Haaga-Helia
ammattikorkeakoulu Oy

Ammatillinen kehittyminen aloittelevana liiketoimintalogiikan ohjelmistokehittäjänä

Meri Eskelinen

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma
2021

Haaga-Helia ammattikorkeakoulu Oy

Tiivistelmä

Päiväys 12.5.2021

Tekijä(t)
Meri Eskelinen

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma

Opinnäytetyön otsikko
Ammatillinen kehittyminen aloittelevana liiketoimintalogiikan ohjelmistokehittäjänä

Sivu- ja liitesivumäärä
87 + 1

Opinnäytetyön otsikko englanniksi
Professional growth as a beginner-level back-end software developer

Tämä opinnäytetyö on suoritettu osana Tietojenkäsittelyn koulutusohjelmaa keväällä 2021. Työ kuvaa päiväkirjamallisena opinnäytteenä aloittelevan liiketoimintalogiikan ohjelmistokehittäjän näkökulmasta sitä, miten työhön siirtyminen ja alati muuttuva ohjelmistoala haastaa tekijänsä päivittäin. viikon seurantajaksossa tekijä raportoi päivittäisiä työtehtäviään ja niiden myötä esiin nousseita oppeja. Päivittäisestä raportoinnista nostetaan viikoittaisiin analyyseihin muutamia aiheita sekä teknisistä että projektityöskentelyn näkökulmista.

Opinnäytetyö suoritetaan kansainvälisessä SaaS-tuotteita tarjoavassa yrityksessä, jonka PSA-järjestelmää kehittävässä tiimissä tekijä työskentelee täysipäiväisesti. Tiimissä on noin 30 työntekijää, joiden työtehtävät on selvästi jaettu asiantuntijoittain. Tekijä työskentelee liiketoimintalogiikan ohjelmoinnin parissa. Kehitettävästä tuotteesta on olemassa sekä vanha, käytöstä tämän vuoden aika poistuva versio sekä uudistettu, myös rajapinnan tarjoava versio.

Ohjelmistokehitys on alati muuttuva ala, jossa aloitteleva ohjelmistokehittäjä törmää uusiin haasteisiin päivittäin. Päivittäinen raportointi tuo hyvin esiin sen, miten vaihtelevia työtehtäviä ohjelmistokehittäjän työnkuvaan kuuluu ja kuinka monia taitoja kehittäjän tulee omata suoriutuakseen työstään. Ohjelmointitaitojen lisäksi työ edellyttää myös tiimi- ja projektityöskentelytaitoja, sekä pehmeitä taitoja.

Viikoittaisista analyyseista voidaan huomata, kuinka pilvipalvelun kehittäminen vaatii tekijältään motivaatiota oppia uutta ja pohtia ohjelman toimivuutta monesta näkökulmasta. Lisäksi jatkuvan integraation ja julkaisuputken määrittelevät vaatimuksia kehittämiselle. Ohjelmistokehittäjän on tärkeä ymmärtää, minkälaiset vaikutukset pienilläkin muutoksilla on isossa järjestelmässä ja kuinka web-sovelluksen sekä rajapinnan kautta tarjottavaa ohjelmistoa voidaan käyttää myös oletetusta käyttäjäpolusta poikkeavasti. Muut tärkeät huomiot työssä koskevat esimerkiksi .NET ympäristössä työskentelyn asettamia vaatimuksia, haasteita siirryttäessä vanhasta järjestelmästä uuteen, sekä kehittämistä etätyöskentelyn aikakautena.

Asiasanat
Ohjelmistokehitys, liiketoimintalogiikka, web-sovellus, oppiminen

Sisällys

1 Johdanto

Tämä opinnäytetyö on kirjoitettu osana Haaga-Helian ammattikorkeakoulun Tietojenkäsittelyn koulutusohjelmaa. Opinnäytetyön tekijä tietojenkäsittelyn tradenomiopiskelija ja ohjelmistokehitykseen profiloitunut Meri Eskelinen.

Päiväkirjamuotoisessa opinnäytetyössä tarkastelen ammatillista kehittymistäni työskennellessäni juniortason ohjelmistokehittäjänä kansainvälisessä ohjelmistotalossa. Opinnäytetyössä lähtötilanteen kartoitus luo perustan tämänhetkisille taidoilleni ja vertaa niitä työnkuvani vaatimaan osaamiseen. Kymmenen viikon seurantajakso sijoittuu aikavälille 15.2.-30.4.2021 ja sen aikana kertyneen päivittäisen raportoinnin avulla voidaan muodostaa syventäviä, viikoittaisia analyyseja työtehtävieni myötä nousseista aiheista. Sekä teknisen osaamisen, että projektityöskentelyn kannalta kiinnostavia oppeja voidaan sitten peilata omaan ammatilliseen kehittymiseeni. Seurantajakson jälkeen voidaan ammattitaitoni kehittymistä tarkastella lähtötilanteeseen verrattuna ja tunnistaa sitä edistäneitä tekijöitä.

Aloitin työskentelyn harjoittelujakson päätyttyä ison kansainvälisen yrityskonsernin tytäryhtiössä. SaaS-, eli pilvipalvelutuotteiden kehittäjänä vuonna 2013 perustetun yrityksen toiminta perustuu räätälöityjen ohjelmistoratkaisujen tarjoamiseen asiakkaidemme liiketoiminnan kehittämiseksi. Yritys myy viittä erilaista ohjelmistoa ja tuottaa suurimmilta osin itse tuotteiden kehittämisen, markkinoinnin, myynnin sekä asiakaspalvelun. Yritys työllisti vuonna 2019 yli 200 henkilöä tehden liikevoittoa noin 23,7 miljoonaa euroa (Suomen asiakastieto s.a.) ja toimii pääosin Suomen markkinoilla. Yrityksen päätoimisto sijaitsee pääkaupunkiseudun ulkopuolella, mutta itse työskentelen yrityksen Helsingin toimipisteessä.

Yrityksen tulevaisuuden tavoitteena on saavuttaa yli miljardin U.S. dollarin yritysarvo yksityisomistuksessa, eli tulla yksisarviseksi. Tavoite ei ole mahdoton, sillä tammikuussa 2021 Statista-nimisen statistiikkaa julkaisevan verkkosivuston tutkimusosasto ennusti SaaS-markkinan arvon nousevan 137 biljoonaan U.S. dollariin vuonna 2020 ja näin tuplaavan arvonsa sitten vuoden 2014. Statista mainitsee suurimpina SaaS-taloina, sekä niin ollen yrityksen kilpailijoina Microsoftin sekä Salesforcen. Yritys pyrkii sen lisäksi kehittämään toimintaansa työnantajana ja sijoittui vuonna 2020 kärkiviisikkoon Suomen Parhaat Työpaikat tutkimuksessa. (Great Place to Work 2020; Statista 2021; Pride 2018, The Wiley Template.)

Tiimini kehittää PSA-järjestelmää, joka tarjoaa käyttäjilleen työkaluja muun muassa projektinhallintaan ja resursointiin, laskutukseen, raportointiin sekä työajanseurantaan. Kuten SaaS-tuotteelle on tyypillistä (Fröschle & Reinheimer 2010, luku 1.1), tuote on joustavasti kustomoitavissa ja sen työkaluista voidaan koota hyvinkin asiakaskohtaisia ratkaisuja. Asiakaskohtaisista kokonaisuuksista eritellään sitten käyttäjäkohtaisia asetuksia ja oikeuksia ohjelmiston työkalujen käyttöön. Ohjelmistosta sekä sen lisäosista maksetaan käytön mukaan. SaaS-tuotteen isoin hyöty yrityksille on liiketoiminnan kannalta välttämättömän ohjelmiston kehityksen, sen ylläpidon ja infrastruktuurin ulkoistaminen ja näin yrityksen resurssien vapauttaminen sen ydintoimintaan (Kavis 2014, luku 2).

1.1 Tietoperusta

Tärkein tietoperustan tarjoaja päivittäisessä työssäni on verkkoselaimen hakukone. Päivittäisessä työssäni törmään useimmiten erittäin rajattuihin ongelmiin - kuten tietynlaisen syntaksin tarpeeseen - ja Google on kaikkein nopein tapa löytää monipuolista tietoa aiheesta kuin aiheesta. Ohjelmistopohjamme on Visual Basic ja C#-ohjelmointikielillä rakennettu ja olio-ohjelmoinnin perusteiden osaamisella pääsee näissä kielissä pitkälle. Tarvitessani sanakirjaa ohjelmointikielien kanssa, löytyy Microsoftin sivuilta loistava tietokanta .NET-kehittäjälle (Microsoft 2021a). Sen lisäksi verkkolehti Medium tarjoaa valtavan määrän ajankohtaisia ICT-alan asiantuntijoiden kirjoittamia artikkeleita.

Aloittelevana ohjelmistokehittäjänä olen hankkinut osaamiseni tueksi Robert C. Martinin kirjat ”Clean Code” (2009), sekä ”Clean Coder” (2011). ”Clean Code” -kirja on käytännönläheinen opas ajattelevaiseen ohjelmistokehitykseen ja kuvaa yksinkertaisen, mutta sivistyneen ohjelmoinnin periaatteita. Kirja käy läpi universaaleja ohjelmointikäytäntöjä, perustelee konkreettisin koodiesimerkein niistä parhaat ja peräänkuuluttaa pitkäkatseista ohjelmistokehitystä. Martin kirjoittaa muun muassa ohjelmakommentoinnista, nimikonventioista sekä intuitiivisista tietorakenteista ja funktioista. ”Clean Coder” on hyvä perusteos siitä syystä, että Martin perustaa väitteensä kymmenien vuosien projektityöskentelyn kokemuksella. Kirjassaan ”Clean Coder” Martin kirjoittaa ohjelmoijan vastuista, kuten ohjelmoinnin laadun tarkastelusta ja tehokkaasta ajankäytöstä. ”Clean Coder” tarjoaa näin päteviä ohjeita aloittelevalle ohjelmistokehittäjälle myös projektiryhmän jäsenenä.

Alemman logiikan ohjelmointiongelmiin käytän Eric Freemanin ja Elisabeth Robsonin kirjaa ”Head First Design Patterns” (2014). Isossa toiminnanohjausjärjestelmässä on paljon käyttöä erilaisille *ohjelmointimalleille*, kuten tehtaille ja yhden instanssin esiintymille - eli Singletoneille, sekä taipuvaiselle hierarkiarakenteelle. Kirja hahmottelee monimutkaisiakin ohjelmointimalleja helposti ymmärrettävällä tavalla käytännön esimerkkejä käyttäen.

Tiimimme projektityöskentely sijaitsee ketterien ohjelmistokehitysviitekehyksien välimaastossa. Työskentelemme kahden viikon sprinteissä, joissa pidämme joka-aamuisen palaverin, mutta sprintin lopuksi tehtävään retrospektiiviin olemme ottaneet käyttöön Agilen Sailboat-harjoitteen. Kattava perusteos ketteristä projektityöskentelyn käytännöistä on Chrisoph Schmidtin kirja ”Agile Software Development Teams” (2015). Kirja käy läpi ICT-alan projektityöskentelyn historiaa, vertaa vanhaa vesiputouskehittämistä nykypäivän iteratiivisiin työtapoihin ja syventyy muutamaan ketterään viitekehykseen tarkemmin.

1.2 Keskeiset käsitteet

Opinnäytetyössäni käsittelen paljon teknisiä käsitteitä, lyhenteitä sekä ammattisanastoa. Työssäni esiintyvät keskeisimmät käsitteet on selitetty tässä luvussa. Suurin osa päivittäisestä työssäni käyttämästä kielestä sisältää yleisesti alalla tunnettuja vieraskielisiä sanoja, jotka on suomennettu työssä kieliopillisista syistä. Suomennetut sanat saattavat olla hankalia tunnistaa. Siksi työssä käyttämäni vieraskielisten sanojen käännökset löytyvät liitteestä 1 Vieraskieliset sanat suomennettuna. Suomennetut sanat voi tunnistaa niiden ensimmäisen esiintymän kursivoidusta muotoilusta.

BDD BDD, eli käyttäytymislähtöinen kehitys on ketterän kehityksen viitekehys, jossa nimensä mukaisesti kehitys tapahtuu ohjelmakäyttäytymisen vaatimusten lähtökohdista. Tällä testilähtöisestä kehityksestä periytyvällä metodilla pyritään luomaan ihmisluettavia kuvauksia siitä, miten kehitettävän ohjelman odotetaan käyttäytyvän. BDD:n katsotaan edesauttavan projektiryhmän sidosryhmien keskinäisessä ymmärryksessä ja kommunikoinnissa. (Elliot 25.5.2019) (Engl. *Behavior-driven-development.*)

Big-O-notaatio Yleistynyt notaatio algoritmin aikavaativuuden merkkaamiseen. Esimerkiksi. Mikäli algoritmi on vakioaikainen - eli suorittaa kiinteän

määrän komentoja syötteen koosta huolimatta – merkattaisiin se Big-O notaatiolla *O(1)*. (Laaksonen 2020, s.7–12.)

CI/CD Jatkuva integraatio ja julkaisu on ohjelmakehitysmuoto, jolloin järjestelmän julkaistua versiota päivitetään jatkuvana prosessina. Jatkuva integraatio mahdollistaa nopean reagoinnin virhetilanteisiin, sekä julkaistun ohjelmiston samanaikaisen kehityksen. (Fowler 1.5.2006.) (Engl. *Continuous integration / delivery*.)

CoP Samasta aiheesta kiinnostuneiden ihmisten ryhmittäminen, jolloin kyseinen ryhmä voi yhdessä syventää ymmärrystään koko ryhmälle mielenkiintoisesta aiheesta. (Jubi & Sankar 2015, 9.) (Engl. *Community of Practice*)

DoD Kuvaus, jolla ketterässä kehityksessä määritellään, milloin tehtävä, tuote tai sen toiminto on valmis. Madan (16.12.2019) määrittelee DoD:in sisältävän kolme komponenttia: liiketoiminnalliset ja toiminnalliset vaatimukset, laatuvaatimukset, sekä ei-toiminnalliset vaatimukset. (Engl. *Definition of Done*.)

ERP ja PSA Usein SaaS-palveluna tuotettu monipuolinen liiketoiminnan prosessien automatisointiin tarkoitettu ohjelmisto. Tyypillisesti tällainen järjestelmä tarjoaa työkaluja projektinhallintaan, logistiikan ja talouden hallinnointiin, sekä niistä raportointiin. Yritykseen hankittu järjestelmä on usein asiakaskohtaisesti räätälöity kokonaisuus. PSA-järjestelmä keskittyy ERP-järjestelmää pienemmälle toiminta-alueelle jättämällä pois esimerkiksi tuotannon- ja logistiikanohjauksen. (Nestell & Olson 2018, 2–3.) (Engl. *Enterprise resource planning, Project Services Automation*.)

Idempotentti Ohjelmistokehityksessä idempotentilla voidaan tarkoittaa metodia, jolla on sen kutsukerroista riippumatta aina sama lopputulos, eikä sillä siis ole sivuvaikutuksia. Esimerkiksi REST-kulttuurissa GET-kutsu, eli tiedonhaku tuottaa aina saman lopputuloksen huolimatta siitä, tehdäänkö kutsu kerran vai miljoona kertaa. (Fielding & Reschke 1.7.2014, 22–23)

IoC Ohjelmistokehityksen periaate, jossa nimensä mukaisesti käännetään järjestelmän ohjaus**.** Esimerkiksi sen sijaan, että joka kerta tiettyä luokkaa tarvittaessa siitä luotaisiin uusi instanssi, alustetaan luokka kerran instanssiksi, joiden ominaisuuksia niitä tarvitsevat luokat sitten kutsuvat tarpeen vaatiessa. "*Don't call us, we'll call you."* (Fowler 26.7.2005) (Engl. *Inversion of Control*.)

Liiketoimintalogiikka

Liiketoiminnan sääntöjä noudattava ohjelmistologiikka tai algoritmi, joka sijoittuu käyttöliittymän ja tietokannan tai muun infrastruktuurin väliin. Liiketoimintalogiikka määrittelee, miten järjestelmän dataa käsitellään, näytetään ja säilötään. (Frankenfield 29.8.2020.)

Quality Circle Yleistynyt projektityöskentelyn prosessi, jossa tiimin sen hetkinen prosessin mahdollisuudet ja haasteet arvioidaan ennakoiden. Laatupiirillä pyritään ennakoimaan tulevia uhkia ja suunnitellaan asianmukaisia muutoksia. (Jubi & Sankar 2015, 9.)

QA Laadunvarmistaja, eli tiimissä työskentelevä automaatio- ja manuaalitestauksesta vastaava henkilö, jonka tehtävä on varmistaa, että ohjelmistokehittäjältä tullut muutos vaikuttaa olemassa olevaan ohjelmistoon hallitusti ja täyttää sille asetetut vaatimukset. (Bogue 2005.) (Engl. *Quality Assurance*.)

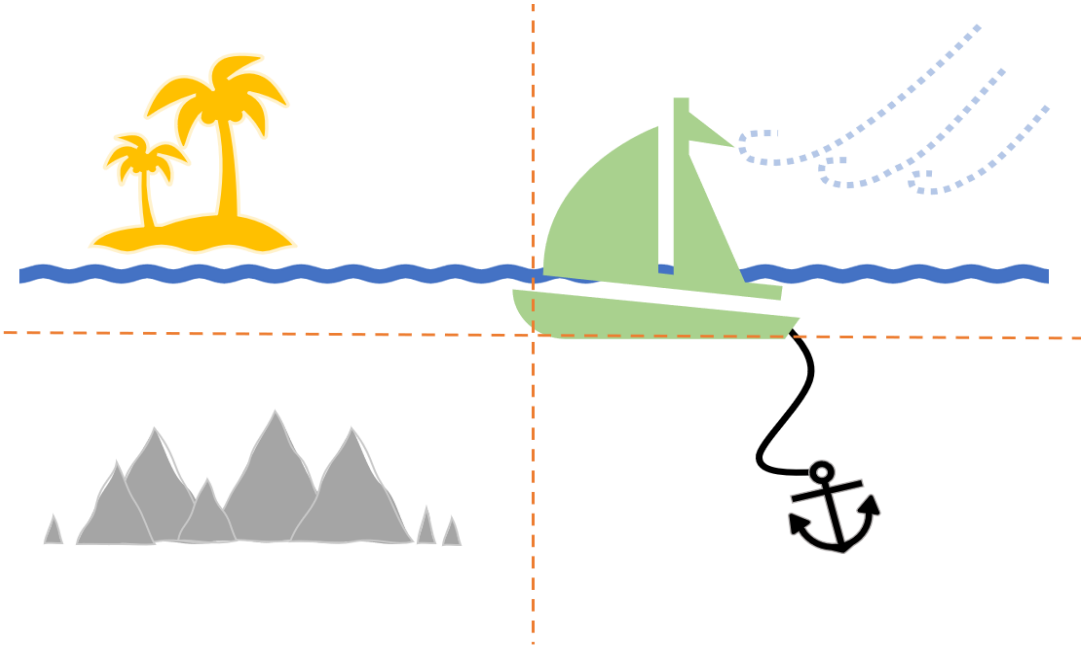
REST HTTP-protokollaan perustuva verkkosovellus arkkitehtuurityyli, jonka periaatteita noudattaen sovellus pyritään luomaan skaalautuvaksi, muokattavaksi, nopeaksi ja luotettavaksi. (Belida & Varanasi 2015.) (Engl. *Representational State Transfer)*

SaaS Palveluntarjoajan ylläpitämällä infrastruktuurilla sijaitseva ja tyypillisesti selaimen kautta tarjottava järjestelmä, josta asiakas maksaa käytön mukaan. SaaS-tuotteen etuja ovat esimerkiksi sen yhteensopivuus useimpiin laitteisiin, sekä ohjelmiston vaatiman kehittämisen, ylläpidon sekä infrastruktuurin kulujen ulkoistaminen. (Blokland, Mengerink & Pol 2013, luku 2.2.) (Engl. *Software-as-a-Service*)

SOAP Xml-pohjainen ja kevyt, verkon välityksellä toimitettu tiedonsiirtoprotokolla. SOAP:in tarkoitus on toimia tiedonsiirrossa järjestelmien välillä niiden alustasta ja ohjelmointikielestä riippumatta. SOAP määrittelee oman sopimuksensa tiedonsiirron muotoon ja tarjoaa siihen turvalliset salaukseen liittyvät säännöt. (Kanjilal 2013, luku 1.1.3.) (Engl. *Simple Object Access Protocol*)

Sailboat Sprintin lopuksi tiimin sisäiseen kommunikointiin käytettävä harjoite. Sailboatin tarkoitus on nostaa esiin sprintin aikana löydettyjä ongelmia ja riskejä, sekä auttaa tiimiä löytämään kollektiivisia tavoitteita ja hahmottamaan sen tarpeita. Sailboatin visuaalisessa ilmeessä (Kuva 1) paratiisisaarelle sijoitetaan tiimin tavoitteet, pohjakiville uhat ja riskit, ankkurille hidasteet, sekä purjeisiin projektia edistävät ja helpottavat asiat. (Torstensson 14.11.2020.)

Singleton Ohjelmistokehityksen *suunnittelumalli*, jossa tilattomasta luokasta annetaan alustaa vain yksi instanssi. Instanssin ominaisuuksia voidaan sitten hyödyntää muussa järjestelmässä tarvittaessa, ilman tarvetta luoda siitä uutta instanssia joka kerta. (Bates, Freeman, Robson & Sierra 2004, 172–188.)



Kuva 1. Sailboat retrospektiivin kaava, muunneltu Torstenssonin (14.11.2020) julkaisemasta kaavasta.

