Haaga-Helia
ammattikorkeakoulu Oy

Python-kehittäjän päiväkirja

Jaakko Heikelä

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma
2017

Haaga-Helia
ammattikorkeakoulu Oy

Tiivistelmä

Päiväys

Tekijä(t)
Jaakko Heikelä

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma

Opinnäytetyön otsikko
Python-kehittäjän päiväkirja

Sivu- ja liitesivumäärä
69 + 4

Opinnäytetyön otsikko englanniksi
Diary of a Python Developer

Tässä päiväkirjatyyppisessä opinnäytetyössä kuvataan Python-kehittäjän työntekoa ja ammatillista kehittymistä kasvuyrityksessä 12 viikon ajanjaksolta. Ajanjakso käsittää aloitusviikon, 10 viikon seurantajakson sekä lopetusviikon.

Aloitusviikolla kuvataan kehittäjän nykyinen työtilanne, työympäristö, työtehtävät ja niissä vaaditut taidot. Lopetusviikolla tarkastellaan kehittäjän ammatillista kehittymistä seurantaviikkojen aikana ja verrataan lopetusviikon tilannetta aloitusviikon tilanteeseen.

Seurantajaksolla tehdään päivittäisiä merkintöjä työtehtävistä sekä niissä kohdatuista haasteista ja ongelmista. Työviikkojen päätteeksi viikon aikana tehdyt havainnot analysoidaan päiväkirjamerkintöjen pohjalta. Viikkoanalyyseissa nostetaan esiin kullekin viikolle sopiva teema ja arvioidaan kehittäjän suoriutumista keskittyen kyseisiin teemoihin. Analysoinnin tueksi tietoa haetaan ulkoisista lähteistä.

Seurantajakson aikana löydetään useita merkittäviä havaintoja kehittäjän työhön liittyen. Havaintoja tehdään niin työn ja työmenetelmien kehittymisestä, kuin analysoinnin tuomista hyödyistäkin.

Asiasanat
Ohjelmistokehitys, ongelmanratkonta, testaus, PostgreSQL, startup

Sisällys

1 Johdanto

Tämä opinnäytetyö on tarkoitettu kuvaamaan ammatillista kehittymistäni ohjelmistokehitystyötehtävissä. Tämä on päiväkirja-tyyppinen opinnäytetyö, jossa seurataan omaa oppimistani ja suoriutumistani arjen työelämässä kymmenen seurantaviikon ajan. Seurantaviikkojen aikana raportoin työssäni kohtaamistani haasteista ja tehtävistä sekä siitä, miten selviydyin niiden ratkaisemisesta ja mitä opin niistä. Raportointiviikot ajoittuvat ajalle 27.2.2017-5.5.2017.

Työpaikkani on Startup-yritys nimeltä 720 Degrees, joka tarjoaa asiakkailleen digitaalisia ratkaisuja sisäilman laadun reaaliaikaiseen tarkkailemiseen ja sen raportointiin. Työskentelen yrityksessä back end-kehittäjänä, eli toiselta nimeltä Python-kehittäjänä osana pientä ohjelmistokehitystiimiä. Itseni lisäksi tiimissä työskentelee projektipäällikkö, toinen back end-kehittäjä sekä kaksi front end-kehittäjää, joista toinen etänä. Muita tiimejä ovat mm. myynti- ja markkinointitiimi, analytiikkatiimi sekä tekninen tuki.

Työtehtäväni vaativat hyvin laajalti kaikkia yleisimpiä back end-kehittäjän taitoja kuten ohjelmointitaitoja, tietokannan suunnittelu- ja toteutustaitoja, versionhallintataitoja sekä kommunikointitaitoja. Kommunikointi englanniksi muiden tiimijäsenten ja tiimien kanssa on välttämätöntä, koska yritykseni on kansainvälinen ja koska kanssani työskentelee ihmisiä eri kansallisuuksista ja kulttuureista.

Muiden taitojen ohella on tärkeää osata hahmottaa laajojen ja monimutkaisten ohjelmistojen toimintaa työssäni. Python-kehittäjänä opin päivittäin uusia asioita eri ohjelmiston komponenteista ja niiden yhteistoiminnasta. Olen myös valmis hakemaan tietoa uusista tekniikoista ja komponenteista, jotka voisivat olla hyödyllisiä ohjelmiston kannalta. Käytämme pääasiassa avoimen lähdekoodin työvälineitä työssämme.

Työtehtävissäni tarvitaan tuntemusta ennen kaikkea Python-ohjelmointikielestä, PostgreSQL-tietokantahallintajärjestelmästä, Git-versionhallinnasta sekä Linux-käyttöjärjestelmästä. Muita avainsanoja jotka ovat tärkeitä mutta eivät yhtä isossa roolissa ovat muun muassa PyCharm-ohjelmointiympäristö, ketterä Scrum-menetelmä, Slack-viestintätyökalu sekä JIRA-tehtävänhallintatyökalu.

Koska Python-ohjelmointikieli on tehtävissäni pääroolissa ja koska tunnen sen perusteet jo hyvin ennen työni alkua, valitsin aiheeseen liittyen avukseni Luke Sneeringerin kirjoittaman teoksen Professional Python. Sneeringeriä (2015, 7) lainaten, tämä kirja on suun-

nattu kehittäjille jotka ovat jo työskennelleet Pythonilla, tuntevat sen hyvin ja haluavat oppia siitä lisää.

Toiseksi päälähteekseni valitsin Simon Riggsin ja Hannu Krosingin tekemän teoksen PostgreSQL 9 Admin Cookbookin, sillä työssäni tarvitsen myös paljon ymmärrystä PostgreSQL-tietokantatekniikasta. PostgreSQL on työni alkuvaiheessa vielä suhteellisen uusi asia, ja josta minulla on paljon opittavaa. Tämä teos tarjoaa opastusta PostgreSQL-tekniikan hallitsemiseen askel askeleelta. Riggsin ja Krosingin (2010, 36) mukaan PostgreSQL on monimutkainen järjestelmä, mutta jokainen matka alkaa aina ensimmäisellä askeleella.

2 Lähtötilanteen kuvaus

Tämä luku kuvaa omaa ammatillisen osaamiseni tilannetta ennen päiväkirjaraportointivaihettani. Luvussa kerrotaan myös sidosryhmistä ja vuorovaikutustilanteista työpaikallani.

2.1 Työtehtävät ja niissä vaadittavat taidot

Ennen oman päiväkirjaseurantani aloittamista olen jo kerennyt työskentelemään yrityksessäni 2 viikkoa kokoaikaisesti. Välttämättömät työkalujen käyttöopastukset sekä ohjelmistoon perehdyttämiset ovat jo suoritettu ja olen päässyt hyvään vauhtiin omissa varsinaisissa työtehtävissäni.

Työtehtävistäni suurimman osan aikaa vievät erilaisten kehitystehtävien toteuttaminen eli tyypillisesti niiden ohjelmointi. JIRA-projektinhallintatyökalun avulla pidetään yllä virtuaalista Scrum-taulua, jossa kehitystiimi organisoi tehtävänjakoa ja seuraa työnkulkua. Valitessani uutta kehitystehtävää tulee minun selvittää, mikä tai mitkä osat ohjelmistosta liittyvät kyseiseen kehitystehtävään. Saatan mahdollisesti neuvotella muun kehitystiimin kanssa tarkemmin haluttavista ohjelmiston muutoksista, kuten ominaisuuksista, parannelmista tai korjauksista.

Luettelo omista työtehtävistäni:

- Ohjelmointi
- Suunnittelutehtävät
- Koodin tarkastaminen yhdessä muun kehitystiimin kanssa
- Tietokannan hallintatehtävät
- Testaus

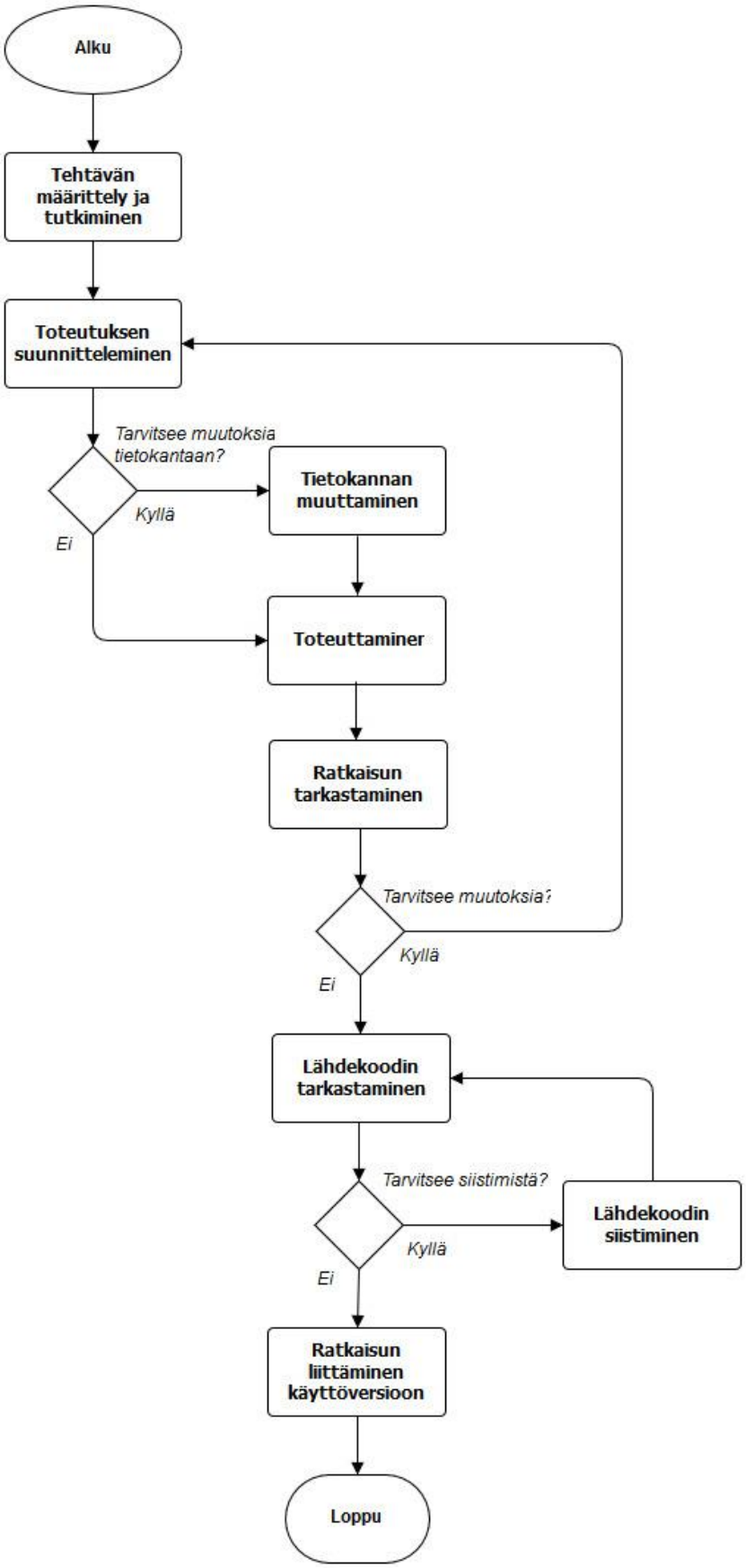
Kaikki luetellut työtehtäväni liittyvät suoraan toisiinsa ja ne voidaan siten ajatella osaksi suurempaa kokonaisuutta. Ennen kehitystehtävien toteuttamista, tulee minun yleensä neuvotella muun kehitystiimin kanssa tarkemmin seuraavaksi tekemästäni tehtävästä, paitsi jos kyseessä on pieni ja selkeä tehtävä. Tämä siksi, että se on osa suunnittelua muutoksen toteuttamista varten. Hyvä suunnittelu säästää turhalta työltä toteuttamisvaiheessa, mikäli toteutuksen valmistuessa huomataan ohjelmiston käyttäytyvän eri tavalla kuin alun perin haluttiin. Jos joudun tekemään muutoksia tietokantaan, nämä muutokset on hyvä suunnitella suunnitteluvaiheessa. Suunnittelupalaverilla on myös se hyvä puoli, että toteutusvaiheessa ei todennäköisemmin tarvitse keskeyttää muiden tiimin jäsenten työrauhaa kysyäkseen heidän mielipidettään johonkin tärkeään asiaan.

Kehitystehtävän suunnittelun jälkeen saatan joutua muuttamaan tietokannan rakennetta tavalla tai toisella. Mikäli tehtävässä on kyseessä uusi ominaisuus, jolla tallennetaan uudenlaista tietoa tietokantaan, tulee minun lisätä joko uusia tauluja tai uusia sarakkeita vanhoihin tietokannan tauluihin. Tämän osuuden kanssa minulla ei vielä ole täyttä itsevarmuutta, joten konsultoin aina kokeneempaa tiimin jäsentä tässä tehtävässä, mikäli muutoksia tietokantaan täytyy tehdä.

Toteutusvaiheen alussa luon Git-versionhallintatyökalulla omaa tehtävääni varten uuden branchin, eli haaran, jossa pystyn toteuttamaan omaa tehtävääni ilman, että toteutustyöni haittaisi ns. käyttövalmista versiota, tai potentiaalisesti toisen tiimin jäsenen samanaikaista ohjelmointityötä. Toteutusvaiheessa muokkaan ohjelmiston lähdekoodia siten, että suunniteltu tavoite täyttyisi mahdollisimman tarkasti. Tavoitteena kehitystehtävässä voi olla uuden ominaisuuden toteuttaminen, vanhan koodin paranteleminen, ohjelmavirheen eli bugin korjaaminen, tai muu tehtävä mikä ei sovi edellä mainittuihin kolmeen kategoriaan.

Testaamista tapahtuu koko ajan toteutusvaiheessa, kun toteutan tehtävää yksi askel kerrallaan. Saatan jokaisen askeleen välissä testata, toimiiko koodini, kuten oletan sen toimivan. Riittävän tiheällä testausvälillä näen, olenko menossa oikeaan suuntaan ratkaisussani. Pyrin myös tallentamaan muutokseni Gitillä aina askelten välissä, jotta voin myöhemmin palata aiempaan vaiheeseen, mikäli huomaan, että johdankin itseni harhaan seuraavalla askeleella ohjelmoidessa. Lopullinen testaus on kuitenkin tärkein vaihe, jotta voidaan minimoida myöhemmin ilmenevät ohjelmavirheet.

Lopuksi, kun olen mielestäni saanut tehtävän valmiiksi ja testannut sitä riittävästi, pyydän muita tiimin back end-puolen jäseniä kokoontumaan palaveriin ohjelmalogiikan tarkastamista varten. Tarkastuspalaverissa saatetaan vielä havaita puutteita tai muutoksen tarpeita ratkaisussani, jolloin palaan takaisin suunnitteluvaiheeseen. Tämä kuvio saattaa toistua montakin kertaa, mikäli valmiit tekemäni muutokset aiheuttavatkin uusia ongelmia ohjelman käyttäytymisessä. Lopulta, kun ratkaisuun ollaan tyytyväisiä, luon Gitillä uuden pull requestin omasta versiohaarastani, mikä tarkoittaa, että pyydän muuta tiimin kokeneempaa jäsentä ”vetämään”, eli liittämään oman toteutusratkaisuni osaksi ohjelman käyttövalmista versiota. Samalla kokeneempi tiimin jäsen tarkastaa lähdekoodini ja pyytää mahdollisesti siistimään tai selkeyttämään sitä. Muutoin hän hyväksyy kirjoittamani koodin ja liittää sen osaksi käyttövalmista ohjelman versiota. Oman tehtäväni tyypillinen eteneminen voidaan yleisesti kuvata oheisella prosessikaaviolla (Kaavio 1).



Kaavio 1. Tyypillinen tehtävän eteneminen prosessikaaviona esitettynä

Työtehtävissäni tarvitaan ennen kaikkea kykyjä loogiseen ongelmanratkontaan. Kehitystehtäväni alkuvaiheessa pitää osata lukea ja ymmärtää toisten ihmisten kirjoittamaa ohjelmakoodia, jotta kehitystehtävän taustat voidaan selvittää. Kehittäjän tulee selvittää, mitä kohtaa ohjelmistossa tulee muuttaa tai mihin kohtaan koodia tulee lisätä.

Suunnitteluvaiheessa kehittäjän pitää pystyä hahmottamaan käsiteltävä tieto luokkina ja olioina, koska se helpottaa suuresti tiedon tallentamista tietokantaan. Kehittäjän tulee hahmottaa lähdekoodin toimintalogiikka ja -järjestys.

Toteutusvaiheessa tarvitaan laajoja ohjelmointitaitoja. Pelkästään perusohjelmointitaidot eivät riitä, vaan kehittäjän pitää osata kirjoittaa koodia mahdollisimman siististi ja järjestelmällisesti. Kehittäjällä pitää olla kyky havaita tilanteet, joissa olemassa olevaa koodia voi mahdollisesti kierrättää, jotta ohjelmakoodin pituus ja suorituskyky pysyisi mahdollisimman optimaalisena.

Kehittäjältä odotetaan innokasta asennetta sekä tehokkaita oppimiskykyjä uusien teknologioiden oppimiseen ja käyttöönottoon. Kehittäjältä odotetaan kykyjä sekä halua löytää uusia ideoita kehitettävän ohjelmiston kehittämiseen.

2.2 Oman nykyisen työn analyysi

Yleisesti ottaen olen päässyt omissa työtehtävissäni erinomaisesti liikkeelle, eikä kehitystyössäni ole toistaiseksi ilmennyt suurempia ongelmia jotka olisivat jäädyttäneet työntekoani pidemmäksi aikaa. Ensimmäisellä työviikollani sain riittävän perehdytyksen kehitettävän ohjelmiston arkkitehtuuriin ja toimintaan. Vaikka en voikaan tietää kaikkia yksityiskohtia ohjelmiston eri osista, on minulla kohtalaisen selkeä käsitys koko järjestelmän kokonaiskuvasta.

Ensimmäisellä viikolla tarvitsin jonkin verran opastusta työympäristön asentamisessa sekä joidenkin perustoimintojen läpiviennissä, kuten tietokantaan yhdistämisessä. Tällä hetkellä olen jo melko itsevarma useimmissa työvaiheissa, enkä pääsääntöisesti tarvitse opastusta työtehtävieni suorittamisessa. Kun kohtaan työssäni jonkin ongelman, pyrin ensi sijassa itse hakemaan ongelmaan ratkaisua, jotten häiritse työtoverieni työrauhaa. Ratkaisun löytäminen saattaa välillä vaatia tiedonhakua internetistä. Tällöin useimmiten löydän ratkaisun lähteistä, joissa on kuvattu sama tai samanlainen ongelma, ja johon on jo löydetty ratkaisu tai eri ratkaisuja, joista jokin toimii oman ongelmani ratkonnassa. Toki toisinaan ratkaisua saattaa olla erittäin vaikeaa löytää. Kun tunnen olevani täysin jumissa ratkaisun löytämisessä, konsultoin kokeneempaa tiimitoveria, jolloin ratkaisu viimeistään löytyy.

Edellisen analyysin perusteella arvioisin itseni taitavaksi suoriutujaksi. Suoriudun työtehtävistäni itsenäisesti, tunnen työssäni käytettävät työkalut ja metodit. Silloin tällöin saatan tarvita apua joissakin asioissa, mutta vastapainoksi pystyn myös opastamaan minun kanssa samaan aikaan aloittanutta työtoveriani hänen ongelmissaan, kun ne liittyvät yhteisiin työvälineisiimme. Kun keksin ideoita tehtävieni läpiviemiseen ja ratkaisemiseen, ovat tiimitoverini yleensä samaa mieltä ratkaisuistani.

2.3 Kehittyminen

Oma ammatillinen kehittymiseni on saanut hyvän alustuksen koulun kursseilta ja työharjoittelusta. Koulussa opin työnteon perusteet, mutta työtehtäväni työharjoittelussa sekä nykyisessä työssäni ovat laajentaneet taitojani merkittävästi. Teen työtä paljon suuremmalla itsevarmuudella ja tunnen että pystyn oppimaan mitä vain nopeasti ja tehokkaasti. Ennen kaikkea, olen halukas oppimaan uutta sekä luomaan järkeviä kehitysideoita työntekoni kohteena oleviin tuotteisiin ja palveluihin. Pystyn hyödyntämään työharjoittelussa saamaani arvokasta työkokemusta nykyisessä työssäni.

Ennen päiväkirjaseurantani alkua olen jo oppinut käytettävien työkalujen perustoiminnot. Olen sisäistänyt itselleni uudet asiat nopeasti ja osaan jo käyttää suurinta osaa niistä ilman kokeneemman tiimitoverin opastusta.

Python-osaamiseni on jo tässä vaiheessa erittäin vahvaa, ja hallitsen syntaksin hyvin, koska olen hankkinut Pythonin perustaidot jo työharjoitteluni aikana. Uudet ohjelmointikielelliset asiat liittyvät lähinnä koodissa käytettäviin moduuleihin ja kirjastoihin. Myös Gitin käyttö sujuu ongelmitta ja ymmärrän ohjelmiston versiopuurakenteen hyvin.

PostgreSQL on minulle uusi teknologia, vaikka itse SQL-kieli onkin itselläni varsin hyvin hallussa. Tietokannan tarkastelussa ja muokkaamisessa käytetään PGAdmin-nimistä työkalua, PostgreSQL-tietokantaa varten jolla pystyn näkemään kaikki tietokannassa olevat taulut sekä luomaan uusia tauluja. Omien ohjelmointiratkaisujeni testaamisessa en käytä yrityksen palvelun käytössä olevaa varsinaista tietokantaa, vaan replikoin itselleni oman lokaalin tietokannan, jotta tallentamani testidata ei täyttäisi yrityksen palvelun oikeaa tietokantaa sinne kuulumattomalla datalla.

Toinen itselleni uusi työkalu on JIRA-projektinhallintatyökalu. JIRA:lla kehitystiimi pitää yllä virtuaalista Scrum-tehtävätaulua, jossa ohjelmiston kehittämistyö on jaettu tehtäviksi. Taulusta näkee, mitä tehtäviä ohjelmaan on suunnitteilla, mitä tehtäviä on parhaillaan työn alla, mitkä tehtävät vaativat tarkastusta ja mitkä tehtävät ovat valmiita. Sain ensimmäisenä

työpäivänäni pikaisen opastuksen JIRA:n käyttämisestä, jossa näytettiin, mitä yksityiskohtia uusista tehtävistä tulee kirjata tehtävän omalle sivulle. Minulle näytettiin, miten tehtäviä siirrellään eri vaihepalstalta toiselle, esim. suunnitteilla oleva tehtävä suunnittelu-palstalta työn alla-palstalle.

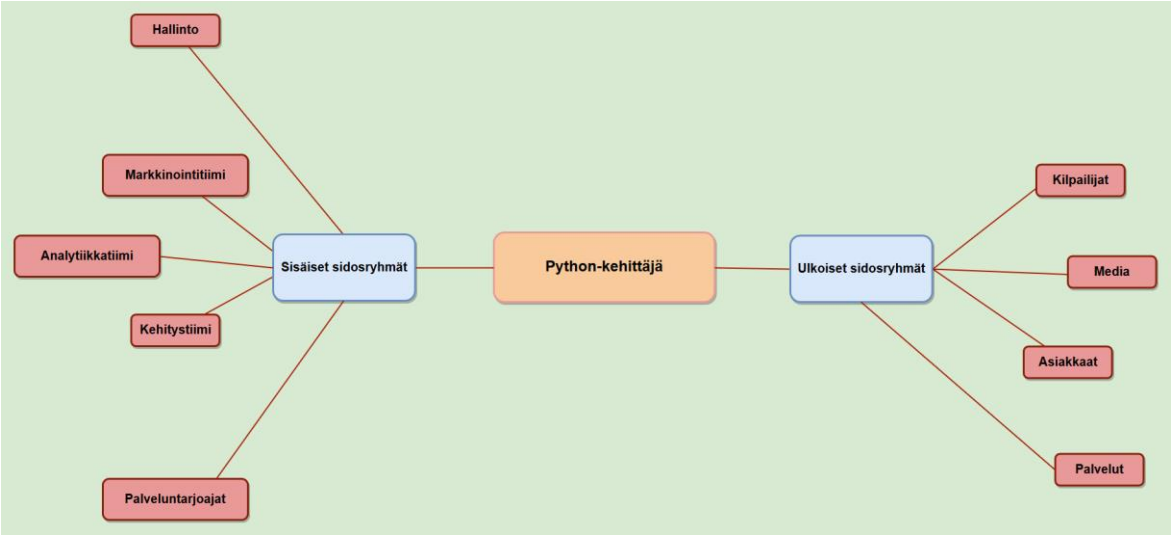
Ohjelmointityössä käytettävä työkalu on nimeltään PyCharm, jonka käytön opettelemisessa ei ollut itselleni juuri ongelmia. PyCharm on ulkoasultaan pitkälti kuten muutkin ohjelmointiin suunnatut tekstieditorit. Siinä on kuitenkin muutamia näppäriä pikatoimintoja, jotka helpottavat lähdekoodin tutkimista, kuten muuttujien määrittelyihin hyppääminen muuttujan nimeä painamalla tai tietyn funktion kutsujien haku.

Amazon Web Services-palvelusta minulla on jonkin verran kokemusta työharjoitteluni kautta, mutta en ole toistaiseksi joutunut tekemisiin sen kanssa uudessa työssäni. Kenties tulen käyttämään sitä myöhemmin.

Yleisesti ottaen minun kannattaa jatkossa panostaa suunnitteluun entistä enemmän, sillä olen toisinaan tehnyt suunnitteluvaiheen hieman hätiköiden. Tämä on näkynyt ohjelmointivaiheessa turhan monimutkaisena ja epäsiistinä koodina.

2.4 Sidosryhmät työpaikalla

Työlläni on vaikutusta useisiin sisäisiin ja ulkoisiin sidosryhmiin. Oleellisimmat sidosryhmät joihin työni vaikuttaa tavalla tai toisella voidaan kuvata oheisella kaaviolla (Kaavio 2).



Kaavio 2. Sidosryhmät

Sisäisiin sidosryhmiin kuuluvat hallinto, markkinointitiimi, analytiikkatiimi, oma kehitystiimini sekä palveluntarjoajat. Omassa kehitystiimissäni vaikutan suoraan tiimitoverieni työmäärään. Kun olen mukana työstämässä tehtäviä, työ jakautuu tiimin kesken tasaisemmin. Back end- ja front end-tiimit tekevät tiivistä yhteistyötä. Molempien tiimien työt vaikuttavat myös toisiinsa. Projektin johto päättää minulle sijoitettavista tehtävistä, riippuen omasta osaamisestani ja näytöistäni. Markkinointitiimin tulee tietää palveluun lisättävistä uusista ominaisuuksista ja muutoksista, jotta he tuntevat markkinoitavan palvelun. Analytiikkatiimi käyttää kehittämäämme ohjelmistoa apuna käyttämämme datan analysoimisessa. Palveluntarjoajilla viitataan muun muassa tuotannollisiin palveluihin, kuten tietokantapalvelimiin. Kun ohjelmistomme ja sen käyttämän datan varastointitarve kasvaa, tarvitsemme laajempia tuotannollisia palveluja.

