdollisimman rehellisesti ja kirjaamaan totuudenmukaisesti työskentelyni olosuhteet. Raporttiin olen kirjoittanut selkeästi toimintani, mutta sisältö on suunnattu enemmän ohjelmistoalaan perehtyneelle kuin kelle tahansa kohderyhmään kuulumattomalle lukijalle. Toteutusta tarkastellessa on huomioitava, että tulokset ovat omiin kokemuksiin pohjautuvia, eikä niitä voida arvioida objektiivisesti.

5.3 Uudet menetelmät ja ratkaisumallit

Käsitykseni sovelluskehittimien tarjoamista apuvälineistä kehitystyön helpottamiseen laajentui opinnäytetyön toteutuksen aikana. Opin käyttämään paremmin sovelluskehittimistä löytyviä toimintoja, jotka nopeuttavat ohjelmakoodin kirjoittamista ja vähentävät virheitä. Erilaisille toiminnoille löytyy usein näppäinoikoteitä, joita en ole aiemmin niin paljoa käyttänyt. Erityisesti ohjelmakoodin automaattinen täydentäminen itse luoduilla pohjilla tuli esiin hyödyllisenä elementtinä silloin, kun tarvitsee työstää paljon samalla tavalla toimivaa koodia, kuten opinnäytetyön aikana tein Djangon REST-sovelluskehyksen kanssa.

Sain kuitenkin vasta pintaraapaisun siitä, mitä sovelluskehittimien toimintoja pystyy käytännön työssä hyödyntämään. Aionkin jatkossa opiskella entistä enemmän sovelluskehittimien eri toimintojen käyttämistä työskentelyssä. Kuvassa 2 (otettu 21.9.2018) näkyy, mitä ohjelmiston tukemia ominaisuuksia sovelluskehittäjä on käyttänyt työnsä tehostamiseksi ja kuinka paljon aikaa koodin täydentäminen on säästänyt. Kuvasta 2 voi havaita, etten ole käyttänyt vielä moniakaan PyCharmin tukemia ominaisuuksia sovelluskehittämisen helpottamiseksi.



Kuva 2. PyCharmin tuottavuusopas.

Myös odotusajan mittaaminen osoitti, miten paljon aikaa voi kulua hukkaan pitkällä ajanjaksolla siihen, että hitaalla työasemalla pystyy työskentelemään. Tehokkuuden edistämiseksi olisikin tärkeää pystyä mittaamaan ohjelmistokehityksen suorituskykyä erilaisin mittarein. Pienimuotoisten tutkimusten tekeminen

prosessin parantamiseksi voisi olla hyödyllistä, jotta pystyttäisiin eliminoimaan paremmin työskentelyyn liittyviä hukkatekijöitä.

Opinnäytetyön toteutusaika osoitti myös, miten lyhyessäkin ajassa ohjelmistokehityksen menetelmät ja tavat voivat muuttua toisenlaisiksi. Siirryimme ohjelmistokehityksen osalta pidemmistä sprinteistä huomattavasti lyhyempiin ja otimme käyttöön lisää ketteriä menetelmiä. Raportointijakson aikana opin soveltamaan paremmin ketterien menetelmien periaatteita. Olin toki kuullut niistä ennen opinnäytetyötä, mutta ne eivät olleet aiemmin merkittävässä roolissa työskentelyssäni. Nyt työtehtävät suunnitellaan pääosin Scrumia soveltaen. Lyhyisiin 1–2 viikon sprintteihin määritetään ne tehtävät, jotka tehdään kyseisellä aikavälillä. Työ vaatii tehtävien pilkkomista yhä pienempiin osiin, koska viikko on varsin rajallinen aika yhtään isompien työtehtävien tekemiseen. Opin kirjoittamaan paremmin tikettejä, joiden tehtävät on jo valmiiksi määritelty hyvin pieniin osiin, jotta ne olisivat tehtävissä lyhyen aikataulun sisällä.