1.3 Anonymiteetti

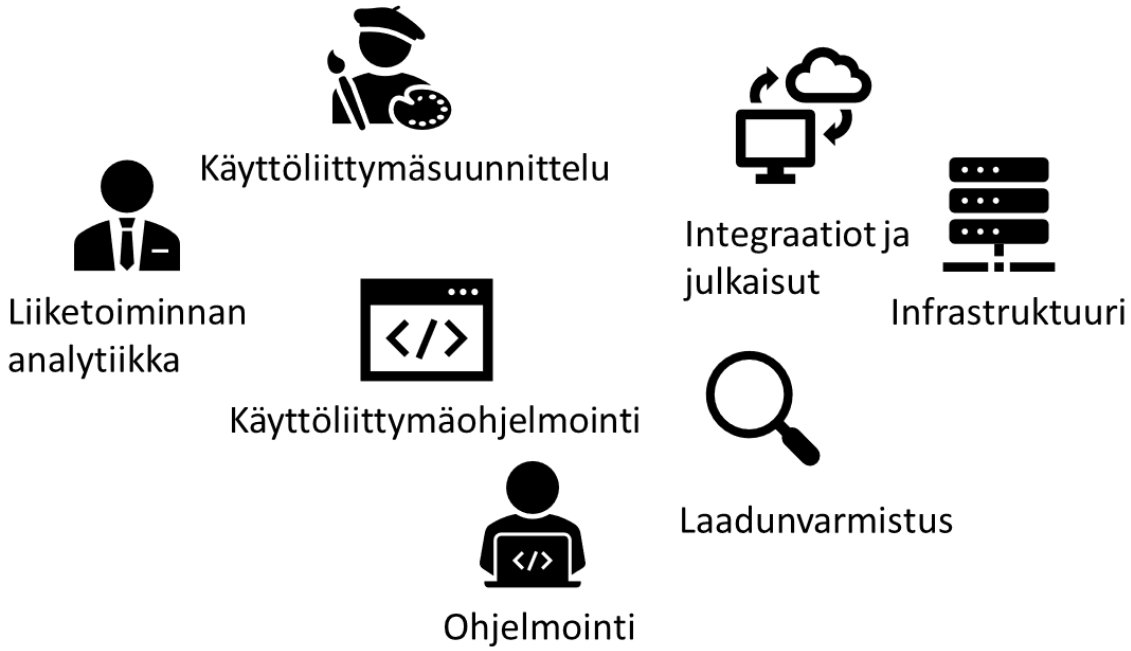
Yritystä ja sen työllistämää henkilöstöä käsitellään työssä anonyyminä yrityksen toiveiden mukaisesti. Nimet, joilla työssä kehitettäviä tuotteita kutsutaan, on muutettu anonyymiyden varmistamiseksi.

1.4 Tutkimuseettiset käytännöt

Opinnäytetyössä ei tulla käsittelemään henkilötietoja tai tutkimuksen ulkopuolella julkaisematonta materiaalia. Opinnäytetyö ja sen tekijä sitoutuu tutkimuseettisen neuvottelukunnan 2012 julkaisemaan ohjeeseen (TENK), sekä ammattikorkeakoulujen rehtorineuvoston julkaisemiin eettisiin ohjeisiin (Arene s.a.).

2 Lähtötilanteen kuvaus

Työskentelen 25 henkilön tiimissä, jossa ohjelmistokehityksen monimutkaiset tehtäväalueet on jaettu asiantuntijakohtaisesti kuvan 2 mukaisesti. Ohjelmointiin liittyvät tehtävät, sekä koodi käyttöliittymän ja siitä alemman ohjelmistologiikan osalta on erotettu omiksi kokonaisuuksiksi. Ohjelmoinnin osalta tiimimme on jaettu pienempiin 6–8 henkilön lähitiimeihin, joihin kuuluu käyttöliittymä- ja alemman tason ohjelmoijia, sekä laadunvarmistaja.



Kuva 2. Projektiryhmän tehtäväalueiden jaottelu.

BAUX-tiimimme koostuu liiketoiminnan analytiikasta ja käyttöliittymäsuunnittelusta vastaavista kollegoistani. Liiketoiminnan analytiikasta vastaavat henkilöt suunnittelevat tuotteen uusia ominaisuuksia käyttökokemus sekä liiketoiminnallinen hyöty-haittasuhde edellä. Analytiikan perusteella käyttöliittymäsuunnittelijat voivat luoda visuaalisia prototyyppejä suunnitelluista toiminnoista. Heidän tärkein tarkoituksensa on kehittää käyttöliittymä mahdollisimman käyttäjäystävälliseksi ja intuitiiviseksi, sekä auttaa muuta tiimiä hahmottamaan tulevia muutoksia.

Käyttöliittymä- eli **UI**-ohjelmoija kehittää nimensä mukaisesti käyttöliittymää prototyypin pohjalta ja kasaa tarvittavat toiminnallisuudet, sekä osan ohjelmalogiikasta konkreettiseksi näkymäksi. Alemman logiikan eli **Core**n ohjelmoija yhdistää lopuksi tietokannan ja käyttöliittymän toimivaksi kokonaisuudeksi. **QA**, eli laadunvarmistus tarkistaa jokaisen

ohjelmamuutoksen toimivuuden manuaalisesti, sekä jatkuvasti uusia automaattisia testejä luoden. **Infrastruktuurin** ja **integraatioiden** vastuuhenkilöt pitävät huolen ohjelmiston kehitykseen tarvittavista alustoista, tietokannasta, sekä ohjelmistojulkaisuista. Omat työtehtäväni sijoittuvat alemman logiikan ohjelmistokehitykseen.

Tiimimme keskeinen tavoite on ottaa vanha tuotteemme – jota tässä yhteydessä kutsutaan nimellä **OldTimer** - pois käytöstä ja siirtyä vanhasta uuteen - **Newbie** - tämän vuoden sisällä. Sen lisäksi tuotteestamme halutaan julkaista julkinen REST API tämän vuoden aikana.

2.1 Työtehtäväanalyysi

Keskeisin työtehtäväni tiimissä on olemassa olevan ohjelman ylläpito, ohjelmistovikojen korjaus sekä uusien toimintojen kehitys. Ylläpitotehtävät päivittyvät laadunvarmistuksen, muiden kehittäjien ja asiakaspalvelunkin kautta tulevin vihreilmoituksin ja kehitysehdotuksin. Ylläpitotehtävät saattavat koskea sekä uutta että vanhaakin järjestelmää. Ylläpitotehtäviin liittyy aika ajoin myös tietokannan käsittelyä, jolloin työskentely vaatii erityistä tarkkaavaisuutta, sekä SQL-tietämystä. Osa ohjelmistostamme generoituu tietokannan mukaan, minkä takia tarkkaavaisuus on tärkeää ja tietokantamuutokset hyväksytetään aina tiimimme infrastruktuurin vastuuhenkilöllä.

Ylläpitotehtävien ja uusien toimintojen kehittäminen tarkoittaa ensisijaisesti olemassa olevan järjestelmän tutkimista, sillä pitkään kehitettävänä ollut monimutkainen järjestelmämme sisältää paljon olemassa olevia ratkaisuja, käytäntöjä, sekä rajoituksia. Yksinkertaisen ongelman ratkaisu on tällaisessa järjestelmässä harvoin yksinkertainen. Nopein ratkaisu saattaakin johtaa pitkälle kantaviin ongelmiin. Uusien toimintojen kehittäminen sisältää myös paljon kommunikointia kehitysryhmän ulkopuolelle. Uusista toiminnoista on suunnitteluvaiheessa tehty selkeät rajaukset ja toimiva prototyyppi, mutta järjestelmän monimutkaisuuden vuoksi toiminnallisuuden vaatimukset täsmentyvät useimmiten vasta kehitysvaiheessa. Tällaisessa ohjelmistokehityksessä on tärkeää omata loogista päättelykykyä, vastuuntuntoa, sekä ongelmanratkaisutaitoja.

Tehtäviini kuuluu keskeisesti myös tekemieni ratkaisujen testaaminen. Testaamiseen sisältyy yksikkötestien suunnittelu ja kirjoittaminen, sekä ratkaisun manuaalinen testaaminen käyttöliittymän, tai REST-kutsujen avulla. Ollessani varma ratkaisun toimivuudesta julkaisen sen arvioitavaksi ja saatan käydä ratkaisua läpi yhdessä laadunvarmistuksenkin kanssa.

Pehmeät taidot ovat yksi tärkeimmistä työkaluistani. Niihin lukeutuvat tiimityöskentely-, esiintymis- ja sosiaaliset taidot sekä ongelmanratkaisukyky (Indeed 3.11.2021). Suurin osa tiimistäni työskentelee Helsingin ulkopuolella ja kommunikointi hoidetaan lähes täysin etäpalaverein, puheluin sekä viestein. Kommunikointi tapahtuu tiimissämme sekä suomeksi että englanniksi. Joka-aamuisten varttipalavereiden, sprintin suunnittelun, sekä sprintin päättymiseen kuuluvien retrospektiivin ja katselmuksen lisäksi olen päivittäin tekemisissä laadunvarmistuksen kanssa uusista muutoksista, vaihdan ajatuksia ohjelman toiminnasta ja pyydän apua itseäni viisaammilta. Aktiivinen osallistuminen palavereihin edistää tiimin kollektiivista ymmärrystä projektin tilanteesta. Vaikka kehitystyöni onkin irrotettu miltei täysin asiakkaista, kuuluu sosiaalinen kanssakäyminen olennaisesti päivittäiseen työskentelyyni. Avoin, kriittinen ja empaattinen lähestymistapa on yksi tärkeimmistä ominaisuuksistasi tiimin jäsenenä. Sen lisäksi koen yleisestä projektityöskentelyn ymmärryksestä ja asiakaslähtöisestä ajattelusta olevan paljon hyötyä kehitystyössäni.

2.1.1 Oma osaaminen suhteessa työtehtäviin

Oma tekninen osaamiseni on mielestäni vielä aloittelijan tasolla ja pidän sitä kohtuullisena suhteutettuna siihen, että kokemusta alasta on ehtinyt kertymään vain muutamia vuosia. Ammattikorkeakoulun ja työharjoittelun kautta kertyneen osaamisen perusteella koen olevani uran alkutaipaleella oleva ohjelmistokehittäjä, joka on juuri herännyt tietoisuuteen kaikesta siitä, mitä ei vielä tiedä.

Vaikka kykenen jo ymmärtämään osia logiikastamme ja työskentelemään melko itsenäisesti, kysyn silti usein apua teknisiin ongelmiin ja tarkistutan tekemäni ratkaisut kokeneemmilla tiimiläisilläni. Koen tarvitsevani tiimiläisiäni enemmän aikaa uuden työtehtävän aloittamiseen. Teknisistä taidoista myös testaaminen on minulle vielä haastavaa, koska esimerkiksi yksikkötestaaminen jäi opinnoissani hyvin vähäiseksi.

Toisena erona kokeneempiin kehittäjiin olen huomannut arastelevani aloitteen tekemistä teknisissä ongelmissa. Tiimissämme rohkaistaan oma-aloitteiseen, projektia hyödyttävään ohjelmakehitykseen, mutten vielä koe omaavani taitoa nähdä tällaisia kehityskohteita tai itsevarmuutta tehdä tällaista aloitetta. Koen olevani vastuuntuntoinen ja näyttäväni sen työskentelyssäni, mutta tarvitsen ohjelmointiini vielä lisää kärsivällisyyttä sekä tarkkaavaisuutta. Opinnäytetyöni aikana on mielenkiintoista tarkastella, miten tekninen ymmärrykseni syventyy ja itsevarmuuteni kehittäjänä kasvaa.

Sosiaalisten ja pehmeiden taitojen saralla koen olevani jo taitava suoriutuja. Olen työskennellyt vuosia erilaisissa asiakaspalveluammateissa ja harjaantunut kommunikoinnin saralla. Esitysten valmistelu ja esiintyminen, sekä tälläkin hetkellä työssäni eniten käytetty viestintä, eli kirjoittaminen tuntuu minulle luonnolliselta. Eniten haasteita kommunikoinnin saralla koen kohdanneeni erittäin teknisistä asioista puhuttaessa, jolloin ammattisanastolla ja teknisten konseptien ymmärryksellä on iso painoarvo. Tähän voidaan lisätä myös se, että vaikka suurin osa työssä käyttämästäni teoriasta on englanniksi, häviää asioiden merkitys välillä käännöksessä. Tärkeätä on myös muistaa, että muun henkilöstön rooleista, työkokemuksesta ja persoonasta riippuen kommunikointi on erilaista.

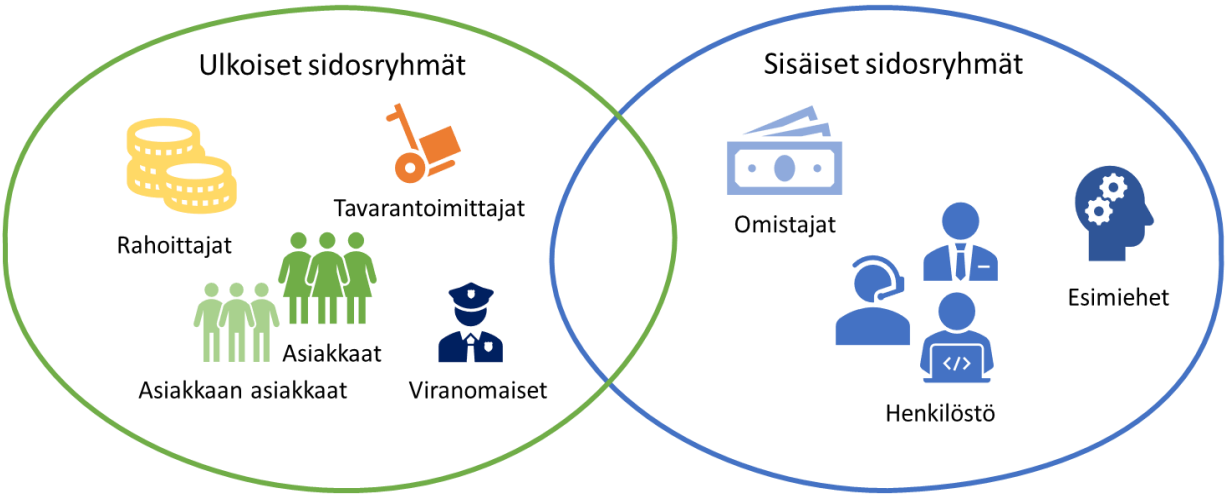
Pehmeiden taitojeni osalta tulen opinnäytetyössäni pohtimaan, miten teknisen ymmärrykseni syventyminen vaikuttaa kommunikaatiokykyihini. Itse olen työssäni usein kokenut saman keskustelun paljon hedelmällisemmäksi, kun käyn sen muutaman vuoden minua kokeneemman kollegan kanssa verrattuna siihen, kun käyn sen kymmeniä vuosia alalla olleen kanssa. Koska kommunikoinnin tärkeys korostuu usein kehittäjäryhmän ulkopuolelle käydyissä keskusteluissa, on kiinnostavaa nähdä, miten kommunikointini kehittyy kokemukseni karttuessa.

Työskentelen ensimmäistä kertaa toimistotyössä ja oikeassa ketterän kehityksen projektiryhmässä. Sain työn aloitukseen yritykseltä kattavat koulutukset, enkä ole kokenut tällaiseen työhön siirtymisessä isompia ongelmia. Etätyöskentelyn aikakaudella normaali toimistotyöskentely on kuitenkin jäänyt vähäiseksi ja uskon tämän vaikeuttavan yrityksen toiminnan hahmottamista kokonaisuutena. Seurantajakson aikana aion kiinnittää huomiota oman hahmotukseni muuttumiseen yrityksen osalta.

2.2 Sidosryhmät työpaikalla

Oman työni näkökulmasta sidosryhmät voidaan jakaa tyypillisesti sisäisiin sekä ulkoisiin sidosryhmiin kuvan 2 mukaisesti. Sisäisiä sidosryhmiä työni kannalta ovat yrityksen omistaja, esimiehet, omat tiimiläiseni, sekä yrityksen muu henkilöstö. Ylemmät esimieheni ohjaavat projektin etenemistä vastaten yritykselle asetettuihin tavoitteisiin ja tekevät kriittisiä päätöksiä projektin rajauksen, aikataulun sekä budjetin kannalta. Lähiesimieheni vaikuttaa vahvasti tekemiseeni luomalla strategioita ja asettamalla tavoitteita, sekä tarvittaessa muuttamalla prosessiemme rakennetta. Oman työni kannalta keskeisimpänä sisäisten sidosryhmien osana on projektiryhmäni, jonka päivittäinen työskentely heijastaa

suoraan omaan työskentelyyni (Concepta 2018). Asiakkaan käyttöön putoava julkaisu toimii tai on toimimatta projektiryhmäni yhteistyön tuloksena.



Kuva 3. Yrityksen sidosryhmät

Projektiryhmän ulkopuolella työskentelyyni vaikuttavat yrityksen muut osastot. Rekrytointi päättää tulevista tiimiläisistäni, henkilöstöhallinto pitää huolta tiimini yleisestä hyvinvoinnista, myynti ja asiakaspalvelu ovat kriittisessä asemassa tuotteen loppukäyttäjien osalta. Lisäksi sisäisenä sidosryhmänä voidaan pitää yrityksen ulkopuolisia, mutta tuotekehityksen osalta vaikuttavia ryhmiä, kuten työkalujen, kehitysalustojen ja ohjelmistokirjastojen ulkoisia tarjoajia (Concepta 2018).

Ulkoisia sidosryhmiä oman työni osalta ovat asiakkaamme, eli tuotteemme ensi- sekä toissijaiset käyttäjät ja heidän asiakkaansa. Asiakkaamme maksavat meille tuotteesta, jota saattavat käyttää asiakkaan oma henkilöstö, sekä mahdollisesti myös asiakkaan asiakkaat. Asiakkaamme käyttävät järjestelmäämme tehostaen sillä liiketoimintaansa käsittelemällä esimerkiksi omien asiakkaidensa prosesseja. Näin ollen järjestelmämme vaikutuksesta epäsuorasti hyötyvät tahot, kuten asiakkaidemme asiakkaat, ovat sen toissijaisia käyttäjiä (Concepta 2018). Yrityksemme toimintaan vaikuttavat toki myös muut ulkoiset sidosryhmät, kuten rahoittajat, IT-kaluston toimittajat sekä viranomaiset.

2.3 Vuorovaikutustaidot tiimissäni

Suurin osa tiimini sisäisestä viestinnästä hoidetaan etätyöskentelyn aikakaudella viestien ja etäpalavereiden välityksellä. Viestein käydyt keskustelut asettavat haasteita monimutkaisten teknisten keskustelujen käymiselle. Lisäksi olen kokenut tärkeäksi aktiivisen osallistumisen etäpalavereissa, joissa suurin osa tiimistäni ei välttämättä viesti

olevansa läsnä videon tai keskustelun välityksellä. Etätyöskentelyn takia tiimimme työskentely on hyvin itsenäistä, jolloin avun saaminen on normaalia hankalampaa ja saatan joutua odottamaan kollegoideni vastauksia normaalia pidempään. Iso osa tiimimme keskusteluista käydään lisäksi englanniksi. Vaikka koen englannin kieleni olevan suhteellisen hyvä, on englannin kielen ammattisanastossani vielä kehitettävää. Tämän kehittymisen koen kuitenkin tapahtuvan luonnostaan työkokemuksen karttuessa.

Seurantajakson alussa koen vielä olevani liian epävarma omasta osaamisestani osallistuakseni teknisiin keskusteluihin koko kehitystiimin palavereissa. Keskusteluihin sisältyy usein paljon teknisiä yksityiskohtia, joita en välttämättä ymmärrä. Pyydän usein myös kollegoiltani palautetta tekemistäni muokkauksista järjestelmäämme. Joudun useimmiten sitten pyytämään lisäselvitystä kollegoideni käyttämiin termeihin, kun ammattisanastoni on vielä kohtuullisen suppea. Asetunkin seurantajakson alussa usein kuuntelijan ja oppijan rooliin, enkä kyseenalaista kollegoideni antamia neuvoja tai ohjelmointiperiaatteita.

3 Päiväkirjaraportointi

Ketterään kehitykseen kuuluu se, että pidämme joka aamu 15 minuutin pituisen palaverin. Palaverissa kahdeksan henkilön lähitiimini käy läpi, mitä kukakin teki edellisenä päivänä ja mitä aikoo tehdä seuraavaksi. Työskentelemme 2 viikon sprinteissä, joka alkaa aina sprintin suunnittelulla ja päättyy koko tiimin retrospektiiviin, sekä lähitiimien kesken pidettäviin sprintin arviointeihin. Joka tiistai pidetään koko organisaation laajuinen Town Hall -palaveri, jonka isännöi jokainen yrityksen tiimi vuorollaan. (Schmidt 2015, 17.)

3.1 Seurantaviikko 1

Maanantai 15.2.2021

Aloitin viikkoni tarkistamalla viime viikolla tekemääni työtä. Sain muutama viikko sitten tehtäväkseni poistaa vanhentuneita tietokantasarakkeita, joista suurin osa on korvattu paremmin salatuilla sarakkeilla. Koska sarakkeille ei enää ollut ohjelmassa näennäistä käyttöä, en ensin uskonut tehtävän käyvän liian vaikeaksi. Totesin tehtävän kuitenkin olevan luultua arkaluontoisempi etsiessäni kyseisten sarakkeiden käyttöä ohjelman koodista ja sitten saadessani korjattavakseni useamman poistoista johtuvaa virheilmoitusta. Yhdeksi ongelmista osoittautui se, etten muistanut tarkistaa tietokannasta kyseisten sarakkeiden *tyhjennettävyyttä*. Kyseinen ongelma ilmeni, kun järjestelmä yritti tallentaa muuttuneita tietoja kantaan ilman yhtä poistetuista, mutta pakollisista tietokantasarakkeista.

Suurin ongelma tällaisessa tehtävässä on kuitenkin vanhan järjestelmän puolella tapahtuvat muutokset, joita on testien puuttuessa vaikea tulkita etukäteen täysin toimiviksi. OldTimer makaa lähes kokonaan funktionaalisen testauksen - jota kehittäjät joutuvat konfiguroimaan ja ajamaan paikallisesti - sekä käyttöliittymästä manuaalisen testauksen varassa, mikä johtaa jäykkään ja haavoittuvaan järjestelmään (Martin 2009, 1124), sekä vaikeasti tulkittaviin virheisiin. Koen tässä olleen minulle arvokas opetus kärsivällisyydestä ja tarkkuudesta.

Tarkistettuani viime viikkoisen työni jäljen, siirryin tämän päivän tavoitteeseen, Newbien uusimpaan toimintoon. Hintalistaus pitää sisällään paljon uusia tehtäviä, joiden vaatimuksia olemme rajanneet BDD:n mukaisesti suunnittelijoiden kanssa. Tänään kyseessä oli massa REST-kutsu, eli PATCH, jolla voidaan muokata useita *tietokokonaisuuksia* samaan aikaan (REST API Tutorials s.a.). PATCH mahdollistaa järjestelmässämme esimerkiksi sen, että kokonaisia asetuksia voidaan muokata ja

tallentaa yksi kerrallaan (Ola 13.11.2018). Massakutsu on tuttu muualta järjestelmästämme, joten aloitin tutkimalla vanhoja ratkaisuja tehtävästä ja noudatin yleistä käytäntöä tehtävässä. Tietokantayhteysongelmien takia en voinut manuaalisesti testata ratkaisuani, mutta sain kirjoitettua yksikkötestit muutoksilleni.