Olennaisimpia ulkoisia sidosryhmiä ovat asiakkaat, media, palvelut ja kilpailijat. Asiakkaille tärkeintä on toimiva ja luotettava palvelu, joka tarjoaa oikeaa tietoa. Työlläni on suuri vaikutus siihen, millaista tietoa ja palvelua asiakkaille tarjotaan sekä miten heidän käyttämä palvelu toimii. Jos jokin osa ohjelmistostamme ei toimi asiakkaan puolella, tarkoittaa se yleensä kiireistä korjaustoimenpidettä kehitystiimissäni, ja mahdollisesti minun kohdallani. Palvelun toiminnallisuudella ja luotettavuudella on vaikutusta myös kilpailijoihin, jotka joutuvat tekemään enemmän töitä. Yrityksemme ja palvelumme ovat läsnä ennen kaikkea sosiaalisessa mediassa ja internetissä. Palvelut-sidosryhmällä viitataan yrityksiin ja palveluihin, jotka tarjoavat korjauspalveluja omalla palvelullamme havaittuihin sisäilmaongelmiin.

2.5 Vuorovaikutustaidot työpaikalla

Vuorovaikutus työssäni on tärkeää niin oman tiimini sisällä kuin tiimien välilläkin. Kehitystiimistäni kaikki yhtä kehittäjää lukuun ottamatta työskentelevät pääsääntöisesti samassa työtilassa, jolloin kommunikointi onnistuu suoraan suullisestikin, mikäli työtoverini ovat paikalla. Yksi kehittäjistämme työskentelee kuitenkin etänä, mistä johtuen pidämme päivittäisiä videopalavereita seurataksemme toistemme tilanteita ja voidaksemme kysyä toistemme kehitystöihin liittyvistä kysymyksistä. Kommunikointi tiimien välillä tapahtuu pääsääntöisesti Slack-viestintäohjelman avulla, vaikka tiimien työtilat sijaitsevatkin samassa rakennuksessa. Tärkeistä asioista ja tapahtumista tiedottamisissa koko yrityksen henkilöstö kootaan yhteiseen palaveriin.

Käytämme kehitystyössämme osittain sovellettua Scrum-metodia. Emme käytä Scrumia liian ”orjallisesti”, vaan ennemminkin pidämme sen toimintoja suuntaa antavina, johtuen osittain kehitystiimimme pienestä koosta. Pyrimme pitämään kehitystiimin sisällä päivittäi-

siä palavereja töidemme etenemisestä edellä mainitun videopalaverin lisäksi, jos kaikki tiimin jäsenet ovat paikalla. Kahden viikon välein pidämme Sprint-palaverin, jossa kerromme yrityksen muille tiimeille sprintin aikana toteutetuista ominaisuuksista ja muutoksista.

Toisinaan vuorovaikutustilanteissa saattaa olla ongelmana tärkeän henkilön poissaolo, mikäli hänen mielipiteensä jonkin tehtävän etenemisessä on tärkeä. Kun esimerkiksi projektin johdolla on muita kiireitä, saattaa oma kehitystehtäväni jäätyä joksikin aikaa. Näissä tilanteissa pyrin katsomaan muita tehtäviä, tai siistimään jo valmiiksi saamaani ratkaisua, mikäli mahdollista.

3 Päiväkirjaraportointi

3.1 Seurantaviikko 1

Maanantai 27.02.2017

Ensimmäisenä seurantapäivänäni työn alla ovat järjestelmän sisäiseen hallintaan tarkoitetut komentorivityökalut, joilla tietokannassa olevaa dataa voidaan lisätä, lukea, päivittää tai poistaa. Olen tähän mennessä saanut valmiiksi ohjelmakoodini kaikkiin toimintoihin, paitsi tietojen poisto-toimintoon. Tämän päivän tavoitteenani onkin poisto-toiminnon ohjelmoiminen loppuun. Tehtävässä ei tule todennäköisesti menemään koko päivää, joten yritän tehtävän valmistuttua luoda dokumentaatiota työkalua varten. Mainitsin kehitystiimini projektipäällikölle jo aiemmin Sphinx-teknologiasta, jolla voi generoida kätevästi siistejä dokumentaatioita käyttäen rst- eli ReStructured Text-tyyppistä tekstimuotoa. Sphinx on suunniteltu ennen kaikkea Python-ohjelmien dokumentointiin, joten yritän käyttää sitä apuna dokumentaation luomisessa.

Päivän työrytmiä sotkivat pitkiksi venyneet palaverit. Joka maanantai-aamu yrityksessämme pidetään perinteinen maanantai aamun kokous, jossa käymme läpi viime viikon tapahtumia sekä mitä opimme niistä. Tällä kertaa tapaamisen käyttötarkoitus kuitenkin muuttui hieman, johtuen ongelmista yrityksen infrastruktuurissa, joita aamun palaverissa pyrittiin yhdessä selvittämään. Iltapäivällä oli toinen palaveri liittyen yleiseen työhyvinvointiin.

Kaikesta huolimatta edistyin jonkin verran komentorivityökalun kehittämisessä, vaikken kerennytkään saamaan sitä valmiiksi päivän aikana. Haasteita tehtävässä riitti, johtuen tietokannasta luettavien taulujen liitostarpeista. Taulujen suhteet toisiinsa olivat melko monimutkaisia ja varsinkin datan poistamista varten tarvittavissa liitoksissa oli paljon haastetta. Tämä tehtävä olikin minulle kaikista haastavin tähän mennessä.

Oma ymmärrykseni tietokannan rakenteesta parani päivän aikana. Ohjelmoidessani ohjelmalogiikkaa, jouduin myös lisäämään uusia funktioita datan poisto-toimintoa varten. Tulin vahingossa lisänneeksi uudet funktiot eri Python-moduuleihin, eli lähdekooditiedostoihin. Koska tarvitsin näitä funktioita molemmissa moduuleissa, yhdistin funktiot toistensa moduuleihin import-lauseilla, jolloin tulin luoneeksi tilanteen, jossa kaksi moduulia ovat riippuvaisia toisistaan. Tämä aiheuttaa Python-koodin lukuvaiheessa niin sanotun deadlockin, eli umpikujan, jossa kumpikaan moduuli ei pysty alustamaan itseään loppuun, koska ne tarvitsevat toistensa resursseja samanaikaisesti. Opin tästä virheestäni paljon uutta.

Tiistai 28.02.2017

Tänään tavoitteenani on suorittaa komentorivityökalun toteuttaminen loppuun. Yritän saada työtehtävään liittyvät suunnitteluongelmat itse ratkaistua. Asetan itselleni kuitenkin aikarajaksi 2 tuntia, jonka jälkeen pyydän apua kokeneemmalta tiimitoveriltani, jotta en jää jumiin tehtäväni kanssa liian pitkäksi aikaa. Tehtävän valmistumisen jälkeen työtavoitteena olisi edelleen dokumentaation luominen samalle ohjelmalle.

Päivän lopussa olen viimeinkin saanut valmiiksi uudet ominaisuudet kaksi viikkoa työn alla olleeseen komentorivityökaluun. Sain jo aamupäivällä oman ratkaisuni valmiiksi viimeisimpään ominaisuuteen, eli tietokannan datan poistofunktioon. Ratkaisun tarkastuspalaverissa projektipäällikkö ei kuitenkaan ollut vielä täysin tyytyväinen ratkaisuuni, koska koodissa oli vielä paljon optimoitavaa sekä tietokantakyselyissä, että käyttöliittymässä.

Jatkoin iltapäivällä ratkaisuni parantelemista muun muassa yhdistämällä kaksi tietokantakyselyä yhteen kyselyyn käyttämällä tietokantataulujen liitosta, eli Join-käskyä. Iltapäivällä pidimme toisen tarkastuspalaverin, jonka totesimme paljon paremmaksi, mutta koodissa oli vielä muutamia yksityiskohtia, jotka kaipasivat parannusta. Yhdistin muun muassa 4 eri funktiokutsua, koska pystyin yhdistämään niihin sijoitetut argumentit samaan kutsuun.

Pian viimeisten korjausteni jälkeen tein versionhallintajärjestelmäämme pyynnön ratkaisuni liittämisestä sisempään haaraan, jonka projektipäällikkö hyväksyi. Samalla siirsin tehtävää Scrum-taulullamme ”työn alla”-palstalta ”vaatii tarkastusta”-palstalle. Tämän jälkeen pääsin aloittamaan dokumentaation luonnin. Käytin aikaisemmin luomaani README.rst-tiedostoa pohjana uudelle dokumentaatiolle. Kerkesin kirjoittaa jonkin verran perustietoja projektista, mutta loput saivat jäädä seuraavalle päivälle. Sain päivän aikana paljon rutiinia koodin optimoinnista, sekä tietokantataulujen liitoksista.

Keskiviikko 01.03.2017

Tehtäväni jatkuvat komentorivityökalun dokumentaatiosta. Tavoitteena on kirjoittaa dokumentaation loppuun rst-muodossa ja lopuksi koittaa saada sen pohjalta generoitua html-sivu Sphinxillä. Mikäli aikaa jää, kysyn projektipäälliköltäni uusia tehtäviä päivän loppupuolelle.

Aamulla pidettiin jälleen palaveri, tällä kertaa liittyen yritykseni henkilöresursseihin. Pääsin kuitenkin hieman myöhemmin työstämään dokumentaatiota Sphinxillä. Kirjoitin ohjelman

perustiedot ja tiiviin käyttöohjeen. Sphinxiä varten jouduin tietysti myös tekemään hieman asetuksia. Tein dokumentaation rst-tiedostoille oman kansion projektin hakemistossa, joita sitten käytin pohjana dokumentaation html-version generoimiseen Sphinxillä. Jouduin aluksi hieman ongelmiin Sphinxin asetusten tekemisessä, mutta muuten kaikki sujui melko mallikkaasti.

Dokumentointityön välissä kehitystiimimme sai viestin, että ohjelmassamme on mahdollisesti muutama kriittinen virhe, joiden varalta ohjelmiston toimintaa tulisi testata käyttöliittymästä käsin. Tästä syystä koko tiimimme käskettiin lopettamaan muu toimintamme, jotta voisimme kaikki auttaa testaamisessa. Teimme tiiminä yhteisen dokumentin, johon kokosimme liudan testitapauksia, jotka kävimme läpi jaotellusti. Tulin itse samalla testanneeksi muutaman tapauksen, jossa löysin erään virheen. Käyttöliittymä ohjasi käyttäjän satunnaisesti väärälle sivulle tiettyjen tapahtumaketjujen jälkeen. Tarkkaa tapahtumaketjua oli kuitenkin vaikea löytää, joten raportoin havainnoistani niin tarkkaan kuin pystyin. Ohjelmiston käyttäytyminen kuitenkin viittasi virheeseen front end-puolella, joten saatoin testaamisen jälkeen keskittyä omiin hommiini. Sillä välin front end-puolen tiimitoverit jatkoivat tapausten tutkimista.

Päivän lopussa sain dokumentaation omasta mielestäni valmiiksi. Mainitsin projektipäällikölleni Tox-skriptityökalusta, jonka tunsin ennestään. Tox on ohjelma jolla voi suorittaa haluamansa testit ohjelmilleen automaattisesti. Testit itsessään pitää yhä toki ohjelmoida. Tox vain helpottaa testien ajossa, kun se luo jokaista testiajoa varten oman virtuaalisen ympäristön, jotta testaajan ei tarvitse huolehtia tarvittavista asetuksista, kuten moduulien asentamisesta. Nämä kaikki voi itse määritellä Toxilla ajettavaan skriptitiedostoon. Toxia voisi käyttää hyväksi myös dokumentaation generoimisen automatisoinnissa, kun halutaan testata, että dokumentaation generoiminen onnistuu. Samoin sitä voisi käyttää hyväksi myös tulevaisuudessa mm. automaatiotestien ajossa. Projektipäällikköni oli erityisen kiinnostunut tiedoistani, joten pienen palaverin jälkeen päätimme lisätä Toxin projektiimme.

Tulin päivän aikana saaneeksi paljon rutiinia testaamiseen, jota en ollut vielä päässyt tekemään uudessa työssäni. Sain samalla hieman vaihtelua omiin työtehtäviini sekä autettua muuta tiimiä.

Torstai 02.03.2017

Jatkan siitä, mihin eilen jäin. Yritän saada ainakin dokumentaation generoimisen Toxilla toimimaan, jonka jälkeen katsomme muun tiimin kanssa, mitkä muut toimet voisivat olla

tällä hetkellä ajankohtaisia. Yksi ideoistani oli myös Flake8-työvälineen ottaminen käyttöön. Flake8:lla voi tarkistaa ohjelman lähdekoodista kaikki tyylivirheet. Oma arvioni on, että lähdekoodista löytyy niitä paljon, joten voisimme tulevaisuudessa helposti korjata ne Flake8:n avulla. Kun olemme Toxin suhteen tilanteeseen tyytyväisiä, tulemme katsomaan projektipäällikkön kanssa itselleni uusia tehtäviä.

Sain päivän alkupuolella Tox-skriptin toimimaan. Säädin sen asetuksissa Flake8- ja Sphinx-testit suoritettavaksi ajon aikana. Toteutin Toxin osana dokumentaation versiohaaraa Gitissä. Lopuksi tein vielä hieman viimeistelyjä dokumentaatioon, kunnes kaikki oli omasta mielestäni valmista ja pyysin tiimitovereitani tarkastamaan dokumentaationi. Projektipäällikkö oli kuitenkin kiireinen, joten jätin hänelle pull requestin versionhallintajärjestelmäämme ja siirryin sillä välin muihin tehtäviin.

Päivän toisena merkittävimpänä tehtävänä oli vanhentuneen tiedon keräys tietokannasta. Tässä tehtävässä kokeneempi tiimitoverini opasti minua. Tulin käyttäneeksi itselleni uutta tekstieditoria nimeltään Sublime Text. Tällä työkalulla pystyin helposti muuttamaan ja järjestelemään tietokannasta poimittua tietoa. Minun ei tarvinnut muuttaa tietoja rivi riviltä, vaan pystyin muuttamaan kaikki rivit yhden rivin muutoksella.

Päivän aikana tulin oppineeksi eniten juurikin Sublime Text-työkalusta ja sen kätevistä toiminnoista. Päiväni tuntui onnistuneen hyvin, varsinkin kun projektipäällikkö kehui dokumentaatiotani ja hyväksyi sen. Päivän päätteeksi kerkesin vielä aloittaa uutta tehtävää liittyen toiseen järjestelmän osaan jossa käytettiin REST-API:a. Oman projektihakemistoni alustamisessa meni jonkin aikaa, mutta sain riittävää opastusta senkin tekemisessä.

Perjantai 03.03.2017

Tänään on tarkoitus jatkaa uudesta tehtävästäni liittyen REST-API projektiin. Tänään tulen toivottavasti oppineeksi hyvin paljon Pythonilla käytettävästä REST-teknologiasta, vaikka tiedänkin siitä jo hieman. Projektin toiminnan opettelemisessa tulee todennäköisesti menemään jonkin verran aikaa.

Päiväni alkoi uuden tehtäväni taustojen selvittämisessä ja ohjelmiston opiskelemisella. REST-teknologia oli minulle jo ennestään tuttu, joten sen ymmärtäminen lähdekoodissa ei ollut minulle ongelma. Pythonille on tarjolla useita eri REST-kehyksiä. Näistä yksi on Flask-RESTful, jonka käytössä ohjelmoijan tulee osata käyttää dekoraattoreita. Itselleni dekoraattorit ovat jo tuttuja työharjoittelustani, niinpä niidenkään ymmärtämisessä minulla ei ollut vaikeuksia.

Jotta pystyin testaamaan ohjelmaa oikeilla REST-kutsuilla ilman varsinaista käyttöliittymää, jouduin kertaamaan Linuxin komentorivillä käytettävän curl-komennon käyttöä. Tällä komennolla pystyin helposti kutsumaan ohjelman käyttämää REST-APIa ilman visuaalista käyttöliittymää. Minun tarvitsi vain määritellä komentoon ohjelmalle lähetettävä data, joka käsiteltäisiin, sekä käyttäjätilini, jolla kirjauduin.

Kun koin ymmärtäneeni ohjelman toiminnasta riittävästi, aloitin tehtäväni ratkaisun suunnittelemista. Tehtävänäni oli toteuttaa ohjelmaan osa, jossa ohjelmaan lähetettävästä datasta karsittaisiin mahdolliset tyhjät merkit ja välilyönnit, jotka olivat varsinaisen merkkijonon alku- tai loppupäässä. Mahdolliset merkkijonon sisällä olevat välilyönnit jätettäisiin huomiotta. Karsiminen koski vain merkkijonoja, eli string-muuttujia.

Tehtäväni sujui aluksi hyvin, kunnes kohtasin funktion, jossa tietokannasta pyrittiin lukemaan rivejä, joilla oli tietty merkkijonotunnus. Tietokantakysely ei jostain syystä toiminut, kuten oletettiin, vaan palautti 0 riviä, kun sen olisi pitänyt palauttaa 1 rivi. Jouduin tutkimaan tietokantaan lähetettävää kyselyä tarkemmin. Selvisi, että ongelma liittyi PostgreSQL-tekniikassa käytettävään ILIKE-operaattoriin. Tutkimiseni aikana en kuitenkaan kerennyt löytämään ongelmaan tarkkaa syytä tai ratkaisua, koska sain muita tehtäviä. Ongelman ratkonta sai siis jäädä maanantaille.

Viikkoanalyysi

Ensimmäisen seurantaviikon jälkeen olen kerennyt työskentelemään erittäin monipuolisten työtehtävien parissa. Viikon aikana esille nousi monenlaista suunnitteluongelmaa omien tehtävieni toteuttamisessa. Aluksi ratkoin ongelmia turhan monimutkaisin tavoin, jolloin toteutuksistani tuli laajempia kuin olisi tarvinnut. Tämä näkyi liiallisena tietokantakyselyiden määränä sekä ohjelman lähdekoodin epäsiisteytenä. Tästä syystä päätin valita tämän viikon teemaksi ohjelmiston toiminnan optimoinnin.

Kehitystyössä tietokannan toimintakykyä tulisi optimoida jatkuvasti sitä mukaa, kun uusia ohjelmiston ominaisuuksia lisätään. Kuten Wang (2011, 3) sanoo opinnäytetyössään, tavoitteena tietokannan toimintakyvyn optimoimisessa on tietokannan suoritustehon maksimointi ja yhteyksien määrän minimointi, jotta saavutetaan suurin mahdollinen suoritusteho. Tietokantakyselyiden määrä tulisi siis pitää minimaalisena. Jo yksikin turha kysely voi olla liikaa, kun kyselyiden määrä kasvaa eksponentiaalisesti uusien asiakkaiden ottaessa palvelun käyttöönsä.

Ohjelman lähdekoodissa taas tulisi aina pyrkiä ennen kaikkea hyvään koodin luettavuuteen. Tämä helpottaa suuresti muiden ohjelman kehittäjien toimintaa, kun heidän tarvitsee käyttää vähemmän aikaa koodin toiminnan tutkimiseen ja ymmärtämiseen. Samalla se helpottaa koodin ylläpidettävyyttä ohjelmoijien omista näkökulmista, kun he tulevat tulevaisuudessa muuttamaan omaa koodiaan, mutta eivät välttämättä muista kaikkia yksityiskohtia siitä. Sneeringer (2015, 260) sanoo, että huolimatta siitä, että koodia usein luetaan enemmän kuin kirjoitetaan, ohjelmoijat useimmiten kirjoittavat koodia olettaen, että heidän ei tarvitse ylläpitää tai edes lukea koodiansa myöhemmin tulevaisuudessa. Siksi Sneeringerin (2015, 260) mukaan yksi tärkeimmistä asioista joita voi tehdä ohjelmoijana, on kirjoittaa luettavaa koodia.

Oman lähdekoodini epäsiisteys näkyi muun muassa järjestelemättöminä funktioina sekä ylimääräisinä funktiokutsuina tilanteissa, joissa yksi funktiokutsu olisi riittänyt, kunhan olisi lähettänyt funktiolle riittävät argumentit. En ollut järjestänyt funktioitani loogisesti omiin lähdekoodi-tiedostoihin, vaan ne olivat siellä, missä niitä ensimmäisenä tarvitsin. Olin kutsunut joitakin funktioita samassa paikassa monta kertaa, lähettäen samalle funktiolle eri argumentit eri kohdista. Sen sijaan, olisin voinut alustaa kaikki argumentit tyhjäksi toiminnan alussa ja lähettää tyhjät argumentit niiden parametrien paikalle, joita ei kyseisessä tilanteessa tarvittu. Argumenttien arvot sijoitin riippuen if-ehtojen täyttymisistä.

Sain kuitenkin tarkastuspalavereissa palautetta koodini siisteydestä, jolloin tulin automaattisesti korjanneeksi ja siivonneeksi lähdekoodiani. Opin näiden virheideni avulla sijoittelemaan koodiani jatkossa paremmin.

3.2 Seurantaviikko 2

Maanantai 06.03.2017

Tänään jatkan siitä mihin jäin viime viikolla. Pyrin ratkaisemaan ongelman liittyen ohjelmiston REST API-funktion tietokantakyselyyn. Mikäli en kuitenkaan löydä ratkaisua ensimmäisen kahden tunnin aikana, pyydän apua muilta tiimitovereiltani. Kun ongelma on saatu ratkaistua, jatkan tehtäväni suorittamista normaaliin tapaan.

Sain ongelman ratkaistua heti aamulla, kun huomasin että virhe liittyikin hakemaani riviin tietokannassa. Yhdellä sarakkeella oli määritelty näiden tietokantaolioiden aktiivisuus-tila tosi- ja epätosi-arvoilla. Käyttämässäni tietokantakyselyssä kuitenkin suodatettiin pois rivit, joilla tämä aktiivisuus-sarake oli määritetty epätodeksi, mikä koski myös hakemaani riviä. Ongelma ratkesi, kun valitsin toisen rivin pääavaimen käyttöön. Tämä olio oli määritelty

aktiiviseksi, jolloin käyttämäni kysely palautti kyseisen rivin. Tämän jälkeen tehtäväni sujui mallikkaasti loppuun asti.

Saatuani tehtäväni valmiiksi, pyysin tiimitoveriltani uutta tehtävää. Tämä tehtävä liittyi myös aiemmin työstämääni REST API-osioon. Tarkoituksena oli uudelleen rakentaa tietty laskenta-algoritmi yhdelle tietokantakyselyn tulokselle. Muutosta tarvittiin, koska tietokannan tauluihin suunniteltiin muutoksia lähitulevaisuudessa. Tämä tehtävä vaikutti haastavalta mutta sain siinä kokeneemman tiimitoverini opastusta. Sain tehtävän ensimmäisen puoliskon valmiiksi päivän loppuun mennessä, kun toinen tehtävään liittyvä funktio piti myös rakentaa uudelleen.

Tiistai 07.03.2017

Jatkan eilisestä tehtävästä. Joudun todennäköisesti tutkimaan lähdekoodin rakennetta hieman lisää ymmärtääkseni koko laskentaprosessin. Sitten pystyn toteuttamaan myös tehtävän toisen puoliskon.

Jouduin opiskelemaan lisää lähdekoodia ymmärtääkseni koko API-funktion toiminnan. Kun ymmärsin mielestäni riittävästi, aloitin funktion muokkaamisen. Tietokantaan oli jo luotu päivitetyt versiot tehtävään liittyvistä tauluista, joten jouduin vain päivittämään tietokantaan tehtävät kyselyt käyttämään päivitettyjä tauluja oikein. Tarvitsin joissakin osissa vielä opastusta tiimitoveriltani, mutta pärjäsin jo paremmin omillani verrattuna eiliseen.

Sain tehtävän valmiiksi puoleen päivään mennessä, jolloin pyysin uutta tehtävää muulta tiimiltäni. Uusi tehtäväni olisi liittynyt aiemmin työstämääni komentorivityökaluun, mutta se edellytti toisen tehtävän toteuttamista API-järjestelmässä, joten jouduin jälleen muokkaamaan toista API-funktiota.

Keskiviikko 08.03.2017

Sain osan uudesta API-funktion muutoksesta valmiiksi eilen, mutta korjattavaa jäi. Jouduimme kehitystiimin kanssa keskustelemaan mahdollisista isoista muutoksista funktioiden rakenteeseen, ylläpitääksemme järjestelmän siisteyttä ja johdonmukaisuutta. Päätimme jatkaa keskusteluja tänään aamupäivällä, joten päiväni tulee todennäköisesti alkamaan palaverista, jonka jälkeen voin jatkaa API-järjestelmän kehittämistehtävää. Tämän jälkeen saatan mahdollisesti päästä muokkaamaan aiemmin työstämääni komentorivityökalua.

Jouduin odottamaan palaveria liittyen API-funktioon ja tietokannan rakenteeseen, koska projektipäälliköllä oli muita kiireitä aamupäivällä. Sillä välin, kun nykyinen tehtäväni odotti yhteisiä päätöksiä, tulin opiskelleeksi hieman front end-puolen käyttöliittymää. Tutkin käyttöliittymämme rakennetta, selailemalla eri sivuja ja painelemalla eri valintoja. Tällä tavoin sain paremman käsityksen, miten back end-puolelta luettua dataa oikein käytettiin käyttöliittymässämme. Löysin samalla yhden kehitysidean liittyen sivuston objekteihin liitetyistä merkeistä. Kerroin tästä myöhemmin muulle kehitystiimille.

Kun lopulta saimme palaverimme pidettyä, päätimme, että oli parasta liittää yhden tietokannan taulun sarakkeet toiseen isompaan tauluun, koska ne olivat koko ajan olleet vahvasti sidoksissa toisiinsa, eikä ensimmäistä tietokannan taulua enää tarvittu. Muokkasin lähdekoodia myös siten, että tietojenkäsittely vastasi tietokannan rakennetta. Kun sain tehtävän valmiiksi, siirsin tehtävän JIRAn Scrum-taululla ’vaatii tarkastusta’-palstalle. Tämän jälkeen pääsin aloittamaan toisen tehtävän, joka liittyi suoraan juuri valmistuneeseen tehtävään.

Toinen tehtävä liittyi komentorivityökaluun, jossa haluttiin antaa käyttäjälle vaihtoehtoisia valintoja tietojen suodattamiseen, kun tietoja luetaan tietokannasta. Kerkesin vielä saamaan tehtävän rungon valmiiksi ilman suurempia ongelmia. Päätin kuitenkin, että toteutuksessa oli vielä hieman parantamisen varaa.

Torstai 09.03.2017

Tänään toteutan loppuun tehtäväni liittyen komentorivityökaluun. Haluan parannella käyttäjälle annettavaa palautetta järjestelmästä, kun tietokannasta luetaan tietoja. Sen jälkeen tarkistutan tehtävän ja kysyn uutta tehtävää.

Sain heti aamupäivällä uuden ominaisuuden valmiiksi. Kun siirsin tehtävän Scrum-taululla ’vaatii tarkastusta’-palstalle, minulla olikin jo kertynyt monta tehtävää tarkastettavaksi. Siksi ennen uutta tehtävää yritin sopia tarkastuspalaveria muun Back End-tiimin kanssa, mutta sain sen vasta iltapäivälle koska heillä oli muita kiireitä aamupäivällä. Sillä välin sain kuitenkin opiskella uutta API-osaa seuraavaa tehtävääni varten.