5.4 Toteutuksen haasteet

Alun perin tavoitteenani oli toteuttaa osaamisen kehittymisen tarkastelu pienempien työtehtävien osalta. Opinnäytetyön suunnitelmassani olin määritellyt nämä kehittymistehtävät kehitystyökalujen vaihtamiseksi, Docker-konttien ja Vagrant-ympäristöön tutustumiseen, ohjelmointikielien erityiseen syventymiseen ja automaattisen dokumentaation. Jo varhaisessa vaiheessa opinnäytetyön toteutusta sain kuitenkin laajan toimeksiannon yhdelle asiakkaalle. Kaivotutkimuksen toteuttaminen veikin käytännössä kaiken käytettävissä olevan työaikani, enkä pystynytkään tekemään muita opinnäytetyöhön suunnittelemiani kehittymistehtäviä. En ehtinyt tästä syystä käyttämään toteutuksen aikana PDCA-menetelmään liittyviä jatkuvia, iteratiivisia syklejä mukaan ottamassani kehittymistehtävässä.

Toinen haaste liittyi ajankäyttöön. Työtehtävien tekemiseen meni paljon aikaa enkä käytännössä pystynyt kirjoittamaan päiväkirjan muistiinpanoja saman päivän osalta puhtaaksi. Kirjasin työpäivän aikana esiin tulleet tehtävät ja huomiot

vihkoon, jotka kirjoitin pääasiassa viikonloppuisin puhtaaksi raporttiin. Tämän takia päivän aikana esiin tulleet huomiot eivät aina enää olleet tuoreessa muistissa silloin, kun kirjoitin ne koneella puhtaaksi raporttia varten.

Kolmas haaste liittyi itse päiväkirjamuotoiseen opinnäytetyöhön. Karelia-ammattikorkeakoulun osalta kyseessä oli varsin uusi toteutusmenetelmä opinnäytetyön tekemiseen. Tehdessäni opinnäytetyön suunnitelmaa ei aiheesta ollut saatavilla vielä paljoakaan tietoa. Tämän vuoksi oli paikoitellen epäselvää, miten opinnäytetyön toteutus tulisi suunnitella ja mitä lopullisessa opinnäytetyössä haluttaisiin tuoda esiin.

Jälkeenpäin ajatellen olisinkin ehkä sittenkin tehnyt mieluummin perinteisemmän opinnäytetyön, mikäli olisin saanut sopivan aiheen töihini liittyen. Perinteisen opinnäytetyön toteutuksen ja raportoinnin vaatimukset olisivat olleet etukäteen selvemmin tiedossa. Toisaalta perinteinen opinnäytetyö olisi vaatinut minua olemaan huomattavasti enemmän pois töistä, mikäli en olisi saanut sovitettua opinnäytetyön aihetta nykyiseen työhöni. Periaatteessa joistakin suunnitelmassa mukana olleista kehittymistehtävissä olisikin voinut ollut aihetta kokonaan oman opinnäytetyön aiheeksi. Kehittymistehtävät yhdessä muiden työtehtävien kanssa tekevätkin mielestäni rajaamisesta perinteistä opinnäytetyötä haastavampaa.

Näen myös hieman kyseenalaisena sen, että päiväkirjamuotoiseen opinnäytetyöhön sovelletaan toistaiseksi samankaltaista arviointia kuin opinnäytetöihin, jotka sisältävät perinteisen IMRD-tyyppisen rakenteen. IMRD-rakenteen puuttumisesta huolimatta opinnäytetyöllä pystytään kehittämään ja osoittamaan tekijän asiantuntijuuden kehittymistä hänen työtehtävissään, mikä on olennaisinta opinnäytetyön oppimisprosessissa (Leinonen 2012, 17). Mielestäni olisikin syytä pohtia, ovatko esimerkiksi erilliset kehittämistehtävät tarpeellisia ammattikorkeakoulun päiväkirjamaisessa opinnäytetyössä.