Tiistai 16.2.2021

Aiemmalla viikolla aloittamani sarakkeiden poisto tarkoittaa minulle myös kaikkien kehittäjiemme *muokkauspyyntöjen* vahtimista. Siinä missä järjestelmästämme generoituminen tietokannan mukaan nopeuttaa tietokannan muutoksien käsittelyä (Miller 7.8.2016), luo se tässä tapauksessa riskin poistetun sarakkeen epähuomiossa takaisin generoimisesta. Näin ollen jokainen päiväni alkaa ja päättyy kaikkien kehittäjien julkaisemien muokkauspyyntöjen läpikäymiseen.

Hintalistaan liittyen pidimme palaverin kehittäjien ja testaajien kesken. Toiminnon rajauksesta löytyi muutama aukko, joiden tiimoilta teimme tarkentavia päätöksiä. Niiden takia tajusin viime viikolla tekemäni tehtävän vaatimusten muuttuneen. Tämä oli BDD:n mukaisesta suunnittelusta huolimatta odotettavissa, sillä tämänkaltaista monimutkaista kokonaisuutta on mahdotonta täysin suunnitella etukäteen (Schmidt 2015, 10–11). Tehtävä liittyi tietokantakyselyn muuttamiseen käyttöliittymäkutsusta saatujen parametrien mukaan ja palaverissa käyttöliittymäohjelmoija huomasi tarvitsevansa lisää tietoja kutsun kautta. Palasin palaverin jälkeen tekemään tarvittavia muutoksia ja jatkoin hahmottelemaan uutta kyselyyn lisättävää kyselysuodatinta.

Tällaisten pienten tehtävien osalta koen olevani hidas aloittamaan uutta, koska järjestelmä ja siihen käytetyt teknologiat ovat minulle vielä suhteellisen uusia. Tämä johtaa siihen, että kynnys kirjoittaa ensimmäinen rivi koodia on korkea. Lisäksi manuaaliseen testaamiseen käytetyn tietokantayhteyden ongelmien takia en ollut kyennyt testaamaan tekemiäni hintalistan muutoksia kunnolla. Päätin siirtyä seuraavaksi päiväksi toimistolle välttääkseni etäyhteydestä johtuvat tietokantaongelmat ja pystyäkseni keskittymään työhöni paremmin.

Keskiviikko 17.2.2021

Aloitin päiväni tarkistamalla muokkauspyynnöt Githubista. Aamupalaverin jälkeen siirryin takaisin eilen aloittamaani tehtävään käyttöliittymässä näkyvän tuotelistauksen suodatukseen liittyen. REST:in kutsuma metodi ottaa vastaan paljon vapaaehtoisia parametreja vaikeuttaen kutsun luettavuutta sen liikkuessa syvemmälle logiikkaan (Martin

2009, 40–43). Ratkaisuni on kuitenkin yhdenmukainen muiden samankaltaisten suodatinmetodien kanssa ja näin ollen siirryin ratkaisuni manuaaliseen testaamiseen.

Vaikka laadunvarmistuksemme testaa Newbieen tehdyt muutokset huolellisesti, pyrin testaamaan kaiken tekemäni manuaalisesti. Vaikka Martinin (2011, 114–115) mukaan laadunvarmistuksen ei pitäisi löytää mitään hyvän ohjelmoijan muokkauksista, on näin suuren järjestelmän kanssa välillä vaikea saada käsiinsä testaukseen sopiva käyttäjä oikeanlaisilla asetuksilla. Tässä tapauksessa pyysin apua testaajaltamme ja yhdessä muistelimme, miten esimerkiksi *lueteltuja* tietotyyppejä lähetetään parametreina REST-kutsussa.

Totesin päivän lopuksi ratkaisuni olevan yksikkötestausta vaille. Tiedän testilähtöisen ohjelmoinnin olevan tiimissämme suosittu lähtökohta, mutta koen silti vielä ymmärtäväni liian vähän järjestelmästä ja yksikkötestaamisesta kyetäkseni hahmottelemaan testit ennen itse ratkaisua. Yksikkötestaaminen jäi opinnoissani hyvin vähälle, joten koen kehittyneeni siinä hurjasti lyhyen urani aikana. Lopulta siirryin suunnittelemaan uusia testejä usealla eri parametriyhdistelmällä ajettuna NUnit-testikirjaston tarjoaman [TestCase()]-attribuutin avulla (NUnit 2018a).

Torstai 18.2.2021

Asetin päivän tavoitteeksi saada kaksi hintalistan suodatukseen liittyvää tehtävää valmiiksi kollegoilta saamieni parannusehdotuksien mukaisesti. Koska suodattimena käytettyjä parametreja otetaan vastaan listoina - joita sitten käsitellään ja siirretään toiseen listaan - voidaan suorituskykyä optimoida alustamalla toisena luotu lista suoraan edeltäjänsä kokoiseksi. Näin järjestelmällä on suoraan oikean kokoinen lista sen sijaan, että se joutuisi joka kerta tilan loppuessa luomaan oikean kokoisen listan ja siirtämään tiedot uudestaan (Laaksonen 2020, 37–39; Microsoft 2021b). Parannusehdotuksien lomassa katsoimme yhdessä laadunvarmistuksen kanssa, että ratkaisu toimii uusien vaatimuksien mukaisesti.

Lounaan jälkeen osallistuin myös palaveriin, jonka tarkoituksena oli aloittaa keskustelu paremman kommunikoinnin saavuttamiseksi BAUX-tiimin ja kehittäjien välillä. BAUX-tiimi on esittänyt toiveen saada nähdä heidän tekemänsä uusien suunnitelmien toteutuksen ennen niiden julkaisua. Kehittäjät taas kokevat, että tulevia suunnitelmia pitäisi käydä läpi ennen niiden lähettämistä kehitykseen, jotta mahdolliset muutokset alkuperäisiin vaatimuksiin ja toiminnallisuuksiin voitaisiin löytää jo aikaisessa vaiheessa. Oman kokemukseni pohjalta nostin esiin ymmärtäväni suunnittelijoiden tarpeen saada nähdä työnsä jäljen toimivana kokonaisuutena ja kannatin etukäteen käytyä yhteistä

vaatimusmäärittelyä. Suunnitelmien arviointi on hyvin yleinen CI/CD-kehityksessä käytetty työkalu, eikä pelkästään vähennä tuotteen jälkiviisasta korjailua, vaan kehittää myös tiimin keskinäistä ymmärrystä yhteisestä strategiasta (Radom 21.3.2019). Palaverissa vaihdettiin ajatuksia ja ehdotuksia, joiden pohjalta nimettiin vastuuhenkilöt viemään toivottuja muutoksia eteenpäin.

Vaikka aloitin päiväni kritiikin siivittämänä, tällainen osaamisen syventyminen työn ohessa tapahtuvassa oppimisessa motivoi minua ja piristi päivääni. Sain myös päiväni tavoitteet täytettyä julkaisemalla molemmat hintalistaan liittyvät ratkaisuni arvioitavaksi ja testattavaksi.

Perjantai 19.2.2021

Päiväni alkoi kollegaltani saamallani palautteella toista suodatinparametria koskien. Ehdotus liittyi ratkaisussani piilevään SQL-injektioksi kutsuttuun riskiin. SQL-injektiosta puhutaan esimerkiksi tilanteessa, kun käyttäjän syöttämällä parametrilla on mahdollisuus kulkeutua sellaisenaan järjestelmän kautta tietokantaan osana SQL-kyselyä. Tällainen parametri voidaan sitten lähettää eteenpäin sellaisessa muodossa, että se käsittelee tietokantaa odottamattomalla tavalla. Vaikka SQL-injektio on yksi yleisimmistä tahallisista tietomurtotyyleistä, voi se aiheutua myös tietämättömästä tai laiskasta ohjelmoinnista. (W3Schools 2021a.)

Ratkaisuni ei tarjonnut yhtä yksiselitteistä riskiä, mutta varsinkin pian julkaistavan julkisen ohjelmointirajapintamme myötä järjestelmän on oltava aukoton. Tällä kertaa aukko paikattiin bittimaskiksi kutsutulla rakenteella, jolla voidaan pakottaa kyseinen parametri ennalta määriteltyyn muotoon (Hilyard & Teilhet 2004). Toinen hyöty kyseisenlaisesta ratkaisusta on sen luettavuus. Kun ohjelmoija näkee kyselyä muodostavan metodin, on hänen helppo tulkita parametrin voivan sisältää vain bittimaskin mukaista tietoa. Näin ollen ratkaisu ei pilaannu ajan kanssa, vaan sen luettavuuden ansiosta sen tarkoitus säilyy yksiselitteisenä (Martin 2009, 175).

Tämän lisäksi kävimme läpi testaajan kanssa ratkaisuni vaatimia testejä ja tarkistimme ratkaisuani yhdessä. Tarkastin, että ratkaisuuni liittyvien luokkien yksikkötestit kattoivat uudet parametrit ja julkaisin ratkaisuni arvioitavaksi. Kokonaisuutena en kokenut saaneeni päivän aikana hirveästi aikaan, vaikka huomasin toisaalta oppineeni paljon uutta ja saaneeni aikaan erittäin kelvollisen ratkaisun.

Viikkoanalyysi

Teknisinä teemoina tältä viikolta nousivat vanhentuneiden sarakkeiden poiston vaatima tarkkuus, sekä käyttäjän syöttämien parametrien käyttö tietokantakyselyssä. Projektityöskentelyn saralla päällimmäisenä aiheena oli *muotoilun* arvioinnin implementointi osaksi tiimin sprinttejä.

Vanhojen sarakkeiden poiston ongelmat johtuvat suurelta osin OldTimerin vähäisestä testaamisesta. Siinä missä Newbien testit ovat automatisoitu ja kattavat miltei koko järjestelmän, OldTimerin vähäiset testit pitää konfiguroida ja ajaa manuaalisesti. Testit eivät myöskään kata läheskään koko järjestelmää, jolloin tehtyjä muutoksia on vaikea todeta toimivaksi. Koska muutoksia on vaikea todeta toimivaksi, joudutaan pohtimaan ovatko muutokset välttämättömiä vai voisivatko ne odottaa vanhan järjestelmän alasajoon asti. Kyseessä olevat sarakkeet on tietokannassa korvattu salatuilla kolumneilla, jolloin tietoturvan kannalta voidaan nämä muutokset tulkita välttämättömiksi. (Martin 2009, 4–6; 124).

Tehtävässä korostui ohjelmoijan työhön liittyvä vastuu (Martin 2011, 9–11). Järjestelmäpäivityksien jatkuva integraatio pääversioon tarkoittaa, että tehdyt muutokset menevät asiakkaiden käyttöön nopealla tahdilla (Fowler 1.5.2006). Vaikka jatkuva integraatio tarkoittaa, että virheisiin voidaan reagoida nopeasti, tarkoittaa se myös isompaa vastuuta järjestelmän jatkuvasta toimivuudesta. Tässä tapauksessa koen, että vaikka osa sarakkeiden käytöstä piiloutui järjestelmään odottamattoman hyvin, olisin voinut suoriutua tehtävästä tarkkaavaisemmin. Opin toisaalta vastuuta, sekä uudenlaisia tapoja etsiä tietoa järjestelmästä, kun jouduin kaivamaan virheitä aiheuttavia sarakkeiden jäännöksiä useaan otteeseen.

Tekninen ymmärrykseni syventyi myös suodatinparametrien käsittelyyn liittyvissä vaatimuksissa. Käyttäjän syöttämä sisältö on tärkeä tarkistaa ja hallita tarkasti sen kulkiessa järjestelmässä. Martin Thoman julkaisema artikkeli ”SQL Injections” nostaa esiin injektioriskin vakavuuden. Vielä vuonna 2020 erittäin suositun korkealaatuisia kuvia ja grafiikoita tarjoavan Freepik-sivuston 8,3 miljoonan käyttäjän tiedot saatiin urkittua SQL-injektion avulla. (Pauli 2013; Thoman 12.10.2020.)

Tärkeä huomio aiheeseen liittyen oli myös se, ettei SQL-injektiossa ole aina kyse tahallisesta toiminnasta. Epähuomion ja tietämättömyyden aiheuttama kyselyvääristymä

on yhtä vaarallinen tapahtuma kuin tahallinenkin ja helppo välttää parantamalla järjestelmän luettavuutta. Käyttäjän syöttämään parametriin puututaan kyseisessä ratkaisussani jo heti ylemmällä tasolla ohjelmaa, joten ongelma oli alemman tason logiikassa sijaitseva SQL-kyselyä muodostava metodi. Ongelman olisi voinut nopeasti ja laiskasti ratkaista jättämällä kommentin parametriin liittyvistä vaatimuksista. Kommenteilla on kuitenkin taipumus vanhentua, eikä sillä voida taata metodia käyttävän ohjelmoijan lukevan tai ymmärtävän ongelmaa. Näin ollen pienellä investoinnilla aikaa ja vaivaa uskon bittimaskin rakentamisen olleen pätevä ratkaisu. (Martin 2009, 53–55; Thoman 12.10.2020.)

Projektityöskentelyn puolesta pääsin tutustumaan tarkemmin muotoilun arviointeihin ohjelmistokehityksen työkaluina. Koen usein uutta toimintoa aloittaessani vaikeaksi hahmottaa, mitä käyttöliittymässä ja käyttäjätarinassa tapahtuu. Siksi ennen ohjelmoinnin aloittamista tehtävä muotoilun arviointi olisi mielestäni erittäin tarpeellinen vaihe. Etukäteen tehdyllä arvioinnilla voitaisi merkittävästi vähentää tuotantovaiheessa huomattuja virheitä (Bliley Technologies 4.2.2020). Sen lisäksi muotoiluun liittyvät kysymykset voitaisiin esittää heti ja käyttäjätarinoihin liittyvä keskustelu käydä suullisesti prototyypin avulla havainnollistaen. Tällöin myös prototyypistä voidaan saada irti kaikki sen potentiaali ja korostaa myös muotoiluun liittyvien päätöksien tärkeyttä (Casacuberta s.a.).

Aurélie Radom kirjoittaa artikkelissaan ”The ‘Design Review’ Between Designers and Developers” (21.3.2019) muotoilun arvioinnin olevan ajankohtaisimmillaan kehityssyklin lopussa. Ramonin mukaan arvioinnin tehtävä on varmistaa, että kehitetty tuote mukailee sen luovaa visiota ja suunniteltua käyttökokemusta. Pohjaten omaan kokemukseensa Radom korostaa UX-tiimin kanssa käytävän keskustelun edesauttavan myös kehitystiimin keskinäistä kommunikointia.

Tiimimme pitää joka sprintin lopuksi usean tunnin palaverin, jossa esitellään uusimpia kehityksessä valmistuneita toiminnallisuuksia. Vaikka palaverissa harvoin jäädään koko tiimin voimin keskustelemaan yksittäisistä demoista, vastaa se mielestäni ainakin osittain kehityksen jälkeistä muotoiluarviointia. Siitä huolimatta koen sekä etukäteen että jälkikäteen pidetyissä muotoiluarvioinneissa olevan paljon potentiaalia. Muotoiluarvioinnissa käyttökokemuksen ja muotoiluun sisältyviin päätöksiin, sekä sen implementoinnin rajoitteisiin päästäisiin käsiksi yhdessä. Tämän vuoksi uskon BAUX-tiimin ja kehityksen välisten palaverien kohentavan myös asiantuntijoiden välistä ymmärrystä toistensa työnkuvia kohtaan.

3.2 Seurantaviikko 2

Maanantai 22.2.2021

Viikkoni piti alkaa viime viikolta jääneellä vanhentuneiden sarakkeiden poistolla tietokannasta. Ajatukseni oli aloittaa sarakkeiden poisto kannasta jo tänään, mutta koska tehtävä tulee vaikuttamaan molempiin järjestelmiimme, varmistin asian vielä tietokannasta ja julkaisuista vastaavalta kollegaltani. Vaikka sarakkeiden poiston ei pitäisi vaikuttaa mihinkään, voidaan muutokset todeta toimiviksi käytännössä vain odottamalla mahdollisia virheilmoituksia. Koska suurin osa järjestelmämme käytöstä tapahtuu kuukauden vaihteessa, oletettavasti tehtävään liittyvät virheilmoitukset tulevat esiin viimeistään silloin. Sovimme näin ollen, että odotan ensi kuun puoliväliin ennen sarakkeiden poistoa kannasta, jolloin voidaan olettaa järjestelmän toimivan ainakin suurimmilta osin odotetusti.

Olen ollut kiinnostunut eräästä atomisuuteen liittyvästä tehtävästä jo hetken aikaa, mutta kokenut sen liian vaikeaksi pystyäkseni tekemään sitä itsenäisesti. Muutama viikko sitten tutustuin kuitenkin atomisuuteen konseptina toiseen tehtävään liittyen ja koin tänään olevani valmis tutustumaan tähän paremmin. Tehtävä liittyi erään MVC-mallimme mukaisen *ohjaimen* vastaanottamiin POST- ja PATCH-kutsuihin, sekä liian isoksi kasvaneen ohjaimen muokkaamiseen. Riippuen vastaanotetusta kutsusta ja sen mukana tulleesta tiedosta, tulee järjestelmän validoida ja tallentaa muutokset yhdenmukaisesti. Ohjain ei kuitenkaan tällä hetkellä takaa tiedonsiirron eheyttä näiden kutsujen osalta. Ehdin päivän aikana vasta tutustumaan tehtävääni, mutta sain sen lisäksi aikaan pieniä muutoksia hintalistauksessa. (REST API Tutorial s.a.)

Tiistai 23.2.2021

Teknisellä puolella aloitin edellisenä päivänä tutustumani ohjaimen uudelleen rakentamisella. Päätin käyttää esimerkkinä toista suurta ohjainta, jossa osa sen käytöksestä on jaettu Strategic-suunnittelumallia muistuttavasti omaan itsenäiseen luokkaansa. Strategic-mallissa alati muuttuva logiikka on irrotettu omaksi kokonaisuudekseen, jota ohjain voi sitten käyttää joustavasti (Bates ym. 2005, 13–24). Koska osa projekteistamme on yhä Visual Basicilla kirjoitettu, jouduin myös kääntämään osan toiminnallisuudesta C#:ksi. En ehtinyt pureutumaan tehtävään sen enempää, sillä muutamaa tuntia lukuun ottamatta päiväni koostui sprintin päättämiseen liittyvistä palavereista.

Joka toinen tiistai päätämme sprinttimme *sprinttikatselmuksella*, lähitiimien retrospektiivillä, sekä yhteisellä retrospektiivillä. Sprintin katselmuksessa jokainen asiantuntijaryhmä esittelee sprintin aikana tuottamansa tulokset (Schmidt 2015, 17). Oman työpanokseni osalta UI-kehittäjämme esitteli hintalistauksen toiminallisuutta siltä osin, kuin se on käyttöliittymänsä puolesta valmis. Koska katselmuksen tarkoitus on näyttää tiimin edistyminen sidosryhmille ja työskentelen paljolti silmän ulottumattomiin jäävien ratkaisujen eteen, en ole paljoa äänessä näissä palavereissa. Vaikka katselmuksessa kuuluu käydä keskustelua tehdyistä muutoksista ja suunnitella tulevaa, venyi palaveri tällä kertaa teknisen keskustelun takia odotettua pitemmäksi. Retrospektiivissä keskustelimme Scrum-käytännöistämme ja sen toimivuudesta, sekä teknisistä aiheista. Tämäkin palaveri venyi odottamattoman pitkäksi, enkä ole varma, oliko keskustelu hedelmällistä.

Keskiviikko 24.2.2021

Tänään pääsin takaisin ohjaimen uudelleenrakentamisen pariin. Näytin tekemiäni muutoksia kollegalleni kuullakseni hänen mielipiteensä muutoksista. Tarkistutan usein tekemääni kollegoillani jo ennen muokkauspyynnon julkaisemista, koska uskon heidän ymmärtävän järjestelmään liittyvistä korkeamman tason konsepteista enemmän kuin minä. Kollegalla hyväksyttämisen jälkeen jatkoin käsittelemään IoC-moduulia, jonka avulla järjestelmässämme rekisteröidään kaikki sen tarvitsemat resurssit dynaamisesti. Rekisteröimme IoC-moduulin avulla tarvitsemamme luokat Singleton-luokiksi, koska emme järjestelmässä tarvitse useita luokkia yhtä instanssia enempää (Bates ym. 2004, 172).

Tehtävässäni ongelma ei ole pelkästään ohjaimen atomisuus, vaan sen hallinnoima tieto. Ohjaimella on paljon enemmän vastuuta kuin ohjaimen oletetaan sisältävän ja vaatii näin ollen uudelleen rakentamista hyvän ohjelmointikäytännön vuoksi (Martin 2009, 137–140). Luettavasta ohjaimesta on myös helpompi ymmärtää, minkälaisia operaatioita se kutsuu. Löysin ohjaimesta muutamia kutsuja, jotka oli loogista siirtää uuteen erilliseen ohjaimeen. Tämän lisäksi päivittelin päivän aikana jo julkaisemiani muutospyyntöjä ja keskustelin laadunvarmistuksen kanssa niiden testaamisesta.

Torstai 25.2.2021

Aamun Scrum-palaverin lisäksi käytin työpäiväni lähes poikkeuksetta atomisuustehtäväni parissa. Kuten aiemmin mainittu, ohjaimen PATCH- ja POST-kutsujen logiikka oli hyvin monimutkainen ja hankalasti luettava. Ohjaimen vastuulla oli monia järjestelmässä

yleensä muualle siirrettyjä tehtäviä, kuten sisään tulevien ja ulos lähtevien tietokokonaisuuksien validointi ja siirtäminen. Ohjaimen vastuisiin kuuluu yksinkertaisuudessaan REST-kutsujen reititys ja oikeanlaisen vastauksen antaminen, eikä varsinainen liiketoiminnallinen logiikka (Addie & Smith 12.5.2019). Tästä syystä siirsin tietokokonaisuuksien validoinnin ja monimutkaisemman käsittelyn niille tarkoitettuihin luokkiin. Silloin kyseiset tehtävät löytyvät mielestäni loogisemmasta paikasta, jolloin siirto hyödyttää myös tiedon hallittavuuden takia.