Alkuiltapäivästä tiimitoverini front end-puolelta pyysi apuani hänelle annettuun tehtävään. Hänen tehtävänsä liittyi ohjelmistovirheen korjaamiseen ohjelmistomme käyttöliittymässä. Hän pyysi minua selvittämään aiemmin löytämääni bugia käyttöliittymästä, jossa käyttäjä ohjattiin toisinaan toiselle sivulle hänen antaessa palautetta. Selvittämisen apuna käytin Firefox-selaimesta löytyvää konsolia, josta näin muun muassa, mitä API-kutsuja käyttöliit-

tymästä tehtiin. Huomasin että bugia ennen esiintyi yksi epäonnistunut API-kutsu. Raportoin tästä tiimitoverilleni, joka vinkkini perusteella löysi bugin nopeasti ja korjasi sen.

Pitkän koodin tarkastuspalaverin jälkeen saimme uudet tehtäväni tarkistettua. Osaan niistä tein pieniä muutoksia, mutta muuten tiimini oli ratkaisuihini tyytyväisiä.

Perjantai 10.03.2017

Uuden API-tehtävän työstäminen jatkuu. Joudun vielä opiskelemaan lähdekoodia ymmärtääkseni funktion riittävän hyvin.

Aamun palaverissa kyselemme toistemme työvaiheista sekä mahdollisista ongelmista. Omalla kohdallani kerron oman tehtäväni haasteista liittyen ohjelmointitehtäväni suuruuteen ja monimutkaisuuteen. Kyseessä on tähän mennessä haastavin tehtäväni tässä työssä, koska muun tiimini mukaan tulen tarvitsemaan tehtävässä tietojen ryhmittely- ja alikysely-taitoja, joista minulla ei vielä ole paljoa rutiinia. Joudun menemään pois omalta mukavuusalueeltani, mikä tuo sinänsä hyvää ja mielenkiintoista vaihtelua työtehtäviini, saadessani enemmän vastuuta.

Jouduin lukemaan tehtävääni liittyvää lähdekoodia pitkään, jotta ymmärsin tehtäväni ongelman perusteellisesti. Tehtävässä API-funktiota oli tarkoitus muuttaa siten, että palautettava data ryhmiteltäisiin ’group by’-lausekkeilla riippuen API-funktiolle annettavista argumenteista. Minun piti myös suorittaa useita alikyselyjä ’subquery’-lausekkeilla tietokantaan, jotta sain juuri ne tiedot valittua, jotka argumenteilla määriteltiin.

Iltapäivällä järjestettiin yrityksessäni joka toinen perjantai järjestettävä Sprint-kokous, jossa kehitystiimimme raportoi muille tiimeille ohjelmistomme käyttöversioon tulevista muutoksista. Jokainen kokoukseen osallistuva sai myös esittää omia kehitysideoitaan ohjelmistollemme. Itse en vielä osannut löytää uusia kehittämisen kohteita, lukuun ottamatta aiemmin raportoimaani puutetta käyttöliittymässä. Koitan olla valppaampi seuraavassa kokouksessa.

Viikkoanalyysi

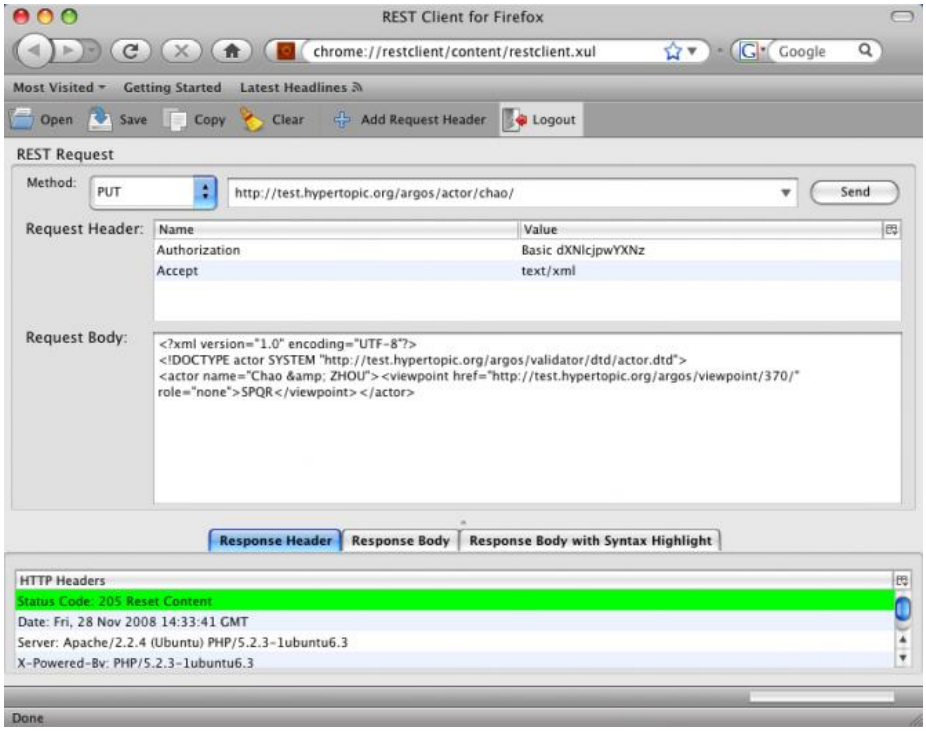
Tulin tehneeksi todella monipuolisia työtehtäviä viikkoni aikana, joissa kaikissa oli omat haasteensa. Samalla kun tein itselleni tuttuja työtehtäviä, löysin myös uusia ja vaihtoehtoisia tapoja ja menetelmiä työtehtäviini. Tämä näkyi esimerkiksi siinä, että opin Linuxin curl-komennon sijasta käyttämään url-osoitteita selaimen kautta REST API-funktioiden

testaamiseen. Selaimessa argumenttien syöttäminen API-funktioille tapahtui eri tavalla, jonka oma tiimitoverini opetti minulle.

Tällä viikolla valitsin teemakseni monipuolisen testaamisen, koska ketterässä tuotekehityksessä tuotteen toimivuutta tulisi testata jokaisen ominaisuuden kohdalta, vaikka oltaisiin kehittämässä vain yhtä tuotteen ominaisuutta. Tämä siksi, että muutokset yhteen ominaisuuteen voivat odottamatta vaikuttaa suorasti tai epäsuorasti toisen ominaisuuden toimintaan. Koska kehitystiimimme on pieni, meillä ei ole testaamiseen erikoistunutta henkilöä tiimissämme. Kuten Väyrynen (2009, 20) sanoo opinnäytetyössään, ketterissä menetelmissä kehitystiimi on yhdessä vastuussa testauksesta, vaikka kehitystiimissä olisi testaukseen erikoistunut henkilö.

API-Ohjelmassa API-funktioille rekisteröidään omat nimiosoitteet. Näitä funktioita voidaan kutsua esim. web-selaimessa, syöttämällä selaimelle ensin palvelimen osoite, jolla ohjelma ajetaan, sekä kutsuttavan funktion osoite. Tässä tapauksessa ajoin ohjelman omalla tietokoneellani, joten käytin palvelimen osoitteena omaa paikallista ip-osoitettani. Funktion osoitteen syöttämisen jälkeen saatoin syöttää funktion argumentit suoraan osoitteen perään käyttämällä merkkejä ’?’, ’=’ ja ’&’. Testatessani kehittämiäni ohjelman API-funktioita, pystyin tällä tavoin toteuttamaan testini nopeammin. Curl-komennossa argumentit määriteltiin curl-ohjelmalle syötettävillä argumenteilla, joiden muokkaaminen oli toisinaan hieman kömpelöä, johtuen Linuxin komentorivin käyttöliittymästä.

REST APIn testaamisessa pelkällä web-selaimella on kuitenkin rajoitteensa. Vain GET-metodien testaaminen onnistuu kirjoittamalla osoite ja argumentit selaimen osoitekenttään, koska käytettävää REST-metodia ei ole mahdollista määrittää osoitekentässä. Tähän en aiemmin keksinyt muuta ratkaisua kuin curl-komennon käyttö muissa kuin GET-metodeissa, kunnes löysin REST Client-nimisen selainlaajennuksen Firefoxiin (Kuva 1). Tämä työkalu tulisi varmasti olemaan kokeilemisen arvoinen APIn testaamisessa tulevalla viikolla.



Kuva 1. REST Client-laajennus FireFoxille (Mozilla Foundation)

Front end käyttöliittymän testaamisessa opin viikon aikana uusia menetelmiä. Opin hyödyntämään FireFoxin kehittäjä-työkaluja paremmin työssäni. Testatessani käyttöliittymän eri elementtien toimintaa, pystyin tarkastelemaan niiden rakenteita ja toimintoja lähdekoodista. Samaisella työkalusetillä pystyin myös seuraamaan sivustolla tapahtuvaa verkkoliikennettä, kuten tapahtuvia REST API-kutsuja. Aikaisemmin saatoin lähinnä arvailla syitä löytämiini bugeihin, mutta näiden työkalujen avulla pystyin havaitsemaan sovelluksen ongelmat paljon tehokkaammin. Tarkastellessani käyttöliittymän lähdekoodia, tulin automaattisesti oppineeksi enemmän front endissä käytettävästä JavaScript-kielestä, vaikka en sitä nykyisissä työtehtävissäni paljoa tarvitsekaan.

Tähän mennessä testaukseni on painottunut niin sanottuun lasilaatikkotestaukseen, koska olen käyttänyt pääasiassa kehittäjän työkaluja testatessani ohjelmistoa. Väyrysen (2009, 10) mukaan lasilaatikkotestauksessa testaus suoritetaan koodaajan näkökulmassa, eli testaaja tuntee toteutuksen. Tavoitteenani tulevaisuudessa olisi myös yrittää toteuttaa enemmän toista testaustapaa, mustan laatikon testausta. Väyrynen (2009, 10) mukaan mustan laatikon testauksessa testaus suoritetaan käyttäjän näkökulmasta, eli testaaja ei kiinnitä huomiota toteutukseen. Käyttämällä enemmän mustan laatikon testausta pystyn löytämään ohjelmistosta bugeja sekä parantelua tarvitsevia ominaisuuksia helpommin, kun en tee asioita yhtä tietoisesti siitä, mitä olen tekemässä.

3.3 Seurantaviikko 3

Maanantai 13.03.2017

Jatkan perjantaina kesken jääneestä tehtävästä, jossa käytän tietojen ryhmittelyä ja alikyselyjä tietokantakyselyissä. Tietokannan rakenteen ymmärtämisessä minulla oli vielä hieman ongelmia.

Tiimitoverini kuvaili tietokannan rakennetta minulle paperille piirrettynä, jotta saisin siitä paremman käsityksen. Pääsin tehtävässäni pikkuhiljaa eteenpäin, ja aloin askel askeleelta ymmärtää tietokannan rakennetta enemmän. Jouduin vielä toisinaan tarkistuttamaan tiimitoverillani alikyselyihin liittyvää syntaksia ohjelman lähdekoodissa. Testatessani ohjelmaa suurimmat ongelmat liittyivät muuttujien tyypityksiin, jotka eivät vastanneet toisiaan. Kun syötin API-funktiolle argumentit, niiden muoto muuttui matkalla funktioon, jossa niitä käytettiin rivien suodattamiseen tietokannasta. Tämä hämmensi tiimitoverianikin. Jouduimme pohtimaan pitkään parasta ratkaisua tilanteeseen, jonka jälkeen päädyimme muuttamaan omaa funktiokirjastoamme, jossa muuttujien tyypitykset määriteltiin.

Tyypitysongelmien jälkeen pääsin hyvään vauhtiin työssäni. Ymmärsin jo alikyselyiden syntaksin ja periaatteen, jolloin pystyin toteuttamaan ne helposti peräkkäin. Jokaisen alikyselyn toteuttamisen välissä tulin testanneeksi APIn toimintaa.

Tiistai 14.03.2017

En saanut eilistä tehtävääni vielä täysin loppuun, mutta sain todennäköisesti suurimmat ongelmat ratkottua, joten tehtävän pitäisi sujua hyvin. Tänään pidän kuitenkin puolet lyhyemmän työpäivän henkilökohtaisista syistä johtuen.

Sain oman ratkaisuni valmiiksi ilman suurempia ongelmia, mutta sen tarkastuspalaverissa havaittiin, ettei tietokantahaku palauttanut dataa oikeassa halutussa muodossa. Projektipäällikkö neuvoikin minua käyttämään muita tekniikoita alikyselyiden sijasta, koska alikyselyt eivät olleet tietokannan optimoinnin kannalta parhaimpia ratkaisuja. Muita käytettäviä tekniikoita olisivat muun muassa with-lausekkeen käyttö sekä datan aliasointi.

Keskiviikko 15.03.2017

Jatkan eilisen tehtävän ratkomista uusin tekniikoin. Saatan tarvita tiimitoverini apua väliajoin ymmärtääkseni enemmän itselle uusien tekniikoiden käytöstä.

Sain opastusta projektipäälliköltäni useaan otteeseen tehtävän edetessä. Pyysin ongelmien ja niiden ratkaisujen kuvaamista paperille piirrettynä, jotta saisin paremman käsityksen tilanteesta. Hän neuvoi minua käyttämään tehtävän ratkomisessa itselleni vähemmän tuttuja SQL-operaattoreita, kuten WITH ja UNION. Tämä ratkaisu toimisi alikyselyjä paremmin tietokannan optimoinnin kannalta. Jouduin opiskelemaan näiden käyttöä paljon eri lähteistä, pääasiassa virallisesta postgresql-dokumentaatiosta. Koska en ollut täysin varma syntaksista, pyysin tiimitoveriani tarkastamaan koodini tietyin väliajoin, jotta tiesin, olinko menossa oikeaan suuntaan.

Sain tehtävään väliaikaisen ratkaisun, jossa tarvittava data palautettiin tietokannasta, mutta sitä ei oltu suodatettu vielä riittävästi. Minun pitäisi vielä tarkastaa suodattimet. Ratkaisun valmistuessa paljastui kuitenkin muita ongelmia API-funktion lähdekoodissa. Tiettyjä muuttujia määriteltiin epämääräisissä paikoissa huonoin argumentein, jolloin tehtävääni lisättiin myös koodin yleinen uudelleenjärjestely.

Torstai 16.03.2017

Jatkan samasta tehtävästä. Yritän saada API-funktiossa havaitut puutteet kuntoon päivän aikana.

Sain tietojen suodattamisen toteutettua oikein, jotta tietokannasta palautetut rivit koskivat vain funktioon argumentoituja tietoja. Toinen suuri korjaus koski osaa argumenteista, jotka funktiolle annettiin REST-pyynnön yhteydessä. Tässä kohtaa minulla oli enemmän haasteita. Jouduin ratkomaan ongelmia liittyen aikaleimoihin eri aikavyöhykkeillä, jolloin minun piti löytää jokin tapa paikallistaa aikaleimojen vertaileminen riippuen aikavyöhykkeestä. Sain yhdessä projektipäällikön kanssa laadittua pohjaratkaisun ongelmalle, mutta syntaksin kanssa minulla oli vielä tehtävää.

Päivän aikana tulin kokeilleeksi viikonloppuna löytämääni REST Client-laajennusta FireFoxille. Osoitteen, metodin ja muiden perusvalintojen määrittely onnistui helposti, mutta REST-pyynnöissä käytettävät headerit osoittautuivat itselleni uudeksi asiaksi. En osannut määritellä metodeille parametreja liitännäisessä, koska niitä ei hyväksytty samassa muodossa kuin kirjoittaessa selaimen osoiteriville. Projektipäälliköni selitti minulle headerien merkityksen vain pintapuolisesti, joten niistä voisi olla hyvä lukea lisää. Hän kuitenkin mainitsi, että käyttää itse Postman-nimistä selainlaajennusta Chrome web-selaimelle, joka saattaisi olla helppokäyttöisempi kuin REST Client. Tulin ladanneeksi ja esivalmistelleeksi sen, mutta en vielä kokeilleeksi sitä käytännössä.

Perjantai 17.03.2017

Yritän ratkaista aikaleimoihin ja aikavyöhykkeisiin liittyvän ongelman tietokantakyselyissä. Tämä havaittu ongelma suurensi tehtäväni laajuutta, joka onkin ollut tähän mennessä haastavin ja monimutkaisin tehtäväni työssäni.

Oikean syntaksin löytämisessä meni aluksi eniten aikaa. Yritin hakea tietoa useista eri lähteistä ongelman ratkomiseen, mutta oikeaa tapaa oli vaikea löytää. Lopulta kysyin neuvoa tiimitoveriltani, jonka avulla ongelma ratkesi. Sen jälkeen minun piti lähinnä hioa muita suodattimia, jotta sain halutun tuloksen.

Ratkaisustani nousi esille uusia ongelmia. Kyselyssäni tein aikaleimojen muunnoksia jokaiselle tietokannasta haetulle aikaleimalle, jolloin tietokantakyselyt veivät paljon enemmän aikaa, mitä suurempia tuloksia tietokannasta haettiin. Ongelman ymmärtäminen vei jonkin aikaa, kunnes projektipäällikkö selitti ongelman minulle. Se, mitä minun pitäisi jatkossa tehdä, olisi ensin haarukoida potentiaaliset tulokset tietokannasta, ennen niiden muuntamista, jotta tietokanta ei rasittuisi liikaa turhista prosesseista.

Viikkoanalyysi

Viikon alussa ongelmani liittyivät muuttujien tyypityksiin. Ongelmana oli teksti- ja tavutyyppisten merkkijonojen käsittely APIssa. Molemmat muuttujatyypit ajavat saman asian, eli toimivat merkkijonojen varastoimiseen, mutta ne varastoivat merkit eri tavalla. Sneeringer (2015, 142) kuvailee kirjassaan, kuinka teksti-merkkijonot varastoivat datan sisäisesti unicode-muodossa, siinä missä tavu-merkkijono varastoi merkit yksittäisinä tavuina. Tavu-merkkijonot ovat huomattavasti rajoitetumpia sisällöltään, sillä ne käyttävät merkkien varastoimiseen ASCII-standardia. ASCII-standardi, eli American Standard Code for Information Interchange on kirjainavaruus, jossa jokaiselle kirjaimelle on vastaava numeerinen arvo, mutta se käsittää vain 128 yleisintä kirjainta.

Unicode-merkkijonoille on tarjolla useampia paljon joustavampia standardeja, kuten UTF-8-standardi. Python Software Foundationin (2017) virallisen Python 2-dokumentaation mukaan UTF-8 on yksi käytetyimmistä standardeista merkkijonojen varastoimisessa. UTF-8 käsittää huomattavasti laajemman avaruuden useampien merkkien varastoimiseen, kuten muiden kuin englannin kielen yleisten kirjainten varastoimiseen. Jos unicode-merkkijonoja yritetään väkisin muuntaa tavu-merkkijonoiksi, voi syntyä ongelmia. Pythonilla ohjelmoidessa unicode-merkkijonoille tulisi aina määrittää käytettävä kirjainavaruus, jos käytetään erikoismerkkejä. Python Software Foundation (2017) huomauttaa, että mikäli

standardia ei määritellä unicode-merkkijonolle Python 2-versiossa, varastoimisessa käytetään yhä ASCII-standardia.

Toinen hämmennystä aiheuttava seikka tehtävässäni oli str- ja unicode-luokkien samanaikainen käyttö. Jotta tiedetään, kumpaa tyyppiä tulisi käyttää, pitää tietää, millä Python-versiolla ohjelmaa toteutetaan. Sneeringerin (2017, 143) mukaan str- ja unicode-luokat ovat käytännössä sama asia Python 2- ja Python 3-versioilla, mutta molempien luokkien funktioita on silti käytössä kummallakin versiolla. Siinä missä unicode-merkkijonot perustuvat str-nimiseen luokkaan Python 3-versiossa, perustuvat ne unicode-nimiseen luokkaan Python 2-versiossa. Niissä on kuitenkin hienovaraisia eroja. Python Software Foundationin (2017) Python 3-dokumentaatiossa huomautetaan, että str-luokka käyttää vakiona UTF-8 standardia merkkijonojensa varastoimisessa.

Omassa työssäni unicode- ja str-luokat useimmiten sekoittuivat keskenään, jolloin tulin aiheuttaneeksi itselleni lisää ongelmia, kun en ollut tarkka muuttujien tyypityksestä. Omissa työkalukirjastoissamme näitäkin sekoiteltiin toisinaan, mutta tärkeintä oli löytää tässä tehtävässä käytettävät moduulikirjastot ja löytää niihin selkeä yhdenmukaisuus. Opin virheistäni paljon uusia piirteitä merkkijonojen käsittelyssä. Tavoitteeksi jatkossa asetan yhdenmukaisuuden tavoittelemisen muuttujatyyppien valitsemisessa, jotta eri muuttujatyypeistä ei aiheutuisi ongelmia niiden vertailemisessa.

Toinen suuremmista haasteistani viikon aikana oli jälleen optimaalinen tietokannan käyttö. Tässä palaan ensimmäisen viikon aikana käsittelemääni teemaan, eli ohjelmiston toiminnan optimointiin. Kohtasin viikon aikana todella monimutkaisia ongelmia tietokannan datan käsittelyssä. Minua opastettiin tiimitovereideni puolesta käyttämään itselleni uusia sekä vähemmän tuttuja tietokanta-tekniikoita ja operaattoreita kuten with-operaattoreita ja alikyselyjä. Aloitin tietokantakyselyn toteuttamisen käyttämällä alikyselyjä, mutta ne osoittautuivat tietokannan suorituskyvyn kannalta haitallisiksi tähän tilanteeseen.

Ensimmäisen ratkaisuni jälkeen, tulin käyttäneeksi with-operaattoreita tietokantakyselyissäni alikyselyjen sijasta. Burlesonin (2013) mukaan esimerkiksi Oracle SQL-tietokannat pystyvät suoriutumaan tehtävistään nopeammin, kun monimutkaiset alikyselyt korvataan globaaleilla väliaikaisilla tauluilla. Minulle selvisi, että siinä missä monimutkaiset alikyselyt joutuivat aina lukemaan tietokannan datan alusta alkaen, with-operaattoreilla pystyin luomaan yhdellä select-lausekkeella väliaikaisen taulu. Tätä taulua pystyin käyttämään koko tietokantasession ajan apuna muihin kyselyihin, jotka käyttivät kyseistä taulua pohjatietona.

Törmäsin suorituskyky-ongelmiin myös siinä tehtäväni vaiheessa, jossa minun piti löytää tietokannasta aikaleimat ja kääntää ne niiden omasta yhteisestä aikavyöhykkeestä vastaamaan toisen aikavyöhykkeen vastaavaa aikaa. Tehtävässäni käytin tätä käännöslauseketta aluksi tietojen suodattimien yhteydessä, mutta tämä ratkaisu osoittautui suorastaan rampauttavaksi ratkaisuksi tietokannan suorituskyvyn kannalta. Tietokanta-ajuri joutui kyselyssään kääntämään kaikki aikaleimat toiselle aikavyöhykkeelle, myös ne jotka eivät olisi voineet edes potentiaalisesti osua haetulle aikavälille. Kuten Riggs & Krosing (2010, 331) toteavat, yksittäisen monimutkaisen kyselyn suorittaminen ei aina ole hidasta itsessään, mutta esimerkiksi satojen tai tuhansien monimutkaisten kyselyjen suorittaminen sekunnissa voi hidastaa tietokantapalvelimen toimintaa huomattavasti. Tästä syystä minun tulisikin aina jatkossa ajatella vaihtoehtoisia tapoja toteuttaa ratkaisuni, jotka rasittaisivat tietokantapalvelinta vain minimaalisesti.

Omassa suunnittelu- ja toteutustyössäni en aina ensin ajatellut suorituskyvyn optimointia, koska tärkeämpää oli ensin toimivan ratkaisun löytäminen sekä sen ymmärtäminen. Niinpä itse optimointi tulee useimmiten jälkikädessä, kun tajutaan että jo toimivaksi todetussa ratkaisussa on vielä parantamisen varaa. On tärkeää, että ohjelmiston käyttöversiossa ongelmat on ratkaistu parhaimmalla mahdollisella tavalla, jotta asiakkaalle ja varsinaiselle käyttäjälle taataan aina paras mahdollinen käyttäjäkokemus.

3.4 Seurantaviikko 4

Maanantai 20.03.2017

Tänään tulen asentamaan itselleni uuden työkoneen, jonka yritykseni oli minulle aiemmin luvannut. Olen tähän mennessä käyttänyt henkilökohtaista kannettavaani työntekoon, joten tämä muutos helpottaa logistiikkaa työssäni. Asennustehtävässä tulee menemään jonkin verran aikaa, mutta luulen pääseväni päivän lopussa vielä jatkamaan viime viikon tehtävääni.

Uuden työkoneen valmisteleminen työkäyttöön sujui suhteellisen nopeasti verrattuna oman henkilökohtaisen kannettavani valmistelemiseen. Vaikka työvaiheissa olikin paljon yksityiskohtia joita en muistanut, pystyin katsomaan mallia omalta kannettavaltani useimmissa tilanteissa. Paljon helpotti se, että pystyin lukemaan Linuxin komentorivin, eli terminaalin käyttöhistoriasta aiemmin käyttämiäni komentoja. Ongelmana oli aluksi, että en nähnyt koko historiaa pelkällä history-komennolla, koska se näytti vain 1000 viimeisintä komentoa. Jotta pystyin näkemään vanhemmat komennot, minun piti avata .bash_history-tiedosto. Tämän avulla minulla ei mennyt kauaa useimpien tarvittavien komentojen selvit-

tämisessä, sekä siinä, mitä riippuvaisuuksia minun tuli asentaa. Yhteensä työkoneen valmisteluun kului aikaa noin neljä tuntia, kun ensimmäisellä kerralla aikaa kului yli päivä.

Päivän lopussa kerkesin hieman työstää aiemmin jatkamaani tehtävää, jossa minun piti muuntaa aikaleimoja käytössä olevien aikavyöhykkeiden mukaan. Tulin muuttaneeksi tietokantasuodattimien rakennetta siten, että tietokannasta luettavia rivejä ei muutettu heti lukemisen yhteydessä, jotta tietokanta ei rasittuisi kyselyistä liikaa. Tulin samalla kokeilleeksi projektipäällikköni suosittelemaa Postman-liitännäistä Chrome-selaimelle, jolla pystyy REST-kutsujen lisäksi suorittamaan testejä API-ohjelmien funktioille. Oivalsin liitännäisen REST-kutsujen teon nopeasti, ja totesin liitännäisen olevan paljon helppokäyttöisempi kuin aiemmin kokeilemani REST Client FireFox-selaimessa. Projektipäällikkö näytti minulle pikaisesti, miten testien luominen Postmanilla onnistuu. Niiden luomisessa tarvitsen kuitenkin JavaScript-taitoja, jotka ovat itselläni hyvin minimaaliset omasta mielestäni. Tulen kuitenkin palaamaan testien luontiin todennäköisesti myöhemmin.

Tiistai 21.03.2017

Jatkan edelleen samasta API-tehtävästä. Olen jo hyvin lähellä lopullista ratkaisua, mutta se vaatii vielä pientä ponnistelua.

Sain aikaleimojen muuntamisen viimein toteutettua, vaikka jouduinkin vielä kysymään ohjeita projektipäälliköltä pariin otteeseen. Koodin tarkastuksessa kuitenkin havaittiin, että osa aikaleimoista jotka olisi pitänyt sisältää tietokantatulokseen jäivät siitä pois tietyissä tapauksissa. Jouduin loppupäivän tutkimaan tämän ongelman syytä, mutta en vielä löytänyt ongelman varsinaista syytä.