Toisaalta voi miettiä, voisiko opinnäytetyötä tehdä keskittymällä yhteen tai kahteen kehittymistehtävään muiden työtehtävien sijaan, mikäli näin on mahdollista järjestää työpaikan kannalta. Kuten aiemmin jo totesin, työpaikalta saamani muut

työtehtävät estivät minua keskittymästä toteutuksen aikana niihin kehittymistehtäviin, jotka olin määritellyt opinnäytetyön suunnitelmassa. Tekijän kannalta olisikin tärkeää sopia jo erikseen toimeksiantosopimuksessa, että työntekijän muut työtehtävät eivät tule syrjäyttämään suunniteltuja kehittymistehtäviä.

Neljäs haaste liittyi salassapitovelvollisuuteen. Koska lähdekoodia ei voitu esittää suoraan työssä, ei vaihtoehtoisia ratkaisumalleja voitu käytännössä esittää, mikäli ne liittyivät itse ohjelmakoodin rakenteeseen. Työn analysoinnista olisikin voinut löytää paljon enemmän toisenlaisia ratkaisumalleja kyseiselle toteutukselle, jos olisin pystynyt sisältämään käyttämääni lähdekoodia osana viikkoanalyysia.

5.5 Osaamisen kehittäminen jatkossa

Jatkossa aion syventyä tarkemmin erityisesti JavaScriptin ohjelmistokehyksiin ja työkalujen käyttöön (webpack, babel). Kaivotutkimuksen tekoprosessi osoitti, että en ollut valmis työstämään näin suurta kokonaisuutta täysin itsenäisesti. Syynä oli työn laajuuden lisäksi puutteet omassa JavaScriptin osaamisessa, sillä mobiilipuoli teetettiin kokonaan eri tavalla kuin mitä pääasiallinen web-käyttöliittymä. En ollut kirjoittanut mobiilipuolen käyttämällä sovelluskehyksellä aiemmin, joten minulla olisi kulunut paljon aikaa sen opiskelussa, ennen kuin olisin päässyt tekemään kunnolla kehitystyötä mobiililiittymän osalta.

Olen hankkinut paljon kirjallisuutta ohjelmistokehitykseen liittyen, mutta en ole ehtinyt vielä lukemaan montaakaan teosta syvällisesti opintojen ja töiden ohella. Valmistumiseni jälkeen aion käyttää jatkossa enemmän aikaa tämän kirjallisuuden lukemiseen. Haluan perehtyä tarkemmin nykyisin tuntemiini ohjelmointikieliin sekä opetella jatkossa uusia kieliä, mahdollisesti sellaisista ohjelmointiparadigmoista, joista minulla ei ole vielä paljoa kokemusta. Esimerkiksi funktionaaliseen ohjelmointiin olisi mielekästä tutustua syvällisemmin.

Tavoitteenani on kehittyä pikkuhiljaa neljännelle ja lopulta viidennelle taitotasolle eli taitavaksi toimijaksi ja asiantuntijaksi. Tavoitteeni saavuttaakseni minun pitää

saada lisää työkokemusta erilaisista työtehtävistä ja syventää osaamistani monipuolisesti. Asiantuntijuuden edistäminen auttaa minua näkemään yhä paremmin, millä valinnoilla ja päätöksillä pystyn saavuttamaan työtehtävieni tavoitteet. Kehitys vaatii minulta avoimuutta, uskallusta ja erityisesti aikaa. (Dreyfus 2014, 179.)

Lähteet

Aranda, M. 2017. What's the difference between JavaScript and ECMAScript? freeCodeCamp.org. https://medium.freecodecamp.org/whats-the-difference-between-javascript-and-ecmascript-cba48c73a2b5. 25.11.2018.