Ohjaimen moninaisten vastuiden siirtäminen vei odotettua enemmän aikaa ja koin loppupäivästä olevani väsynyt tähän pikkutarkkaan tekemiseen. Tiedän tehtävässä olevan vielä tekemistä yksikkötestien ja itse atomisuuden varmistamiseksi. Olin kuitenkin tyytyväinen saamaani tulokseen ja sovin laadunvarmistuksen kanssa seuraavaksi päiväksi yhteisen testaushetken.

Perjantai 26.2.2021

Tänään suurin osa päivästäni meni atomisuustehtävän sijasta laadunvarmistuksen kautta tulleisiin muutoksiin hintalistauksessa. Yhdestä hintalistasta voidaan tehdä useita versioita, joille halutaan sitten luoda uusi oletustyötuntihinta. Olin jo aiemmin tällä viikolla lisännyt oletustyöhinnan luonnin uuden hintalistan lisäämisen yhteydessä. Kun hintalistasta voidaan tehdä uusia versioita, huomattiin oletustyöhinnan uupuvan uusista versioista. Etsimme yhdessä kokeneemman kollegani kanssa oletushinnan lisäämiselle sopivinta sijaintia, sillä vaikka itse muutos ei ollut teknisesti haastava, hankaloitti hintalistauksen monimutkainen logiikka tehtävääni. Muutoksen jälkeen siirtyessäni testaamisvaiheeseen huomasin käyttämiltäni luokilta puuttuneen yksikkötestit, joten pääsin luomaan kokonaisia testiluokkia itsekseni.

Oletushintalisäyksen jälkeen siirryin suodatinparametrien tarkentamiseen. Emme olleet huomanneet, että käyttöliittymän tarvitsema suodatettujen tuotteiden laskuri ei toiminut oikein uusilla muutoksilla. Osasin aavistaa, että ongelma johtui jo aiemmin suodatusta tehdessä ihmettelemästäni hankalasta SQL-kyselystä. Kysely on monimutkainen ja yhdistää useita toisistaan riippuvia tauluja, minkä takia suodatuksen lisääminen piti alun perinkin rakentaa manuaalisesti tietokannassa testaamalla. Kyseessä on luultavimmin vanhasta järjestelmästä siirretty kysely, johon ei sittemmin ole enää sen riippuvaisuuksien takia haluttu koskea. Vaikka tiedän, että kyselyn uudelleen järjestely olisi hyvän ohjelmointikäytännön mukaista (Martin 2009, 14), en uskaltanut lähteä purkamaan kyselyä. Pyrin silti pitämään omat lisäykseni luettavina ja ehdin päivänä päätteeksi

lähettää ratkaisuni laadunvarmistukseen testattavaksi. En päässyt testaamaan ohjaimeen liittyneitä muutoksiani laadunvarmistuksen kanssa, mutta olin tyytyväinen päivän aikaansaannoksiin.

Viikkoanalyysi

Tämän viikon teemoiksi nousivat teknisinä aiheina REST-kutsujen tiedon eheyden varmistaminen, sekä toisistaan riippuvien operaatioiden atomisuus. Projektityöskentelyn näkökulmasta tutustuin tällä viikolla sprintin päättämiseen liittyvien palavereiden hallintaan.

REST-kutsut eivät ole idempotentteja, eli takaa samaa lopputulosta samalla useita kertoja lähetetyllä kutsulla (Halonen 8.2.2019; Taulukko 1). Näin ollen niiden tuoma datan yhtenäisyys pitää taata järjestelmässä liiketoiminnallisella tasolla. Tulen käyttämään ohjaimen operaatioiden atomisuuden varmistamiseen .NET-ympäristön tarjoamaa transaktiorajausta, joka on yleisesti käytössä järjestelmässämme.

Taulukko 1. REST-kutsut ja niiden vastineet CRUD-funktioissa, sekä niiden käyttötarkoitukset (REST API Tutorials s.a.)

REST-kutsu	***CRUD***	***Käyttötarkoitus***
POST	CREATE	Luo uusi resurssi, yleensä toisesta resurssista riippuva resurssi. Tarvitsee muuta logiikkaa taatakseen eheän tiedonsiirron, ei idempotentti.
GET	READ	Lue resurssi, palauttaa haettu tai haetut resurssit xml- tai json-muodossa. Ei muuta tietoa, idempotentti.
PUT	UPDATE	Luo uusi resurssi tai muokkaa kerralla koko olemassa olevaa resurssia. Oikein käytettynä idempotentti.
PATCH	UPDATE	Muokkaa olemassa olevaa resurssia tai sen osia. Vaatii validoinnin alemmassa logiikassa, ei idempotentti.
DELETE	DELETE	Poista resurssi. Resurssin eheyden näkökulmasta idempotentti.

Atominen logiikka tietojenkäsittelyssä voidaan rinnastaa transaktioon, jonka aikana voidaan suorittaa useita toisistaan riippuvia, mutta irrallisia operaatiota (Mehmood 9.2.2019). Esimerkiksi kun PATCH-kutsussa ensin tallennetaan tietokantaan uusi kalenteriaktiviteetti, jolle sitten yksi kerrallaan lisätään osallistujia, liittyvät aktiviteetin ja siihen liittyvien osallistujien tallentamiset yhteen. Mikäli jossain kohtaa aktiviteetin ja

viimeisen osallistujan tallentamisen välissä järjestelmässä sattuu virhe, voi osa tiedoista olla jo kirjoitettu talteen ja osa jäädä kirjoittamatta. Mikäli jokin transaktion sisällä tapahtuvista operaatioista epäonnistuu, palautetaan järjestelmä takaisin tilaan, jossa yhtäkään operaatiota ei ole vielä suoritettu (Microsoft 2021c).

Ohjaimen uudelleen rakentaminen helpotti transaktion sijoittamista koodiin. Osa ohjaimen operaatioista oli sijoitettu ohjaintasolla transaktion sisään, joka riitelee ohjaimen yleisten vastuiden kanssa (Addie & Smith 12.5.2019). Lisäksi osa transaktion sisällä olevista operaatioista ei vaikuttanut toisiinsa ja ne voitiin näin ollen siirtää pois transaktion sisältä. Koska transaktio on eristetty toiminto, joka voi luoda järjestelmässä lukkoja ja viiveitä, on transaktion rajaus tärkeä pitää mahdollisimman pienenä (Habib 6.1.2014). Uudelleen rakentaminen mahdollisti turhien toimintojen poistamisen transaktiosta, sekä transaktion siirtämisen järjestelmän liiketoiminnalliseen logiikkaan.

Käydessäni keskustelua kyseisestä tehtävästä kollegoideni kanssa esiin nousi kysymys siitä, oliko kyseinen muutos tarpeellinen tiimin tavoitteiden kannalta. Tiimin tärkeimpiä lyhyen tähtäimen tavoitteita ovat vanhan järjestelmän alas ajo ja uuden järjestelmän toimintojen valmistuminen. Koska kyseessä oleva ohjain on jo käytössä ja todettu toimivaksi, ei tehtävän tekeminen välttämättä edistänyt tiimille asetettuja tavoitteita ja ollut tärkeä sprintin DoD-määritelmän mukaan (Madan 16.12.2019). Toisaalta koska ohjaimen toiminta ei ole ollut täysin atomista tähän mennessä, on siitä johtuvia ongelmia noussut jo tuotannossa. Sen lisäksi muutokseni helpottavat järjestelmän testausta sekä lisää niiden kattavuutta. Näin ollen uskon tehtävän olleen ajankohtainen ja opettavainen.

Projektityöskentelyn näkökulmasta pohdin tällä viikolla paljon sprintin päätöspalavereitamme ja niissä käytävän keskustelun rajausta. Useimmiten keskustelua syntyy hyvin teknisistä asioista, josta hyötyvät vain tiimin kehittäjät. Kun puolet tiimistä ei ymmärrä ja näin ollen hyödy tällaisesta keskustelusta, jää se silloin katselmuksen tavoitteiden ulkopuolelle (Schwaber & Sutherland 2020, 9). Sen sijaan yhteistä aikaa voitaisiin käyttää esimerkiksi tiimin yhteisten tavoitteiden käsittelemiseen ja niistä keskustelemiseen. Otin tähän kantaa lähitiimini retrospektiivissä.

Tiimissämme retrospektiivit hoidetaan pienemmissä ryhmissä ja niiden tulokset esitellään seuraavana päivänä koko tiimin retrospektiivissä. Olemme käyttäneet Sailboat-kehystä ryhmämme retrospektiiviin, jonka avulla otamme esiin sprintin aikana tulleita. Kuten retrospektiivin luonteeseen tyypillisesti kuuluu (Schwaber & Sutherland 2020, 10), nousee ryhmässämme keskustelun aiheita liittyen projektityöskentelyyn, tiimiin, kommunikointiin ja

prosesseihin, sekä teknisiin haasteisiin. Myös retrospektiivit koostuvat usein hyvin teknisistä keskusteluista ja uskon sen johtuvan siitä, että lähitiimini koostuu lähes poikkeuksetta ohjelmistokehittäjistä. Pyrin itse usein ottamaan esiin muita teemoja, kuten projekti- ja ryhmätyöskentelyn tai kommunikoinnin.

Mielestäni tiimissäni voitaisiin ottaa käyttöön Scrum-palavereiden aikaraja, eli time boxing. Aikarajaukset ovat yleinen ketterässä ohjelmistokehityksessä käytetty työkalu, jonka avulla tiimin keskittyminen käsillä olevaan tehtävään pysyy paremmin hallinnassa (ScrumInc s.a.). Aikataulutettu keskustelu pysyy mielestäni paremmin aiheessa ja vähentää turhaa keskustelua asettamalla painetta keskustelun laatuun.

3.3 Seurantaviikko 3

Maanantai 1.3.2021

Aloitin viikkoni tuttuun tapaan tarkistamalla julkaistut muokkauspyynnöt, jonka jälkeen siirryin suoraan ohjaimen atomisuuteen liittyneeseen tehtävään. Lisäsin ohjaimesta liiketoiminta logiikkaan siirrettyjen operaatioiden ympärille transaktion rajaten sen sisälle vain tietokantaan liittyvän tiedonkäsittelyn. Sen jälkeen tutkin muutoksia koskevien luokkien yksikkötestejä ja totesin niiden vaativan uudelleenkirjoittamista sekä lisäyksiä tekemieni muutoksien takia.

En koe vielä ymmärtäväni tarpeeksi järjestelmästämme kirjoittaakseni ratkaisujani puhtaasti testilähtöisellä kehitysmallilla. Pyrin kuitenkin yksikkötestien kirjoittamisessa irrallisiin, riippumattomiin testeihin, jotka tietäisivät mahdollisimman vähän. Riippumattomat testit ylläpitävät luettavaa järjestelmää, kun muutoksia tehdessä voidaan luottaa siihen, että muutoksien jälkeen ajetut testit paljastavat muutoksista johtuvat ristiriidat ohjelman toimivuudessa. Käytin loppupäivän kirjoittaessani ohjaimelta uupuvia yksikkötestejä ja koin loppupäivästä saaneeni suhteellisen paljon aikaan. (Martin 2009, 124–132.)

Tiistai 2.3.2021

Suurin osa päivästäni meni atomisuustehtävän yksikkötestaamiseen. Vaikka uskon kehittyneeni yksikkötestaajana paljonkin sitten harjoitteluni, olen huomannut kirjoittavani vielä toistaiseksi implementaatioon nojaavia testejä. Implementaatioon nojaavalla testillä tarkoitetaan sellaista testiä, jossa testissä otetaan huomioon implementaatiosta tiukasti riippuvia asioita ja joka näin aiheuttaa lisätyötä aina implementaatiota muutettaessa. Mikäli esimerkiksi testi nojaa vahvasti tiettyihin metodikutsuihin, joiden paikkaa sitten siirretään, joudutaan järjestelmää muokatessa palaamaan lopuksi muuttamaan myös testi. (Dodds 2020.)

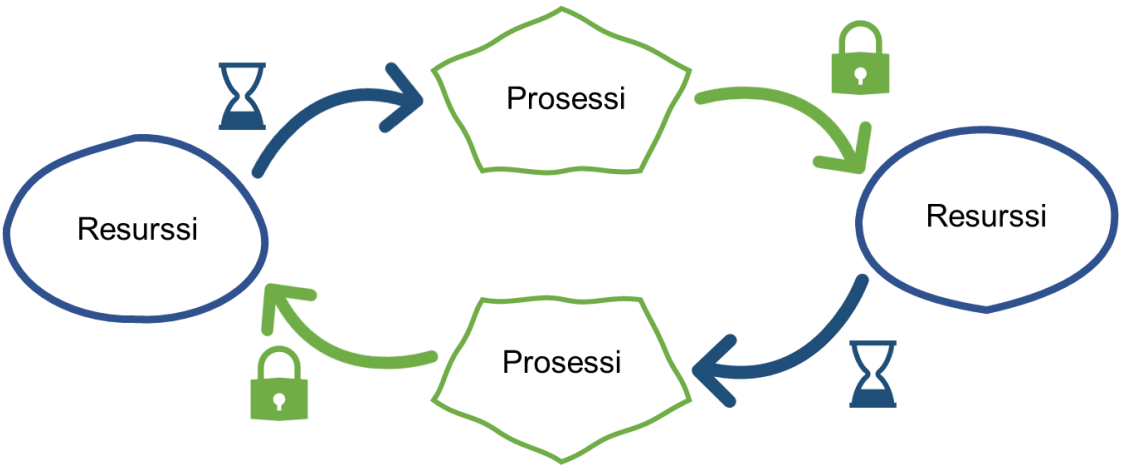
Ilmoittauduin aiemmin tänä vuonna mentoroitavaksi yrityksen sisäiseen juniortason ohjelmistokehittäjille tarkoitettuun ohjelmaan. Mentoriohjelmaan käytetään 5 työtuntia viikossa ja olen yksi kolmesta mentoroitavasta. Tapasimme tänään ensimmäistä kertaa ja kävimme loppupäivästä läpi, minkälainen mentoriohjelmamme tulee olemaan. Sovimme myös tapaamisen ensi viikolle, johon mennessä meidän tulee aloittaa Martin (2009) Clean Code kirjan lukeminen, sekä pohtia motivaatioitamme ohjelmoijana kehittymiseen. Ehdin

palaverin jälkeen käydä vielä läpi tapaamisessa saamani materiaalit ja todeta atomisuustehtäväni olevan valmis manuaaliseen testaamiseen.

Keskiviikko 3.3.2021

Käytin aamupäivän lisäten atomisuusratkaisustani vielä uupuvia yksikkötestejä, sekä manuaalisesti testaten uudelleen rakennettua ohjainta. Ohjain reitittää kaikki kalenterimerkintöihin liittyvät muutokset, jonka takia koin helpoimmaksi testaustavaksi laittaa jokaiselle *reitille* omat *pysäytyspisteet* ja sitten tarkkailla datan kulkemista ajaen järjestelmää *virheenkäsittelytilassa*. Nyt kun ohjaimen yksikkötesti kattavat enemmän kuin pelkästään tekemäni muutokset, vahvistavat ne ohjaimen ja siihen liittyvän logiikan luotettavuutta (Martin 2011, 80–81). Viimeisenä silauksena julkaisin muokkauspyyntöni arvioitavaksi Github-alustalle, sillä julkaisu laukaisee kaikki järjestelmälle kirjoitetut yksikkötestit. Näin minun on helppo löytää paikat, joissa muutokseni saattoivat tehdä odottamattomia muutoksia.

Tutustuin loppupäivästä infrastruktuurista vastaavan asiantuntijamme ilmoittamaan tikettiin kannassa syntyvästä deadlock- eli *umpikujatilanteesta*. Umpikuja johtuu yleensä tilanteesta, jossa kaksi prosessia varaa resursseja ja odottavat samalla varattuja resursseja pystyäkseen jatkamaan työskentelyä (GeeksforGeeks 6.8.2020). Kun toinen prosessi panttaa ensimmäisen prosessin vaatimaa resurssia, ei ensimmäinen näin pääse etenemään ja vapauttamaan toisen prosessin tarvitsemaa resurssia. Näin ollen kumpikin prosessi odottaa toistaan, eikä kumpikaan pääse etenemään (Kuva 4).



Kuva 4. Demonstraatio umpikujasta, jossa kaksi prosessia varaa toistensa varaamaa resurssia ja odottavat seuraavan resurssin vapautumista. Kuva muunneltu GeeksforGeeks sivustolla esitellystä kuvasta. (GeeksforGeeks 6.8.2020.)

Selvittäessäni REST-kutsua, joka laukaisee kyseisen tapahtumasarjan, löysin järjestelmästämme uuden virheen ja raportoin siitä useammalle kollegalleni. Koska kyseinen virhe esti umpikujatehtävän tarkemman selvittämisen, päätin siirtyä seuraavana päivänä etsimään ratkaisua ensin siihen.

Torstai 4.3.2021

Päiväni koostui pitkälti edellisenä päivänä löytämäni virheen korjaamisesta. Käytin ison osan päivästä tutustumalla järjestelmän osaan, joka sijaitsee REST-kutsun ja sen vastaanottavan ohjaimen välissä. En ymmärrä kutsun ja ohjaimen välissä olevaa rakennetta vielä täysin, enkä siis tiennyt mistä virhettä olisi paras etsiä. Virheen etsimisessä meni odotettua kauemmin, koska osa logiikasta on .NET-rungon tarjoamaa valmista rakennetta, jolloin esimerkiksi kiinnostavaan koodiriviin pysäyttäminen ohjelman ajon aikana oli vaikeaa.

Pyysin kollegaltani apua virheen etsimiseen ja yhdessä paikansimme ongelman REST-kutsusta tulevan tiedon konvertointiin. Konvertoiva attribuutti periytyi .NET-rungon tarjoamasta luokasta ja näin ollen käytti .NET tarjoamia metodeja päivämäärien konvertoinnissa ja validoinnissa. Validoidessaan päivämääriin liittyvää tietoa ottaa runko huomioon sen hetkisen suoritettavan kulttuurin. Kulttuuri tarjoaa .NET-rungossa työkalut erilaisten aikaformaattien, kalenterien sekä tietotyyppien järjestelyyn maa- tai kielikohtaisesti. Koska järjestelmässämme kulttuuri on kiinni sillä hetkellä tietoa käsittelevästä järjestelmästä, oli konvertoinnilla järjestelmässä hyvä mahdollisuus epäonnistua. Opin, että tällaiset muutokset liittyvät vahvasti valittuun runkoversioon ja ymmärsin minkä takia olemme viime aikoina keskustelleet mahdollisuudesta siirtyä .NET-rungon uuteen versioon. (Microsoft 2021d.)

Teimme tarvittavat muutokset rungosta periytettyyn luokkaan lisäämällä mahdollisuuden validoida päivämääriä tietotyyppinä ottamatta huomioon sen hetkistä kulttuuria. Koska virhe vaikutti poistuvan muutoksien myötä, päivitin lopuksi luokkaan liittyvät yksikkötestit ja lisäsin niihin muutaman uuden tapauksen tehdäkseni niistä kattavammat. Vaikka en kokenut saaneeni päivän aikana paljoa aikaan, uskon oppineeni paljon uutta järjestelmästämme, sekä .NET-rungon käytöstä sen osana.

Perjantai 5.3.2021

Aloitin päiväni opiskelemalla eilen ratkaisemaamme REST-kutsun tiedon konvertointiin liittyvää ongelmaa. Koska kokeneempi kollegani ymmärtää .NET logiikkaa kattavasti ja

osasi näin oikoa tehtävän ratkaisua, koin tarpeelliseksi tehdä lisäselvitystä asiasta. Halusin ymmärtää itsekin syyn alkuperäisen ratkaisun toimimattomuuteen, sekä tutustua .NET-rungon osittain julkisena löytyvään lähdekoodiin. Kävimme yhdessä läpi virheeseen johtaneen lähdekoodin, ja lopulta en pelkästään oppinut kulttuureihin liittyvistä ongelmista .NET-rungossa, vaan löysin myös uusia tapoja tutustua siihen.

Kävin myös keskustelua virheen löytymiseen johtaneesta alkuperäisestä tehtävästä. Umpikujaan johtaneet useat käyttöliittymästä lähetetyt REST-kutsut on ohjelmoitu pitkän aikaa sitten, emmekä keksineet syytä viiden kutsun lähettämiseen yhden sijasta. Mikäli kutsut voitaisiin korvata yhdellä kutsulla, voitaisiin tiedon eheys varmistaa ja umpikuja välttää alemmassa logiikassa. Tämän lisäksi osa kutsuista toistaa samoja tietokantakyselyjä ja näin ollen toimintaa voisi tältäkin osin tehostaa.

Loppupäivästä käyttöliittymäkehittäjä ilmoitti myös, että mobiilisovelluksemme tekee samaisia kutsuja omalla tavallaan, eikä kutsujen yhdistäminen välttämättä olekaan ainoa tarvittava muutos. Koska totesimme suunnitelman muuttuneen päivän aikana useaan kertaan, sovimme palaverin ensi viikoksi. Palaverin tavoitteena on ottaa mukaan kokeneempia kehittäjiä, joilla saattaa olla tietoa alkuperäisen ratkaisun syistä, ja käydä keskustelua siitä mitä ratkaisuun tarvitaan.

Viikkoanalyysi

Tällä viikolla pääsin kirjoittamaan paljon yksikkötestejä ja syventämään ymmärrystäni niiden kehittämisestä. Teknisinä teemoina nousivat tällä viikolla myös .NET-rungon käyttö osana järjestelmäämme, sekä umpikujatilanteiden syyt, sekä tavat välttää niitä. Projektityöskentelyn osalta ei tällä viikolla korostunut käsiteltäviä teemoja.

Martin mainitsee kirjassaan kolme testilähtöisen ohjelmistokehityksen periaatetta:

1. Tuotantokoodia kirjoitetaan vain sen verran, kuin yksikkötestin läpäisemiseen vaaditaan.
2. Yksikkötestiä kirjoitetaan vain sen verran, että yksikkötesti saadaan epäonnistumaan.
3. Tuotantokoodia kirjoitetaan vain sen verran, että tällä hetkellä epäonnistuva testi saadaan onnistumaan. (Martin 2009, 123–124.)

Tunnistan, etten vielä osaa ajatella kehittämistä testilähtöisesti ja voisin pyrkiä muuttamaan työskentelytapojani siihen suuntaan. Mielestäni Martinin mainitsemat periaatteet soveltuvat kuitenkin hyvin myös jälkikäteen lisättyjen yksikkötestien

kirjoittamiseen. Toisaalta Martin (2009, 133) korostaa, että mikäli testit kirjoitetaan vasta tuotantokoodin jälkeen, voi testien kirjoittaminen olla vaikeaa.