Tulin päivän aikana keskustelleeksi muun kehitystiimin kanssa automaatiotesteistä. Autoin projektipäällikköä testien toteuttamisessa havaitun ongelman kanssa. Yksi testifunktio aiheutti poikkeuksen, koska sille syötettyjä argumentteja oli ohjelman mukaan vähemmän kuin funktiossa määriteltyjä parametreja, vaikka kaikki näytti olevan kunnossa. Sitten huomasin, että parametrien välistä puuttui pilkku, mikä tietenkin korjasi tilanteen. Tämä näyttää sen, miksi ohjelmoidessa on aina hyvä olla apuna toinen henkilö, silloin kun ongelma ei millään ratkea itsekseen.

Keskiviikko 22.03.2017

Yritän ratkaista aikaleimoissa havaitun ongelman. Mikäli en löydä ongelman syytä ensimmäisen kahden tunnin aikana, pyydän apua.

Löysin ongelman ratkaisun hyvinkin nopeasti, kun vertasin omaa uutta lähdekoodia vanhan toteutusversion lähdekoodiin. Tosin tälläkin kertaa hämmennystä aiheutti oma kirjoitusvirhe koodissa, jolloin tulin käyttäneeksi päivämäärä-muuttujien rajaamisessa eri funktiota kuin luulin. Vertailemalla koodeja rinnakkain, tämä virhe löytyi helposti.

Koska projektipäällikkö ei ollut tänään paikalla, loin Gitissä pull requestin ja aloin työstää uutta, huomattavasti yksinkertaisempaa tehtävää, jossa tietyt API-funktiot jaettiin kahteen eri API-funktioon. Tarkoituksena oli eristää tietokantaolioiden aktiivisuus-sarakkeen päivittäminen omaan päätepisteeseensä, turvallisuusseikkojen vuoksi. Näitä tietokantaoliota oli toki monta, joten APIa piti muuttaa monesta paikkaa. Tehtävä sujui hyvin suoraviivaisesti, vaikka välillä tulikin ongelmia yleisten kirjoitusvirheiden vuoksi.

Torstai 23.03.2017

Tehtävästä jäi vain pieni osa jäljelle, joten minun pitäisi saada kaikki valmiiksi heti aamulla. Sen jälkeen pystyn todennäköisesti pitämään uuden tarkastuspalaverin sekä tälle tehtävälle, että eilen valmistuneelle tehtävälle.

Kaikki sujui suunnitelmien mukaan, ja sain hyvää palautetta tarkastuspalaverissa. Funktioiden jakotehtävän tarkastuksessa riitti, että näytän demon vain yhdestä muuttuneesta osasta, koska kaikki muutettavat osat toimivat samalla periaatteella. Projektipäällikkö tarkistaisi vielä koodini myöhemmin yksityiskohtaisemmin, liittäessään ratkaisuni käyttöversioon. Tätä ei voitu kuitenkaan vielä tehdä, koska ominaisuutta ei oltu toteutettu ohjelmistomme front end-puolella.

Toinenkin tehtävä meni läpi tarkastuksesta, mutta projektipäällikkö pyysi minua kuitenkin luomaan testejä APIn toimintaa varten Chrome-selaimen Postman-laajennuksella. Kuten aiemmin mainitsin, tällä työkalulla pystyi luomaan näppäriä automaatiotestejä tarkistamaan esim. palauttaako kutsuttava REST API-funktio oikeaa dataa käyttäjälle. Testeissä asetettiin funktiokutsuille eri parametreja, ja tarkistettiin, vastasiko palautettu data tiettyä arvoa. Sain tehtäväni aikana paljon rutiinia JavaScript-syntaksiin, joka ei ole itselläni niin hyvin hallussa kuin Python.

Perjantai 24.03.2017

Sain tehtäväkseni lisätä uuden parametrin toiseen API-funktioon. Parametrina olisi merkkijono, joka ottaisi vastaan käyttöliittymässä käyttäjän antaman vapaaehtoisen kommentin.

Parametrin lisääminen onnistui odotetun helposti. Ominaisuuden testaamista vaikeutti kuitenkin hieman funktioihin tarvittavat asetustiedostot, jotka itseltäni puuttuivat. Tiimini sanoi kuitenkin, että asetustiedostojen asettaminen olisi vienyt turhan paljon aikaa, mikä ei ollut tarpeellista tässä tehtävässä. Niinpä tulin vain poistaneeksi riippuvuudet lähdekoodista, jotka kyseisiä asetustiedostoja tarvitsivat. Tämä oli helppo tehdä muuttamalla kyseiset rivit lähdekoodista kommenteiksi. Lisäsin vain #-merkin rivien alkuun testaamisen ajaksi, jolloin kyseistä riviä ei luettu ohjelmassa ajon aikana.

Myöhemmin annoin projektipäällikön nopeasti tarkistaa ratkaisuni. Koska muutos oli lopulta hyvin yksinkertainen, emme katsoneet tarpeeksi varata erikseen aikaa tarkastuspalaveriin. Muutokseni toimi hyvin, mutta ennen kuin se voitaisiin ottaa käyttöön, piti muutoksen näkyä myös front end-puolella, eli ohjelmiston käyttöliittymässä. Iltapäivällä pidettiin perinteinen Sprint-palaveri, jossa kerroimme yrityksen muille tiimeille ohjelmiston käyttöversioon tulevista muutoksista. Koska yrityksemme oli laajentunut ja avannut uuden toimipisteen Yhdysvaltoihin, käytimme videoyhteyttä kommunikoidaksemme uuden toimipisteen työntekijöiden kanssa.

Sprint-palaverin jälkeen kerkesin vielä aloittaa uutta tehtävää, joka liittyy seurantaviikolla 2 työstämääni tehtävään laskenta-algoritmeista. Tässä tehtävässä oli tarkoituksena päivittää loput REST API-funktiot käyttämään uutta laskenta-algoritmiamme.

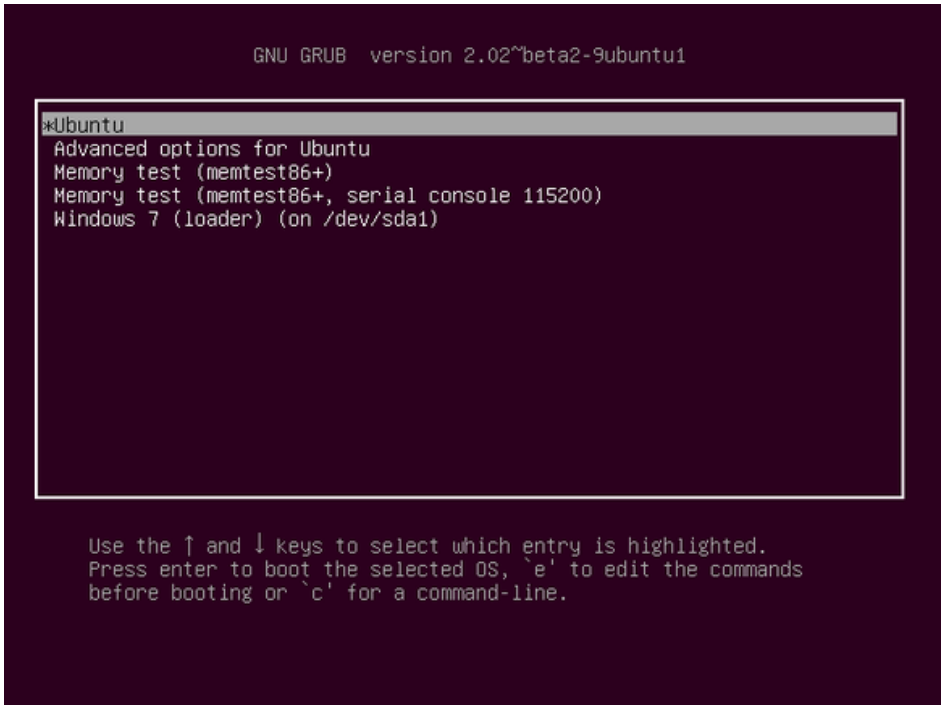
Viikkoanalyysi

Asentaessani uutta työkonettani alkuviikolla, tulen oppineeksi uusia yksityiskohtia Linux-käyttöjärjestelmän toiminnasta ja käytöstä. Samalla saan rutiinia käyttöjärjestelmän asentamiseen, jota en ole tehnyt pitkään aikaan. Asennus tapahtuu USB-tikulle kirjoitetulla Linuxin asennusohjelmalla. Asennettava käyttöjärjestelmänä toimii viimeisin Ubuntu-käyttöjärjestelmä, joka pohjautuu Linuxin Debian-distribuutioon. Cobbaut (2015, 6) määrittelee Linux-distribuution Linuxin ytimen, eli kernelin päällä toimivaksi ohjelmistokokoelmaksi. Linuxin Debian-distribuutio on vain yksi monista Linux-distribuutioista. Cobbautin (2015, 6) mukaan distribuutiolla voidaan koota palvelinohjelmisto, järjestelmätyökalut, dokumentaatio sekä useat työpöytäohjelmat keskitettyyn ja turvalliseen ohjelmistoarkistoon. Debian-distribuutioon kuuluva Ubuntu-käyttöjärjestelmä sopii omaan kehitystyöhöni parhaiten, koska tunnen sen parhaiten Linux-käyttöjärjestelmistä.

Itse käyttöjärjestelmän asentaminen oli hyvin suoraviivaista, vaikkakin jouduin kysymään muilta neuvoa levyosien määrittelyssä. Ongelmia ilmeni, kun halusimme poistaa työkoneeseen esiasennetun Windows 10-käyttöjärjestelmän, mutta jättää sen varmuuskopio-

levyosion talteen. Kun tietokonetta yritettiin käynnistää uudelleen, se ei tuntemattomasta syystä löytänyt Ubuntun käynnistysohjelmaa, jolloin jouduimme käynnistämään Ubuntun USB-tikun kautta. Kokeilimme korjata ongelmaa monin eri tavoin, mutta mikään ei tuntunut tehoavan. Epäilimme ongelman aiheuttajaksi Windowsin varmuuskopio-levyosiota. Koetimme hakea tietoa kyseiseen ongelmaan liittyen, mutta emme löytäneet mitään varmaa tiedonlähdettä, jolla korjaaminen olisi onnistunut. Epäilymme osoittautuivat todeksi, kun päädyimme viimeisenä vaihtoehtona asentamaan käyttöjärjestelmän uudestaan siten, että antaisimme asennusohjelman pyyhkiä myös varmuuskopio-levyosion pois. Tämän jälkeen käyttöjärjestelmä käynnistyi normaalisti, vaikkakin Windowsin varmuuskopio-tiedostot olivat poissa.

Vaikkakin levyjen pyyhkimisen hyvänä puolena oli levytilan vapautuminen, toinen, turvallisempi vaihtoehto asennukseen olisi ollut Ubuntun asentaminen rinnakkain Windows-käyttöjärjestelmän kanssa, jolloin mitään Windowsin tiedostoista ei olisi menetetty. Hoffmania (2015) lainaten se on ideaalisin tapa asentaa Linux-käyttöjärjestelmä, koska pääsy takaisin Windows-järjestelmään on aina mahdollista uudelleenkäynnistyksen avulla. Tosin, ongelmien ilmetessä omassa asennuksessani tämän vaihtoehdon valitseminen olisi todennäköisesti ollut jo liian myöhäistä, jotta korjaustoimenpiteiltä olisi vältytty. Kun Linux asennetaan rinnakkain Windows-käyttöjärjestelmän kanssa, voi käyttäjä valita, tietokonetta käynnistäessään, kumman käyttöjärjestelmistä ajaa. Oheisessa kuvassa näkyy käyttäjälle käynnistyksen yhteydessä näytettävä GRUB-käynnistysmenu (Kuva 2).

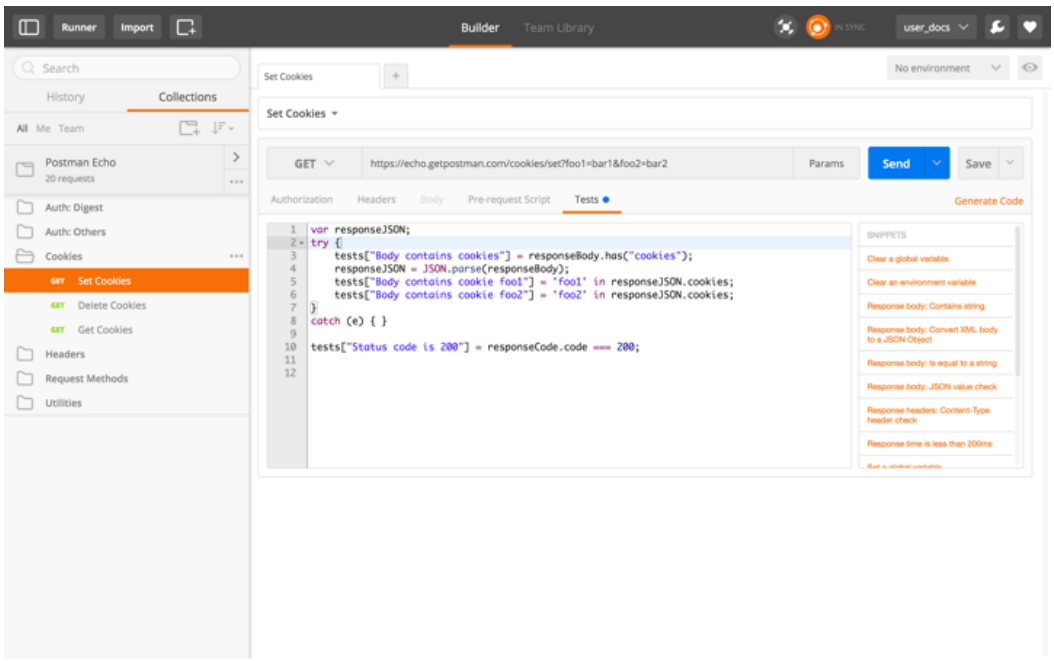


Kuva 2. GRUB-käynnistysmenu (Hoffman, 2015).

Viikon toiseksi merkittäväksi oppimisaiheeksi valitsin testaamisen, jossa löysin kokonaan uusia ulottuvuuksia, ottaessani käyttöön JavaScriptillä kirjoitettavat REST API-testit. Tässä palaan seurantaviikolla 2 käyttämääni teemaan, monipuoliseen testaukseen.

Kirjoitin testejä Google Chrome-selaimen Postman-liitännäisellä, jolla testien kirjoittaminen onnistui ilman suurempaa vaivaa. Postman luo ympäristön, jolla voi kirjoittaa ja ajaa testejä jokaiselle API-pyynnölle ilman huolta mistään ylimääräisistä asennuksista (Postdot Technologies, Inc 2016). API-testeillä tulin testanneeksi ohjelmistomme API-funktioita tarkistamalla, vastasiko funktioista palautettu JSON-data odotuksia. Loin monta eri testitapausta, joissa testasin muun muassa, onnistuiko API-kutsu ylipäätään, täsmäsikö palautettu data tiettyä arvoa sekä pysyikö APIn vastausaika alle tietyn aikarajan. Loin testitapauksia, joilla testattiin, ettei API palauttanut myöskään ylimääräistä dataa.

Samalla kun kehitin omia testaustaitojani, kehitin myös JavaScript-tuntemustani, mikä saattaisi olla tulevaisuudessa hyödyksi myös silloin kun joudun auttamaan front endin ongelmien ratkonnassa. Tärkeintä oli kuitenkin, että opin syntaksin riittävän hyvin testaamista varten. Testaamisessa oli tärkeää, etten testannut yhtä funktiota vain tietyillä parametreilla, vaan loin useita tapauksia useilla eri argumenteilla, joita funktioille syötettiin. Postmanissa tämä oli helppo toteuttaa luomalla ns. kokoelmia, jotka käsittivät tietyn API-funktion url-osoitteen, siihen käytettävät kirjautumistunnukset, sille syötettävät argumentit sekä sille ajettavat testit, jotka suoritettaisiin API-kutsun jälkeen. Alla olevassa kuvassa näkyy esimerkki Postmanilla toteutettavista testeistä (Kuva 3).



Kuva 3. Testien toteuttaminen Postman-liitännäisellä (Postdot Technologies, Inc 2016).

Kun olin saanut testit toimimaan tietyin argumentein, pystyin kokeilemaan, miten argumenttien muuttaminen vaikutti niiden läpäisemiseen. Lopulta, kun kaikki vaikutti olevan kunnossa, tallensin Postmanilla luomani testiympäristöt JSON-muotoon, jotta voisin tallentaa tekemäni testit myös versionhallintajärjestelmään. Tällä tavoin toinen kehittäjä voi avata luomani testiympäristöt ja ajaa niiden sisältämät testit omalla koneellaan. Totesin Postmanin olleen suuri kehitysaskel monipuoliseen testaamiseen niin omalla kohdallani, kuin projektimme kannalta.

3.5 Seurantaviikko 5

Maanantai 27.03.2017

Jatkan siitä, mihin jäin perjantaina, eli uusien laskenta-algoritmien käyttöönottamisesta useammissa eri paikoissa. Kerkesimme tiimitoverini opastuksella toteuttaa jo mahdollisen ratkaisun, mutta se vaatisi vielä testaamista.

Testatessani uutta muutostani huomasin funktion palauttavan dataa oudossa muodossa, eikä tiimikaverinikaan osannut auttaa asiaan. Sitten huomasin, että tulin vain yksinkertaisesti kutsuneeksi väärää päätettä samassa API-ohjelmassa, jolloin ongelma ratkesi. Muistutuksena itselleni, että mikäli ohjelman toiminnasta katoaa kaikki järki, on syytä palata perusasioiden äärelle. Tällä kertaa ongelman ratkomiseen tosin ei mennyt liian kauaa. Sen jälkeen ratkaisu todettiinkin toimivaksi.

Toinen tiimimme front end kehittäjistä, joka työskentelee etänä, kommentoi viikonloppuna viime viikolla valmistunutta ratkaisuani tehtävään, jossa työstin uutta versiota API-ohjelmastamme noutamaan aikaleimoja ja muuta dataa eri aikavyöhykkeiltä. Hän pyysi minua dokumentoimaan uuden version käyttöohjeen, jonka olin unohtanut tehdä. Uudessa API-versiossa joitakin funktioille annettavia parametreja oli lisätty, poistettu tai nimetty uudelleen, mutta en ollut selittänyt niiden merkitystä mitenkään. Tämän takia uuden ominaisuuden käyttöönotto viivästyi. Onneksi minulla oli valmiit esimerkit laadittuna Postman-ohjelmassa, jotka pystyin kätevästi selittämään tiimitoverilleni. Hän kiitti minua pian hyvästä käyttöohjeesta, mutta kertoi sen jälkeen ohjelmassa esiintyvästä virheestä hänen kokeillessaan APIa etäpalvelimella.

Pienen selvittelyn jälkeen huomasin, että olin myös unohtanut lisätä uuden API-funktion päätepisteen niin sanottuun keskus-APIin, joka ajettiin etäpalvelimella, toisin kuin omalla koneellani olin tähän mennessä ajanut vain API-ohjelmistomme tiettyjä osia. Pienen korjausliikkeen kautta päädyin luomaan uuden pull requestin versionhallintajärjestelmäämme,

koska aiemmin valmistunut ratkaisuni oli jo liitetty ohjelman käyttöversioon. Kohtasin ensimmäistä kertaa versio-konfliktin uudessa työssäni. Vaikka en ollut selvittänyt konflikteja pitkään aikaan, tämä sujui kuitenkin ilman suurempia ongelmia. Käytin tosin apuna versionhallinnan dokumentaatiota väliajoin.

Kun aiemmat virheeni oli saatu korjattua, pääsin aloittamaan tehtävää, jossa tarkoituksena oli luoda uusi API, joka vastaanottaisi käyttäjältä kuva-tiedoston ja lähettäisi sen varastoitavaksi Amazon Web Servicesin ylläpitämään tiedostojen varastointi järjestelmään. Pääsin ensimmäistä kertaa tekemisiin Amazon Web Services-tekniikan kanssa, mikä olikin uutta ja mielenkiintoista vaihtelua omiin työtehtäviini.

Tiistai 28.03.2017

Tulin eilen loppupäivästä tehneeksi jonkin verran tutkimusta liittyen uuteen tehtävääni ja löysinkin jo hyviä esimerkki-ratkaisuja aiheeseen liittyen. Sain uuden API-funktion rungon kasaan jo eilen, mutta lisättävää ja muutettavaa vielä varmasti löytyisi.

Jouduin hieman korjailemaan uuden API-funktion asetuksia ennen kuin sain sen ajettua virheittä. Testaamisvaiheessa kohtasin toki uusia virheitä lähdekoodissani. Vaikka en tässä vaiheessa vielä ymmärtänytkään kaikkia yksityiskohtia esimerkin pohjalta luomastani ominaisuudesta, pystyin virhesanomien avulla ja pienen tiedonhaun avulla ymmärtämään tärkeimpien osien toiminnan ja korjaamaan virheet. Muutaman korjauksen jälkeen sainkin jo API-funktion toimimaan täysin kuvan lähettämisessä tiedostojen varastoimispalveluun. Löysin Postman-liitännäisestä kätevän ominaisuuden kuvan lähettämiseen tiedosto-objektina, jossa minun tarvitsi vain kertoa sille kuvan tiedostopolku omassa käyttöjärjestelmässäni. Se että en kohdannut vaikeampia ongelmia itselleni uuden tekniikan kanssa yllätti minut. Näyttäessäni ratkaisun projektipäällikölle, oli hän tyytyväinen ominaisuuden toimivuuteen, mutta pyysi vielä muutoksia kuvien nimeämiskäytäntöihin sekä selkeyttämään apin käyttöä.

Kollegallani kehitystiimin front end puolella sen sijaan oli suurempia ongelmia oman ratkaisunsa kanssa liittyen uuteen kuvan tallennus-ominaisuuteen. Kyse oli itseni kanssa samaan aikaan yrityksessä aloittanut front end-kehittäjä, joka työskentelee samassa toimitilassa kanssani. Hänen tehtävänään oli lähettää käyttöliittymässä tehty kuva tekemääni API-funktioon argumenttina, mutta tehtävä osoittautui erittäin haastavaksi. Ongelmana oli argumentin lähettäminen tiedosto-objektina JavaScriptillä, mihin emme löytäneet selkeää ratkaisumallia. Yritin parhaani mukaan auttaa ongelman selvittämisessä, kunnes projektipäällikkö tuli mukaan avuksi.

Koska olin edennyt omissa työtehtävissäni hyvin, sain loppupäivästä seuraavaksi opiskella itselleni uutta kehityksen alla olevaa Kinesis Client-projektia, jossa käytettiin Amazon Web Servicesin tarjoamaa Kinesis-tekniikkaa.

Keskiviikko 29.03.2017

Tiimitoverini selitti minulle tärkeimmät osat projektista, jota opiskelin. Tänään saan toivottavasti riittävästi ymmärrystä projektin toiminnasta, mahdollisesti jo puoleen päivään mennessä, jotta pääsen työstämään ensimmäistä kehitystehtävää liittyen projektiin.

Tämä päivä sujui enimmäkseen ongelmien parissa. Olin suuren osan ajasta auttamassa kollegaani kehitystiimin front end-puolelta, jolla oli edelleen ongelmia kuvan lähettämisen kanssa API-funktioon. Ongelmaan ei millään tuntunut löytyvän toimivaa ratkaisua. Syynä tuntui olevan tapa, jolla kuvaa käsiteltiin käyttöliittymässä. Myöhemmin pidimme toisen kokeneemman front end kehittäjän kanssa videopalaverin kysyäksemme neuvoa ongelmaan.

Tulin päivän aikana opiskelleeksi lisää Amazonin Kinesis-tekniikasta ja opin lisää Kinesis Client-projektin toiminnasta opiskelemalla lähdekoodia. Yrittäessäni testata ohjelman ajoa kohtasin kuitenkin ongelmia. Vaikka olin määritellyt ohjelman ajoon tarvittavat kirjautumisasetukset ennalta määriteltyyn kokoonpanotiedostoon, ongelma vaikutti silti olevan, että ajettava ohjelma ei löytänyt sitä. Kokeneempi tiimitoverinikaan ei kyennyt löytämään ongelmaan ratkaisua, joten päätimme jättää ongelman ratkonnan huomiselle, koska projektipäällikkö oli tänään poissa töistä. Hän tietäisi ongelmasta todennäköisesti enemmän.

Torstai 30.03.2017

Yritän selvittää ongelman liittyen kirjautumistietoihin Kinesis Client-projektissa. Toivon mukaan projektipäällikkö on saatavilla avuksi, jotta pääsen ohjelmiston kehittämisessä eteenpäin.

Jatkoimme ongelman ratkomista pitkälle puoleen päivään. Projektipäällikkönikään ei löytänyt ongelman tarkkaa syytä, mutta heti alussa selvisi, että syy ei liittynyt kirjautumistietojen löytymiseen. Syynä oli, että Python-ohjelma joka ajettiin sisäisesti Java-ohjelman kautta, ei tuottanut minkäänlaista tulostetta ohjelman toiminnasta, kun sen olisi pitänyt tulostaa paljonkin tietoja sen toiminnasta. Lopulta tulin tarkistaneeksi oman koneeni Javan version ja vertasin sitä projektipäällikön koneen Java-versioon, jossa ohjelma toimi kuten piti. Sel-

visi, että minulla oli uudempi Java-versio, joten päätin varmuuden vuoksi poistaa uudemman Javan koneeltani ja asentaa vanhemman version tilalle. Heti, kun sain vanhan version Javasta asennettua, ajoin ohjelman, joka alkoi heti tulostaa tietoja. Vaikka tästä ongelmasta ei oltu mainittu missään dokumentaatiossa, sain jälleen kerran osoituksen siitä, että Javan versio voi aiheuttaa hyvin paljon komplikaatioita ohjelmien toiminnassa.

Ongelman ratkettua pääsin tarkemmin tutkimaan ohjelman toimintaa. Tehtävänäni oli käyttää Pythonin sisäänrakennettua loki-moduulia, jolla pystyisin tallentamaan tietoja ohjelman toiminnasta erillisiin lokitiedostoihin, jotta tiedon prosessointia voitaisiin tarkastella niistä käsin. Jouduin hieman kertaamaan loki-tekniikkaa Pythonilla, vaikka olin kohdannut sitä ennenkin.

Perjantai 31.03.2017

Sain eilen tietojen tallennuksen lokeihin toimimaan osittain. Ongelmia oli kuitenkin itse ohjelman toiminnassa, joka tuntemattomasta syystä ei kutsunut tiettyjä ajastettuja funktioita, joissa lokitallennusta tuli tehdä. Saatan joutua pyytämään neuvoa tiimitoveriltani tämän selvittämisessä.

Ongelmamme Kinesis Client-ohjelmassa saatiin lopulta korjattua, kun muuntelimme sen käyttämää asetuskokoonpanoa. Totesimme, että järjestelmässämme olisi tulevaisuudessa vielä paljon kehitettävää, jotta asetuskokoonpanojen muokkaamiselta vältyttäisiin jatkossa. Seuraavaksi pystyin testaamaan toteuttamaani lokitallentamista ajamalla ohjelman. Käytin Pythonin sisäänrakennettua lokityökalua tulostamaan tietoja ja tapahtumia ennalta määrittelemääni lokitiedostoon. Yksi virhe, jonka sen käyttämisessä aluksi tein, oli se, että en asettanut lokitallentimen viestimistasoa, joka määrittelee sen, minkä tyyppiset viestit tallentuvat tiedostoon, kuten info- tai virhe-tyyppiset viestit. Määrittelemätön viestimistaso aiheutti sen, että mitään viestejä ei tulostettu lokitiedostoon.