Arpaci-Dusseau, A. & Arpaci-Dusseau, R. Distributed Systems. Teoksessa Operating Systems: Three Easy Pieces. http://pages.cs.wisc.edu/~remzi/OSTEP/dist-intro.pdf. 25.11.2018.

Atlassian Corporation Plc. 2018. Jira | Issue & Project Tracking Software. https://www.atlassian.com/software/jira. 23.11.2018.

Brainwy Software Ltda. 2018. PyDev. http://www.pydev.org. 25.11.2018.

Carbonnelle, P. TOP IDE index. https://pypl.github.io/IDE.html. 26.11.2018.

Collins-Sussman, B. & Fitzpatrick, B. & Pilato, M. 2013. Version Control Basics. Teoksessa Version Control with Subversion. http://svnbook.red-bean.com/en/1.7/svn.basic.version-control-basics.html. 25.11.2018.

Crockford, D. 2018. Code Conventions for the JavaScript Programming Language. http://crockford.com/javascript/code.html. 26.11.2018.

Datical. 2018. Liquibase | Database Refactoring | Liquibase. https://www.liquibase.org. 23.11.2018.

Django Software Foundation. 2018a. The Web framework for perfectionists with deadlines. https://www.djangoproject.com. 25.11.2018.

Django Software Foundation. 2018b. Templates | Django Documentation. https://docs.djangoproject.com/en/2.1/topics/templates. 25.11.2018.

Django Software Foundation. 2018c. Models | Django Documentation. https://docs.djangoproject.com/en/2.1/topics/db/models. 23.11.2018.

Dreyfus, S. 2004. The Five-Stage Model of Adult Skill Acquisition. Bulletin of Science, Technology & Society 24 (3). Thousand Oaks: Sage Publications, 177–181.

Eclipse Foundation. 2018. Eclipse Desktop & Web IDEs. https://www.eclipse.org/ide. 25.11.2018.

Ecma International. 2017. The JSON Data Interchange Syntax. ECMA-404 2nd Edition. http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf. 25.11.2018.

Esterbook, Chuck. 2001. Using Mix-ins in Python. Linux Journal. https://www.linuxjournal.com/article/4540. 25.11.2018.

Haikala, I. & Mikkonen, T. 2011. Ohjelmistotuotannon käytännöt. Helsinki: Talentum Media Oy.

Hirsjärvi, S. & Remes, P. & Sajavaara, P. 2009. Tutki ja kirjoita. Helsinki: Tammi.

Hovi, A. 2004. SQL-opas. Jyväskylä: Docendo Finland Oy.

Hunt, T & Thomas, D. 2015. The Pragmatic Programmer. Boston: Addison-Wesley.

JetBrains s.r.o. 2018. PyCharm: the Python IDE for Professional Developers. https://www.jetbrains.com/pycharm/. 23.11.2018.

Keypro Oy. 2018a. Meistä. Keypro Oy. https://www.keypro.fi/fi/meista. 9.6.2018.

Keypro Oy. 2018b. Referenssit. Keypro Oy. https://www.keypro.fi/fi/referenssit. 9.6.2018.

Keypro Oy. 2018c. Keyaqua – Keypro. Keypro Oy. https://www.keypro.fi/fi/tuotteet/keyaqua. 13.12.2018.

Lagstedt, A. & Kotila, H. 2015. Päiväkirjamainen opinnäytetyö vauhdittaa valmistumista. Teoksessa Kotila-Mäki (toim.) 21 tapaa tehostaa korkeakouluopintoja. 153–154. Haaga-Helian julkaisut. http://urn.fi/URN:ISBN:978-952-6619-78-1. 24.11.2018.

Lang, J-P. 2014. Overview – Redmine. http://www.redmine.org/projects/redmine/wiki. 23.11.2018.