Uskon, että testien kattavuus ja niiden hyvä toteutus on arvokasta järjestelmän toimivuuden kannalta myös jälkikäteen. Isossa järjestelmässä, kuten OldTimer ja Newbie, muutokset voivat vaikuttaa yllättävissä paikoissa. Tasdik Rahmanin (13.3.2019) kirjoittaman artikkelin ”What should and should not be tested in unit tests?” mukaan arvokkain yksikkötestin ominaisuus onkin, että testi paljastaa, milloin järjestelmää muutetaan.

Rahman (13.3.2019) nostaa esiin myös toisen periaatteen testaamiseen: Mikäli testattava koodi on erittäin suoraviivainen tai vastaavasti erittäin monimutkainen, yksikkötestaamiseen ei kannata käyttää kohtuuttomasti aikaa. Toisaalta Martin (2009, 123–124) nostaa esiin ajatuksen, että resurssien käyttö yksikkötestaamiseen palkitsee pitkässä juoksussa, kun testilähtöinen kehittäminen nopeuttaa kehitysprosessia.

Oma testaamiseni nojaa vielä vahvasti tuotantokoodin toteutukseen. Koen toistaiseksi hankalaksi ajatuksen kirjoittaa testiä ennen tuotantokoodia, kun en yleensä ole varma siitä mitä kaikkea lopullisen ratkaisun tekeminen vaatii. Uskon tässä olevan kyseessä myös kerralla liian ison konseptin ajattelu. Pyrin kehittämisessäni hahmottamaan tiedonkulun järjestelmässä käyttöliittymästä saapuvasta kutsusta tietokantakyselyyn asti, kun ratkaisun kannalta kriittistä saattaa olla yksittäisen metodin tai jopa yhden rivin muutos. Testin kirjoittaminen ennen ratkaisua saattaakin olla juuri, mitä minun kuuluisikin tehdä.

Umpikujatilanteen puolesta ymmärsin tällä viikolla, kuinka .NET-runkokin sisältää rajoituksia. Käyttöliittymän kautta umpikujaa selvittäessäni löytämäni virhetila johtui .NET-rungon luokasta periytetty luokkamme, joka näin ollen nojasi .NET-rungon tarjoamaan metodiin. Opin, että .NET-runkoon nojatessa on tärkeää ymmärtää sen ottavan kulttuurin huomioon konvertoidessaan tietotyyppejä. Konvertointiin tarjottiin järjestelmässämme alun perin valmiiksi tietyssä formaatissa olevia kirjainjonoja. Rungon luonnollinen oletuskulttuuri silloin, kun kulttuuria ei ole erikseen määritelty, on prosessin kulttuuri. Näin ollen, kun konvertoinnissa prosessi tarjoaakin konvertointiin vääränlaista kulttuuria - joka on hyvin mahdollista selaimen kautta käytettävässä järjestelmässä – voidaan konvertoinnin onnistumista kaikissa tilanteissa pitää epävarmana. (Microsoft 2021e, TypeConverter.cs; TypeDescriptorContext.cs; DateTimeConverter.cs.)

Itse umpikujatilanteen kannalta opin tällä viikolla, miten kyseisenlainen tilanne voi muodostua käyttöliittymän toistuvien kutsujen kautta. Esimerkiksi käyttöliittymän puolesta

tilanteen saattoi muodostaa yksinkertaisesti se, että kutsuja lähettävä painike ei lukkiutunut kutsujen lähettämisen jälkeen ja mahdollisti näin painikkeen peräkkäisen painamisen. Kun yksi painallus lähetti järjestelmään viisi kutsua, kasvaa kutsujen määrä moninkertaisesti jokaisen ylimääräisen painalluksen myötä. Tämän kautta ymmärsin paremmin, mitä click happy eli käyttäjän kärsimätön painikkeiden painelu käyttöliittymässä tarkoittaa konseptina ja mihin se voi johtaa. Ongelma on yleinen idempotenttien REST-rajapinnan kutsujen kanssa, jotka ruuhkauttavat järjestelmän helposti. Painikkeen lukitseminen ensimmäisen painalluksen jälkeen voi ratkaista ongelmamme ainakin osittain. (Korneeva 23.10.2020.)

3.4 Seurantaviikko 4

Maanantai 8.3.2021

Päiväni alkoi käymällä kollegani kanssa keskustelua edellisellä viikolla julkaisemastani muokkauspyynnöstä merkkijonojen konvertointiin liittyen. Kollegani pyysi minua tekemään yksikkötestit ongelmasta, jonka muutokseni ratkaisivat. Koska alkuperäinen ongelma oli konvertoinnin huomioon ottama kulttuuri ja ratkaisuni myötä annetulla kulttuurilla ei enää ole väliä, koin aluksi hankalaksi testata muutosta. Kollegani kuitenkin ehdotti, että lähestyn ongelmaa testilähtöisemmin. Hän kehotti minua ottamaan ratkaisuani edeltävän version järjestelmästä ja kirjoittamaan oletettavasti epäonnistuvan testin alkuperäiseen ongelmaan. Ajaessani saman testin sitten uudessa versiossa tuli sen onnistua ja näin testi on kirjoitettu testaamaan juuri sitä ongelmaa, jonka takia muutos tarvittiin. Näin olin kirjoittanut testin testilähtöisellä ohjelmistokehitysmallilla (Martin 2009, 122–123) ja kollegani oli tyytyväinen kirjoittamiini muutoksiin.

Testin kirjoittamisen lisäksi käytin aikaa umpikujatilanteeseen johtaneen toiminnon ratkaisun hahmottelemiseen. Tarkoitus on viiden erillisen REST-kutsun sijaan lähettää käyttöliittymästä vain yksi kutsu, joka ajaa tarvittavat operaatiot yksi kerrallaan välttäen näin umpikujaan johtavat ristikkäiset prosessit. Kävimme myös iltapäivästä mentorointiryhmäni kanssa läpi, mikä kutakin meistä motivoi kehittymään ohjelmoijana ja mistä tahto osallistua mentorointiohjelmaan kumpuaa.

Tiistai 9.3.2021

Miltei koko päiväni kului sprintin päättämiseen liittyvissä palavereissa. Osa tiimistämme osallistui aiemmalla viikolla Scrum-koulutukseen ja siitä inspiroituneena sprintin päätöstä aikataulutettiin uudestaan. Aloitimme heti aamusta sprinttikatselmuksen, jossa keskustelu muuttui taas mielestäni liian yksityiskohtaiseksi ja tekniseksi. Scrum-viitekehyksen työkaluja tarjoava yritys ScrumDesk käyttää sprintin arvostelun aikataulukseen pohjaa, jossa kehitystiimin demolle on varattu maksimissaan 15 minuuttia kahden tunnin palaverista (ScrumDesk s.a.). Palaverissamme yksi kehittäjistä demosi tekemiään optimointeja yhteensä 25 minuuttia, suurilta osin puhdasta koodia näyttämällä.

Lisäksi keskustelimme lähitiimini retrospektiivissä muutamista teknisistä haasteista projektissamme ja suunnittelimme seuraavan sprinttimme. Koen itse välillä vaikeaksi arvioida työtehtävien kestoa, kun välillä helpoksi luultu tehtävä voikin paljastaa isomman ongelman järjestelmässä ja viedä yhden työpäivän sijasta viisi.

Keskiviikko 10.3.2021

Tänään olin mukana rakentamassa **WebJobiksi** kutsuttua kokonaisuutta Newbiehen tehtävän uuden integraation tueksi. Integraatio on laskujen hallinnointiin tarkoitettuun järjestelmään ja WebJobin tarkoitus on lähettää järjestelmässämme luodut laskut toiseen järjestelmään. WebJob lisätään pilveemme, jolloin se pysyy käytössä kellon ympäri ja sen ajo voidaan laukaista ajastetuin väliajoin (Microsoft 16.10.2018).

Pyysin päästä tekemään kyseistä tehtävää kokeneemman kehittäjän kanssa yhteistyössä, jotta ymmärrykseni pilvessä tapahtuvista prosesseista kasvaisi. Kävimme yhdessä läpi, miten uuden itsenäisen projektin rakentaminen järjestelmässämme toimii, sekä minkälaisia muutoksia järjestelmän julkiasuputkessa ajettavat skriptimme kaipaavat tällaisessa tilanteessa. Pääsin myös paketoimaan ja rakentamaan ensimmäisen **NuGet**-pakettini, jonka infrastruktuurista vastaava kollegani voi asentaa pilveemme. Käytimme koko päivän integraatioon liittyen, suunnittelupalaverin sekä WebJobin rakentamisen muodossa.

Torstai 11.3.2021

Päivän teema oli olemassa olevien muokkauspyyntöjen ylläpitäminen. Koska järjestelmäämme kehitetään jatkuvasti isolla volyymillä, parikin päivää vanha muokkausjulkaisu vanhenee nopeasti. Tällä kertaa eniten ongelmia tuotti atomisuuteen liittynyt muokkauspyyntöni, joka erosi paljon uusimmasta pääversiostamme ja oli näin ollen myös altis yhdistämiseen liittyviin konflikteihin.

Osa järjestelmämme yksikkötesteistä on alun perin kirjoitettu **JustMock**-kirjastoa hyödyntämällä ja sittemmin todettu sen aiheuttavan ongelmia testien ajossa erilaisissa ympäristöissä. JustMock tarjoaa työkaluja yksikkötestiin tarvittavien osien matkimiseen (Telerik JustMock s.a.). Konfliktit liittyivät yksikkötesteihin, joista olin tehnyt kattavammat ja joista oli sittemmin toisen kehittäjän muutoksien myötä poistettu JustMockin käyttö. Pääsin kuitenkin opettelemaan samanlaisten toimintojen suorittamisen testeissä ilman kyseistä kirjastoa ja olin tyytyväinen saamiini tuloksiin.

Sain atomisuuteen liittyvästä tehtävästä myös palautetta liittyen metodiin, jota kokeneempi kollegani uskoo järjestelmän kutsuvan erittäin usein. Kollegani ehdotti, että koska kutsuttava osa puolestaan kutsuu tietokantakyselyä suorittavaa metodia, olisi ongelmaan hyvä ratkaisu etsiä tietoa ensin välimuistista. Mikäli välimuistista ei löydy kyseistä tietoa, voidaan tietokantakysely tehdä. Näin tieto voidaan saada nopeasti, vaikka kutsuja

esitettäisiin miljoona ja samalla estetään miljoonien peräkkäisten tietokantakyselyjen tekeminen (Microsoft 30.3.2017).

Perjantai 12.3.2021

Aamupäiväni hurahti mentorointipalaverissa, jossa kävimme läpi työmme ohessa esiin tulleita uusia termejä, joihin kaipasimme selvennystä ja keskustelimme erilaisista ohjelmointiin liittyvistä konsepteista. Meidän tuli myös jokaisen etukäteen valita mainituista termeistä yksi ja selittää se palaverissa toisillemme. Oma termini oli tietorakenne puna-mustapuu, joka oli itsellenikin miltei uusi termi. Koska puna-mustapuu on tasapainotettu binäärihakupuu, koin aiheen helpoksi lähestyä (GeeksforGeeks 13.10.2020; Sambol 25.8.2016, 0:00-0:50). Binäärihakupuu tuli minulle tutuksi vaihdossa, jossa käsittelimme puita peliohjelmoinnissa hyödynnettävinä tietorakenteina.

Mentoroitavana olon lisäksi pyrin sulkemaan kaikki aukinaiset tehtäväni ja saamaan muokkauspyyntöni läpi, ettei ensi viikon lomani ajaksi jäisi mitään järjestelmämuokkauksia vanhentumaan. Tämä tarkoitti pitkälti muokkauspyyntöjen testiajojen tilan tarkistamista ja laadunvarmistuksen kanssa kommunikointia. Lisäksi osallistuin kaikkien tuotteemme kanssa työskentelevien tiimien keskinäiseen kuukausittaiseen palaveriin. Palaverissa käydään läpi esimerkiksi myynnin, markkinoinnin ja asiakaspalvelutiimien tilanne. Palaveri ei ole pakollinen, mutta koen saavani paremman käsityksen tuotteemme asiakaspinnasta osallistumalla siihen.

Päätin lopettaa työt hieman aiemmin, koska tiesin jäljellä olevan tehtäväni vaativan enemmän työstämistä kuin mitä olisin tänään ehtinyt tekemään. Koin jännittäväksi lähteä lomalle, sillä mikäli joku muutoksistani aiheuttaisi ongelmia tuotannossa, en olisi paikalla korjaamassa niitä.

Viikkoanalyysi

Teknisinä teemoina tällä viikolla tutustuin puna-mustapuuhun tietorakenteeseen ja opin välimuistin hyödyntämisestä useiden samanlaisten tietokantakutsujen vähentämisessä. Projektityöskentelyn kannalta pohdin tälläkin viikolla tapoja kehittää ketteriä käytäntöjämme.

Puna-mustapuu tuli vastaan mentorini linkittämässä videossa *ylisuunnittelua* ohjelmistokehityksessä käsittelevällä videolla. Me mentoroitavat saimme kerätä listan termejä, jotka ovat tulleet vastaan töissämme ja joita emme vielä ymmärrä. Koska toisille

opettaminen on yksi tehokkaimmista tavoista oppia itse, otimme jokainen itsellemme yhden termin, jonka sitten opetimme muille. Valitsin puna-mustapuun, koska tutustuin vuosi sitten puutietorakenteisiin vaihdossa ja puna-mustapuu oli minulle uusi konsepti. Puna-mustapuu on tasapainotettu binäärihakupuu, jonka *solmuja* väritetään ja siirrellään päittäin solmun sukulaisista riippuen. Näin puun tyypillisimpien operaatioiden eli haun, lisäyksen, sekä poiston aikavaativuutta voidaan pienentää ja ylläpitää. (GeeksforGeeks 13.10.2020; Jarrett 4.5.2018; Sambol 25.8.2016).

Koska olin itse tutustunut binäärihakupuihin sekä muihin tyyleihin käyttää puita tiedon säilyttämiseen, oletin muiden mentoroitavien olevan tuttuja konseptin kanssa. Kun kysyin, kuinka tuttu binäärihakupuu muille oli, sain vastaukseksi hiljaista muminaa ja tajusin joutuvani selittämään myös binäärihakupuun tarkoituksen. Tämä hämmensi minua ja koin opetukseni jääneen tarvittua vajaammaksi, koska en ollut valmistautunut selittämään puutietorakenteen perusteita. Osittain se kuitenkin kasvatti itseluottamustani, kun tajusin olevani tuttu muille tuntemattoman teorian kanssa.

Puna-mustapuu ei käytännössä ole työni kannalta hirveän hyödyllistä teoriaa. Puna-mustapuuta ei tapaa nykypäivän web-ohjelmoinnissa paljoa, vaikka se on yleisesti käytössä. Puna-mustapuuta on käytetty paljon erittäin alhaisen tason logiikassa, kuten Linux-käyttöjärjestelmässä tai MySQL:n indeksoinnissa. Uskon sitä koskevan teorian olevan kuitenkin tärkeää ymmärtää. (GeeksforGeeks 13.10.2020.)

Tutustuin tällä viikolla myös välimuistin hyödyntämiseen nopeassa tiedonhaussa. Tallentamalla välimuistiin tietoa, joka muuten saatettaisiin hakea miljoona kertaa tietokannasta, voidaan järjestelmää tehostaa ja nopeuttaa. Kehittäessä SaaS-tuotetta on tärkeä ottaa huomioon, kuinka usein tietynlaisia pyyntöjä saatetaan enimmillään kutsua. Tiedostamalla reitin olevan käytössä miljoonia kertoja sekunnissa voidaan järjestelmästä tunnistaa sen suorituskyvyn heikkoja kohtia. (Microsoft 30.3.2017)

Omassa tapauksessani pääsin hahmottamaan, miten objekti tallennetaan välimuistiin käyttäen identifioivaa nimeä. .NET Framework tarjoaa objektin tallentamiseen tällaisen mahdollisuuden, jolloin myös identifioivan nimen kartoittaminen on tärkeää. Mikäli joku muu järjestelmässämme sattuisi tarvitsemaan samaa välimuistiin kirjoitettua tietoa ja objektin nimi on intuitiivinen, voi seuraava tietoa tarvitseva kehittäjä hakea suoraan valmiiksi tallennettua tietoa välimuistista. Teemana välimuistiin tallentaminen järjestelmäkoodilla oli uusi asia ja olin iloinen löytäessäni uuden tavan tehostaa järjestelmäämme. (Tutorialspoint, s.a.a)

Tämän viikon sprintin päätökseen liittyneessä katselmuksessa keskustelu syveni hyvin tekniseksi ja erään kehittäjämme demo vaati neljäsosan koko tiimin katselmuksesta. Otin aiheen esille lähitiimini kanssa retrospektiivissä. Ehdotin, että jatkossa Scrum-palavereita vetäisi yksi virallinen henkilö, joka päättää mikä kuuluu katselmukseen ja mikä ei. Tyypillisesti Scrumiin kuuluu Scrum Masteriksi kutsuttu henkilö, jonka vastuulla on ylläpitää tiimin projektityöskentelyä Scrumin viitekehyksen sisällä. Tähän kuuluu esimerkiksi tiimiläisten kouluttaminen ymmärtämään Scrumin tarjoamia kehittämistä tehostavia työkaluja, sekä työkalujen oikeanlaisen käytön varmistaminen. (Schwaber & Sutherland, 2020. 6–7). Schmidt (2015, 16) mainitsee kirjassaan Scrum Masterin vastuulla olevan myös mahdollisten esteiden ja hidasteiden poistaminen tiimin tieltä. Tämä mielestäni voidaan nähdä myös keskustelun, sekä tiimin keskittymisen ohjauksena.

Tällä hetkellä epävirallisena isäntänä koko tiimin laajuisissa palavereissa on toiminut toinen tuoteomistajistamme. Koska tätä vastuuta ei kuitenkaan ole varsinaisesti annettu hänelle, luo se epävarmuutta koko tiimiin siitä kuka, miten ja millä aikataululla sprintin päättämiseen liittyvät palaverit toteutetaan. Tällä viikolla tuoteomistajamme joutui lähtemään muutamaa minuuttia ennen palaverin virallista lopetusta ja uskon tiimimme istuneen useita minuutteja turhaan palaverissa, jonka olisi voinut jo päättää. Koska vastuuta palaverin isännöinnistä ei virallisesti ollut annettu kenellekään, ei kukaan kokenut oikeudekseen päättää palaveria.

En tiedä, onko tiimimme olevan vielä valmis varsinaiseen Scrum Masteriin, mutta uskon tämän tyylisestä fasilitaattorista olevan hyötyä kehitystyömme kannalta. Henkilöllä olisi hyvä olla asiantuntemusta ymmärtää Scrumia työkaluna. Lisäksi henkilön olisi tärkeä ymmärtää kaikkien tiimien asiantuntijoiden kannalta, milloin käyty keskustelu on hyödyllistä koko tiimille ja milloin olisi syytä siirtää keskustelu katselmuksen ulkopuolelle. Näin ollen teknisellä tasolla keskustelua ymmärtävällä isännällä olisi myös auktoriteetti keskeyttää katselmuksen ulkopuolelle kuuluva keskustelu ja siirtää tiimin huomio takaisin koko tiimin kannalta tärkeisiin aiheisiin.

3.5 Seurantaviikko 5

Maanantai 22.3.2021

Tutustuin tänään ennen lomaani esiin nousseeseen ongelmaan umpikujatilanteen aiheuttamien REST-kutsujen yhdistämisessä. Ongelma vaikutti johtuvan päällekkäisistä transaktiorajauksista. Transaktioluokan eristystaso määrittelee, mitä ja miltä prosesseilta transaktio lukitsee resursseja käsittelynsä ajaksi (Microsoft 8.12.2019). Opin, että järjestelmässämme yleistynyt käytäntö on käyttää .NET-rungon tarjoamasta luokasta periytyvää luokkaa. Koska olin itse käyttänyt suoraan .NET-rungon TransactionScope-luokkaa ja sen oletuksena saama eristystaso oli erilainen kuin muualla järjestelmässä (Microsoft 2021f), syntyi järjestelmässä päällekkäisten transaktioiden toisistaan poikkeavien eristystasojen johdosta virhe. Korjasin virheen kutsumalla omaa luokkaamme ja siirryin kirjoittamaan yksikkötestejä kaikille muutoksilleni.

Päivään sisältyi myös tiimissämme uudelleen käyttöön otettu sprinttikatselmukseen valmistautumiseen tarkoitettu palaveri. Palaveri otettiin käyttöön, koska haluamme jakaa katselmuksessa tapahtuvan esiintymisen ja kysymyksiin vastaamisen vastuuta tasaisemmin lähitiimini kesken. Suuren osan tiimimme demoista hoitaa käyttöliittymäohjelmoijamme ja olemme pohtineet, jääkö katselmuksissa tästä johtuen jotain tärkeää sanomatta alempaan logiikkaan liittyen. Koin palaverin hyödylliseksi oman osamme katselmuksen läpikäyntiin, jotta olisimme mahdollisimman tehokkaita itse katselmuksessa. Lyhyt ja tehokas katselmus pitää tiimin mielenkiinnon ja keskittymisen tärkeissä asioissa (Bowes 9.8.2014), joten sen yhdessä valmistelu voi mielestäni vain hyödyttää tiimiä.

Tiistai 23.3.2021

Suurin osa päivästäni kului sprintin päättämiseen liittyvissä palavereissa. Aloitamme sprintin päätöksen katselmuksella, jossa tällä hetkellä jokainen neljästä tiimimme sisällä jaetusta lähitiimistä esittää sprinttinsä tulokset. Siinä missä toisella kehitystiimistämme oli enemmän näytettävää, oma tiimimme työskenteli viime sprintin suurimmaksi osaksi piiloon jäävän logiikan kanssa. Itse mainitsin katselmuksessa muutaman tekemäni muutoksen, joita en kyseisestä syystä voinut havainnollistaa demolla, mutta joiden uskoin kuitenkin olevan mielenkiintoista tietoa useimmille tiimiläisilleni.

Scrumista poiketen varsinaista kehityskeskustelua emme toistaiseksi ole käyneet katselmuksissa (Bowes 9.8.2014), vaikka demon aikana katselijoita kehotetaankin usein

esittämään tarkentavia kysymyksiä esiteltävästä toiminnallisuudesta. Uskon osan tästä aiheutuvan myös siitä, ettei katselmuksiin osallistu tiimin ulkoisia tai edes kaikkia sisäisiäkään sidosryhmiä. Tiimissämme pohditaan kuitenkin tällä hetkellä paljon projektityöskentelyämme kehittäviä uudistuksia, joihin pureuduimme retrospektiivissämme.