Toteutettuani Kinesis-lokitallentimen päädyin jatkokehittämään jälleen omaa API-järjestelmäämme. Tällä kertaa kyseessä tosin oli uudemman version toteuttaminen ohjelmistostamme. Koska muutin aiemmin laskenta-algoritmien logiikkaa vanhassa API-versiossa, sain tehtäväkseni päivittää laskenta-algoritmit myös uudessa API-versiossa. Toteuttaminen sujui suoraviivaisesti, mutta minun pitäisi vielä testata muutosta ensi viikolla.

Viikkoanalyysi

Tällä viikolla kohtasin paljon erilaisia haasteita ja erikoisia tilanteita liittyen monenlaiseen viestintään ja kommunikaatioon. Siksi päätin valita viikon teemaksi ennakoivan ja selkeän viestinnän.

Edellisellä viikolla työskentelin ahkerasti suuremman tehtävän parissa, jossa muunnettiin tietokannasta löytyviä aikaleimoja eri aikavyöhykkeille. Uusi funktio, jonka lisäsin API-järjestelmään, toimi pohjimmiltaan samalla tavalla kuin toinen funktio kyseisessä järjestelmässä, mutta tämän funktion oli tarkoituksena toimia uutena versiona kyseisestä funktiosta. Uuden funktion käyttämisessä oli kuitenkin monia eroavaisuuksia verrattuna aiempaan versioon. Tästä syystä en olisi voinut olettaa, että toinen työntekijä, joka ei ole ollut mukana ratkaisuni kehittämisessä voisi tietää, miten kyseistä funktiota käytettäisiin. Kuten Tervola (2015) toteaa, joissakin yrityksissä tunnutaan edelleen uskovan, että tieto lentää kuin itsestään työpisteestä toiseen. Tämä tuntui kyseisellä hetkellä pätevän omalla kohdallanikin.

Olin toki maininnut argumenttien tekniset tiedot lähdekoodissa. Kuten Haapajärvi (2012, 22) sanoo Pro gradu-tutkielmassaan, dokumentointia tehtäessä pitäisi miettiä kuka dokumenttia tarvitsee. Front end kehittäjällä ei aina välttämättä ole kykyjä lukea funktioni toimintaa pelkän lähdekoodin avulla. Toteutukseni puutteellinen dokumentaatio vaikutti suoraan kehitystiimini toimintaan, koska front end-kehittäjä ei osannut käyttää uutta toteutustani. Ennakoivalla dokumentoinnilla toteutukseni olisi voitu ottaa käyttöön jo viikonlopun aikana, mutta virheeni myötä käyttöönotto viivästyi.

Dokumentaation puute aiheutti myös sen, että toteutustani ei päästy kunnolla testaamaan käyttöliittymästä käsin, jolloin tekemäni ohjelmointivirhe paljastui vasta saatuani dokumentaation kuntoon. Olisin voinut vaihtoehtoisesti myös viestittää toteutukseni käyttäjälle yksinkertaisen käyttöohjeen jollakin kommunikaatiovälineellä, joka olisi ajanut saman asian tilapäisesti. Tästä opin sen, että yrityksen sisäisellä viestinnällä, tässä tapauksessa työn dokumentoinnilla on suuri merkitys myös ohjelmistokehityksessä, koska sen puutteet voivat aiheuttaa ongelmia ja viivästyksiä muiden kehitystiimin jäsenten työssä. Puutteellinen dokumentaatio voi aiheuttaa myös muita ongelmia epäsuorasti, kuten ohjelmointivirheiden löytymistä myöhään tässä tapauksessa.

Myös itse lähdekoodin kommentointia voidaan pitää tärkeänä dokumentoinnin kannalta. Ohjelman lähdekoodia harvoin tulee kehittämään ainoastaan yksi ja sama henkilö läpi yrityksen elinkaaren. Kommentit lähdekoodissa viestivät muille ohjelmistokehittäjille siitä,

miten koodi toimii ja miten sitä käytetään. Haapajärveä (2012, 17) mukaillen ohjelmistoa kehitettäessä pelkkä koodi ei yksin ole riittävä vaan se tarvitsee lisäksi hiukan tekstiä, jotta se olisi ymmärrettävää. Omassa työssäni tämäkään puoli ei aina toteutunut, mutta pyrin ainakin lisäämään kommentteja aina, kun koodin rakenteessa oli erikoisia piirteitä, jotka johtuivat jonkin toisen ohjelmiston osan erityisestä piirteestä.

Toinen erikoinen tilanne liittyi kehittämääni API-funktioon, jossa käyttäjä pystyi lähettämään kuvatiedoston Amazon-tiedostovarastoon. Tiimimme uudempi front end-kehittäjä yritti liittää omaa käyttöliittymäratkaisuaan API-funktiooni. Myös tässä tilanteessa dokumentaationi oli puutteellista, mutta sillä oli vähemmän merkitystä, sillä funktioni tarvitsi vain yhden parametrin, tiedosto-objektin. En kuitenkaan voinut tietää, missä muodossa kuva olisi käyttöliittymässä, joten minun oli tehtävä selväksi kehitystiimini front end-kehittäjille, missä muodossa heidän pitäisi kuva lähettää API-funktiolle. Se mitä tein kuitenkin oikein, oli se, että osallistuin kyseisen ongelman ratkontaan front end-kehittäjien kanssa. Pidimme aiheeseen liittyen videopalaverin, jossa esittelin APIn toimintaa jakamalla näyttökuvani muille kehittäjille, jolloin he saivat paremman käsityksen APIn toiminnasta.

Se viestintäpuoli, missä onnistuin kuluneella viikolla, oli reaaliaikainen viestintä Slack-viestintätyökalulla. Kun sain tiimitoverini mainitsemat viat korjattua, tulin heti kirjoittaneeksi hänelle osoitetun viestin Slackissa. Tällä tavoin hän ei jäänyt epätietoisuuteen siitä, milloin hänellä olisi lupa ottaa ratkaisuni uudelleen käyttöön käyttöliittymässä. Vaikka kirjoitinkin kohdennetun viestin, käytin siihen kehitystiimin kanavaa, sen sijaan että olisin kirjoittanut hänelle yksityisviestin. Tällä tavoin muutkin tiimitoverini olivat tietoisia tehtävän kulusta.

Se, miten parhaiten voisin parantaa viestintää kehitystiimini sisällä olisi aikatauluttaa tärkeimmät viestintään liittyvät toimenpiteet. Määrittelemällä etukäteen esim. milloin luon toteutukseni dokumentaation, voisin jatkossa auttaa itseäni olemaan unohtamatta työni dokumentaatiota. Dokumentaation tekeminen muodostuisi lopulta rutiiniksi. Tilannetta voisi helpottaa se, että aloitan työni dokumentaation jo tehtävän toteuttamisen aikana, jolloin voisi paremmin huomata mitä on jäänyt tekemättä, mikäli yritän määritellä tehtävää valmiiksi millään tavoin.

> *”Entä miten välttäisi sisäisen viestinnän sudenkuopat? Suunnittelulla, tietysti. Sisäinen viestintä pitää aikatauluttaa kuin muukin yrityksen liiketoiminta. Täytyy laatia strategia mitä kerrotaan, koska, milloin, millä välineellä ja kenelle.” (Tervola 2015.)*

3.6 Seurantaviikko 6

Maanantai 03.04.2017

Lomapäivä.

Tiistai 04.04.2017

Tulen testaamaan viime perjantaina tekemäni muutoksen API-järjestelmän uudessa versiossa. Odotan testauksen sujuvan suoraviivaisesti, kuten viimeksikin.

Vastoin odotuksiani, muutokseni testaaminen aiheutti tavallista enemmän hankaluuksia. Aluksi ongelmat johtuivat väärin nimetyistä ja sijoitetuista asetustiedostoista, mutta sen jälkeen hankaluuksia aiheuttivat uudessa API-versiossa käytettävät tietokantaskeemat. Koska käytettävä tietokantaskeema ei sisältänyt kaikkia tarvittavia tietoja aikaisemmin käyttämästäni skeemasta, jouduin päivittämään kyseisen skeeman taulut uusilla riveillä. Lopulta sain kuitenkin funktioni toimimaan, mutta huomasimme tiimitoverimme kanssa, että funktio ei palauttanut odotettua dataa. Tutkimme lähdekoodia hieman ja huomasimme että jotkin tietokantakyselyistä tarvitsisivat vielä lisää suodattimia.

Iltapäivällä projektipäällikkö kokosi kehitystiimini back end-jäsenet kokoukseen, jossa suunnittelimme kokonaan uutta, joskin alkuun pienikokoista projektia kehitettäväksi. Projektina oli luoda uusi työkalu, jota käyttäisimme ns. ’bufferina’ aiemmin käyttämäni Kinesis-tekniikan kanssa. Työkalulla pystyisi jaksottamaan Kinesiksen tuottamaa dataa tiettyyn aikaikkunaan sekä lukemaan aikaikkunoihin sijoitettua dataa. Kokouksessa kävimme läpi eri toteutusvaihtoehtoja samalla kun projektipäällikkö koulutti meitä paremmin ymmärtämään Kinesiksen toimintaa, joka varsinkin itselleni oli vielä melko tuntematonta, vaikka sainkin jo jonkinlaisen peruskäsityksen aikaisemmassa Kinesis-tehtävässäni.

Keskiviikko 05.04.2017

Sain eilisen päätteeksi vastuulleni uuden Kinesis-tehtävän luomisen ja aloittamisen. Tarkoituksena oli tehdä ns. buffer-työkalu, johon voisi syöttää ja vetää ulos sen ryhmittelemää dataa tietyin aikavälein. Loin JIRA-työkalulla uuden tehtävän jossa selitin uuden projektin tarkoituksen. Tänään pyrin saamaan muut työn alla olevat tehtävät pois alta, jotta pääsen aloittamaan uuden projektin suunnittelun ja toteuttamisen.

Tutkin uudemman API-järjestelmän versiota vertailemalla sen lähdekoodia vanhempaan versioon. Ongelmiemme syytä oli aluksi vaikea löytää, mutta lopulta löysimme ongelman juuren API-järjestelmien eriävistä tavoista syöttää parametrit tietokannan käsittelijä-funktioille.

Iltapäivästä pääsin aloittamaan uutta Kinesis buffer-työkaluprojektia, josta puhuimme jo eilisessä kokouksessa. Tämä oli ensimmäinen kerta, kun pääsin aloittamaan uuden ohjelmistoprojektin työstämistä alusta alkaen. Projektipäällikkö hoiti toki projektikansion luomisen versionhallintajärjestelmään, mutta muuten sain vastuulleni hoitaa projektin alussa tarvittavat osat ja asetukset. Sain hyvää rutiinia uuden Git-kansion kuntoon laitossa, jota en ollut tehnyt pitkään aikaan. Tarvittavia toimenpiteitä olivat muun muassa gitignore-tiedoston määrittäminen, jolla pystyin määrittämään versionhallintajärjestelmästä pois jätettävät tiedostot ja hakemistot, sekä ohjelman main-funktion luonti.

Tarvittavien aloitustoimenpiteiden jälkeen luonnostelin ohjelman ydintoiminnan paperille. Konsultoin projektipäällikköä muutamaan otteeseen, kun halusin varmistusta projektiin liittyvissä yksityiskohdissa. Kun olin saanut mielestäni riittävän selkeän kuvan toteutettavasta ohjelmasta, aloin toteuttaa ohjelman ydintoimintaa. Loin tehtävään pala palalta lisää toimintoja ja ominaisuuksia, kunnes sain jonkinlaisen perusratkaisun osaan projektista.

Torstai 06.04.2017

Tarkoituksenani on jatkaa eilen työstämääni Kinesis buffer-työkalua. Kerkesin eilen saamaan työkalun syöttö-funktion rungon toteutettua, mutta se vaatisi vielä hieman hiomista.

Päiväni kului pääasiassa Kinesis buffer-työkalun parissa. Olin jo aiemmin saanut valmiiksi työkalun syöttö-metodin, joten nyt työstin sen luku-metodia. Yksi suunnittelutyön suurimmista haasteista oli organisoida palautettavan datan rakenne. Olen tähän mennessä enimmäkseen käyttänyt Pythonin dict-olioita API-järjestelmästä palautettavan datan rakenteen määrittelemiseen, mutta tällä kertaa päätin luoda palautettavalle datalle oman luokan, jota käyttäisin apuna tehtävässä.

Yksi suurimpia haasteita tehtävässä oli iteroida data oikein riippuen funktiolle annetuista argumenteista. Tämä johtui siitä, että funktion palauttamat data-oliot käyttivät avaiminaan aikaleimoja. Minun piti keksiä, miten iteroisin kaikkien aikaleimojen läpi kronologisessa järjestyksessä, kun ne saattoivat olla sijoitettuna bufferiin missä järjestyksessä tahansa. Ongelman ratkaisemiseen löysin avuksi Pythonin sorted-funktion, jolla pystyin järjestä-

mään aikaleimat kronologiseen järjestykseen. Jouduin kuitenkin vielä konsultoimaan omaa projektipäällikköä, koska tarvitsin lisätietoa palautettavan datan yksityiskohdista.

Perjantai 07.04.2017

Lomapäivä.

Viikkoanalyysi

Kulunut viikko oli hieman lyhyempi sisällöltään johtuen ottamistani lomapäivistä. Työstämäni työtehtävät olivat silti opettavaisia ja ammatillista kehittymistä tapahtui siinä missä muinakin viikkoina. Alkuviikosta jouduin testaamaan tekemiäni muutoksia API-järjestelmän uudessa versiossa. Ohjelmaa testatessa löytyi uusia ongelmia lähdekoodin sekä tietokannan rakenteesta, jotka tosin eivät heti selvinneet. Jouduin ratkomaan haastavia ongelmia ja tilanteita, joiden syyt olivat toisinaan hieman epäselviä. Samoin loppuviikosta työskentelin uuden ohjelmaprojektin parissa, jossa kohtasin lukuisia suunnitteluun liittyviä ongelmia. Viikon teemaksi nostan oma-aloitteisen ongelman ratkonnan, mikä onkin helppo valinta johtuen tehtävien luonteesta.

Olen tarvinnut työssäni alusta alkaen ongelmanratkontakykyjä sekä innokasta asennetta työhöni. Kuten Metsänheimo (2016, 3) toteaa opinnäytetyössään, ohjelmointi on pääasiassa jatkuvaa ongelmanratkontaa. Siksi onkin tärkeää, että pyrin lähestymään ongelmaa tehokkain metodein. Ohjelman rakenteelliset ongelmat tulisi pyrkiä ratkaisemaan jo suunnitteluvaiheessa, jotta toteuttamisvaiheessa vältyttäisiin turhilta lisätöiltä. Hyvä suunnittelukaan ei kuitenkaan aina takaa ongelmatonta työtä, sillä ongelmat voivat aiheutua myös esim. muiden toteuttamasta ohjelmointityöstä, tai käytettävistä työkaluista.

Jotta mahdollisuuteni ratkaista ongelmani itsenäisesti paranevat, tulee minun testata toteuttamaani ratkaisua mahdollisimman monipuolisesti. Mikäli itse testit eivät paljasta ongelman juurta, saatan tarvita mahdollisesti debug-työkaluja testaustyössäni. Yksi yleisimmin käyttämistäni debug-työkaluista on Pythoniin sisäänrakennettu pdb-moduuli. Pdb-moduuli määrittelee interaktiivisen lähdekoodin debug-työkalun Python-ohjelmille (Python Software Foundation, 2016). Tällä työkalulla pystyn asettamaan lähdekoodiini ns. välietappeja, jotka käynnistävät ohjelmaa ajettaessa väliaikaisen Python-konsolin. Konsolissa pystyn lukemaan ohjelmassa määriteltyjä muuttujia sekä kokeilemaan niiden toimintaa. Käytännössä näen siis ohjelman sisällöllisen toiminnan tarkasti. Tästä on suurta hyötyä, kun haluan tarkistaa, että määrittelemäni muuttujat ja funktiot toimivat juuri niin kuin oletan niiden toimivan.

Pdb-moduulia tarvitsin kuluneella viikolla varsinkin uudemman API-version kehittämisessä, kun halusin tarkistaa muuttujien arvot ja formaatit API-kutsuja tehdessäni. Toisinaan saatoin käyttää Pythonin sisäänrakennettua print-funktiota tulostamaan muuttujien arvoja suorittaessani tiettyjä metodeita. Näin pystyin kokeilemaan esim. missä kohtaa jokin ohjelman funktio lakkaa toimimasta, kun ohjelma ei tulosta tiettyä määrittelemääni string-lausetta.

Kuluneella viikolla sain paljon uutta arvokasta kokemusta ja rutiinia ongelmanratkonnasta Pythonilla. Vaikka olinkin ajoittain hämmentynyt kehittämieni ohjelmien oudoista käyttäytymisistä, selkenivät ne viimeistään käyttäessäni apuna monipuolista testausta ja debug-työkaluja. Usein saatan tarvita kyseisiä metodeita pelkästään auttamaan itseäni ymmärtämään monimutkaisten ohjelmien toimintalogiikkaa.

3.7 Seurantaviikko 7

Maanantai 10.04.2017

Jatkan buffer-työkalun kehitystyötä. Nyt viime viikosta viisastuneena toivon pääseväni sujuvasti eteenpäin ratkaisun toteuttamisessa.

Pidemmän viikonlopun palautumisen ansiosta minulla oli hyvä keskittymiskyky heti aamusta alkaen. Viime torstaina tekemistäni muistiinpanoista oli nyt selkeästi hyötyä. Vielä torstaina osa tehtävääni liittyvistä seikoista oli vielä epäselkeitä, mutta nyt kerratessani tehtävää tuoreella mielellä, pystyin heti havainnoimaan ohjelmani tarvitsemat puutteet ja muutokset. Nyt tiesin myös tarkalleen mitä kysymyksiä minun piti vielä esittää projektipäällikölle, jotta saisin ratkaisuni toimimaan hänen odottamallaan tavalla.

Tehtäväni vei silti lähes koko päivän aikaa. Työtä oli paljon varsinkin aikaleimojen iteroimisen toteuttamisessa, kun minun piti myös asettaa bufferin ikkunan luku-metodille muuttuja, joka määrittelee, mistä aikaleimasta metodi aloittaa laskemaan aikaleimoja. Aikaleimat tallennettaisiin bufferiin minuutin tarkkuudella, joten minun piti iteroida niitä minuutin jaksoissa. Tämä onnistui hyvin Pythonin datetime-moduulin timedelta-olioilla, joiden koon pystyin määrittelemään yhdeksi minuutiksi. Lisätessäni yhden minuutin kokoisen timedelta-olion iteroija-muuttujaan, sain kasvatettua sitä yhdellä minuutilla, ja niin edelleen.

Odottaessani projektipäällikön vapautumista tehtäväni tarkastukseen, pyysin tiimitoveriltani uutta tehtävää, jota voisin silmäillä sillä aikaa. Sain tehtäväkseni ensimmäisen bugin korjaamisen työssäni. Etänä työskentelevä front end-kehittäjä raportoi, että sivu jossa

sisään kirjautuvalle käyttäjälle näytettiin perustiedot ei näyttänyt käyttäjälle back endissä määriteltyä tervetuloa-viestiä. Tulin testanneeksi APIa, josta viesti ja muut tiedot palautettiin, mutta totesin viestin palautettavan siitä normaalisti. Tulin siis kommentoineeksi tehtävää testini kuvankaappauksen kera, jossa totesin, että virheen täytyy liittyä front endiin. Kenties joskus toiste pääsisin korjaamaan toista bugia.

Päivän lopussa tarkistimme projektipäällikköni kanssa Kinesis buffer-ohjelmani toiminnan. Hän oli tyytyväinen ratkaisuuni, mutta keskusteltuamme hetken totesimme, että voisimme toteuttaa automatisoidut yksikkötestit testaamaan ohjelmani toimintaa. Testit lisättäisiin suoraan osaksi tätä tehtävää. Testit olisi helppo toteuttaa nyt, kun toteutus on vielä tuore.

Tiistai 11.04.2017

Tulen toteuttamaan päivän aikana yksikkötestit tekemääni Kinesis buffer-moduuliin. Saatan joutua kysymään ohjetta projektipäälliköltä, miten hän on tähän mennessä toteuttanut omat yksikkötestinsä.

Sain yksikkötestit toteutettua nopeasti ja suoraviivaisesti heti sen jälkeen, kun projektipäällikkö oli näyttänyt minulle, kuinka testit voi ajaa suoraan PyCharm-tekstieditorista. Pyrin tekemään vähintään yhden testitapauksen jokaista moduulini käyttötapausta varten. Tässä tapauksessa se tarkoitti vähintään yhtä testitapausta jokaisen luokan metodille. Tulin testanneeksi mm. ikkunan luomisen onnistumista, datan syöttömetodin toimimista bufferissa sekä ikkunan lukemista bufferista. Testasin myös, että metodit palauttivat poikkeuksen tai ei mitään, mikäli annoin metodeille virheellisiä argumentteja.

Testien toteuttamisen jälkeen tulin tutkineeksi toista bugia järjestelmässämme. Kyseessä oli ohjelmistomme admin-työkaluun liittyvä bugi, jossa tiettyjen tietokantaolioiden uudelleennimeäminen ei toiminut tietyssä paikassa. Tulin analysoineeksi API-kutsujen toimintaa web-selaimeni kehittäjätyökalupaneelista. Jotkin API-kutsuista epäonnistuivat tuntematomasta syystä. Kun kuitenkin testasin niitä Postman-liitännäisellä, kaikki tuntui toimivan hyvin. Bugi vaikutti lopulta liittyvän front endiin, jolloin tulin kommentoineeksi tehtävää muutaman näyttökaappauksen kera.

Loppupäivästä sain tehtäväkseni aloittaa vanhan datan siirtämisen Amazon Web Services-palvelun RDS-tietokantapalvelimelta Amazonin Redshift-tietokantapalvelimelle. Tarkoitus oli siirtää data käyttämällä Amazonin omaa DMS-palvelua, eli Data Migration Serviceä. Minulta kuitenkin puuttui tarvittavat käyttöoikeudet näiden palveluiden käyttöön, joten jouduin odottamaan projektipäällikköä niiden asettamisessa.

Keskiviikko 12.04.2017

Tulen työskentelemään eilen mainitsemani datan siirtotehtävän parissa. Koska tehtävässä käytettävä tekniikka on minulle lähes täysin uutta, odotan tehtävän olevan todella haastava itselleni.

Datan siirtotehtävän ohessa tulin päivän aikana opiskelleeksi lisää käyttämästämme Amazonin Kinesis-tekniikasta. Sain tehtäväkseni luoda yksikkötestit bufferi-tehtävän tapaan myös Kinesikseltä saadun datan prosessointiohjelmalle. Aina kun minulla oli ylimääräistä aikaa, kuten odottaessa projektipäällikköä, pyrin suunnittelemaan toteutettavia yksikkötestejä ohjelmaamme.

Datan siirrossa ongelmia ilmeni heti alussa, kun tietokantapalvelimien yhteyksissä ilmeni vikaa. Aluksi viat johtuivat vääristä asetuksista, mutta pian ilmeni, että RDS-tietokantapalvelimen käyttämää PostgreSQL-tekniikkaa ei tuettu Amazonin DMS-palvelussa. Tämä esti datan siirtämisen kokonaan DMS-palvelun avulla. Jouduimme siis keksimään muun tavan siirtää data pavelimelta toiselle.

Kuulin analytiikkatiimissä työskentelevän kollegani käyttäneen työssänsä aiemmin jotain työkalua datan siirtämisessä, joten tulin konsultoineeksi häntä. Hän esitteli minulle tuntemansa Pentaho-työkalun, jolla pystyi kätevästi tekemään monipuolisia datan siirto- ja muunnostehtäviä. Pienen esittelyn jälkeen ymmärsin ohjelman käyttämisen perusteet, mutta ohjelmassa oli huomattava määrä toimintoja, joiden vaikutuksia saatoin vain arvailla. Tuleva datan siirto olisi loppupeleissä yksinkertainen, koska kyseessä oli tiedon siirto ainoastaan yhdestä tietokantataulusta, mutta ohjelmalla pystyisi tarvittaessa tekemään suuria ja monimutkaisiakin datan siirroksia. Dataa pystyisi samalla muuntelemaan haluamallaan tavalla siirtämisen välissä.

Torstai 13.04.2017

Kerkesin eilen saamaan tietokantayhteydet kuntoon Pentaho-työvälineellä. Tulin testanneeksi nopeasti tiedonsiirtoprosessia. Ongelmana oli, että lähdetietokannassa olevan tietokantataulun aikaleimaa esittävä sarake oli määritelty aikavyöhykkeen sisältäväksi aikaleimaksi. Kohdetietokannassa aikaleima-sarake oli määritelty ilman aikavyöhykettä. Joudun tekemään datan muunnosta siirtämisen välissä.

Aamulla pidimme pyynnöstäni palaverin analytiikkatiimissä työskentelevän kollegani kanssa. Palaverissa hän antoi minulle perusteellisemman opastuksen Pentaho-ohjelman

käyttämiseen. Palaveria ennen minulla oli vain aavistus siitä, mitä työvälineellä pystyisi käytännössä tekemään, mutta kiitos kokeneen kollegani hyvän opastuksen, sain paremman käsityksen käyttämäni ohjelmiston toiminnasta.

Palaverin jälkeen alkoi varsinainen ongelmanratkonta datan siirtämisessä. Kollegani neuvoi minua muuntamaan aikaleima sopivaan muotoon raa'alla merkkijonon 'search and replace'-toiminnolla. Tällä pystyin hakemaan aikaleimoista eri symboleita ja korvaamaan ne toisella, jotta sain ne tarvittavaan muotoon. Projektipäällikkö huomautti kuitenkin, että puhtaampi tapa tehdä muunnos olisi tehdä konversio suoraan lähdetietokannan sql-lukulausekkeessa.

Saimme lopulta tiedonsiirron toimimaan, mutta seuraavana ongelmana oli, että datan syöttäminen kohdetietokantaan oli aivan liian hidasta. Selvisi, että tämä johtui ensi sijassa siitä, että kohdetietokanta oli eri verkossa kuin oma paikallinen työasemani. Tällöin verkkojen välityksellä tapahtuva datan syöttö tietokantapalvelimeen rajoittui vain 5 riviin per sekunti, kun tavoitteena oli lukea tuhansia rivejä sekunnissa. Redshift-tietokantapalvelin ei myöskään ollut suunniteltu ensi sijassa tällaiseen käyttöön.

Kokeilimme useita eri ratkaisuja, kuten datan siirtämistä tietokantapalvelimen kanssa samassa verkossa olevalta virtuaalikoneelta, mutta vauhti oli vieläkin liian hidas. Lopulta päätimme vaihtaa ratkaisumallia. Sen sijaan, että olisimme syöttäneet dataa suoraan tietokannasta toiseen, päätimme syöttää lähdetietokannasta luetun datan tekstitiedostoon, ja ladata se Amazon S3-tiedostovarastoon, josta voisimme sitten ladata datan kohdetietokantaan. Kerkesimme kokeilla uutta ratkaisumallia pikaisesti, mikä vaikutti jo paljon lupaavammalta. Päivä oli kuitenkin jo lopussaan, joten varsinainen siirto tapahtuisi kuitenkin vasta pääsiäisen jälkeen.