Leinonen, R. 2012. Ammattikorkeakoulupedagogiikan kehittäminen. Opiskeluorientaatiot ja opinnäytetyön vertaistilanteet opiskelijoiden asiantuntijuuden kehittymisen tukena. Oulun yliopisto. Kasvatustieteiden tiedekunta. Väitöskirja. http://jultika.oulu.fi/files/isbn9789514298448.pdf. 5.12.2018.

Lutz, M. 2013. Learning Python. Sebastopol: O'Reilly Media.

McConnell, S. 2015. Code Complete 2nd Edition. Redmond: Microsoft Press.

Mercurial. 2018. https://www.mercurial-scm.org/about. 26.11.2018.

Mozilla. 2018a. HTML: HyperText Markup Language. https://developer.mozilla.org/en-US/docs/Web/HTML. 25.11.2018.

Mozilla. 2018b. import – Javascript | MDN. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/import. 16.9.2018.

Oracle Corporation. 2018. Database Net Services Reference. https://docs.oracle.com/cd/B19306_01/network.102/b14213/tnsnames.htm. 23.11.2018.

Python Software Foundation. 2018. PEP 8 – Style Guide for Python Code. https://www.python.org/dev/peps/pep-0008. 26.11.2018.

Quality Knowhow Karjalainen Oy. 2018a. Lean Six Sigma DMAIC. Quality Knowhow Karjalainen Oy. http://www.sixsigma.fi/fi/six-sigma/dmaic. 9.6.2018.

Quality Knowhow Karjalainen Oy. 2018b. Lean Six Sigma. http://www.sixsigma.fi/fi/lean/yleinen/lean-ja-johtaminen. 10.6.2018.

Radigan, D. Why code reviews matter (and actually save time!). Atlassian Agile Couch. https://www.atlassian.com/agile/software-development/code-reviews. 10.6.2018.

Rehkoph, M. 2018. Agile Epics: Definition, Examples & Templates. Atlassian Agile Couch. https://www.atlassian.com/agile/project-management/epics. 5.12.2018.

Roihuvuo, J. 2017. Päiväkirjamuotoinen opinnäytetyö–Ohjeet opiskelijalle. Karelia-ammattikorkeakoulu. https://moodle.karelia.fi/mod/resource/view.php?id=75183. 25.11.2018.

Roser, C. 2016. The Key to Lean – Plan, Do, Check, Act! AllAboutLean.com. 19.4.2016. https://www.allaboutlean.com/pdca. 10.6.2018.

Sencha Inc. 2016. MVC Application Architecture. https://docs.sencha.com/extjs/4.2.6/#!/guide/application_architecture. 26.11.2018.

Sencha Inc. 2018. Sencha Ext JS. https://www.sencha.com/products/extjs. 25.11.2018.

Sorva, J. 2018. Ohjelmointi 1. https://plus.cs.hut.fi/o1/2018/wNN/glossary. 25.11.2018.

Vincent, W. Django Best Practices. William S. Vincent. https://wsvincent.com/django-models-best-practices. 25.11.2018.