Aiemmasta käytännöstä poiketen tuoteomistajamme osallistuivat lähitiimini retrospektiiviin. Koska koen tuoteomistajien omaavan paljon kehittäjiä enemmän tietoa ja kokemusta projektityöskentelyn hallinnoimisesta, uskon heidän olevan hyödyksi siihen liittyvissä palavereissa. Kävimme keskustelua Scrumin työkalujen hyödyntämisestä ja siitä, millä tavalla tiimimme työskentely tällä hetkellä poikkeaa Scrumista. Esiin nousi myös yleisten kehittämiseen sovellettavien käytäntöjen puute. Ehdotin jonkinlaisen työpajan pitämistä käytäntöjen sopimiseksi, koska olemme käyneet aiheesta keskustelua aiemminkin. Päätimme kuitenkin jäädä odottamaan tulevia tiimimuutoksia ennen työpajan eteenpäin puskemista.

Jään usein teknisessä keskustelussa hiljaisemmaksi, mikäli en koe ymmärtäväni keskusteltavasta konseptista tarpeeksi. Omaan mielestäni kuitenkin paljon ymmärrystä pehmeistä taidoista ja puhun mielelläni myös projektityöskentelystä. Näin ollen koin itseni erittäin hyödylliseksi tänään, kun kävimme paljon projektityöskentelyyn liittyvää keskustelua.

Keskiviikko 24.3.2021

Tänään sijoitin suurimman osan työtunneistani mentorointiohjelmaan liittyvän ohjelmointitehtävän ratkaisemiseen. Tehtävässä tuli luoda liiketoimintalogiikka alennuslihasovellukselle, jossa käyttäjät voivat ilmoittaa sekä etsiä alennuslihaa myyviä kauppoja. Tehtävässä luetellaan useampi käyttäjätarina, joiden perusteella tuli kirjoittaa liiketoimintalogiikka täyttämään tarinoista esiin nousseet vaatimukset. Pitkän tähtäimen tavoitteena mentorointiohjelmassa on tehdä samainen tehtävä useamman kerran ja näin seurata omien taitojen kehittymistä. Pyrin tekemään ohjelmassani päätöksiä tasapainotellen optimaalisuuden ja yksinkertaisuuden välillä välttääkseni ICT-alalla tyypillisesti esiintyvän ylisuunnittelun (Young 29.4.2017, 1:45-3:10).

Tapasimme mentorointiryhmäni kesken iltapäivästä ja kävimme läpi luomamme ratkaisut. Omani sai positiivista palautetta juuri sen yksinkertaisuudesta ja luettavuudesta. Keskustelimme myös, miten valitsemani luokkien ja metodien nimet sopivat saamieni käyttäjätarinoiden käsitteistöön ja tiedostin ajatelleeni asiaa alitajuisesti ohjelmaa

kirjoittaessani. Mentorini puolsi Martinin (2009, 17–30) näkemystä selkeän nimeämisen tärkeydestä ja korosti, että käyttäjätarinasta voidaan usein poimia yhteisesti ymmärrettäviä käsitteitä ohjelmoinnissa käytettäviksi nimiksi.

Keskustelimme myös ratkaisustani tehdä yhdestä liiketoimintalogiikkaa sisältävästä luokasta Singletonin, jota mentorini piti ainakin ratkaisemallani tavalla alustettuna vanhanaikaisena ratkaisuna. Tapaamisen lisäksi ehdin käydä keskustelua kollegoideni kanssa umpikujatilannetehtävääni liittyvien parametrien nimeämisestä, sekä tehtävän vaikutuksesta IoC-infrastruktuuriin. Sain päivän aikana mielestäni suhteellisen paljon aikaan, vaikken ehtinytkään pureutua normaaleihin työtehtäviini kunnolla.

Torstai 25.3.2021

Päiväni alkoi ennen lomaani koodista poistamieni tietokantasarakkeiden poistamisella kannasta. Sekä vanha, että uusi tuotteemme käyttää samaa tietokantaa, jonka takia jännitin poistoista aiheutuvia muutoksia. Pyysin avukseni kokeneemman kollegan, jonka kanssa selvitimme sarakkeisiin liittyviä riippuvuuksia kannassa. Pääsin sarakkeiden lisäksi pitkästä aikaa käsittelemään myös indeksejä, eli usein tarvitun tiedon hakemisen nopeuttamiseen tarkoitettuja osoittimia (Tutorialspoint s.a.b) sekä näkymiä, eli taulusta etukäteen muodostettuja kyselyitä (Tutorialspoint s.a.c).

Sarakkeiden poiston lisäksi tarkistin aiempiin tehtäviini liittyvät muokkauspyynnöt ja jatkokehitin mentorointiohjelmaan liittyvää ratkaisuani. Huomasin ohjelmoidessani pyrkiväni ratkaisemaan tehtävän paljon monimutkaisemmin, kuin mitä olin edeltävänä päivänä halunnut. En kokenut saaneeni päivän aikana hirveästi aikaan, jonka lisäksi koin epäonnistuneeni katsellessani kasaan kirjoitettua ohjelmaani. Totesin päivän lopuksi, ettei ratkaisun ylityöstäminen paranna lopputulosta ja jätin tehtävän toiselle päivälle.

Perjantai 26.3.2021

Ehdin päivän aikana tekemään kaikenlaista pientä sekalaista tehtävää, kuten uuden hintalistausominaisuuden käyttöönottamista. Yritin myös ratkaista asiakaspalvelusta tullutta virheilmoitusta kalenterisynkronoinnin ongelmasta, jossa asiakas on vastaanottanut sähköpostiinsa kalenterimuutoksista johtuvia virheellisiä ilmoituksia. En kuitenkaan saanut toistettua ongelmaa, enkä tiennyt miten lähteä tilannetta purkamaan. Toisaalta myöskään asiakaspalvelija ei ollut saanut virhetilannetta toistettua testiympäristössämme. Muutaman tunnin selvittelyn jälkeen keskustelin ongelmasta lähitiimivastaavani kanssa ja päädyimme jättämään tehtävän tältä päivältä selvittämättä.

Koin epäonnistuneeni tehtävässä, vaikka ratkaisuun annetut eväät olivatkin vähäiset ja yritin minkä voin. Päivinä, joina teen paljon pieniä asioita, tunnen harvoin saaneeni aikaan paljoa. Tällaisina päivinä on vaikea motivoitua, varsinkin jos tarjolla ei ole uutta varsinaista tehtävää. Päätin käyttää ylityötuntejani ja lopettaa työviikkoni normaalia aiemmin.

Viikkoanalyysi

Viikkoni sisälsi monia pienempiä teknisiä tehtäviä, mutta isompana konseptina nousi mentorointiohjelmasta saamani tehtävänanto: käyttäjätarinoiden mukaisen järjestelmän rakentaminen tyhjästä. Sen lisäksi pohdin tällä viikolla paljon Scrum-palavereissa esille nousseita ajatuksiani tiimimme häilyvästä DoD-määrittelystä, sekä kehittäjiemme yleisistä ohjelmointikäytännöistä.

Kirjoitin mentorointitehtävän ratkaisua ensin mahdollisimman yksinkertaisesti ja suoraviivaisesti käyttäjätarinoita vastaavaksi. Järjestelmän kasvaessa huomasin kuitenkin pyrkiväni jatkuvasti ratkaisemaan tehtävän vaatimukset kattavammin, jolla oli mielestäni suora vaikutus järjestelmän monimutkaisuuteen. Aloittelevana ohjelmoijana ajattelen ongelmien ratkaisut usein liian isona kokonaisuutena. Koen, että yksi isoimpia aloittelevan kehittäjän kompastuskiviä onkin pyrkimys ratkaista koko ongelma kerralla. Siinä, missä kokeneempi ohjelmoija osaa pilkkoa tehtävän pienempiin hallittaviin osiin, joudun itse usein peruuttamaan kehittämässäni ratkaisussa pystyäkseni paremmin hahmottamaan ratkaisuun vaadittavat komponentit.

Young argumentoi Build Stuff -konferenssin avauspuheessaan, että tietojenkäsittelyn koulutuksessa keskitytään usein ratkaisemaan ongelmat kokonaisuudessaan. Vedoten kokemukseensa juniorkehittäjien rekrytoinnista Young kertoo, kuinka suurin osa vastavalmistuneista hakijoista keskittyy tietorakenteista kysyttäessä niiden Big-O-notaatioon ajattelematta, miten käyttötarpeesta riippuen toinen tietorakenne voi olla hyödyllisempi. Siinä, missä punamustapuulla on mahdollisuus olla Splay-puuta nopeampi, Splay-puu optimoi eniten haettua tietoa kierrättämällä sitä lähemmäs *juurta*. (Young 29.4.2017, 8:03-11:50.)

Ajattelemalla tietorakenteen käyttötarkoitusta ja liiketoiminnallisia vaatimuksia voidaan kehityksessä tunnistaa järjestelmän realistiset ongelmat ja näin välttää pyrkimystä ratkaista kaikkea. Young näyttää kuvaa pyörästä ja avaruusraketista, kertoen avaruusraketin olevan huono pyörä. Mikäli liiketoiminnalliset vaatimukset tarvitsevat pyörän, on turha rakentaa avaruusrakettia. Young nostaa myös esiin, ettei kaikissa

järjestelmän ongelmissa ole kysymys teknisestä ongelmasta, vaan välillä on kyse liiketoiminnallisesta riskistä. (Young 29.4.2017, 2:05-2:30; 10:30-12:02.)

Laaksonen myös (2017, 12) ottaa oppikirjassaan ”Tietorakenteet ja algoritmit” esiin Big-O notaation: ”Mitä hyötyä on määrittää algoritmin aikavaativuus? Hyötynä on, että aikavaativuus antaa meille arvion siitä, kuinka *hyvä* algoritmi on eli miten suuria syötteitä sillä voi käsitellä tehokkaasti.”. Vaikka Laaksonen myöhemmin mainitsee, että algoritmin todellinen ajankäyttö riippuu monista asioista, keskitytään oppikirjassa juuri Youngin hahmottelemalla tavalla pitkälti algoritmin nopeuteen irrallisena kokonaisuutena.

Uskon, että vaikka algoritmien aika- ja muistivaativuuden hahmottaminen on tärkeää, ovat Youngin nostamat seikat totuudenmukaisia. Sen sijaan, että pyritään luomaan kaikkein optimaalisin algoritmi teorian pohjalta, tulee kehityksessä miettiä ratkaisun todellisia käyttötarkoituksia ja riskejä (Young 29.4.2017, 2:05-11:50). On hyvä muistaa, että aloitteleva kehittäjä ei välttämättä ymmärrä kyseenalaistaa oppikirjan tarjoamia vastauksia ja olenkin oppinut ajattelemaan valmiiksi tarjottuja ratkaisuja kriittisesti vasta työelämässä. Tästä hyvänä esimerkkinä on esimerkiksi viime viikolla noussut .NET-rungon tarjoaman RangeAttribuutin oletuskulttuurin aiheuttama virhe järjestelmässämme.

Tällä viikolla pääsin ymmärtämään enemmän DoD-määrittelyn tärkeydestä. Projektityöskentelyn kannalta koen olevani vahvoilla tiimissämme. Vaikka tiimimme soveltaa ketterän kehityksen viitekehyksiä, uskon kollektiivisen ymmärryksen niiden tehokkaasta käyttämisestä olevan vielä vajavaista. Toisaalta lähitiimissäni kyse voi olla myös siitä, että kollegani ovat hyvin keskittyneitä järjestelmämme tekniseen toteuttamiseen. Teknisen toteutuksen aiheuttamien haasteiden valossa projektityöskentelyyn liittyvät asiat eivät välttämättä tunnu niin tärkeiltä. Hyvänä esimerkkinä toimii DoD-määrittelyn uupuminen päivittäisestä työskentelystämme. Tiimissämme vallitsee pitkälti vapaus ja luotto siitä, että tekijä itse osaa määritellä milloin ratkaisu on valmis julkaistavaksi.

Schwaber ja Sutherland (2020, 12) kuvaavat Scrum-oppaassaan DoD-määrittelyn rajaavan sprinttiin asetetuille tehtäville maaliviivan. DoD-määrittelyn tarkoitus on tuoda läpinäkyvyyttä tiimin sisällä jaettujen tehtävien etenemiseen, sekä luoda raamit sille mikä sprintin aikana tehdystä täyttää tuloksien julkaisuun vaaditut vaatimukset. Scrum Guidessa myös mainitaan, että mikäli tuotteen parissa työskentelee useampi itsenäisiä sprinttejä tuottava tiimi, tulee tiimien määritellä DoD yhteisesti. DoD:n mukaisesta

tuottamisesta on pääasiassa vastuussa kehittäjät, joiden tulee noudattaa yhdessä sovittuja rajauksia sprintin tuloksista.

Myös Madan (16.12.2019) kirjoittaa artikkelissaan “DONE Understanding Of The Definition Of “Done”” hyvän DoD-määrittelyn lisäävän läpinäkyvyyttä. Madanin mukaan DoD:n puuttumisesta johtuva läpinäkymättömyys aiheuttaa tiimissä epärealistisia odotuksia, epämääräisiä aikatauluja, sekä virheellistä sprintin arviointia. Määrittelemätön DoD voi Madanin mukaan aiheuttaa tuoteomistajalle vaikeuksia hahmottaa tuotteen kehitystä ja haitata näin muiden Scrumin työkalujen hyödyntämistä. Näin ollen DoD:n voidaan katsoa kulkevan käsikädessä Scrumin muiden hyötyjen kanssa ja sen irrottaminen muusta ketterästä työskentelystä haittaa tiimiä. Lisäksi Madan uskoo läpinäkyvän työskentelyn kasvattavan tiimin itsevarmuutta, joka taas Schmidtin (2005, 43–45) mukaan rohkaisee tiimiä asettamaan korkeampia tavoitteita sekä vastaamaan paremmin tuleviin haasteisiin.

DoD-määrittelyn puuttuminen aiheutti tiimissämme tällä viikolla keskustelua ja totesimme kaipaavamme muitakin yhdessä määriteltyjä käytäntöjä työskentelyymme. DoD-määrittely voisi tuoda varmuutta siihen, että kun kehittäjä laittaa muutoksen tuotantoon, tuoteomistaja tietää tämän täyttävän sille asetetut vaatimukset. DoD:llä voitaisiin tiimissäni myös määritellä, milloin ja kuinka tarkasti esimerkiksi yksikkötestausta tarvitsee tehdä. Pidän tiimissämme vallitsevasta luottamuksesta, jonka perusteella jokaiselle on annettu vapaus itse määritellä, milloin testi on paikallaan. Uskon kuitenkin yhteisten pelisääntöjen haastavan tiimiä myös tarkempaan työskentelyyn. DoD-määrittely olisi hyvä opas tiimiin uutena tulevalle kehittäjälle ymmärtää, mitkä ovat tiimin silmissä tärkeät lähtökohdat hyväksyttävän julkaisun aikaansaamiseen. Lisäksi DoD-määrittely ohjaisi katselmointejamme yksinkertaisempaan suuntaan, kun vain määrittelyä vastaavat tuotokset käydään läpi (Schwaber & Sutherland 2020, 12).

3.6 Seurantaviikko 6

Maanantai 29.3.2021

Tuoteomistajamme pyysi tänään muutoksia Newbien hintalistausominaisuuteen. Ominaisuus on tähän mennessä tehty Oldien toiminnallisuuden pohjalta, mutta paranneltu versio oli mielestäni ominaisuus, jonka käyttäjä luultavasti odottaisi olevan olemassa. Ominaisuuden tulee tarkistaa, löytyykö kannasta tietyillä raameilla olemassa olevaa oletushinnastoa ja mikäli ei, asettaa uutena luotu hinnasto oletukseksi.

Tärkeä oppi tässä tehtävässä oli kyseisenlaisen tiedon hakeminen tietokannasta optimaalisesti. Järjestelmästämme löytyy useampi kohta, jossa tietokannasta haetaan ensin kaikki annettuihin raameihin osuvat rivit ja palautetaan ne listana kutsuvalle metodille. Sekä tietokantakysely että metodi laskivat saatua sisältöä, vaikka kyseisessä tilanteessa kyselyn tuottamien tuloksien hyödyntämiseen riittää pelkkä tarkistus siitä, löysikö se tietokannasta hakukriteereillään ylipäätään mitään (Bertucci, Gallelli, Rankins & Silervstein 2012, luku 45.2.2; Vadgama 13.1.2021).

Hinnastotehtävän lisäksi kävin läpi mentorointitiimiläisteni tekemät ratkaisut saamaamme tehtävään ja annoin niistä palautetta. Toisten ratkaisujen läpikäyminen auttoi näkemään omat ratkaisuni tehtävän suhteen uudessa valossa. Lisäksi, koska kävimme ratkaisut ja kommentit vielä yhdessä läpi, sain paljon uusia näkökulmia tehtävän kehittämiseen.

Tiistai 30.3.2021

Aamuni alkoi hinnastotehtävän lisätutkimuksella, kun huomasin hinnastoja käsittelevän logiikan tekevän epäjohdonmukaisia ja kömpelöitä operaatioita ennen tietokantaan tallentamista. Kollegamme kanssa tätä tutkiessamme huomasimme, että osa tiedon validoinnista tapahtuu kummallisessa paikassa. Epäjohdonmukainen logiikka viittaa järjestelmässämme yleensä siihen aikakauteen, kun osa logiikasta oli pakon edessä siirrettävä Oldiesta Newbieen uuden järjestelmän käyttöönoton yhteydessä. Tällöin työskennellään *perintökoodin*, eli vanhentuneiden järjestelmäratkaisujen kanssa. Yleisin tapa korvata perintökoodia on refaktoroida vanhentuneet ratkaisut uusilla ja tuntui luonnolliselta ratkaisulta tähänkin tehtävään.

Otin käyttöön Martinin (2009, 14; 138–139) mainitsemat hyvän ohjelmoinnin kaksi käytäntöä:

1. *Jätä leirintäalue aina siistimpään kuntoon, kuin missä löysit sen.*

   Optimoin tehtäviä tietokantakutsuja, sekä refaktoroin järjestelmän osat loogiseen järjestykseen niin, ettei operaatioita tehdä useampaa kertaa turhaan. Lisäsin järjestelmään myös ehtoja, joiden mukaisesti osa sen hetkisessä tilanteessa tarpeettomista operaatioista voidaan jättää tekemättä. Muutin myös osan tietokantakutsuista liiketoimintalogiikassa tehtyihin tarkistuksiin, jolloin turhat tietokantakutsut vähenevät.

2. *Luokalla pitäisi olla vain yksi vastuu – yksi syy muuttua.*

   Saadun tiedon validointi kuuluu luokalle, jonka vastuulla on tarkistaa saadun tiedon pätevyys. Osa tiedon pätevyyden tarkistuksesta tapahtui alun perin keskellä liiketoimintalogiikkaa, vaikka järjestelmästämme yleisin tapa validointiin on sille erikseen rakennettu validaattori. Siirsin validointivastuun kokonaisuudessaan sille tarkoitettuun luokkaan.

Kävin päivän aikana keskustelua hinnasto-ominaisuuden tuotelistauksiin liittyvistä muutoksista, jotka käyttöliittymäohjelmoijamme nosti esiin. Tuotelistaus kaipaa lisää muutoksia, joten kävimme uudet vaatimukset läpi ja otin muutoksien alemman logiikan rakentamisen vastuulleni. Sain mielestäni päivän aikana paljon aikaiseksi, kun alkuperäisen tehtävän lisäksi löysin ja kehitin järjestelmästä löytämiäni epäkohtia.

Keskiviikko 31.3.2021

Käytin aamupäivän hinnaston lisäämiseen liittyvien muutoksien manuaaliseen testaamiseen. Vaikka tein mielestäni kattavat yksikkötestit, on kehityksessä hyvä tähdätä siihen, ettei laadunvarmistus löydä mitään (Martin 2011, 114). Omassa työskentelyssäni olen myös huomannut, että kehittäminen hidastuu huomattavasti, mikäli poikkeukselliset käyttäjätarinat jäävät yksinään laadunvarmistuksen vastuulle. Laadunvarmistus joutuu ensin tutkimaan, johtuuko testien poikkeukset testiympäristöstä tai viallisesta testistä, vai onko kyse yksinkertaisesti muutosten aiheuttamasta ongelmasta. Sen jälkeen muutokset palautuvat kehittäjälle, joka ongelmien muuttamisen jälkeen käy vielä läpi yksikkötestit, pyytää hyväksynnät muilta ohjelmoijilta ja lähettää sitten muutokset takaisin laadunvarmistukseen. On siis huomattavasti nopeampaa ottaa reunatapaukset huomioon jo mahdollisesti yksikkötesteissä, sekä mikäli mahdollista, varmistaa käyttäjätarinan vaatimusten täyttyminen manuaalisesti.

Hinnastomuutoksien lisäksi pidimme mentoriryhmäni kanssa palaverin, jonka yhteydessä mentorimme antoi tehtäväksemme suunnitella jonkinlainen koulutus, jonka voisimme pitää yrityksen kesätyöntekijöille. Sovimme etukäteen työpajalle aiheen ja suunnittelimme

kutsut, jotka sitten lähetimme yrityksen kehitystiimeille. Aiheeksi valitsimme testilähtöisen ohjelmistokehittämisen ja keskustelimme alustavasti siitä, miten haluamme työpajan toteuttaa. Vaikka oikean työpajan pitäminen alan ammattilaisille jännittää, olen silti innoissani siitä, mitä kaikkea tulen oppimaan työpajan järjestämiseen liittyen.

Torstai 1.4.2021

Eilen juuri ennen työpäivän loppua sain laadunvarmistajalta viestin porsaan reiästä hinnastomuutoksessani. Oldiesta peräisin oleva kummallinen logiikka, jonka uudistin hinnastotoimintoa rakentaessani, oli aiheuttanut sivuoireen. Ongelmaksi muodostui se, ettei yksi hinnaston tallentamiseen käytetty reitti toiminutkaan enää loogisesti. Pyrin testaamaan muokkaukseni manuaalisesti mahdollisimman perinpohjaisesti, mutta tällaisissa tapauksissa huomaan, kuinka paljon opittavaa minulla vielä on tuotteestamme.