Perjantai 14.04.2017

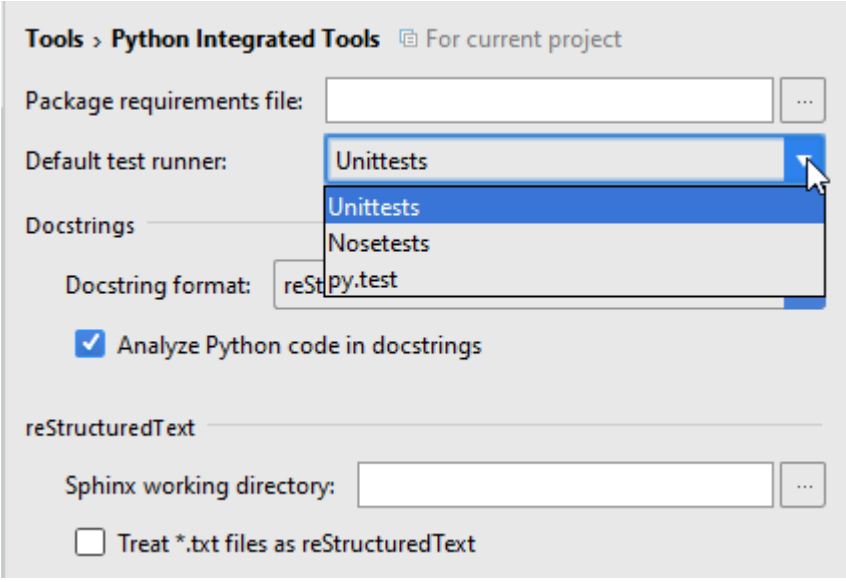
Pääsiäinen.

Viikkoanalyysi

Kuluneeseen viikkoon mahtui paljon testausta ja ongelmanratkontaa. Testaaminen oli kuitenkin eniten esillä työssäni, joten tämän viikon teemaksi päätin valita jälleen monipuolisen testaamisen.

Alkuviikosta sain valmiiksi uuden Kinesis buffer-ohjelmani toteuttamisen. Jotta saatoin kutsua projektia valmiiksi, minun oli kuitenkin vielä toteutettava ohjelmalle yksikkötestit. Yksikkötestit helpottaisivat tulevaisuuden kehitystyötä, koska kehittäjän ei tarvitsisi manuaalisesti testata jokaista testitapausta erikseen, mikä nopeuttaisi kehitystyötä jatkossa huomattavasti. Vaikka ohjelmani ei ollutkaan vielä suuri, testit olivat silti yhtä aiheellisia kuin missä tahansa muussakin ohjelmassa. Salehia (2013, 25) lainaten, yksikkötestien tärkeys ei välttämättä näy projektin alkuvaiheessa, mutta niiden hyödyt tulevat esiin projektin myöhemmissä vaiheissa, kun lähdekoodista tulee monimutkaisempaa, tarvitaan lisää ominaisuuksia tai kun regressio-vikoja ilmenee lisää.

Testien ajamisessa käytin apuna pytest-kehystä. Pytest on erittäin monipuolinen ja tehokas testaamiskehys Python-ohjelmille, jolla ohjelmoidut yksikkötestit voidaan ajaa nopeasti ja kätevästi. Pytest voidaan helposti integroida eri työvälineisiin, kuten PyCharm-tekstieditoriin, jossa se voidaan konfiguroida ajettavaksi automaattisesti aina kun ohjelman lähdekoodiin tallennetaan muutoksia, jolloin kehittäjä saa suoraan palautetta lähdekoodinsa toiminnasta (Kuva 4).



Kuva 4. Pytest-kehyksen käyttöönotto PyCharm-ohjelmointiympäristössä (JetBrains s.r.o. 2017).

Florian Bruhin kehuu kehittämäänsä pytest-työkalua Swiss Python Summit-konferenssissa sen yksinkertaisuudesta ja selkeydestään. Verrattuna Pythonille kehitettyyn unittest-kehykseen, pytest on parempi työkalu muun muassa siksi, että se antaa testaajalle tarkat tiedot siitä, missä kohtaa jokin testi epäonnistui. Kuten Bruhin (2016, 3:34) sanoo, pytest-kehyksellä testaaja ei ainoastaan näe mikä testi epäonnistui, mutta myös millä muuttujien arvoilla epäonnistuneet testit suoritettiin, sekä miten epäonnistuneiden

testien tuloksiin päädyttiin. Bruhinin konferenssiesitys kertoo pitkälti, miksi kehitystiimissäni suositaan pytest-kehystä.

Tulin luoneeksi testitapauksia ohjelmaani monipuolisesti ja kattavasti. Testasin, että kaikki funktiot toimivat odotetulla tavalla niille annetuilla parametreilla. Tein myös testitapauksia, joissa varmistin, että funktiot epäonnistuivat, kun annoin niille virheellisiä argumentteja. Joissakin testitapauksissa varmistin erityisesti, että kutsuttu funktio heitti poikkeuksen, kun testasin sitä odottamattomilla argumenteilla.

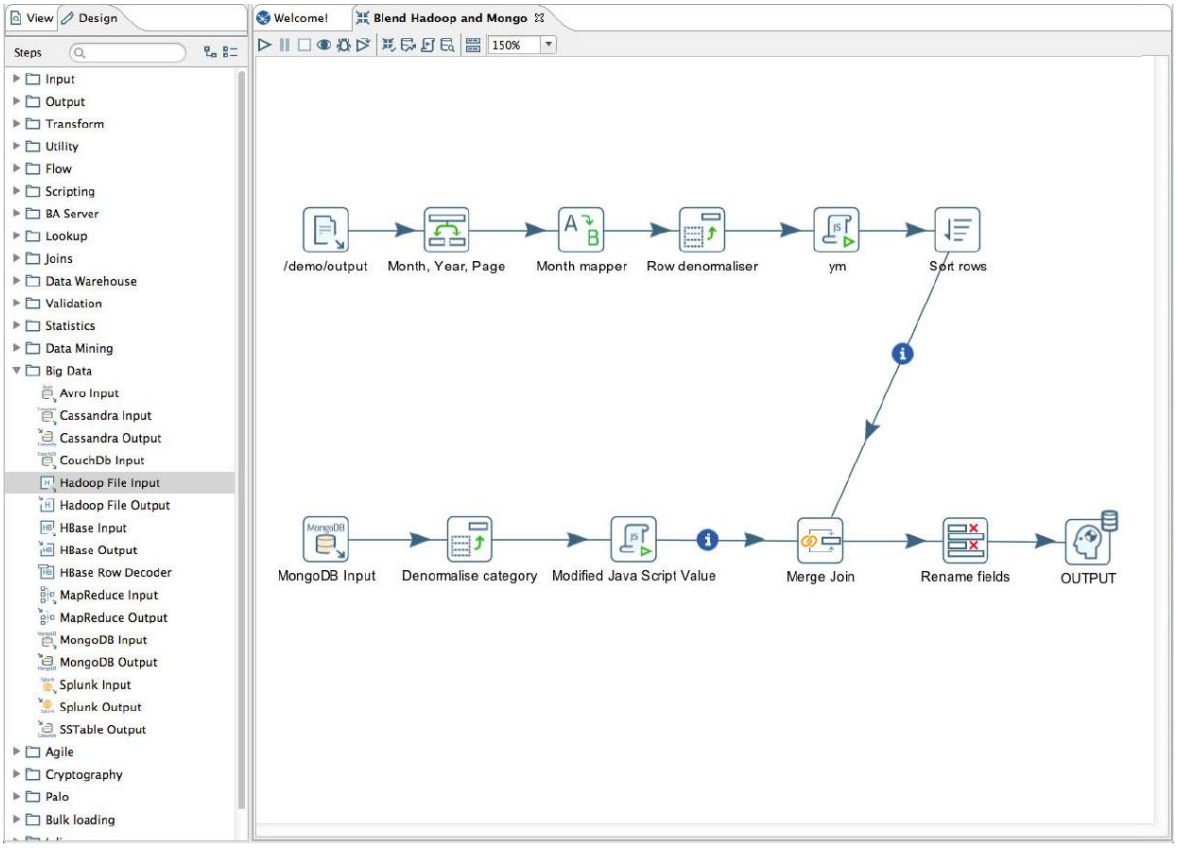
Pythonilla testituloksia voidaan tarkistaa Pythoniin sisäänrakennetulla assert-funktiolla. Aina varsinaisen ja odotetun tuloksen vertailu ei kuitenkaan onnistu pelkällä assert-kutsulla. Näin voi olla, mikäli halutaan tarkistaa, että testistä kutsuttu funktio heittää poikkeuksen, sen sijaan että tarkistettaisiin jonkun palautettavan muuttujan arvo. Tätä varten minun piti opetella pytestin tarjoama raises-funktio, jossa nousi esiin jälleen pytest-kehyksen helppokäyttöisyys.

Testejä ohjelmoidessani olisin voinut olla kuitenkin optimaalisempi ja käyttää enemmän pytest-kehyksen ominaisuuksia. Sen sijaan, että tein useita testifunktioita testaamaan samaa ohjelman funktiota eri parametrein, olisin voinut käyttää pytestin parametrize-dekoraattoria yhden testifunktion kohdalla. Parametrize-dekoraattorilla pystyisin määrittelemään testin ajettavaksi useaan kertaan, ja jokaiselle kerralle määrittelemään eri parametrit. Pythonin dekoraattoreiden käyttö on suositeltavaa, kun tiettyä ohjelman osaa, eli funktiota tai muuttujaa, halutaan ns. paketoida toisen valmiiksi määritellyn funktion sisälle. Sneeringeriä (2015, 16) lainaten, dekoraattoreiden käyttöä voidaan kuvailla lauseella: ”Haluan tämän tietyn, kierrätettävän palan toiminnallisuutta näihin tiettyihin paikkoihin.”

Jotta olisin voinut saada ohjelmoimistani testeistä enemmän irti jo ohjelman kehittämisvaiheessa, olisin minun kannattanut soveltaa kehitystyöhöni TDD-menetelmää, eli Test Driven Developmentia. Sen sijaan, tulin käyttäneeksi perinteistä yksikkötestaus-menetelmää, jossa ohjelmoin yksikkötestit vasta sen jälkeen, kun olin toteuttanut ohjelman päätoiminnallisuuden. TDD-menetelmässä yksikkötestit toteutetaan ensin, jolloin testien tarkoituksena on osoittaa puuttuva tai virheellinen toiminnallisuus ohjelmaa kehitettäessä. Testien toteuttaminen ennen ohjelmalogiikan toteuttamista olisi ollut itselleni haastava, mutta opettavainen menetelmä ohjelmani kehitystyöhön. Kuten Hartikainen (2016) sanoo, vaikein asia kehittäjille TDD-menetelmässä on juurikin se, että ohjelman yksikkötestit tulee kirjoittaa ennen lähdekoodin kirjoittamista. Salehin (2013, 28) sanoin, TDD on tehokas tekniikka, sillä se auttaa hallitsemaan ohjelman lähdekoodia ja suunnittelua paremmin,

mutta se voi myös olla kaksiteräinen miekka, mikäli sitä ei käytetä oikein, koska testien toteuttamiseen voi kulua paljon aikaa ja projektin aikataulu voi epäonnistua.

Loppuviikosta työskentelin datan integroimiseen tarkoitetulla Pentaho-työvälineellä, kun tarkoituksena oli siirtää vanhaa dataa Amazon Web Servicesin RDS-tietokantapalvelimesta Amazonin Redshift-palvelimelle. Kyseessä oli enemmänkin järjestelmäspesialistin, kuin ohjelmoijan työtehtävä, mutta tehtävä annettiin kuitenkin minulle, koska kehitystiimissämme ei ollut erikseen näihin tehtäviin erikoistunutta työntekijää ja koska tunsin Amazonin jo ennestään. Tässä työtehtävässä laajensin huomattavasti omaa asiantuntemustani Pentaho-työvälineestä ja datan integroinnin toteuttamisesta. Ohessa kuva Pentaho-työvälineestä (Kuva 5).



Kuva 5. Datan Integroiminen Pentaho-työvälineellä (Pentaho Corporation, 2017).

Testaamistaitoja tarvitsin Pentahon käytössä myös siinä mielessä, että datan siirron aikana piti selvittää, missä siirrossa käytettävässä osassa muodostui pullonkaula, kun siirron nopeus oli liian hidas. Pystyin selvittämään hitaimman osan testaamalla, miten vaikuttaisin siirron nopeuteen, mikäli vaihtaisin operaatiossa käytettäviä osia, esimerkiksi vaihtamalla kohdetietokannan tilalle paikallisen tekstitiedoston, johon prosessoitu data tulostettaisiin. Työtehtävässäni käyttämäni operaatio perustuu ETL-tekniikkaan, eli ”Extract – Transform – Load”-tekniikkaan, jossa prosessoitava tieto puretaan ensin määritellystä lähteestä,

muunnetaan haluttuun formaattiin, ja sen jälkeen ladataan määriteltyyn kohteeseen. ETL:ssä tiedon kulku perustuu prosessin hitaimpaan osaan, jolloin on tärkeää, että kaikki prosessin osat toimivat mahdollisimman optimaalisesti. Mikäli ETL-prosessissa on mitään, mikä rajoittaa tiedon kulkua, vaikkakin vain yksi askel, vaikuttaa se koko tiedon kulkuun prosessissa (Pentaho Community).

3.8 Seurantaviikko 8

Maanantai 17.04.2017

Pääsiäinen.

Tiistai 18.04.2017

Jatkan siitä, mihin jäin ennen pääsiäistä, eli datan siirtämisestä. Nyt tarkoituksena olisi kohdentaa muunnettava data ensin Amazonin S3-tiedostovarastopalveluun, josta se voitaisiin edelleen ladata Redshift-tietokantapalvelimelle.

Projektipäällikkö kertoi aamulla suorittaneensa datan siirron itse viikonlopun aikana. Kun infrastruktuurilliset ongelmat oli saatu kuntoon viime torstaina, itse datan siirto oli helppo, mutta aikaa vievä prosessi. Siispä pääsin jatkamaan muita töitä.

Tämä päivä kului yksikkötestien luomisessa aiemmin työstämääni Kinesis Client-ohjelmaan. Kyseessä oli suhteellisen laaja määrä ohjelman funktioita, jotka tarvitsivat testejä. Hyödynsin viikonlopun aikana opiskelemaani teoriaa testien toteuttamisesta Pytest-kehyksellä ja siten säästin useita rivejä testien koodissa käyttämällä muun muassa parametrize-dekoraattoreita, jolloin minun ei tarvinnut kirjoittaa useaa testifunktiota erikseen eri parametrien testaamiseen.

Haasteena tänään oli mock-olioiden eli ns. testiolioiden luominen testejä varten, sillä testien siisteyden vuoksi pyrin luomaan globaalit oliot käytettäväksi useille eri testifunktioille. Määrittelemällä normaalit globaalit oliot testitiedostoihin, eri testifunktiot jakoivat samat oliot keskenään, kun tavoitteena oli eristää kaikki testit toisistaan. Ratkaisu tähän löytyi pytest-kehyksen fixture-dekoraattorista, jolla pystyin määrittelemään, mitkä muuttujat tai oliot toimisivat mock-olioina Pytestin testiympäristössä. Testifunktioissa minun tarvitsi vain lisätä parametreina haluamani mock-oliot.

Keskiviikko 19.04.2017

Pääsin hyvään vauhtiin yksikkötestien luonnissa Kinesis Client-ohjelmaan. Eri lähdekooditiedostoissa tuntui olevan samanlainen rakenne, joten testien luonnin pitäisi jatkua suoraviivaisena nyt kun olen jo toteuttanut kahden lähdekooditiedoston testikokoelman.

Testien luonti eteni hyvin, kunnes kohtasin jälleen ongelman mock-olioiden määrittelyssä. Osa testaamistani funktioista käyttivät tietokantahakuja, jotka minun piti korvata mock-olioilla. Haasteena oli ratkaista se, miten saisin tietokantoja käsittelevät funktiot korvattua testaamieni funktioiden sisällä. Ratkaisuna toimi lopulta mock-kirjaston patch-funktio, jota projektipäällikkö neuvoi minua käyttämään. Patch-funktiolla pystyin määrittelemään testaamieni funktioiden sisäiset muuttujat tai funktiot yleiseksi mock-olioksi, jolle pystyin testissäni asettamaan mitä tahansa toimintoja tai paluuarvoja. Mock-olio korvaisi oikean funktiossa käytettävän muuttujan tai funktion testin ajon aikana.

Yksi päivän aikana tekemistäni havainnoistani oli outo syntaksivirhe liittyen aikaleima-olioiden luontiin Pythonilla. Python 2.7-versiolla on tarjolla mm. datetime-luokka, jonka instanssin pystyy luomeen helposti käyttämällä luokan parametrillista konstruktoria. Konstruktoriin voidaan antaa parametriksi vuosi, kuukausi, päivä, tunti, ja niin edelleen. Olin tottunut syöttämään kuukausiparametrin muodossa 01, 02, 03 jne. Python-ohjelmointiympäristö hyväksyi useimmat näistä, paitsi 08 ja 09, joissa se ilmoitti syntaksivirheestä. Tämä oli itselleni erittäin outoa, kunnes selvisi, että 0-alkuiset numerot tulkitaan Pythonissa vakiona oktaalinumeroiksi. Koska oktaalijärjestelmässä on kantalukuna kymmenen sijaan kahdeksan, Python hyväksyy vain oktaalinumerot 0-7.

Torstai 20.04.2017

Jatkan yksikkötestien toteuttamisesta. Jäljellä on vielä joitakin ohjelman apufunktioita, joiden testit puuttuvat.

Heti aamusta sain tietää, että aikaisemmin tekemässäni koodissa oli bugi, kun front end-kehittäjäkollegani oli raportoinut virheestä API-järjestelmän toiminnassa. API-funktiot palauttivat dataa väärässä formaatissa. Kun aloitin virheenratkonnan, löysin toisen virheen. Tuntemattomasta syystä yksi kyseiseen tehtävääni liittämä commit oli jäänyt pois ohjelman käyttöversiosta. Vaikka kyseinen commit ei ollutkaan kriittinen ohjelman toiminnan kannalta, hämmensi se minua silti suuresti, samoin kuin tiimikavereitanikin. Tutkiessamme ohjelman versiopuurakennetta Gitissä, emme löytäneet järkevää selitystä commitin katoamiselle. Huomasimme kuitenkin, että brancheja yhdistäessä olimme joutuneet selvit-

tämään kaksi konfliktia. Pidimme todennäköisimpänä, että konflikteja selvittäessämme olimme tulleet vahingossa poistaneeksi tekemäni commitin. Korjasin ongelman käyttämällä Gitin cherry pick-toimintoa, jolla pystyin poimimaan versiohistoriasta kyseisen commitin ja liittämään sen uuteen branchiin.

Itse bugin korjaaminen oli loppupeleissä suoraviivaista, vaikkakin jouduin hetken aikaa hahmottamaan ongelman syytä. Selvisi, että minulla oli tarpeettomia sarakkeita tietokantakyselyyn määritellyssä group by-lausekkeessa. Niiden poistaminen korjasi ongelman. Loppupäivästä kerkesin vielä hetkeksi palata eilen työstämieni yksikkötestien pariin.

Perjantai 21.04.2017

Jatkan yksikkötestien kirjoittamisesta. Tavoitteena olisi saada ne tänään valmiiksi, mikäli ei tule muuta kiireellisempää tehtävää.

Testien kirjoittaminen sujui hyvin ilman suurempia ongelmia. Olin jo ratkaissut suurimmat pulmat liittyen yksikkötestien suunnitteluun ja toteuttamiseen viime päivinä, joten jäljellä olevat testitapaukset olivat itselleni yksinkertaisia. Yksikkötestit Kinesis Client-ohjelmassa olivat osa suurempaa kokonaisuutta, jota kutsuimme kehitystiimissämme nimellä ’Quality Assurance’, eli suomeksi laadunvarmistukseksi. Tämä tiesi sitä, että muut ohjelmistomme osat tarvitsivat vielä laadunvarmistusta, eli tässä tapauksessa yksikkötestien kirjoittamista. Saatuani testit kuntoon Kinesis Client-ohjelmalle, tulin keskustelleeksi tiimitoverini kanssa siitä, mitkä muut ohjelmiston osat olivat tärkeimpiä laadunvarmistuksen kannalta. Keskustelun päätteeksi tulin luoneeksi kaksi uutta tehtävää JIRAssa. Tehtävät liittyivät yksikkötestien toteuttamiseen API-järjestelmämme käyttämiin apufunktioihin. Tulisin todennäköisesti toteuttamaan näitä testejä myös tulevilla viikoilla.

Tulin päivän aikana korjanneeksi myös bugin liittyen API-järjestelmäämme. Yksi API-funktio ei palauttanut dataa halutussa järjestyksessä. Ongelma oli kuitenkin korjattu nopeasti lisäämällä yksi tietokannan taulukon sarake tietokantakyselyn group by-lausekkeeseen.

Viikkoanalyysi

Kulunut viikko oli jälleen hyvin testauspainotteinen. Samalla tulin korjanneeksi kaksi bugia järjestelmässämme. Bugien havainnoimisessa kohtasin oudon tilanteen versionhallinnassa, kun yksi tekemäni commit oli nähtävästi kadonnut kehittämäni ohjelman versiohistoriasta. Todennäköisin syy löydöksiemme perusteella tähän oli se, että olin itse pyyhkinyt

koodini pois aiempaa konfliktia selvittäessä. Tämä muistutti minua siitä, kuinka huolellinen konflikteja selvittäessä tulisi olla. Koodin rakenne tulisi tarkistaa molemmista ohjelman versioista, kun niitä yhdistetään. Tämä tilanne oli kuitenkin vain kertaluontoinen, joten kuvaavin teema tälle viikolle on mielestäni ohjelmiston laadunvarmistus ketterin menetelmin. Se sivuaa samalla aiemmin käyttämääni teemaa, monipuolista testaamista.

Yksikkötestien ohjelmoiminen vei suurimman osan aikaa koko viikon työajasta, koska testattavalla ohjelmalla ei ollut niitä vielä lainkaan, ja lähdekoodi oli jo kasvanut melko laajaksi. Kuten aiemmin todettua, vaikka testien ohjelmoiminen vaikuttaakin toisinaan melko puuduttavalta ja aikaa vievältä työltä, voi sillä olla suuri vaikutus pidemmällä aikavälillä, kun se voi säästää aikaa useilta turhilta korjaustoimenpiteiltä myöhemmin. Riittävä yksikkötestien määrä taas auttaa takaamaan kehitettävän ohjelman laadun sen käyttöönotossa. Kuten Saleh (2013, 25) sanoo kirjassaan, huonolaatuisissa ohjelmissa on usein pidempi testausjakso jokaisessa ohjelman käyttöönotossa, koska ne sisältävät suuremmalla todennäköisyydellä enemmän vikoja per muutos lähdekoodissa, mikä luo enemmän paineita projektinhallinnalle, kehittäjille ja testaajille.

Luodessani yksikkötestejä Kinesis Client-ohjelmalle, jouduin samalla opiskelemaan lähdekoodia, koska minun tuli tietää, miten mikäkin testattava funktio toimi tarkalleen. Testaamisesta oli siis myös hyötyä siinä, että opin tuntemaan kehitettävän järjestelmän paremmin, koska en ollut itse kirjoittanut ohjelman lähdekoodia. Toisaalta, yksikkötestit auttavat myös kehittäjiä ymmärtämään lähdekoodin toimintaa ja miten ohjelman eri funktioita käytetään. Kuten Salehkin (2013, 25) sanoo, hyvät yksikkötestit voivat olla hyvä viittauksen kohde dokumentaatiolle, koska ne sisältävät testitapaukset ohjelmiston käyttötapauksista. Tämän lisäksi, yksikkötestit näyttävät kehittäjälle, miten järjestelmän rajapintoja käytetään, mitkä taas kertovat järjestelmän nykyisestä rakenteesta (Saleh 2013, 25).

Perjantaina valmistuneet yksikkötestit auttaisivat ohjelman kehittäjiä tulevaisuudessa varmistamaan ohjelman toimivuus uusien muutosten käyttöönotossa. Testejä oli toki vielä paljon toteuttamatta, kun otettiin huomioon koko kehittämämme ohjelmisto. Siksi onkin hyvä kartoittaa aina väliajoin, missä ohjelmiston osissa yksikkötestejä ei vielä käytetä apuna laadunvarmistuksessa. Useimmille puuttuville testeille olikin jo luotu niitä kuvaavat tehtävät JIRAssa. Tulin kuitenkin luoneeksi kaksi uutta tehtävää, joissa kerroin järjestelmän osista, joissa yksikkötestejä ei oltu vielä toteutettu.

Viikon loppupuolella tulin korjanneeksi kaksi bugia ohjelmistostamme. Toinen niistä liittyi omaan, aikaisemmin toteuttamaani ominaisuuteen. Määrittelemäni tietokantakyselyt eivät olleet täysin oikein määriteltyjä. Tällä ei ollut useimmissa käyttötapauksissa mitään vaiku-

tusta, mutta pyydettäessä suurempaa alaa dataa tietokannalta, datan lajitteleminen ei toiminut halutulla tavalla. Tämä osoitti sen, että aikaisemmin Postmanilla tekemäni API-testit eivät olleet olleet riittävän kattavia, tai olin huonoksi onnekseni valinnut huonot parametrit käytettäväksi testattavissa kutsuissa. Toki suurin osa testitapauksista olivat olleet kattavia, minkä vuoksi järjestelmässä havaittu virhe ei ollut tämän kriittisempi.

Testaaminen on toki vain osa laadunvarmistusta, ainakin omassa tapauksessani, kun toimin itse oman kehittämistyöni testaajana. Testaamalla kehittämääni ratkaisua ohjelmaan paljastan vain omat kehitystyössäni tekemäni virheet. Varsinainen laadunvarmistus tapahtuu kuitenkin vasta, kun ryhdyn vikojen korjaustoimenpiteisiin, tai mikäli testit ovat kattavat eivätkä ne osoita vikoja. Toisenlaisessa tilanteessa testaajan rooli voisi toki olla toinen, jolloin testaajan rooli saatetaan usein väärinymmärtää. Antti Niittyviitalla on tähän osuva näkökulma:

> *Testaajan todellinen tehtävä ei ole rakentaa luottamusta tuotteen toimivuudesta. Testaajan todellinen tehtävä on tuhota väärin perusteltu luottamus. Silloin ei ole kyse laadunvarmistuksesta, silloin on kyse testauksesta. (Niittyviita 2011.)*

Vikojen korjaamisessa kaikki testaaminen keskitetään bugin vaikuttamiin osiin ohjelmistossa. Mikäli bugi on jonkun toisen henkilön raportoima, joudun tavallisesti havainnollistamaan virheen yrittämällä toistaa sen ohjelmaa käytettäessä. Tällöin on tärkeää, että bugin kuvaus ja tapahtumakulku on kuvattu mahdollisimman tarkasti. Birdiä (2012) mukaillen, et voi olla varma, että korjaat virheen, ellet pysty toistamaan samoja askelia, nähdä virheen tapahtuvan itse ja toistaa samat askeleet uudestaan korjattuasi sen, nähden virheen poistuvan.