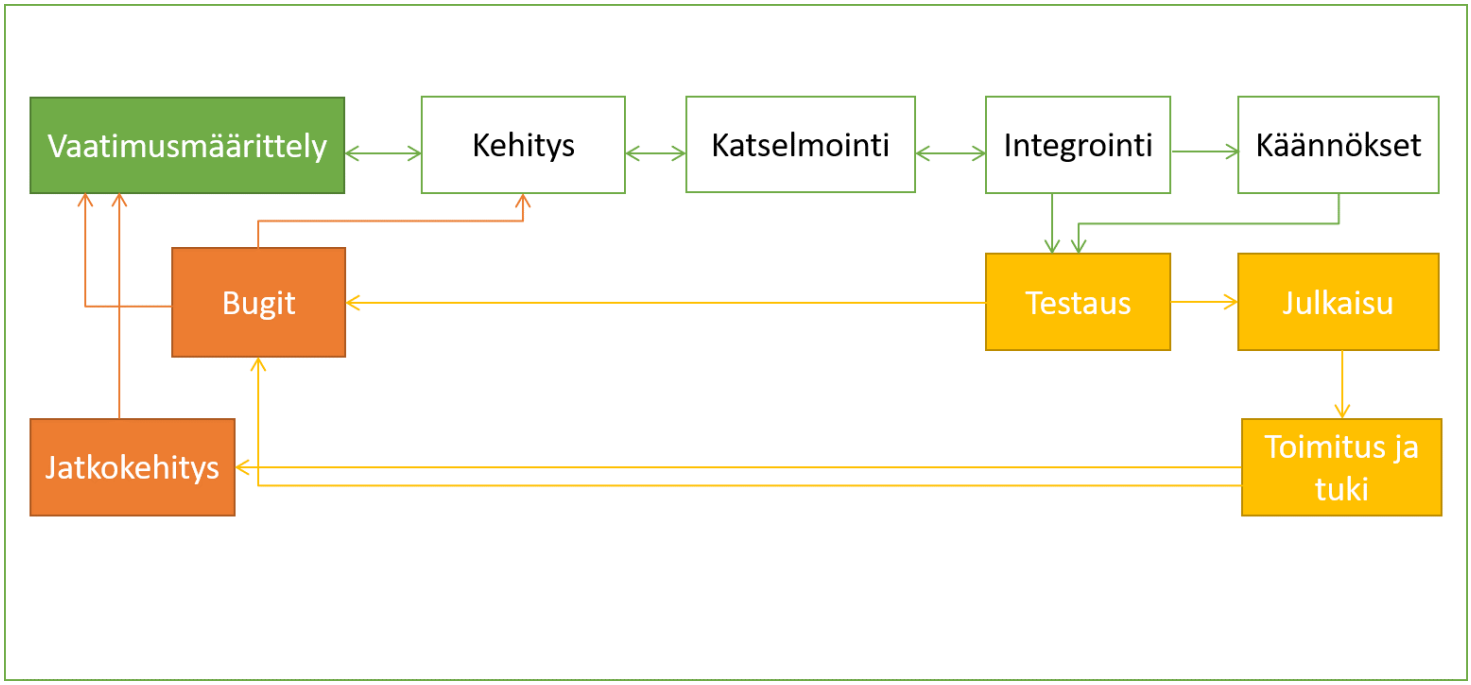
Wales, M. 2014. 3 Web Dev Careers Decoded: Front-End vs Back-End vs Full Stack. The Official Udacity Blog. https://blog.udacity.com/2014/12/front-end-vs-back-end-vs-full-stack-web-developers.html. 10.6.2018.

World Wide Web Concortium (W3C). 2004. Web Services Architecture. https://www.w3.org/TR/2004/NOTE-ws-arch-20040211. 23.11.2018.

World Wide Web Concortium (W3C). 2008. Extensible Markup Language (XML) 1.0 (Fifth Edition). https://www.w3.org/TR/REC-xml. 25.11.2018.

W3Schools.com. 2018. CSS Introduction. https://www.w3schools.com/Css/css_intro.asp. 23.11.2018.

Ohjelmistokehityksen prosessi tiivistetysti



A3-raportti sovelluskehityksen nopeuttamiseksi

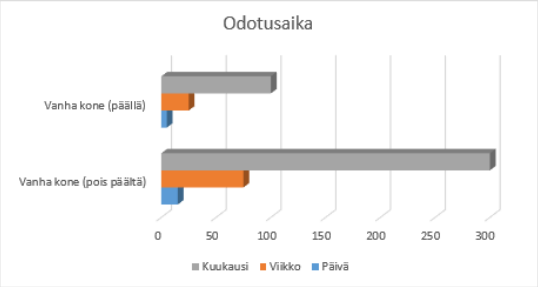
Aihe: Sovelluskehityksen nopeuttaminen

1. Taustatiedot - PLAN

- Käytössä on Eclipse-sovelluskehitin.
- Kehitystyö on hidasta.
- Nykyinen työkone on vanha.