Laadunvarmistajamme on työskennellyt tuotteen parissa vuosia ja osaa ottaa huomioon sen, mitä vanhassa järjestelmässä on mahdollista tehdä. Hän osaa näin olettaa samaisen toiminnon tarjoavan samat toiminnallisuudet myös uudessa järjestelmässä. Kyseisessä tapauksessa tallennus tapahtui sellaisen REST-kutsun kautta, jota en ollut ottanut huomioon testatessani ratkaisua. Tällaiset tilanteet muistuttavat minulle, kuinka tarkasti järjestelmää pitää tutkia, ennen kuin voi todeta uusien muokkauksien toimivan saumattomasti olemassa olevien vaatimusten kanssa.

Hinnastomuutoksien valmistuessa huomasin olevani vailla uutta tekemistä. Pyysin päästä seuraamaan kollegani työskentelyä. Kyseessä oli hinnaston tuotelistaukseen liittyvä tehtävä ja vaati samaisen monimutkaisen tietokantakyselyn muokkausta, jota olin itsekin käsitellyt samaiseen toimintoon liittyen. Koska pääsiäinen pyhineen lyhentää sprinttiämme, eikä *tehtävälokillamme* ollut minulle konkreettista tekemistä, päätin aloittaa pääsiäislomani lounaan jälkeen.

Viikkoanalyysi

Teknisinä oppeina tällä viikolla nousi järjestelmässä tarvittavien tietokantakyselyiden optimointi, .NET-rungolla rakennetun järjestelmän muuttujien tyhjennettävyysmahdollisuuden poistaminen, jonka lisäksi jäin tällä viikolla pohtimaan perintökoodin käsittelyä. Muuhun työskentelyyni liittyen huomasin uusia hinnasto-ominaisuuden vaatimuksia määritellessämme ottavani aiempaa itsevarmemmin vastaan uusia tehtäviä. Huomio ilahdutti, sillä koen työssäni suhteellisen paljon itse itselleni asettamaa suorituspainetta.

Uutta hinnastoa lisätessä halusimme järjestelmän tarkistavan, mikäli kannasta löytyy jo olemassa oleva oletushinnasto uuden hinnaston mukaiselle valuutalle. Samanlainen tarkistus oli tehty jo perintökoodissa, mutta ratkaisua oli mahdollista optimoida huomattavasti. Perintökoodi kutsuu tietokannasta kaikki organisaatiolla määritellyt hinnastot annetun valuutan perusteella. Kysely käännetään sitten järjestelmän ymmärtämiksi tietotyypeiksi ja tuloksena saadun listan pituus lasketaan. Mikäli listan laskutoimitus palautuu takaisin nollana, voidaan tarvittavat jatkotoimenpiteet tehdä.

Huomattavasti optimaalisempi tapa on päätellä jo tietokannassa, löytääkö kysely rivejä. Sen lisäksi löytyneetkin rivit on turha laskea, kun ehdon täyttymiseen riittää yhdenkin rivin löytäminen. .NET-rungossa tarkistusmetodi Any() on laskumetodia Count() nopeampi, koska tarkistukselle riittää, että listasta löydetään ensimmäinen alkio. Laskumetodi taas laskee koko listan, joka voi sisältää miljoona alkiota. Samalla tavalla kannassa on nopeampaa tarkistaa kyseisenlainen tilanne EXISTS tarkistuksella, kuin laskea rivit COUNT komennolla. Tarkistuksen tekeminen tietokannassa nopeuttaa operaatiota myös sen takia, että sillä vältetään saadun tuloksen kääntäminen tietokannan vastauksesta järjestelmän ymmärtämäksi objektiksi ja tulos voidaan palauttaa pelkkänä bittinä. (Bertucci ym. 2012, luku 45.2.2; Vadgama 13.1.2021.)

Mielenkiintoisena uutena asiana tällä viikolla oli myös .NET-rungon uuden version tarjoama mahdollisuus poistaa *viitetyyppisten* muuttujien tyhjennettävyys. Tyhjennettävyyden poistamisella voidaan varmistaa, ettei tyhjiä tietotyyppejä voida käyttää. Kyseinen .NET projektiin lisättävä attribuutti tuli esiin yhden mentoroitavan osallistujan alennuslihasovelluksesta. .NET-rungossa viitetyyppien tyhjennettävyys on oletuksena mahdollista, mutta tyhjennettävyyden estäminen poistaa esimerkiksi tarpeen tarkistaa tyhjiä palautuksia pitkin koodia. Tyhjennettävyyden poiston etuna on esimerkiksi se, että metodi palauttaa aina loppuun asti käsitellyn objektin tyhjän sijasta. Lisäksi järjestelmässä voidaan tällöin luottaa siihen, että tiedon puuttuminen käsitellään aina heti. (Bugayenko 13.5.2014; Microsoft 2021g 2020.)

Omissa tuotteissamme emme vielä käytä .NET-rungosta versiota, joka tukisi tyhjennettävyyden poistamista, sen hyödyistä huolimatta. Uskon tyhjennettävyyseston kannustavan kuitenkin laadukkaaseen ohjelmointiin, varsinkin kun kyseessä on iso kehitystiimi ja sitäkin isompi järjestelmä. Mikäli metodit eivät voi palauttaa tyhjää, on niiden rakenne pakko ajatella tarkemmin. Toisaalta isossa järjestelmässä, jossa käsitellään paljon asiakkaiden antamaa dataa ei ikinä voida luottaa siihen, että mikäli jotain pyydetään, jotain myös saadaan. Olen ymmärtänyt tämänhetkisessä työssäni, että juuri

tämä on tärkeä muistaa erityisesti web-sovellusta kehitettäessä. (Howard & LeBlanc 2005, Never Trust User Input.)

Tällä viikolla nousi taas esiin se, kuinka perintökoodi hidastaa uuden luontia ja tuo haastetta kehitystyöhöni. Pääkkö kirjoittaa perintökoodista artikkelissaan ”Kaikki koodi on jossain vaiheessa legacya”. Pääkkö kuvailee, kuinka koodi voidaan määritellä perinnöksi, ”kun siitä tulee pullonkaula kehityksen suhteen. Tämä voi ilmetä skaalautuvuuden suhteen, laajennettavuuden suhteen tai vaikkapa ymmärrettävyyden suhteen.”. Hyvä huomio kirjoituksessa on mielestäni juuri se, että kaikki koodi vanhentuu väistämättä jossain vaiheessa. Vanhentuminen ilmenee omassa tuotteessamme juuri Any() ja Count() tyylisten metodien käyttöeroina. Silloin, kun alkuperäinen oletushinnaston haku on oletettavasti kirjoitettu, Any() metodi ei ole ollut käytettävissä. (Microsoft 2021h; Pääkkö, 23.10.2018.)

Pääkkö (23.10.2018) luettelee perintökoodin huonoiksi puoliksi järjestelmän luettavuuden, pirstaleisen toiminnan, sekä dokumentaation ja testien puutteen. Vanha järjestelmämme täyttää kaikki Pääkön esiin nostamat ongelmat. Martin (2009, 53–55; 175) mainitsee tämän kaltaisten ongelmien kantavan teknistä velkaa. Kuten Pääkkökin kirjoittaa, ei perintökoodi kuitenkaan tarkoita suoriltaan, että koodi on ollut tekohetkellä huonoa vaan pikemminkin sitä, miten vaatimukset ja järjestelmä sen ympärillä on muuttunut. Pääkkö kehottaa lukijaa kirjoittamaan kommentteja järjestelmän rajatapauksien käsittelyyn, kun taas Martin muistuttaa kommenttien olevan alttiita vanhentumiselle.

Tästä päättelen, että kommentoinnilla voidaan mahdollisesti merkitä vanhentumiselle alttiita rajatapauksia silloin, kun metodin uudelleenjärjestely, -nimeäminen tai luotto sen luettavuuteen ei riitä. Tällainen on mielestäni tärkeä muistaa erityisesti, kun työskennellään jatkuvalla tahdilla julkaistavan tuotteen kanssa ja asiakkaat kokevat nopeasti tehtyjen muutoksien vaikutukset.

3.7 Seurantaviikko 7

Tiistai 6.4.2021

Pääsiäislomalta paluu alkoi tiimissämme suoraan sprintin päättämisellä. Aamun sprinttikatselmuksessa lähitiimeillämme oli yllättävän vähän esitettävää, jonka totesimme johtuvan mitä luultavimmin pääsiäisen lyhentämästä sprintistä. Tuttuun tapaan käyttöliittymäohjelmoijamme esitteli kaiken, mitä lähitiimilläni oli näytettävää. Koen katselmukset usein vaikeaksi paikaksi nostaa esiin omia aikaansaannoksiani. Uskon itsevarmuuteni teknisestä näkökulmasta uupuvan vielä sen verran, että koko tiimin kesken pidetyssä palaverissa en ole varma, milloin aikaansaamani muutokset ovat kiinnostavia muulle tiimille.

Koska muut tiimimme sprintin päätökseen liittyvät palaverit oli siirretty seuraavalle päivälle, käytin loppuosan päivästäni uuden laadunvarmistajamme kanssa keskustellen eräästä muokkauspyynnöstäni. Kehitin myös eteenpäin mentorointiohjelman alennuslihasovellusta, jonka olin aloittanut alusta sitten viime kerran. Koska tulemme toisten mentoroitavien kanssa käsittelemään testilähtöistä ohjelmistokehitystä työpajassamme, olemme sopineet tuottavamme uuden version tehtävästä testilähtöisesti. Koin testilähtöisyyden yllättävän hankalaksi aloittaa uutta projektia, enkä päässyt tehtävässä kovin pitkälle.

Keskiviikko 7.4.2021

En asettanut suuria tavoitteita päivälleni, sillä yli puolet päivästäni oli etukäteen varattu sprintin retrospektiiville, sekä seuraavan sprintin suunnittelulle. Retrospektiivissä tiimillämme oli enimmäkseen hyvää sanottavaa sprintistä ja ainut esiin noussut kritiikki oli tehtävien loppuminen ennen sprintin päätöstä. Sprinttiin valittujen tehtävien keston ja kompleksisuuden yliarviointi johti tässä sprintissä siihen, että loppusprintistä osa tiimistä jäi vaille tähdellistä tekemistä. Tehtävien aliarviointi on kuitenkin yliarviointia ongelmallisempaa, kun yliarvioidussa sprintissä mahdollistetaan yhä ketterä reagointi kiireellisiin, yllättäviin tehtäviin (Dalmjin 7.10.2019). Koska kyseinen sprintti toteutettiin ensimmäistä kertaa näin suunnitelmallisesti, on tehtävälokin suunnittelemisessa ymmärrettävästi vielä opittavaa. Otimme keskustelun huomioon suunnitellessamme seuraavaa sprinttiä ja lisäsimme tehtävälokiimme muutaman ylimääräisen, mutta pieneltä vaikuttavan tehtävän.

Uuden lokin mukaan tulen aloittamaan sprinttini uuden integraation asetuksiin liittyen. Sovimme käyttöliittymäohjelmoijan kanssa seuraavaksi aamuksi palaverin, jossa irrotamme prototyypin mukaisesti kokonaisuuden pienemmiksi tehtäviksi. Tulemme samalla käymään läpi esimerkiksi, minkälaisia reittejä ja minkälaista REST-*mallia* aiomme käyttää asetuksien siirtämiseen käyttöliittymän ja tietokannan välillä. Tutustuin myös integraation dokumentaatioon ja koska sama integraatio on olemassa Oldien puolella, kokeilin miten asetukset ovat toimineet alun perin. Lisäksi käytin loppupäivästä hetken aikaa opiskellessani muutamaa mentorini esiin nostamaa konseptia, joista tulemme keskustelemaan tapaamisessamme perjantaina.

Torstai 8.4.2021

Aloitin päiväni sisartuotteemme - **Invoicy** - asetuksiin liittyvän tehtäväkokonaisuuden pilkkomisella. Invoicy on verkkolaskutukseen painottunut ohjelma ja integraation avulla asiakas voi lähettää järjestelmästämme laskunsa Invoicyyn, joka puolestaan automatisoi näin järjestelmästämme saapuvan laskutuksen. Tavoitteenamme oli käydä läpi asetuksien toiminnallisuuksista luotu prototyyppi ja tarkentaa tehtäväkokonaisuutta. Lisäksi tällaisella palaverilla on usein myös tärkeä asema alkuperäisen suunnitelman tarkentamisella.

Vaikka Oldie tarjoaa hyvän esimerkin monille Newbieen lisättävien toiminnallisuuksien kannalta, suunnitellaan Newbien toiminnallisuus niin käyttökokemuksen, kuin implementaation kannalta uudestaan. Usein toiminnallisuudesta myös poistetaan käyttämättömiä tai vanhentuneita osia ja lisätään parannuksia. Käyttökokemuksen, sekä toiminnon suunnittelu etukäteen on Oldien esimerkkiä noudattaenkin hankalaa etukäteen (Schmidt 2015, 10–11), minkä takia toiminnon siirtyessä kehitykseen protosta herää usein tarkentavia kysymyksiä. Tällä kertaa käyttöliittymässä oli suunniteltu näytettävän eräs merkityksetön tieto, jota ei säilytetä omassa järjestelmässämme ja se pitäisi näin hakea joka kerta Invoicyltä erikseen. Tuoteomistajan luvalla poistimme kyseisen vaatimuksen rajauksestamme.

Palaverissa pilkkomisen jälkeen päätin aloittaa integraatioasetuksien ensimmäisen osan. Koska integraatio on Newbien puolella täysin uusi kokonaisuus ja kollegallani oli vielä kesken siihen liittyvän projektipohjan rakentaminen, päädyin kuitenkin enimmäkseen suunnittelemaan tulevia lisäyksiä järjestelmään. Olin silti tyytyväinen siihen, miten saimme pilkottua tehtävät, jonka lisäksi pystyin tekemään pieniä, projektipohjasta riippumattomia muutoksia integraatioon liittyen.

Perjantai 9.4.2021

Asetin päivän tavoitteeksi aloittaa integraatioasetuksien rakentamisen. Kyseessä on asetukset, jotka on tallennettu yhtenä xml-dokumenttina tietokantaamme. Lisäksi asetuksia säilyttävään tietokantakolumniin on tallennettu muutakin, kuin kyseiseen toimintoon käytettyä tietoa ja asettaa näin haastetta asetuksien käsittelemiselle. Tämä toiminnallisuus vaatii siis odotettua monimutkaisempaa käsittelyä ohjaimesta tietokantakyselyyn. Päädyin käyttämään suuren osan aamupäivästäni käymällä läpi tehtäväni asettamia rajauksia ja tutkimalla, miten kyseisentyylisiä ratkaisuja on tehty muualla järjestelmässämme.

Integraatiotehtävän lisäksi tapasimme mentoriryhmäni kanssa. Ennen tapaamista meidän tuli tutustua fitness-funktioksi kutsuttuun konseptiin, sekä siihen miten IoC-periaatetta on hyödynnetty omissa tiimeissämme. Fitness-funktio voi olla järjestelmään implementoitu tai manuaalinen toimenpide, jolla voidaan yksikkötestin tavoin testata järjestelmän käyttäytymistä. Yksikkötesteistä poiketen fitness-funktion tarkoitus on kuitenkin ensisijaisesti mitata sille osoitetun järjestelmän toimintaa ja laskea, täyttääkö se sille asetetut vaatimukset. Tavoitteeni oli ymmärtää konseptia sen verran, että mentoriohjelmassa totuttuun tapaan pystyn selittämään muulle ryhmälle sen perusteet. Kävimme mielenkiintoista keskustelua aiheesta muun ryhmän kanssa ja olin tyytyväinen omaan puheenvuorooni aiheesta. Lisäksi esittelin järjestelmämme IoC-rakennetta, jonka totesimme ryhmäni kanssa olevan automatiikkansa puolesta edistyksellinen. Työpäiväni päättyi hilpeissä merkeissä organisaation järjestämässä Stay at home -juhlassa. (Ford, Kua & Parsons 2017, luku 2.)

Viikkoanalyysi

Uutena konseptina tällä viikolla löysin fitness- funktiot. Projektityöskentelyn puolesta taas pohdin tällä viikolla paljon yhteisiä käytäntöjä ja sitä, miten ne dokumentoimattomanakin siirtyvät tiimissä kokeneilta jäseniltä uudemmille.

Tällä viikolla tiimimme projektityöskentelyssä nousi esiin, kuinka paljon kommunikoinnin hallintaa ohjelmistokehitystiimissä tarvitaan. Olen useasti törmännyt tilanteeseen, jossa pyydän neuvoa rutiininomaiseen tehtävään, jonka jokainen tiimiläiseni osaa suorittaa, muttei siitä löydy dokumentoitua ohjeistusta. Ohjeistus yleistyneestä käytännöstä tehtävä vaihtelee sen perusteella, keneltä kysyn neuvoa, koska sovittua käytäntöä ei ole. Yleensä ensimmäinen keneltä kysyn ei osaa auttaa, mutta tietää keneltä ohjeistuksen voisi mahdollisesti saada.

Kyseessä onkin juuri kollegoideni kokemuksen tuomista opeista ja ymmärryksestä. Tällöin voidaan puhua hiljaisesta tiedosta, jonka katsotaan olevan tiimin jäsenien yksilöllistä osaamista organisaation hyvistä käytännöistä, normeista ja prosesseista. Hiljaista tietoa voi olla vaikea tunnistaa, kopioida sekä jakaa organisaatiossa ja se asettaa näin haasteita kollektiiviselle ymmärrykselle. Hiljaisen tiedon isoin ongelma on organisaation muistinmenetys, eli tilanne, jossa tärkeää tietoa menetetään työntekijöiden vaihtuessa. Lisäksi varsinkin ohjelmistokehityksessä tapahtuva jatkuva muutos vaikeuttaa tiedon jakamista, kun dokumentoitu tieto vanhenee nopeasti. (Davičienė & Raudeliuniene 2010, 822–824; Jubi & Sankar 2015, 8.)

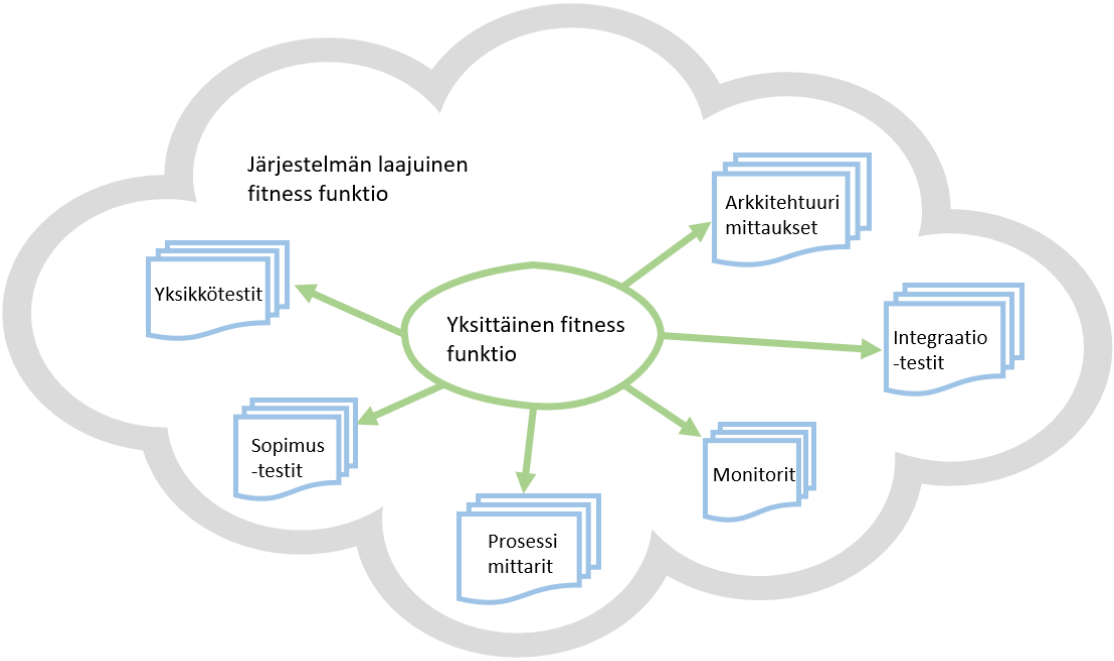
Iso osa ketterää projektityöskentelyä on tiedonjakamiseen ja kollektiivisen ymmärryksen ylläpitämiseen liittyvä kommunikointi. Työskentelymme on tällä hetkellä isossa murroksessa ja uusien työtapojen opettelu tiimin sisäiseen kommunikointiin olisi nyt otollista. Jubi ja Sankar (2015, 9) listaavat tapoja muuttaa hiljainen tieto konkreettisiksi käytännöiksi ketterässä työskentelyssä:

- Scrum-palavereissa jaetut vinkit ja kokemukset
- Hyvin dokumentoidut aivoriihet
- Sprinttikatselmuksien jälkeiset palautekeskustelut
- Quality Circle (ks. Luku 1.2 Käsitteet)
- CoP (ks. Luku 1.2 Käsitteet)
- Junior-tason tekijän yhteistyö kokeneen kollegan kanssa

Yhteisen ymmärryksen käsittelyyn löytyy siis useita työkaluja, joita voitaisiin tiimissämme hyödyntää. Omaan työskentelyyni koen hiljaisen tiedon kannalta olevan tärkeää ohjelmoinnissa, avoimien keskustelujen käynnissä, sekä muutoksieni dokumentoinnissa. Koska kommentit tapaavat vanhentua järjestelmässä (Martin 2009, 53–55), on ohjelmoinnissa hyvä kirjoittaa itse metodi mahdollisimman yksiselitteiseksi. Aiempaa avoimempi keskustelu on koko tiimin yhteisissä palavereissa haastavaa ja vaatii itsevarmuutta, mutta voi parhaimmillaan hyödyttää koko tiimiä. Omien muutoksieni dokumentointia harrastan nyt jo mahdollisimman paljon kirjoittamalla ylös tehtäväjärjestelmäämme kattavat kuvaukset kaikesta tekemästäni, mutta dokumentointiin voisi etsiä muitakin työkaluja. Uudet työtavat voisivat hyödyttää koko tiimiä, jolloin muu hiljainen tieto saataisiin dokumentoitua paremmin.

Tällä viikolla uutena konseptina noussut fitness-funktio tuntui aluksi hyvin abstraktilta käsitteeltä. Tätä saattaa selittää se, että fitness-funktio on enemmänkin konsepti kuin

kiveen hakattu metodi (Ford ym. 2017. luku 2). Fitness-funktio voidaan myös osoittaa testaamaan kuvan 5 tavoin järjestelmää sekä koko sen laajuudelta, että sen yksittäistä osaa. Rinnastankin fitness-funktion peliohjelmoinnista tuttuun näennäiseen tekoälyyn, jolloin tapahtumiin soveltuva reaktio lasketaan ennalta määritellyin säännöin (Buckland 2005, Introduction).