Ketterässä Scrum-menetelmässä bugien korjauksia ei tavallisesti oteta huomioon määriteltäessä yhden Sprintin laajuutta. Tämä tarkoittaa, että yksittäiset korjaustehtävät sovitetaan mukaan käynnissä olevaan Sprintiin riippumatta siitä, onko nykyisessä Sprintissä enää tilaa uusille tehtäville. Soveltaessamme Scrumia osittain kehitystiimissämme, priorisoimme bugien korjaustehtävät korkealle, koska ne liittyvät suoraan kehitettävän tuotteen laadunvarmistukseen. Mikäli emme priorisoisi pieniäkään bugeja korkealle, saattaisimme joutua mittaviin ongelmiin pidemmällä aikavälillä. Tällainen lähestymistapa johtaa bugien kasaantumiseen ja useamman viikon päästä niiden korjaaminen voi viedä paljon enemmän aikaa (Abhilash c 2012).

3.9 Seurantaviikko 9

Maanantai 24.04.2017

Tällä viikolla luvassa on normaalista poikkeava viikko, sillä yrityksemme on kutsuttu Hackathon-tapahtumaan, jossa tarkoituksena on kehittää täysin uusi tuoteprototyyppi alusta alkaen. Tapahtuma käsittää koko viikon, mutta varsinainen toteuttamistyö tulee todennäköisesti alkamaan aikaisintaan tiistaina. Sitä ennen projektini johto tulee keskustelemaan kehitettävän tuotteen vaatimuksista tapahtuman järjestäjien kanssa ja suunnittelemaan tuotetta. Varsinaisia tehtäviä aloitetaan todennäköisesti jakamaan tiistai-aamuna, jolloin osa tiimistä työskentelee tapahtumapaikassa ja muut toimistolla, mukaan lukien minä.

Tämän päivän vietin vielä yksikkötestien parissa. Tällä kertaa kohteena oli API-järjestelmän apufunktioiden testaaminen. Yksi tehtävistä oli päivittää vanha Pythonin unittest-kehystä käyttävä testitiedosto pytest-kehykselle. Samalla parantelisin itse testejä käyttämällä enemmän parametreja, kuin vanhoissa testeissä.

Yhdessä testitapauksessa kohtasin suuren haasteen, kun minun piti tehdä erään funktion käyttämästä lokaalista dict-oliosta mock-olio. Funktiossa dict-olioon syötettiin avain-arvo-pareja, mutta en pystynyt testaamaan funktion ulkopuolelta, onnistuiko datan syöttö normaalisti. Koska testifunktiossani käytin mock-olioita, ne eivät toimineet samalla tavalla, jollen erikseen muokannut niiden käyttäytymistä. Ratkaisu ei ollut helppo, mutta löytyi lopulta stackoverflow-ongelmanratkontayhteisön sivustolta, jossa neuvottiin asettamaan mock-olio uudelleenohjaamaan sille syötetty data testifunktiossa käyttämääni dict-olioon. Tämä onnistui Python-olioden sisäänrakennetuilla getitem- ja setitem-metodeilla.

Tiistai 25.04.2017

Eilen loppupäivästä pidimme kehitystiimin kanssa lyhyen videopalaverin liittyen tällä viikolla alkavaan Hackathon-tapahtumaan. Tiedämme jo, että saamme käyttää projektissa osittain omaa dataamme sekä pohjana omia projektejamme, joita sitten muokattaisiin tarpeen mukaan. Alkupäivästä projektin johto todennäköisesti suunnittelee projektia lisää, kunnes iltapäivällä alkavat varsinaiset kehitystyöt. Siihen asti jatkan yksikkötestien ohjelmoimista.

Aamupäivällä pidimme pienen palaverin tiimikaverini kanssa, jossa pyrimme suunnittelemaan päivän aikana toteutettavia tehtäviä Hackathon-projektiamme varten. Käyttäisimme projektin pohjana omaa API-järjestelmäämme, jota muokkaisimme tarpeen mukaiseksi.

Pyrimme suunnittelemaan, mitä lisäfunktioita tulisimme tarvitsemaan ja miten ne toimisivat. Jaoimme toteutustehtävät siten, että kollegani vastasi API-järjestelmän rajapinnan toteuttamisesta, sillä välin kun minä työstäisin tietokannan käsittelyfunktioita. Tulin samalla ohjelmoineeksi funktion, jolla pystyisi lähettämään REST API-kutsuja muihin ulkoisiin API-järjestelmiin, koska tehtävämme vaati sitä.

Toteuttamistehtävissä minulla ei ollut juuri ongelmia. Välillä jouduin tosin kysymään lisää järjestelmän rakenteesta ja yksityiskohdista. Pidimme myös videopalaverin muiden tiimin jäsenten kanssa, jossa kävimme yhdessä läpi projektin tilannetta sekä tavoitteita sen rakenteen parantelemiseksi.

Keskiviikko 26.04.2017

Sain eilen tärkeimmät tietokantatason funktiot toteutettua. Itselläni ei ole vielä täyttä varmuutta seuraavista tehtävistäni, joten konsultoin tiimikavereitani aamun palaverissa siitä, mitä askeleita ottaisin seuraavaksi projektissa.

Aamulla selvisi, että olemme jo saaneet Hackathon-projektin ensimmäisen iteraation valmiiksi muilta osin, paitsi API-järjestelmän rajapinnan, jota tiimitoverini parhaillaan työsti. Muut työn osat odottivat tämän osan valmistumista, joten sain tehtäväkseni palata toistaiseksi työstämään oman API-järjestelmämme tehtäviä siihen asti, kunnes projektipäällikkö antaisi lisäohjeita.

Tulin päivän aikana luoneeksi uuden osan API-järjestelmäämme, jolla pystyi tallentamaan uudenlaisia resursseja tietokantaan sekä lukemaan niitä. Tehtävä sujui rutiinilla ilman suurempia ongelmia. Lisäksi tulin selvittäneeksi muun muassa kahta eri bugia, joista toinen selvisi, kun huomasin, että tietokantakyselyssä käytetty globaali filtteri oli määritelty väärin.

Torstai 27.04.2017

Mikäli Hackathon-projektiin ei liity uusia tehtäviä minua varten vielä aamulla, jatkan eilen aloittamastani bugin selvittämisestä. Kyseessä on tietokantakyselyihin liittyvä bugi, joka aiheuttaa joidenkin rivien poisjäämisen kyselyiden tuloksista, joiden ei pitäisi jäädä pois.

Jatkoin bugin selvittämistä aamusta alkaen. Jouduin pitkään lukemaan monimutkaista lähdekoodia, jotta ymmärsin, miten tietokantakyselyiden tulokset suodatettiin ja rakennettiin. Lopulta ongelman syy selvisi, kun huomasin että tuloksista suodatettiin pois rivejä

olioista, joilta puuttui tärkeä osa toisesta tietokannan taulusta. Vaikka tauluja ei varsinaisesti oltu liitetty toisiinsa, ei tällaista tilannetta kuitenkaan saanut olla käytännössä. Ongelma ei ollut siis lähdekoodissa, vaan tietokannan taulun puuttuvissa riveissä, jonka korjasimme yksinkertaisesti lisäämällä puuttuvat rivit tietokantaan. Tilanteen aiheuttaja oli toistaiseksi vielä hieman epäselvä.

Loppupäivästä minulla ei ollut paljoa tehtävää, mutta tulin kuitenkin tutkineeksi järjestelmämme admin-käyttöliittymään liittyvää outoa käyttäytymistä tilanteessa, jossa toisinaan olioiden uudelleennimeäminen aiheutti ongelmia käyttöliittymässä. En kuitenkaan kyennyt toistamaan outoa käytöstä.

Perjantai 28.04.2017

Työtehtäväni olivat edenneet tähän mennessä sujuvasti, eikä tiimitoverillani ollut eilen loppupäivästä tarjota minulle uutta tehtävää, joten katsomme uudestaan tilannetta tänä aamuna. Projektipäällikkö on vielä tämän päivän Hackathon-tapahtumassa paikan päällä, joten emme todennäköisesti pääse vielä suunnittelemaan uusia ideoita ja tehtäviä hänen kanssaan.

Tiimitoverini kertoi Kinesis Client-ohjelman tarvitsevan lisää tietoja tallennettavaksi Amazon Web Servicesin tarjoamalle DynamoDB-tietokantapalvelimelle, ja luettavaksi sieltä. Toteutin tietokantayhteydet ohjelman metodeihin löytämieni esimerkkien mukaisesti. Data tallennettaisiin DynamoDB-tietokannassa sille varattuun tauluun. Koodini testaaminen kuitenkin tyssäsi siihen, kun huomasimme, että AWS-tililläni ei ollut vielä valtuuksia lukea tai tallentaa dataa DynamoDB-tietokantaan. Vain projektipäällikkö pystyisi antamaan minulle nämä valtuudet AWS-järjestelmässä. Hän oli kuitenkin kiireinen Hackathon-tapahtumassa, joka huipentuisi tänään, joten hän ei olisi kerennyt antaa minulle niitä.

Tallensin työni Gitiin ja sain jatkaa API-järjestelmämme integraatiotesteistä, joissa testattiin järjestelmän toimivuutta tietokannan kanssa. Tulin opiskelleeksi aiemmin toteutettuja integraatiotestejä, jotta sain paremman kuvan testien rakenteesta sekä niissä käytetyistä metodeista. Huomasin, että testeissä käytettiin Pytest-kehyksen sijasta yleisempää unittest-kehystä. Yritin ajaa vanhoja testejä PyCharmissa, mutta testidata vaikutti olevan pahasti vanhentunut nykyisestä tietokannan rakenteesta. Päivä oli kuitenkin jo lopussaan, joten tutkiminen ja korjaaminen sai jäädä ensi viikolle.

Viikkoanalyysi

Vastoin odotuksiani, kulunut viikko oli sangen normaali. Odotin pääseväni osallistumaan Hackathon-projektin kehittämiseen enemmän, mutta työmäärä olikin odotettua pienempi ja keskitetty tiettyihin ohjelmisto-osiin. Niinpä osallistuin projektin kehitystyöhön ainoastaan tiistaina. Muilta osin työtehtäväni olivat varsin vaihtelevia, kuten uusien ominaisuuksien toteuttamista, bugien korjausta sekä testien ohjelmoimista. Tälle viikolle valitsin teemaksi monipuolisen ohjelmistokehityksen.

Tähän mennessä olen saanut paljon rutiinia hyvinkin erilaisista tehtävistä. Työni ei ole siis pelkkää uusien ominaisuuksien ohjelmoimista ohjelmistoon, vaan minun tulee olla valmis myös testaamaan kehittämäämme ohjelmistoa, varmistamaan sen laatu sekä tekemään muita operatiivisia tehtäviä järjestelmäämme, kuten tietokantapalvelimen ylläpitotehtäviä. Tämä perustuu kehitystiimimme käyttämään DevOps-malliin, jossa ohjelmistotuotteen kehitys- ja tuotanto tapahtuu saman tiimin toimesta ja jossa työntekijöillä on laaja tuntemus molemmista osa-alueista. Kaiken kaikkiaan DevOps on kombinaatio niitä kulttuurillisia filosofioita, käytäntöjä ja työvälineitä jotka parantavat yrityksen kykyä toimittaa ohjelmia ja palveluja nopealla vauhdilla, kehittäen ja parantaen tuotteita nopeammalla tahdilla kuin yritykset jotka käyttävät perinteisiä ohjelmistokehitys- ja infrastruktuurin hallintaprosesseja (Amazon Web Services, Inc 2017).

Työni alussa työtehtäväni keskittyivät lähinnä itse ohjelmiston kehittämiseen, enkä juuri tuntenut kehitystiimissäni käytettävää DevOps-menetelmää. Tähän mennessä olen oppinut erittäin paljon uutta myös työni operatiivisista tehtävistä, kuten palvelumme käyttämän infrastruktuurin ylläpitotehtävistä. Näiden tehtävien rutinoituminen auttaa minua jatkossa toimimaan itsenäisemmin, mikäli kohtaan ongelmia tuotantopuolella. Tämä tarkoittaa myös sitä, että opin entistä enemmän käyttämään uusia työkaluja, jotka ovat sopivia juurikin DevOps-mallin toimintaan. Kuten Amazon Web Servicesillä (2017) sanotaan, nämä työkalut auttavat ohjelmistokehittäjiä suoriutumaan itsenäisesti tehtävistä, jotka muutoin tarvitsisivat tukea muilta tiimeiltä, mikä ennestään parantaa kehitystiimin nopeutta.

Kuluneella viikolla minun tarvitsi toisinaan hallita ohjelmistomme käyttämää infrastruktuuria. Tämä tapahtui muun muassa silloin, kun jouduin hallinnoimaan Amazon Web Servicesin tarjoamaa DynamoDB-tietokantapalvelinta. Viikon aikana tulin jatkaneeksi myös yksikkötestejä viime viikolta. Tulin oppineeksi lisää pytest- ja unittest-kehyksistä. Yksikkötesteissä kohtaamani ongelma, jossa jouduin luomaan Pythonin dict-oliosta mock-olion, opetti minulle uusia asioita itse Python-kielen ominaisuuksista.

Merkittävin ominaisuus jonka opin tällä viikolla Pythonista oli niin kutsuttujen ’magic methodien’ käyttö. Sneeringeriä (2015, 69) lainaten, Python-luokkiin voidaan tarvittaessa määritellä iso liuta metodeita, joita kutsutaan, kun luokista muodostettuja olioita käytetään tietyissä tilanteissa. Niitä ei siis tarvitse kutsua erikseen ohjelman lähdekoodista. Python-olioilla on omat vakio-käytösmallit, eli metodit joita oliot noudattavat tietyissä tilanteissa. Näitä käytösmalleja voidaan muuttaa määrittelemällä olioille tai niiden luokille omat magic methodit, joilla voidaan korvata vakiokäytös. Magic methodin tunnistaa lähdekoodissa metodin nimessä käytettävistä alaviivoista, joita on metodin nimen kummallakin puolen kaksi kappaletta.

Magic methodeja tulin tarvinneeksi, kun minun piti muuttaa yksikkötestissä käytettävän mock-olion käytöstä silloin, kun sille syötettiin avain-arvo pareja, kuten dict-olioille. Tämä onnistui, kun määrittelin mock-oliolle getitem ja setitem-magic methodit, joilla muutin olion käyttäytymistä siten, että se uudelleenohjasi sille syötetyt arvot toiseen, oikeaan dict-olioon samalla, kun sen kautta pystyi lukemaan dict-olion arvoja. Toisaalta, getitem-metodin määrittäminen testien koodiin ei olisi ollut välttämätöntä, koska ohjelman lähdekoodissa mock-oliota ei olisi kuitenkaan luettu, sille olisi vain syötetty arvoja. Sneeringeriä (2015, 87) mukaillen, ei ole vaadittua, että jokainen olio jolle määritellään getitem-metodi tarvitsisi erikseen määriteltyä setitem-metodia.

Pytest-kehyksestä tulin oppineeksi sen, että sillä pystyy ajamaan myös unittest.Testcase-tyylisiä testejä, joissa yhteen Testcase-luokkaan on koottu kokoelma eri testimetodeita. Odotin alun perin niiden ajamisen olevan mahdollista vain itse unittest-kehyksellä. Pytestin unittest-tuen tarkoituksena on mahdollistaa testikokoelmien, jotka on kirjoitettu ajettavaksi unittest-kehyksellä, ajaminen myös pytest-kehyksellä (Holger krekel and pytest-dev team, 2015).

Analyysin perusteella arvioisin oman ammattitaitoni kehittyneen viikon aikana usealla eri ohjelmistokehityksen osa-alueella. Viikkooni sisältyi uuden oppimista erityisesti Python-ohjelmointikielestä, pytest-kehyksestä ja testiautomaatiosta sekä Amazon Web Servicesin hallintatyökaluista.

3.10 Seurantaviikko 10

Maanantai 01.05.2017

Vappu.

Tiistai 02.05.2017

Näillä näkymin jatkan siitä, mihin jäin perjantaina, eli integraatiotesteistä. Mikäli projektipäällikkö on paikalla, pääsen toivottavasti myös testaamaan Kinesis Client-ohjelmaan tekemiäni muutoksia.

Päivän aikana pidetyt palaverit sotkivat jonkin verran työrytmiä. Pääsin kuitenkin eteenpäin testidatan päivittämisessä. Minulle selvisi myös, että oikean tietokannan sijasta minun kuuluisi käyttää testejä varten replikoitua testitietokantaa. Kun sain pääsyn testitietokantaan, pääsin varmistamaan, että kaikki testidata jonka olimme määritelleet testitiedostoissa tallentuisi oikein. Jouduin lisäämään joitakin sarakkeita testitietokantaan sekä laajentamaan testidataolioita koodissamme.

Kun tietokannan eheyden todettiin olevan kunnossa ja kun testidata tallentui ajon aikana kantaan, ongelmaksi muodostui kuitenkin se, että testejä ajettaessa tietokantakyselyt eivät palauttaneet dataa tuntemattomasta syystä. Joutuisin siis vielä huomenna selvittämään, puuttuiko testidatasta vielä olennaisia osia, jotka aiheuttivat datan suodattumista pois tuloksista.

Keskiviikko 03.05.2017

Tavoitteenani on korjata integraatiotestien käyttämä testidata itsekseni ja päästä jatkamaan testitapausten luontia. Tietokantakyselyt vaikuttivat melko monimutkaisilta, joten ongelman syyn selvittämisessä voi mennä tovi.

Pidimme aamulla monta pitempää palaveria. Yhdessä niistä keskustelimme kehitystiimin kesken aiemmin toteuttamastani ominaisuudesta Kinesis Client-ohjelmaan, jota pääsin vasta tänä aamuna testaamaan. Tehtävässäni paljastui useita komplikaatioita ominaisuuteen liittyen, ja pitkän pohdinnan jälkeen päätimme, että ominaisuus ei ollut kannattava tällä hetkellä johtuen sen työmäärän ja hyödyn välisestä huonosta suhteesta. Kenties palaisimme tehtävään myöhemmin uudestaan.

Jätin integraatiotestit toistaiseksi väliin, koska sain tärkeämpää tehtävää. Tehtävänäni oli parannella erästä tietokantakyselyä API-järjestelmässä siten, että eri aikaleimoihin ryhmitelty rivien lukumäärä tuloksissa näytettäisiin myös niiden aikaleimojen osalta, joissa tuloksia ei ollut. Tulisin siis näyttämään lukumäärän nollana kyseisten aikaleimojen kohdalla. Tehtävä osoittautui haastavaksi, sillä tiimitoverini neuvoi minua jälleen käyttämään alikyselyjä. Jouduinkin tehtävän aikana kysymään neuvoa useaan otteeseen. Lopulta kun

saimme tietokantakyselyn ”toimimaan”, huomasimme että se oli kuitenkin aivan liian hidas. Tulimme pohtineeksi eri vaihtoehtoja, kuten tuloksien näyttämisessä käytetyn aikaikkunan laskemista Python-lähdekoodissa, tietokantakyselyn sijasta. Lopulta päätin lähteä tähän suuntaan.

Torstai 04.05.2017

Yritän selvittää, miten parhaiten rakentaisin aikaikkunan, jota käytettäisiin eilen työstämäni tietokantakyselyn tulosten näyttämiseen. Mikäli en löydä kunnollista ratkaisua kahden ensimmäisen tunnin aikana, pyydän apua jälleen tiimitoveriltani.

Pitkän suunnittelusession jälkeen lähdin työstämään aikaikkunan rakentamista puhtaasti Pythonin avulla, muokkaamatta tietokantakyselyä ollenkaan. Jouduin käymään läpi tietokantakyselystä palautettavan datan rakennetta, jotta ymmärsin, mihin kohtaan minun pitäisi lisätä dataa. Tulin muokanneeksi datasta rakennettua dict-oliota. Käyttämällä Pythonin for- ja in- operaattoreita onnistuin tehtävässäni lopulta hyvin. Varmistin, että dict-olioon luotaisiin nolla-arvot niille aikaleimoille, joita siinä ei vielä esiintynyt.

Tehtävän jälkeen projektipäällikkö pyysi minua poistamaan erään suodattimen toisesta tietokantakyselystä. Suodatin hidasti tietokantakyselyä merkittävästi, koska siinä käytettiin postgreSQL:n ilike-operaattoria. Ilike-operaattorilla vertaillaan erilaisia merkkijonoja keskenään. Tehtävän ratkaisu oli nopea, yhden koodirivin muutos.

Kollegani teknisen tuen tiimistä raportoi tämän jälkeen admin-käyttöliittymässämme esiintyvästä bugista, joka aiheutti joidenkin tietojen ’piiloutumista’ käyttöliittymässä. Tämä johtui siitä, että käyttöliittymässä näytetty data ei suodattunut oikein API-järjestelmässä. Päätin ottaa uudenlaisen lähestymistavan käyttöön tietokantakyselyiden tutkimisessa. Poistin tietokantakyselyistä yhden suodattimen kerrallaan ja kokeilin, miten se vaikutti APIsta palautettavaan dataan. Tällä tavoin löysin ongelman, joka oli väärä operaattori kyselyn suodattimessa.

Loppupäivästä löysin vielä syyn integraatiotestien ongelmaankin. Selvisi että tietokantasessio, eli transaktio, jolla testidata syötettiin tietokantaan, oli eri, kuin sessio jolla testidataa luettaisiin tietokantakyselyissä. Tämä johtui siitä, että testidatan luomiseen käytettyä sessiota ei lähetetty argumenttina testeissä ajettaville tietokantakyselyille, vaan niissä alustettiin uudet sessiot. Kun kysyin asiasta tiimiltäni, ihmettelimme, miten testit on ylipäätään ajettu aiemmin onnistuneesti. Kyseessä oli kuitenkin erittäin vanhat testit, joista kellään ei ollut juuri muistikuvaa.

Perjantai 05.05.2017

Nyt kun integraatiotestien ongelman syy selvisi, tavoitteenani on todennäköisesti ensin korjata kaikki vanhat testit toimiviksi. Tämä tapahtuisi syöttämällä testidatan luomiseen käytetty sessio parametrina tietokantakyselyfunktioille, jolloin funktiot pystyisivät lukemaan samassa sessiossa luotua dataa.

Loin uuden tehtävän JIRAlla, jossa kerroin vanhojen integraatiotestien korjaamisen tarpeesta. Tämä selkeyttäisi tehtävänjakoani sekä ohjelmiston versiointia. Aloitettuani testien korjaamisen, työskentelin siten, että syötin alkuperäiset sessiot parametreina tietokantakyselyille ja niitä käyttäville funktioille. Koska kaikissa testeissä tietokantakyselyjä ei käytetty suoraan, vaan toisen ylemmän funktion kautta, jouduin usein passittamaan session useamman funktion kautta.

Alkupäivästä testien korjaaminen sujui paremmin, koska ensimmäiset korjaamani testit eivät käyttäneet kovin monimutkaisia funktioita. Iltapäivällä vastaan tuli kuitenkin funktioita, joissa korjattavaa oli muuallakin kuin tietokantasession määrittelyssä. Joidenkin testien odotetut tulokset olivat selvästi vanhentuneet, eivätkä ne siten vastanneet testattavista funktioista saatavia tuloksia. Päivän lopussa jäin jumiin yhden testitapauksen kanssa, jossa vertailtiin suuria dict-olioita keskenään, joiden rakenteet olivat hyvin monimutkaisia. En kyennyt löytämään korjattavaa kohtaa olioista, koska niitä selvästi muunneltiin useassa kohtaa testeissä. Keskittymisenikin oli hieman kateissa, joten testien korjaaminen saisi jatkua ensi viikolla.

Viikkoanalyysi

Valitsin viimeiseksi teemaksi järjestelmän integraation varmistamisen, koska työskentelin pääosan viikon työajastani API-järjestelmämme integraatiotestien parissa. En ollut vielä aiemmin työskennellyt näiden testien kanssa, joten työskentely oli erittäin opettavaista. Järjestelmän osien integraatiotestaamista tulisikin tehdä säännöllisemmin, kuin mitä tähän mennessä olin tehnyt. Testien vanhentuneisuus osoittikin sen, miten kauan edellisestä testauksesta oli kulunut aikaa. Oli korkea aika laittaa automatisoidut testit kuntoon.

Mitä integraatiotestaus tarkalleen ottaen on? Integraatiotestaamisessa testataan järjestelmän komponenttien yhteistoimintaa, jossa tavoitteena on löytää sellaisia virheitä, jotka eivät ole tulleet esiin yksikkötesteissä (Jyväskylän yliopisto). Kun esimerkiksi testataan jonkin ohjelman toimivuutta tietokannan kanssa, testataan juuri niitä ohjelman funktioita,

jotka kommunikoivat tietokannan kanssa. Tällöin on hyvä varmistaa, että testiympäristö on asennettu oikein. Tämä tarkoittaa muun muassa, että testitietokanta on määritelty ja valmisteltu vastaamaan järjestelmän oikeaa tuotannollista tietokantaa, jotta testejä varten luotu testidata ei potentiaalisesti haittaisi sitä. Siitäkin huolimatta, että käytämme integraatiotesteissä eri tietokantaa kuin ohjelmaa ajettaessa, käytämme transaktiota testidatan syöttämisessä tietokantaan. Transaktiossa tietty sarja tietokantatoimintoja määritellään yhdeksi toiminnaksi. Toiminta peruutetaan kokonaisuudessaan, mikäli yksikin osa epäonnistuu. Integraatiotesteissä tämä toiminta peruutetaan joka tapauksessa testien suorittamisen jälkeen, jotta tietokanta jäisi aina puhtaaseen ja eheään tilaan. Tällöin testidatan muokkaaminen on helppoa testeistä käsin.

Kun järjestelmän koko ja monimutkaisuus kasvaa, myös mahdollisten toimintavirheiden määrä eri komponenttien välillä kasvaa. Tällöin kannattaa turvautua regressiotestaukseen, eli suorittaa aiemmin suoritetut testit uudestaan, joilla pyritään varmistamaan, etteivät uudet lisäykset ja muutokset aiheuta ongelmia aiemmin tehtyjen elementtien toiminnassa (Jyväskylän yliopisto). Tästä syystä on hyvä huolehtia siitä, että aiemmin toteutetut automatisoidut integraatiotestit ovat ajan tasalla. Tällöin ne on helppo ajaa, kun uusia elementtejä toteutetaan järjestelmään ja mahdolliset virheet voidaan heti havaita. Kuten Rauhala (2010, 29) toteaa, myös järjestelmän virheiden korjaukset voivat aiheuttaa uusia virheitä.

Ongelmia integraatiotestien korjaamisessa aiheutti erityisesti se, ettei kenelläkään ollut selkeää muistikuvaa siitä, miten testit oli aiemmin ajettu onnistuneesti. Tämä saattoi johtua siitä, että testit oli toteuttanut aiempi työntekijä, joka ei enää työskennellyt kehitystiimissämme. Jouduin itse selvittämään testeihin liittyvät ongelmat perin pohjin. Koska testeissä käytettävät tietokantafunktiot eivät palauttaneet dataa ollenkaan, löysin helpon lähestymistavan toimintavirheen jäljittämiseen. Poistin tietokantakyselystä suodattimen pois yksi kerrallaan, kunnes suodattimia ei ollut enää ollenkaan. Tällöin kaiken luettavan datan olisi viimeistään pitänyt näkyä, mutta näin ei kuitenkaan ollut. Tämä tarkoitti, että dataa ei ylipäätään ollut luettavissa tietokannassa. Syyksi paljastui lopulta tietokantasessio, eli transaktio, jossa testidata tallennettiin tietokantaan, oli eri kuin sessio, jossa dataa yritettiin lukea tietokannasta. Testidata syötettiin testitietokantaan, mutta datan lukufunktiot yrittivät yhä lukea sitä tuotantopalvelimelta.