2. Nykyinen tilanne - PLAN

- Eclipse ei integroidu kaikkeen Djangoon liittyvään koodiin eikä mahdollista esim. kaikkien ohjelmaluokkiin liittyvien koodien määrittelytiedostojen aukaisemista.
- Työaikaa kuluu paljon hukkaan tietokoneen käynnistämisessä, sovelluskehittimen toimintojen käytössä.



3. Tavoitteet- PLAN

- Ohjelmakoodin kirjoittaminen on nopeampaa ja tehokkaampaa. Ota käyttöön kevyempi sovelluskehitin ja opettele ohjelmistokehitystä nopeuttavia oikoteitä (näppäinoikotiet, koodipohjat).
- Työaikaa kuluu vähemmän "hukkaan" odottelun vuoksi.

4. Juurisyyn analyysi - PLAN

- Työkone on vanha ja heikkotehoinen. Toiminta hidastuu erityisesti iltapäivällä ja virtuaalikoneita käytettäessä.
- Kaikista ohjelman käyttöä helpottavista toiminnoista ei ole vielä kokemusta tai osaamista (näppäinoikotiet, koodipohjat).
- Yhden näytön käyttäminen vaatii jatkuvaa aktiivisen ikkunan vaihtamista, koska tarvittava tieto ei ole yhtä aikaa esillä.

5. Ratkaisuehdotusten kerääminen - PLAN

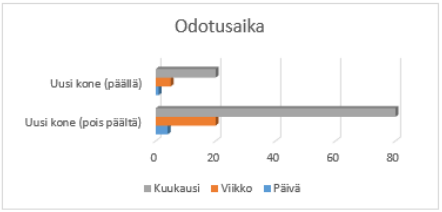
- Tietokoneen päivitys tai tehokkaamman tietokoneen hankinta
- Tietokoneen pitäminen päällä sammuttamisen sijaan
- Sovelluskehittimen vaihto toiseen, jota muut työntekijät ovat kehuneet (PyCharm)
- Sovelluskehittimen eri toimintojen syvällisempi opettelu (kustomoidut koodipohjat, näppäinoikotiet)
- Firefoxin kehitystyökalujen vaihto Chromeen
- Toisen näyttöpäätteen hankkiminen

6. Toteutus - DO

- Siirrytty PyCharmin käyttöön Eclipsestä.
- Opeteltu käyttämään enemmän sovelluskehittimen tarjoamia oikoteitä.
- Tehty Live Template -pohjia nopeuttamaan ohjelmointia.
- Siirrytty käyttämään Chromen kehitystyökaluja.
- IT-osastolta saatu vanha näyttöpääte kehitystyötä varten ja myöhemmin uusi työasema.

7. Ratkaisun varmistaminen – CHECK

- Chromen kehitystyökalut toimivat nopeammin ja ovat yhtä helpot käyttää kuin Firefoxin vastaavat.
- PyCharm tarjoaa enemmän näppäinoikoteitä ja paremmat työkalut ohjelmakoodin automaattiseen täydentämiseen / koodipohjien luomiseen.
- Uusi työasema toimii huomattavan paljon nopeammin (odotusaika minuuteissa), vaikka kone sammutettaisiin joka päivä.



8. Seuranta – ACT

- Uusi työasema vaatii paljon konfiguroimista, tarvittavat oikeudet puuttuvat. Ohjelmointiympäristön pystyttäminen vie vielä lisää aikaa, mikä estää kehitystyön tekemistä sillä välin.
- PyCharmin väripohja estää joitakin virheistä kertovia ilmoituksia näkymästä kunnolla. Aseta käyttöön väripohja, jolla ilmoitukset näkyvät selvemmin.

PVM ja versio: 22.11.2018