Sessioiden korjaamisen jälkeenkin integraatiotesteissä oli vielä muuta korjattavaa. Koska testejä ei oltu päivitetty pitkään aikaan, odotetut testitulokset eivät enää vastanneet testattavista funktioista saatuja tuloksia. Olen samaa mieltä Konstin (2007, 25) kanssa siitä, että tilanne on erittäin huono uusille testaajille, mikäli testitapaukset eivät ole ajan tasalla. Kun

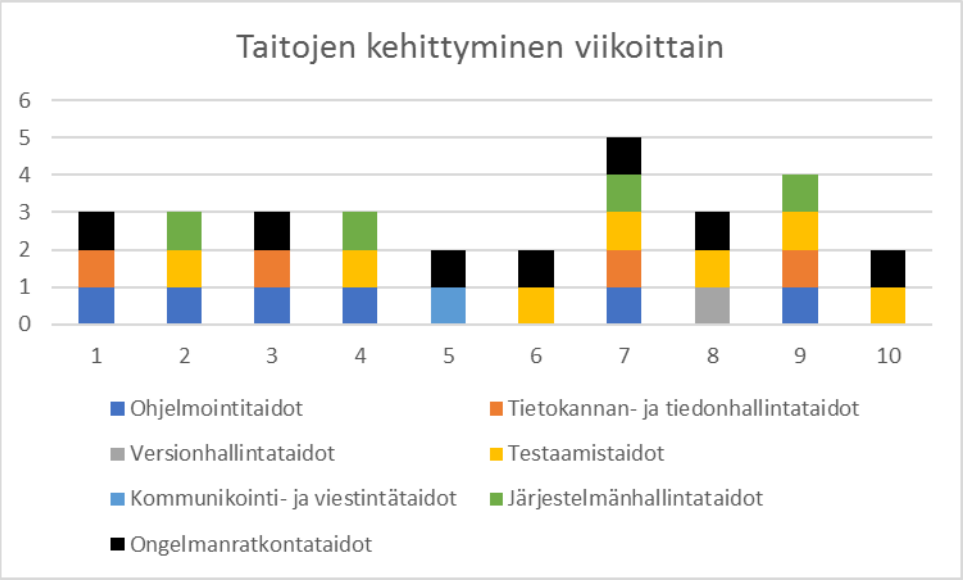
testitapaukset pidetään ajan tasalla, voidaan olla entistä varmempia testauksen laadusta ja tuloksista (Konsti 2007, 25). Siksi minun kannattaisi myös itse panostaa testien ja testitapausten ylläpitämiseen tulevaisuudessa, kun saan testit jälleen toimimaan odotetulla tavalla. Näin muiden kehittäjien ei tarvitsisi joutua minun tilanteeseeni.

4 Pohdinta ja päätelmät

Tässä luvussa analysoidaan omaa ammatillista kehittymistäni seurantaviikkojen pohjalta. Luvussa tullaan käymään läpi muun muassa seurantaviikkojen aikana tekemäni merkittävimmät havainnot, uudet opitut asiat sekä uudet menetelmät. Luku sisältää myös pohdintaa ja mietiskelyä siitä, miten olen pystynyt hyödyntämään päiväkirjassa tapahtunutta työn analysointia sekä miten työtäni voitaisiin kehittää jatkossa.

4.1 Kehittyminen

Omat ammatilliset taitoni ovat kehittyneet merkittävästi 10 seurantaviikon aikana. Kehitystä on tapahtunut useilla eri alueilla. Oheisella kaaviolla voidaan kuvata eri taitojeni kehittymistä sen perusteella, kuinka ne olivat esillä viikkoanalyyseissani (Kaavio 3).



Kaavio 3. Taitojen kehittyminen viikoittain

Kaaviossa olen antanut taidoille aina yhden pisteen per viikko silloin, kun ne ovat analyysieni perusteella mielestäni kehittyneet. Taidoilla tarkoitetaan kyseisiä taitoja kokonaisuudessaan, ei niinkään mihinkään tiettyyn työkaluun tai menetelmään liittyviä taitoja. Järjestelmänhallintataidoilla kuvaan sekä käyttöjärjestelmän hallintaan, että tuotantopalvelimien hallintaan liittyviä taitoja.

Kaaviosta voidaan päätellä, että ainakin ohjelmointitaidot, ongelmanratkontataidot ja testaamistaidot ovat kehittyneet eniten suhteessa muihin työssäni tarvittaviin taitoihini. Tämä todistaakin sen, että ohjelmointityöni on pääasiassa jatkuvaa ongelmanratkontaa. Samalla testaamistaitoni kehittyivät, kun pääsin muun muassa ohjelmoimaan kattavia automaatiotestikokoelmia projekteihimme. Versionhallintataitoni sen sijaan olivat hyvin harvoin työs-

säni koetuksella, johtuen siitä, että hallitsin jo työni alussa kaikki tärkeimmät versionhallintamenetelmät. Kommunikointinikin on ollut lähes jatkuvasti tehtävieni tasolla. Hallitsen Englannin kielen erinomaisesti, joten en ole joutunut sen kanssa ongelmiin vuorovaikutustilanteissa. Muutamia englanninkielisiä ammattisanoja olen toki oppinut lisää.

Aloitusviikolla arvioin itseni taitavaksi suoriutujaksi yleisellä tasolla. Seurantajakson jälkeen arvioisin itseni taitavan suoriutujan ja kokeneen asiantutijan välimaastoon, koska työni on ollut erittäin sujuvaa ja olen pystynyt integroimaan itseni kehitystiimiin osaavaksi asiantuntijaksi. Toisaalta suuressa ja monimutkaisessa tietojärjestelmässä on vielä monia itselleni tuntemattomampia osia, joihin tarvitsen useimmiten kokeneemman kehittäjän opastusta.

4.2 Uudet ratkaisumallit ja menetelmät

Seurantaviikkojen aikana löysin useita uusia menetelmiä työni tekemiseen. Näistä merkittävimmät liittyivät ongelmanratkonta-tilanteisiin, jolloin opin ratkomaan ongelmia tehokkaammin. Yksi metodeista oli se, että ohjelmien bugeja havainnoidessani pyrin tarkastelemaan ohjelman kaikkia elementtejä ajon aikana käyttämällä pdb-moduulia. Mikäli joissakin elementeissä esiintyi poikkeavuuksia, saatoin päätellä bugien syyt usein siitä, missä kyseiset elementit määriteltiin lähdekoodissa. Yritin aina ensi sijassa ratkaista ongelmat itsenäisesti, jolloin minulla oli parempi mahdollisuus oppia tekemistäni havainnoista. Tähän käytin myös apuna sitä, että asetin itselleni usein tietyn aikarajan aamuisin ongelmienratkontaan, jonka jälkeen vasta pyysin apua ongelmaani. Tämän tein siksi, että olin yleensä työpäivän alussa terävimmilläni, jolloin minulla oli parempi mahdollisuus hahmottaa ja ratkaista ongelmat. Työpäivän lopussa sen sijaan keskittymiseni saattoi toisinaan olla kateissa, jolloin muuten helposti huomattava virhe jäi usein huomaamatta. Mikäli en kuitenkaan löytänyt ratkaisua aikarajaan mennessä, tulin konsultoineeksi tiimitoveriani, jotta en jäänyt jumiin liian pitkäksi aikaa työssäni.

Löysin työni aikana uusia työkaluja REST API-funktioiden testaamiseen. Aluksi käytin pelkästään Linux-terminaalissa käytettävää curl-komentoa, jonka lisäksi opin käyttämään verkkoselaimen osoitekenttää. Näistä kahdesta siirryin lopulta käyttämään Postman-ohjelmaa, jolla testaaminen onnistui kätevimmin ja jossa oli kaikki minulle tärkeimmät toiminnot ja ominaisuudet. Samalla opin käyttämään verkkoselaimista löytyviä kehitystyökaluja, joilla pystyin muun muassa tutkimaan verkkosivuilta tehtyjä REST-kutsuja.

Tiedonhallinnassa löysin uusia ratkaisumalleja. Opin optimoimaan tekemiäni tietokantakyselyitä tiivistämällä niiden toimintaa, jolloin pystyin myös vähentämään niiden määrää.

Näin tapahtui esimerkiksi silloin, kun korvasin tietokantakyselyssä tehdyt alikysely-ratkaisut väliaikaistaulu-ratkaisulla tietyissä tilanteissa, jolloin tietokanta-ajurin ei tarvinnut toistaa samaa pohjakyselyä tietokantaan useita kertoja. Itse tietokantakyselyissä tapahtuvia virheitä opin selvittämään paremmin viimeisellä seurantaviikolla. Kyselyt eivät usein palauttaneet dataa ollenkaan. Tällöin otin käyttööni menetelmän, jossa poistin kyselyistä suodattimia yksi kerrallaan, jotta löysin ongelmallisen suodattimen, tai sen, että datan lukeminen ei onnistunut lainkaan.

4.3 Havainnot ja oppiminen

Uudet oppimani asiat painottuivat enimmäkseen tiedonhallintaan ja testaamiseen. Tiedonhallinnassa opin käyttämään alikyselyjä sekä tietojen ryhmittelyä tietokantakyselyissä. Testaamisessa opin muun muassa käyttämään mock-olioita oikein sekä tekemään kattavampia testejä. Python-ohjelmointikielestä löysin uusia pieniä yksityiskohtia, mutta enimmäkseen sen ominaisuudet olivat jo entuudestaan tuttuja. Front endissä tulin oppineeksi jonkin verran lisää JavaScriptin ominaisuuksista testatessani käyttöliittymiä tai REST API-funktioita. DevOps oli mielestäni merkittävin ammattitermi jonka opin työni aikana. Siihen liittyen tulin oppineeksi useita projektimme tuotannolliseen puoleen liittyviä tehtäviä, kuten datan siirtoa.

Työni aikana tulin huomanneeksi, miten monipuolinen Python-ohjelmointikieli todellisuudessa on, kun opin enemmän sen edistyneemmistä ominaisuuksista, kuten Python-olioiden magic methodeista. Näillä ominaisuuksilla pystyn tulevaisuudessa tarvittaessa muuntelemaan olioiden sekä itse ohjelmien käyttäytymistä juuri haluamakseni.

Merkittävin yleinen havaintoni on se, kuinka nopeasti pystyin oppimaan ja sisäistämään itselleni uudet työkalut ja tekniikat. Samalla opin suhteellisen nopeasti ymmärtämään laajojen ja monimutkaisten järjestelmien toimintaa. Ohjelmistokehitystyöhöni sisältyy uusien asioiden oppimista päivittäin, tavalla tai toisella. Enimmäkseen uuden oppiminen liittyi kehitettävän järjestelmän toimintaan, mutta joinakin päivinä opin myös käyttämään uusia työvälineitä ja tekniikoita. Nopeasta oppimisestani on paljon hyötyä tulevaisuudessakin, kun pystyn toimimaan työssäni tehokkaasti sekä laajentamaan tietojani ja taitojani koko ohjelmistokehitysalalla.

Työni alussa nostin tavoitteeksi panostaa suunnitelmallisuuteen työssäni. Seurantaviikkojen aikana opitut asiat ja työtehtävistäni saatu rutiini auttoivat minua kohti suunnitelmallisempaa työtä, kun pystyin hahmottamaan kehitystyöni kohteet paremmin. Työni alussa sen sijaan minulla oli usein epäselvempi kuva kehitettävän järjestelmän rakenteesta.

4.4 Jatkokehitys- ja tutkimusmahdollisuudet

Työni jatkokehityksen aiheeksi voisi mahdollisesti ottaa seurannan jossa keskityttäisiin enemmän front end-taitojeni kehittämiseen. Tämä toki vaatisi sitä, että saisin vastuuta kyseisistä tehtävistä työssäni. Tällä hetkellä olen yhä titteliltäni Python-kehittäjä, eli back end-kehittäjä, mutta mikäli saisin enemmän rutiinia ja itsevarmuutta front end-tekniikoihin, voisin potentiaalisesti kehittyä moniosaavaksi full stack-kehittäjäksi. Full stack-kehittäjällä tarkoitetaan ohjelmistokehittäjää, jolla on kyky työskennellä sujuvasti kaikissa ohjelmistokehityksen osa-alueissa, aina ohjelmiston käyttöliittymästä tietokantatasoon. Tämä voisi siis itsellänikin olla tulevaisuuden tavoitteena. Vaikka en olekaan yhtä rutinoitunut front end-tekniikoiden käytössä, on minulla kuitenkin kohtalaisen hyvä käsitys niiden toiminnasta. Kuten aiemmin todettua, työtehtäväni opettivat minulle jonkin verran uusia asioita front end-puolelta.

4.5 Työn analysoinnin hyödyntäminen

Olen onnistunut tehostamaan oppimistani viikoittaisen analysoinnin avulla, kun olen esimerkiksi opiskellut itselleni uusia tekniikoita erilaisista ulkoisista lähteistä työajan ulkopuolella. Analysoinnissa olen pystynyt syventymään viikon aikana kohtaamiini haasteisiin ja ongelmiin paremmin, jolloin olen ollut huomattavasti viisastuneempi seuraavan viikon alussa. Näin ollen minun ei ole tarvinnut käyttää ylimääräistä aikaa tiettyjen asioiden kertaamiseen työviikon alussa.

Yleisesti voidaan todeta, että päiväkirjamerkintöjen ja viikkoanalyysien tekeminen on auttanut minua laajentamaan tietojani ohjelmistokehityksessä ennestään. Tämä taas auttaa minua eteenpäin urallani, kun tiedostan työni analysoinnin tuoman hyödyn jatkossakin.

Lähteet

Abhilash c. 2012. Agile/Scrum Bug fixing: Myths & dysfunctions. Luettavissa: http://www.theagileschool.com/2012/03/agilescrum-bug-fixing-myths.html. Luettu: 22.4.2017.

Amazon Web Services, Inc. 2017. What is DevOps? Luettavissa: https://aws.amazon.com/devops/what-is-devops/. Luettu: 30.4.2017.

Bird, J. 2012. Fixing Bugs - If You Can't Reproduce a Bug, You Can't Fix It. Luettavissa: https://dzone.com/articles/if-you-cant-reproduce-bug-you. Luettu: 22.4.2017.

Bruhin, F. 5.2.2016. Pytest-kehittäjä. Pytest: Rapid Simple Testing. Pytest. Konferenssi-video. Rapperswil, Sveitsi. Näytettävissä: https://www.youtube.com/watch?v=rCBHkQ_LVIs. Nähty: 16.4.2017.

Burleson Consulting. 2013. Oracle WITH clause to simplify complex SQL. Luettavissa: http://www.remote-dba.net/oracle_10g_tuning/t_oracle_with_sql_clause.htm. Luettu: 19.3.2017.

Cobbaut, P. 2015. Linux Fundamentals. Luettavissa: http://linux-training.be/linuxfun.pdf. Luettu: 26.3.2017.

Haapajärvi, J. 2012. Muutokset ohjelmistokehityksen dokumentointiin yrityksen siirtyessä käyttämään ketteriä ohjelmistonkehitysmenetelmiä. Luettavissa: http://jultika.oulu.fi/files/nbnfioulu-201302131034.pdf. Luettu: 2.4.2017.

Hartikainen, Jani. 2016. What’s the difference between Unit Testing, TDD and BDD? Luettavissa: http://codeutopia.net/blog/2015/03/01/unit-testing-tdd-and-bdd/. Luettu: 16.4.2017.

Hoffman, C. 2015. Dual-booting Linux with Windows: What you need to know. Luettavissa: http://www.pcworld.com/article/2955460/operating-systems/dual-booting-linux-with-windows-what-you-need-to-know.html. Luettu: 26.3.2017.

Holger krekel and pytest-dev team. 2015. Support for unittest.TestCase / Integration of fixtures. Luettavissa: https://docs.pytest.org/en/latest/unittest.html. Luettu: 30.4.2017.

JetBrains s.r.o. 2017. PyCharm 2017.1 Help / Code Running Assistance. Luettavissa: https://www.jetbrains.com/help/pycharm/2017.1/code-running-assistance.html#d7018e217. Luettu: 22.4.2017.

Jyväskylän yliopisto. Testauksen tasot. Luettavissa: http://smarteducation.jyu.fi/projektit/systech/Periaatteet/suunnittelun-periaatteet/testaus/testauksen-tasot. Luettu: 7.5.2017.

Konsti, M. 2007. ULKOISTETUN TESTAUSPROJEKTIN PROSESSI. Luettavissa: http://www.theseus.fi/bitstream/handle/10024/10351/TMP.objres.1010.pdf?sequence=2. Luettu: 7.5.2017.

Metsänheimo, M. 2016. Ohjelmistokehittäjän arki suuryrityksessä. Luettavissa: http://www.theseus.fi/bitstream/handle/10024/108467/Ohjelmistokehittajan%20arki%20suuryrityksessa.pdf?sequence=1. Luettu: 9.4.2017.

Mozilla Foundation. RESTClient, a debugger for RESTful web services. Luettavissa: https://addons.mozilla.org/en-US/firefox/addon/restclient/. Luettu: 11.3.2017.
Niittyviita, A. 2011. Testaus ja laadunvarmistus napit vastakkain. Luettavissa: https://ohjelmistotestaus.fi/2011/06/09/testaus-ja-laadunvarmistus-napit-vastakkain/. Luettu: 22.4.2017.

Pentaho Community. Wiki. My transformation is running slow, what do I do?!. Luettavissa: http://wiki.pentaho.com/pages/viewpage.action?pageId=13176788. Luettu: 16.4.2017.

Pentaho Corporation. 2017. Data Integration. Luettavissa: http://www.pentaho.com/product/data-integration. Luettu: 16.4.2017.

Postdot Technologies, Inc. 2016. Writing Tests. Luettavissa: https://www.getpostman.com/docs/writing_tests. Luettu: 26.3.2017.

Python Software Foundation. 2017. Python 2.7.13 documentation. Luettavissa: https://docs.python.org/2/. Luettu: 19.3.2017.

Python Software Foundation. 2017. Python 3.6.1rc1 documentation. Luettavissa: https://docs.python.org/3/. Luettu: 19.3.2017.

Rauhala, E. 2010. Ohjelmistotestauksen suunnittelu - Case: A-lehdet Oy:n laskujen tulostusohjelma. Luettavissa: http://www.theseus.fi/bitstream/handle/10024/23687/Rauhala_Eija.pdf?sequence=1. Luettu: 7.5.2017.

Riggs, S. & Krosing, H. 2010. PostgreSQL 9 Admin Cookbook. Packt Publishing. Yhdistynyt kuningaskunta. Luettavissa: http://ebookcentral.proquest.com/lib/haaga/reader.action?docID=952059. Luettu: 25.2.2017.

Saleh, H. 2013. JavaScript Unit Testing. Packt Publishing. Birmingham, Yhdistynyt Kuningaskunta. Luettavissa: http://ezproxy.haaga-helia.fi:2077/lib/haagahelia/reader.action?docID=10648835. Luettu: 16.4.2017.

Sneeringer, L. 2015. Professional Python. John Wiley & Sons, Incorporated. Yhdysvallat. Luettavissa: http://ebookcentral.proquest.com/lib/haaga/reader.action?docID=4187169. Luettu: 25.2.2017.

Tervola, M. 2015. Vältä sisäisen viestinnän sudenkuopat. Luettavissa: http://www.talouselama.fi/tyoelama/valta-sisaisen-viestinnan-sudenkuopat-3397543. Luettu: 2.4.2017.

Väyrynen, V. 2009. Onnistunut testaus ja ketterät menetelmät. Luettavissa: http://www.theseus.fi/bitstream/handle/10024/7221/Vayrynen_Virve.pdf. Luettu: 11.3.2017.

Wang, Q. 2011. PostgreSQL database performance optimization. Luettavissa: http://www.theseus.fi/bitstream/handle/10024/29980/ThesisReport_Qiang_Wang.pdf. Luettu: 4.3.2017.

Liitteet

Liite 1. Ammattikäsitteet

- Aliasointi – Jonkin ohjelmallisen olion uudelleen nimeäminen ohjelman lähdekoodissa
- Back end – Ohjelmiston osa, jossa tietoa tallennetaan, luetaan, muokataan tai poistetaan tietokannasta.
- Branch – Versionhallinnassa tietystä ohjelman versiosta käytettävä termi, jossa ohjelman kehitystyö tapahtuu
- Bugi – Ohjelman toimintavirhe
- Chrome - Verkkoselain
- Curl – Linuxin komentorivissä käytettävä komento, jolla pystyy kutsumaan eri verkko-osoitteita tai tekemään REST API-kutsuja
- Data – Tietokantaan tallennettava tieto
- Deadlock – Ohjelman tila, jossa kaksi prosessia ovat ns. umpikujassa, koska ne tarvitsevat toistensa varattuja resursseja suorittaakseen itsensä loppuun
- Debug – Ohjelman toiminnallisen virheen havainnollistamiseen ja korjaamiseen tähtäävä toiminta
- Dekoraattori – Operaattori, jolla tietty toistuva toiminnallisuus voidaan asettaa ohjelman eri funktioihin tai muuttujiin
- DevOps – Ohjelmistokehityksen toimintamalli, jossa ohjelmiston kehittäminen ja tuotannolliset operaatiot on jaettu saman kehitystiimin kesken
- Dict – Pythonin Dictionary-olio, johon pystyy tallentamaan dataa avain-arvo muodossa (avaimella viitataan arvoon)
- DynamoDB - Amazon Web Servicesin tarjoama tietokantapalvelin
- Firefox - Verkkoselain
- Flake8 – Pythonin lähdekoodin tarkistamiseen tarkoitettu ohjelma, joka löytää koodista kaikki tyylivirheet ja raportoi ne
- Flask – Web-kehys Python-ohjelmille
- Flask-RESTful – Laajennus Flask-kehykseen jolla pystyy nopeasti rakentamaan REST API-teknologiaan perustuvia ohjelmia
- Front end – Ohjelmiston osa, jossa tieto esitetään ohjelmiston käyttäjälle ja jonka kautta käyttäjä on vuorovaikutuksessa ohjelmistoon
- Full stack – Ohjelmistokehittäjä, jolla on kyvyt työskennellä sujuvasti kaikissa ohjelmistokehityksen osa-alueissa
- Git – Versionhallintatyökalu ohjelmistokehitykseen

- Hackathon – Ohjelmistokehitys-tapahtuma, jossa tavoitteena on kehittää uusi pala ohjelmistoa nopealla aikataululla
- Integraatiotesti – Testi, jolla testataan järjestelmän yhden tai useamman osan toimivuutta toisen osan kanssa, esim. sovelluksen testaus tietokannan kanssa
- Java – Ohjelmointikieli
- JavaScript – Ohjelmointikieli, jota käytetään interaktiivisten verkkosivujen toteuttamisessa
- JIRA – Tehtävänhallintatyökalu ohjelmistokehitykseen
- Kinesis – Amazon Web Servicesin tarjoama tekniikka isojen data-määrien prosessointiin
- Konflikti – Tilanne versionhallinnassa, jossa ohjelman lähdekoodin yhdistäminen toiseen ohjelmaversioon ei onnistu automaattisesti, koska samaa koodia on muokattu molemmissa versioissa
- Konsoli – Näkymä, johon ohjelmalla voidaan helpoiten tulostaa ohjelman käyttämiä tietoja
- Linux – Käyttöjärjestelmä
- Magic method – Python-olioille asetettava metodi, jota ohjelma kutsuu tietyissä tilanteissa ilman, että metodia tarvitsee kutsua lähdekoodissa
- Mock – Testaamisessa käytettävä termi oliosta, jota käytetään testiympäristössä jäljittelemään oikeaa oliota ohjelman ajoympäristöstä
- Pentaho – ETL-työkalu, jolla pystyy integroimaan dataa halutussa muodossa
- PostgreSQL – Tietokannan hallintajärjestelmä
- Postman – Chrome-verkkoselaimen laajennus, jolla pystyy tekemään REST API-kutsuja visuaalisen käyttöliittymän kautta
- Pull request – Versionhallinnassa käytettävä termi pyynnölle yhdistää valmis ohjelman ominaisuus tai muutos ohjelman käyttövalmiiseen versioon
- PyCharm – Ohjelmointiympäristö, tekstieditori, joka on suunniteltu Python-ohjelmien kehittämiseen
- Pytest – Kehys Python-ohjelmien automaatiotestien ohjelmoimiseen ja ajamiseen
- Python – Ohjelmointikieli
- Redshift – Amazon Web Servicesin tarjoama tietokantapalvelin
- REST Client – FireFox-verkkoselaimen laajennus, jolla pystyy tekemään REST API-kutsuja visuaalisen käyttöliittymän kautta
- S3 – Amazon Web Servicesin tarjoama tiedostovarastopalvelu
- Scrum – Ohjelmistokehityksen menetelmä, jossa kehitettävää ohjelmistoa tuotetaan iteratiivisesti, pala kerrallaan
- Slack – Ohjelma reaaliaikaiseen viestintään

- Sphinx – Dokumentaation generoimiseen tarkoitettu ohjelma joka tukee rst-tyyppisiä tekstitiedostoja
- Sprint – Scrum-metodissa käytettävä ajanjakso, jonka aikana tietyt ohjelman ominaisuudet pyritään saamaan valmiiksi asiakkaalle käyttöä varten
- Sprint-palaveri – Kokous, jossa viimeisimmän Sprintin tuloksia tarkastellaan
- Startup – Termi jota käytetään yleensä yrityksestä, joka on nuori ja kasvuhakuinen, eli vastaavana terminä kasvuyritys
- String – Muuttujatyyppi, joka käsittelee merkkijonoja
- Sublime Text – Monipuolinen tekstieditori edistyneeseen tekstin muokkaamiseen
- Tox – Ohjelma, jolla voi kätevästi ajaa erilaisia testejä automaattisesti
- Transaktio - Tietty sarja toimintoja tietokannassa, jotka määritellään yhdeksi toiminnaksi, joka peruutetaan, mikäli yksikin osa epäonnistuu
- Unicode - Muuttujatyyppi, joka käsittelee merkkijonoja
- Unittest – Kehys Python-ohjelmien automatisoiduille yksikkötesteille
- UTF-8 standardi – Kirjainavaruus
- Yksikkötesti – Testi, jolla testataan vain pientä osaa ohjelmasta, eli tyypillisesti vain yhden funktion toimintaa

Liite 2. Lyhenteet

- API – Application Programming Interface, eli ohjelmointirajapinta, jolla eri ohjelmat voivat kommunikoida keskenään
- ASCII - American Standard Code for Information Interchange, eli kirjainavaruus
- AWS – Amazon Web Services
- DMS –Database Migration Service, eli AWS tiedonsiirtopalvelu
- ETL – Extract, Transform, Load, eli menetelmä, jolla dataa voidaan lukea määritellystä lähteestä, muuntaa haluttuun muotoon ja ladata sen jälkeen määriteltyyn kohteeseen
- JSON - JavaScript Object Notation, eli formaatti, jolla halutut tiedot voidaan esim. tulostaa siistissä, luettavassa muodossa
- RDS - Relational Database Service, eli AWS tietokantapalvelu
- REST API - Representational state transfer, Application Programming Interface
- RST (reST) – ReStructured Text, eli tekstitiedostotyyppi, jolla luodaan dokumentaatio rakenne generoitavaksi Sphinx-ohjelmalla
- TDD – Test Driven Development, ohjelmistokehitysmenetelmä jossa ohjelman testit luodaan ennen ohjelman toiminnallisuutta