Kuva 5. Järjestelmän laajuinen fitness-funktio verrattuna yksittäiseen fintess funktioon, kuva muunneltu kirjassa ”Building Evolutionary Architect” esitetystä kaaviosta (Ford ym. 2017, luku 2).

Funktio voidaan mentorini mukaan toteuttaa myös manuaalisena tarkistuksena, eli projektityöskentelyn osana ja näin rinnastaa jonkinlaiseksi DoD-määrittelyksi. Mentorimme kertoi itse käyttäneensä funktiota rajoittamaan erään luomansa *rajapinnan* käyttöä. Rajapinnan tarkoituksena oli toimia mukana *Sisustaja*-nimisessä ohjelmointimallissa, jossa luokalle voidaan lisätä vastuita dynaamisesti ja joustavasti (Bates ym. 2004, 92–93). Mentorini kertoi joutuvansa usein perustelemaan kollegoilleen, että fitness-funktio varmistaa Sisustaja-mallin käytännöllisyyden estämällä uusien metodien ja toiminnallisuuksien lisäämisen. Tällainen rajaus varmistaa näin rajapinnan alkuperäisen tarkoituksen ylläpidon ja estää sen väärinkäyttämisen. Käytännön esimerkki helpotti minua ymmärtämään fitness-funktion sitä, miten fitness-funktio eroaa yksikkötesteistä.

Yksikkötestit ovat nimensä mukaisesti tarkoitettu yksikön testaamiseen, kun taas fitness-funktiolla voidaan testata laajempia kokonaisuuksia (Ford ym. 2017, luku 2).

Huomasin, ettemme omassa järjestelmässämme hyödynnä tällä hetkellä fitness-funktioita ja pääsin pohtimaan, mihin voisimme sitä käyttää. Hyvä käyttökohde funktiolle voisi olla liiketoimintalogiikassa kulkevan prosessin aikavaativuuden mittaaminen. Järjestelmästämme löytyy sekä perittyä että huolimattomalta ohjelmoinnilta näyttävää logiikkaa, jossa algoritmin aikavaativuus ja monimutkaisuus on moninkertainen sen syötteeseen verrattuna. Tällaisen funktion kirjoittaminen olisi potentiaalinen haaste minulle niin ohjelmoinnin näkökulmasta kuin itsevarmuudenkin kannalta. Kyseisenlainen kehityskohde vaatisi tiimin hyväksynnän ja tarkoittaa näin ollen sitä, että minun tulee ensin esittää asia tiimille ja saada sille kannatusta.

3.8 Seurantaviikko 8

Maanantai 12.4.2021

Viikkoni alkoi suoraan Invoicyn asetuksien rakentamisella Newbieen. Päivän tavoitteena oli suunnitella alempi logiikka, sekä tietokantakysely hakemaan oikeat asetukset tietokannasta. Ongelma tietokantakyselyn muodostamisessa on, että kyseiset asetukset on tallennettu xml-muotoisena. Xml-muotoista kolumnia ei Microsoft SQL Serverissä voida verrata valmiilla metodeilla (Microsoft 2021i), minkä takia taulujen yhdistäminen JOIN-metodilla ja duplikaattien poistaminen löydetyistä riveistä DISTINCT- tai GROUP BY -metodeilla ei ollut mahdollista (Thakur 17.11.2017). Kollegani neuvojen pohjalta lähestyin ongelmaa uudelta kantilta hakemalla asetuksia muiden yhdistävien taulujen kautta. IN- ja UNION-metodeilla sain yhdistettyä kyselyyn tarvittavat taulut ja haettua vain olemassa olevat asetuksia sisältävät kolumnit (W3Schools 2021b; W3Schools 2021c).

Tietokantakyselyn jälkeen jäin pohtimaan, miten xml-muotoisten asetuksien asettaminen liiketoimintalogiikan ymmärtämään muotoon olisi järkevintä. Aiemmassa samankaltaisessa tilanteessa lisäsin järjestelmään *kääreluokan*, joka vastasi xml-asetuksien sisältämät tiedot. Tämä ratkaisu helpotti huomattavasti tietojen käsittelyä käyttöliittymän ja tietokannan välillä. Samainen ratkaisu yksinkertaisti huomattavasti myös käyttöliittymästä saatujen tietojen validointia sekä asetuksien yleistä käsittelyä. Useiden xml-dokumentin elementtien lukeminen ja kirjoittaminen tietotyypistä toiseen myös yksinkertaistui huomattavasti. Lisäksi ohjelmointi on helpompaa, kun kääreluokan kautta asetuksia voi käsitellä C#:in ymmärtäminä tietotyyppeinä xml-elementtien sijaan.

Kirjoitin kääreluokan ja tajusin tarvitsevani uusimman tietokantakyselyn lisäksi muutakin tietoa kannasta. Koin loppupäivästä vaikeaksi keskittyä ja hahmottaa, miten tietoja olisi paras hankkia ja säilyttää järjestelmässä. Asetin huomisen tavoitteeksi rakentaa tarvittavat tietokantakyselyt, sekä niitä kutsuvan ja käsittelevän *palvelun* metodit.

Tiistai 13.4.2021

Tänään ymmärsin paremmin tiimimme tavoitteiden vaikutuksia organisaatiotasolla. Kävimme läpi tiimin kesken vuoden ensimmäisen kvartaalin sujumista ja pohdimme, miltä esimiehen asettamat tavoitteet seuraavalle kvartaalille vaikuttavat. Vanhan käyttöliittymän alas ajo oli tiedettävästi epärealistinen tavoite jo viime kvartaalissa siihen verraten, kuinka keskeneräinen uusi käyttöliittymä on. Kyseistä tavoitetta lukuun ottamatta olimme kuitenkin saavuttaneet kvartaalin päämäärät. Esimiehemme kävi myös läpi tiimimme

strategiasuunnitelman, jonka tarkoituksena on auttaa tiimiä hahmottamaan, missä tilanteessa tiimi on nyt ja mihin ollaan pyrkimässä (IfM 10.2.2018, 0:10-0:18). Lisäksi esimiehemme hahmotteli tuotteemme kehityksen *ajurit*, joiden perusteella tuotteemme tärkeimmät vaatimukset on määritelty (Croft 2.6.2014, 0:06-1:05).

Tällainen palaveri auttaa hahmottamaan koko tiiminlaajuista kehitystyötä ja verrata omaa työskentelyä suhteessa tuotteemme tavoitteisiin. Nämä palaverit ovat myös harvoja kertoja, kun kaikkia tiimin jäseniä osallistetaan aktiivisesti keskustelemaan lähitiimin ulkopuolella. Palaverin lisäksi edistin integraatioasetuksiin liittyvää tehtävääni, sekä kävin mentoriryhmän kanssa keskustelua ohjelmistoarkkitehtuurista testilähtöisen kehityksen näkökulmista. Vaikken kirjoittanut montaa riviä koodia, koin kaikkiaan olleeni suhteellisen tuottava tänään.

Keskiviikko 14.4.2021

Halusin saada tämän päivän aikana valmiiksi integraatioasetuksiin liittyvän tehtävän, ensimmäisen GET-reitin rakentamisen. Olin ehtinyt jo maanantaina rakentaa ohjaimen, mallin ja sitä käsittelevän luokan, sekä tarvittavien tietojen hakemiseen liittyneet tietokantakyselyt. Ehdin aamupäivästä kirjoittamaan lisäyksilleni vielä yksikkötestit, jonka jälkeen hain uusimmat muutokset *pääratkaisustamme*.

Pääratkaisuumme on tämän viikon aikana ehtinyt tulemaan paljon muutoksia, jonka lisäksi oma versioni on perujaan kollegani vielä pääratkaisuun yhdistämättömästä versiosta. Koska sekä perimäni versio, että pääratkaisu sisälsi paljon omasta versiostani poikkeavia muutoksia, jouduin selvittelemään versioiden välisiä ristiriitoja hakiessani uusimmat muutokset omaan versiooni. Ristiriidat aiheuttavat itselleni ajoittain turhautumista, mikäli ne kohdistuvat esimerkiksi projektitiedostojen konfigurointiasetuksiin. Tällaiset tiedostot ja niiden muokkaaminen on minulle vielä hieman uutta ja sotken välillä muutokseni yhdistäessäni versioita. Aiemmista virheistä oppineena otin nyt kopion ehjästä, yhdistämättömästä versiosta ennen ristiriitojen selvittämistä. Näin olisin pystynyt yrittämään yhdistämistä helposti uudestaan, mikäli ensimmäinen versioni olisi mennyt rikki.

GET-reitin manuaalisen testaamisen jälkeen totesin muokkausteni toimivan halutulla tavalla. En kuitenkaan voinut vielä julkaista muokkauspyyntöä, koska muokkaukseni riippuvat kollegani versiosta. Sen sijaan tutustuin esimiehemme kehotteesta Zero-Bugs -konseptiin, jossa jatkuvan integraation ympäristössä järjestelmävirheiden korjaaminen asetetaan aina uuden tuotannon edelle. Konseptissa tarkoitus on sekä tehokkaasti hallita

virheitä että päästä eroon vanhentuneista virheilmoituksista. Konsepti oli mielestäni erittäin mielenkiintoinen ja päätin tutkia asiaa tarkemmin omalla ajallani. (Hilton 5.1.2021.)

Torstai 15.4.2021

Aamuni koostui pitkälti koko organisaation laajuisessa vuoden ensimmäisen kvartaalin retrospektiivissä. Palaverissa organisaation esimiehet esittävät tiimiensä tulevan kvartaalin tavoitteet, edellisessä kvartaalissa saavutetut tulokset sekä löydetyt kehittämiskohteet. Tällainen palaveri auttaa minua hahmottamaan yrityksemme suuntaa isommassa mittakaavassa.

Osallistuin kuitenkin normaalisti aamuiseen Scrum-palaveriimme. Olimme aiemmin mentorini kanssa sopineet, että selvitämme mikäli fitness-funktioita käytetään omissa tiimeissämme. Lisäksi saimme tehtäväksemme esitellä fitness-funktio tiimillemme ja näin syventää omaa ymmärrystämme aiheesta. Otin asian esiin aamupalaverissamme ja koin itsevarmuuteni kasvavan päästessäni kertomaan itseäni paljon kokeneemmille kollegoille mistä on kyse. Lisäksi kerroin, kuinka voisimme fitness-funktioiden avulla implementoida koodin kompleksisuuden sekä aikavaativuuden analyysiä ja samalla vahvistaa yhteisesti sovittuja ohjelmoinnin käytäntöjä (Paul & Wang 11.1.2019). Kollegani innostuivat ajatuksesta ja sovimme tutkivamme asiaa lisää, jonka jälkeen jatkoin keskustelua mentorini kanssa sopivien työkalujen löytämiseksi.

Ehdin loppupäivästä vielä aloittaa uuden integraatioasetuksiin liittyvän PATCH-reitin suunnittelun. Kuten aiemmin viikolla totesin, xml-muotoiset asetukset tuovat haastetta tehtävään ja päätin luoda aiemmasta integraatiotehtävästäni tutun kääreluokan arvojen käsittelyn helpottamiseksi. Sain myös periytettyä uuden luokan edelliselle kääreluokalle kirjoittamastani pohjaluokasta, jolloin voin hyödyntää sen metodeja. Asetin seuraavan päivän tavoitteeksi jatkaa mentorointiohjelmasta saamaani uutta tehtäväpatteristoa sekä jatkaa PATCH-reitin rakentamista.

Perjantai 16.4.2021

Saimme aiemmin tällä viikolla mentoriltamme uuden Katan, eli tehtäväpatteriston, jonka tarkoituksena on kehittää ja vahvistaa ohjelmointitaitoja useiden toistojen kautta. Tehtävässä on kyse eräänlaisesta keilausheittoja vastaanottavasta liiketoimintalogiikasta, jonka tulee laskea keilauksien pisteet normaalia monimutkaisemmalla pisteytysmenetelmällä.

Kata-tehtävä on tarkoitus suorittaa aina tunnin sisään uudestaan ja uudestaan, jonka jälkeen omaa kehitystä voidaan tarkastella konkreettisten implementaatioiden välillä. Tehtävä oli odotettua haastavampi, mutta mielekäs. En tosin suoriutunut siitä yhtään niin hyvin, kuin ensivaikutukselta olin kuvitellut. Päätin toistaa harjoituksen vielä viikonlopun aikana ja tällä kertaa sisällyttää harjoitukseen myös testilähtöisen implementaation. (Thomas 2016.)

Kata-tehtävän lisäksi hahmottelin kääreluokkaa integraatioasetuksien PATCH-reittiä varten. Kävin aiheesta myös keskustelua kollegani kanssa, joka työstää omasta ratkaisustani tiukasti riippuvia muutoksia. Koin onnistuneeni työssäni hyvin, kun vertaillessamme versioitamme samojen asetuksien käsittelystä kollegani myönsi ratkaisuni olevan omaansa parempi. Päivän päätteeksi ehdin vielä kirjoittamaan muutamia testejä tekemilleni muokkauksille ja koin saaneeni päivän aikana aikaan suhteellisen paljon.

Viikkoanalyysi

Viikon teknisenä aiheena pohdin syitä xml-muotoisen tiedon säilyttämiseen järjestelmässämme, eli SOAP-protokollaa ja sen eroja nykyaikaisempaan REST API-rakenteeseen. Sen lisäksi pohdin, miten tiimimme projektityöskentely voisi hyötyä esimieheni esiin nostamasta Zero-Bugs-konseptista.

Tietokantaan on tallennettu xml-muotoisia asetuksia alun perin luultavasti sen takia, että Oldien kehittämisen aikaan SOAP-protokolla oli tuotteessamme pääasiallinen API. SOAP, eli Simple Object Access Protocol, tarjoaa REST:in tavoin protokollan jakaa tietoa järjestelmien välillä API-infrastruktuurin kautta. Kuten REST, SOAP on yksinkertainen ja kevyt käyttää, eikä REST:in tavoin ole riippuvainen suorittavan järjestelmän tyypistä tai ohjelmakielestä. REST:istä poiketen SOAP tukee kuitenkin vain xml-muodossa siirrettyä tietoa ja käyttää pyyntöihinsä vain POST-metodia. SOAP on lisäksi tiukasti palvelimeen sidottu protokolla, kun taas REST on arkkitehtuuri ja näin vähemmän sidottu palvelimeen. (Kanjilal 2013, luku 1.1.3; Upwork Staff 19.4.2017.)

REST on tänä päivänä joustavuutensa takia erittäin suosittu pohja API:lle ja SOAP valitaankin nykyään vain sen REST:iä tiukempien turvallisuuskäytäntöjensä takia. SOAP:in luotettavuus johtuu sen tarjoamista turvallisuuteen liittyvistä ominaisuuksista, kuten ACID – *atomisuus*, *johdonmukaisuus, eristäytyminen*, *kestävyys* – sopimusten noudattaminen. SOAP:in kattavampi turvallisuus onkin otollinen yritystason järjestelmille, joissa turvallisuus on järjestelmän tärkein vaatimus. Näin ollen SOAP on ollut luonnollinen

valinta Oldien integraatioiden käsittelyyn järjestelmässämme. (Kanjilal 2013, luku 1.1.3; Upwork Staff 19.4.2017.)

Newbie nojaa täysin REST-infrastruktuuriin tietojen siirrossa, jolloin xml-muotoisen tietokantakolumnin käsittely tuntuu uudessa järjestelmässä kömpelöltä. Newbiessä turvallisuus on järjestetty eri tavalla ja REST-infrastruktuuri mahdollistaa tiedonsiirron joustavasti JSON-objektien avulla. Sen lisäksi REST tarjoaa kaikki CRUD-operaatiot, joka mahdollistaa tietokokonaisuuden yksittäisen jäsenen muokkaamisen. (Kanjilal 2013, luku 1.1.3.)

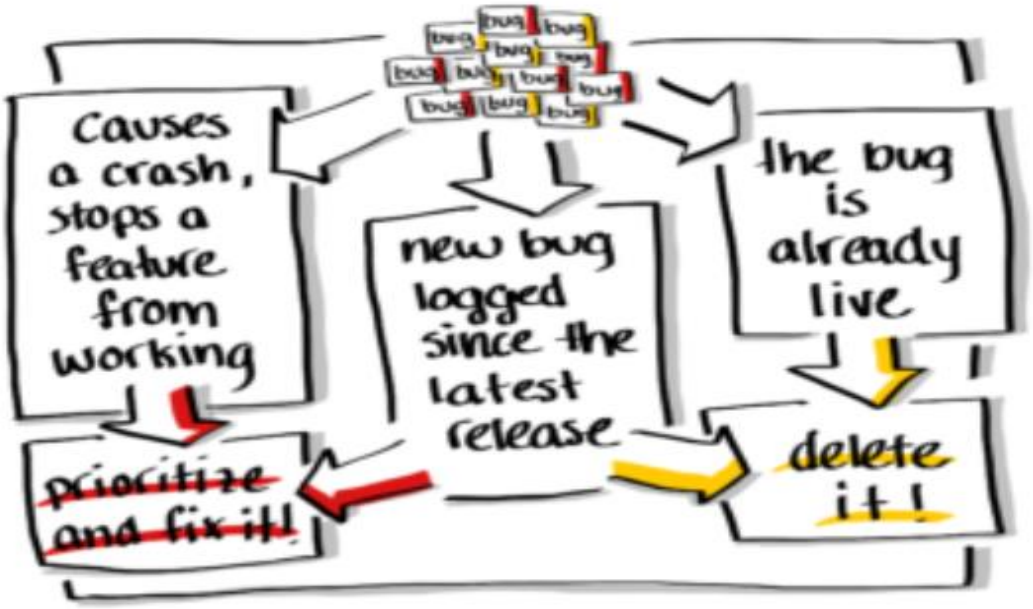
Omissa opinnoissani olen opiskellut SOAP:ia teknologiana hyvin vähän. Tämä luultavasti johtuu siitä, että REST:in julkaisu vuonna 2000 muutti pysyvästi API:en kehittämistä tarjoamalla aiempaa joustavampia tapoja rakentaa web-sovelluksia (Coline 26.1.2017). Vuonna 2021 valmistuvalle ohjelmistokehittäjälle REST-infrastruktuurin ymmärrys on SOAP-protokollan tuntemusta paljon tärkeämpää. Hyvä huomio tässä on kuitenkin se, kuinka jo vuonna 2003 ensimmäisen kerran julkaistu tuotteemme tuo mukanaan riippuvaisuuksia kaikenlaisiin teknologioihin.

Kun palvelujamme on tarjottu aiemmin SOAP:in kautta, on uudessakin tuotteessa tärkeä ottaa huomioon vahvasti siihen luottavat asiakkaat. On kuitenkin ollut opettavaista työskennellä tuotteen kanssa, jossa myös Newbien puolelle on tuotu osa vanhoista teknologioista. Tällaiset riippuvuudet aiheuttavat usein haasteita, mutta samalla opettavat juniortason tekijälle paljon ohjelmistokehityksen historiasta ja periaatteista.

Jatkuvalla integraatiolla tarjoamamme palvelu vaatii myös projektityöskentelyn kannalta tärkeitä uudistuksia. Esimieheni linkittämä artikkeli ”Implement a zero-bug policy” (Hilton, 5.1.2021) summaa, kuinka järjestelmävirheiden kerääntyminen isossa järjestelmässä johtaa epäselviin aikatauluihin ja kehityksessä hukattuihin resursseihin. Konseptissa on tarkoitus priorisoida järjestelmästä tehdyt virheilmoitukset uuden toiminnallisuuden yli ja vähentää näin esimerkiksi keskeneräisten toimintojen julkaisemista. Tarkoitus on myös keventää virheilmoituksien käsittelyä yksinkertaistamalla sitä. Tärkeää on ymmärtää, ettei konseptissa ole tarkoitus korjata jokaista sisään tulevan virheilmoitusta, vaan pikemminkin päättää mikä potentiaalisesti vaikuttaa asiakkaaseen ja on pakko korjata. Virheet, jotka voidaan todeta tarpeeksi pieniksi tai asiakkaan käyttökokemuksen kannalta merkityksettömäksi poistetaan. Tällöin pyritään myös erottamaan, mitkä sisään tulevista virheilmoituksista kuvailevat järjestelmävirheitä, mitkä kehitysehdotuksia ja mitkä puuttuvia toimintoja.

Myös Sundman (5.2.2015) kirjoittaa virheilmoituksien ylläpitämisen haasteista. Sundman kuvaa, kuinka tyypillinen kriittisyysasteikolla virhejaottelu ei välttämättä riitä tehokkaaseen virheidenkäsittelyyn kehitystiimissä. Vaikeasti hallittava virhelista sisältää erilaisia virheilmoituksia, jotka sitten vanhentuvat ennen kuin kukaan ehtii tutkimaan niitä. Sundman ehdottaa Hiltonin (5.1.2021) tavoin suoraviivaista ratkaisua: virheilmoituksia ei enää järjestetä kriittisyyden mukaan ajatuksella, että niihin palata sitten kun aikaa on, vaan virheilmoitus arvioidaan heti sen tullessa. Mikäli virhe luokitellaan käyttäjään negatiivisesti vaikuttavana tekijänä, se sijoitetaan tehtävälistan kärkeen ja korjataan heti. Mikäli virhettä ei koeta näin tärkeäksi, se poistetaan. Näin tiimi ei enää käytä aikaa virheilmoitusten ylläpitoon, eikä tutkimattomat virheilmoitukset jää roikkumaan.

Omassa järjestelmässämme priorisoimme tällä hetkellä virheilmoitukset juuri kriittisyysasteikolla, jossa pienen kriittisyyden virheilmoitukset saattavat vanhentua järjestelmässä puolikin vuotta. Osasyy tähän on varmasti Newbien, sekä Public REST API:mme priorisointi, jolloin vain erittäin kriittisiksi virheiksi asetetut virheet lasketaan sprinttiin mukaan. Tiimissä on pohdittu myös jonkinlaisen virhevastaavan roolia, joka mahdollisesti vaihtuisi jokaisen sprintin ajaksi. Tämä ei ole vielä toistaiseksi kuitenkaan saanut paljoa kannatusta, kun kukaan kehittäjistä ei ole innokas ottamaan roolia vastaan kokonaisen sprintin ajaksi.



Kuva 6. Vanhan virhelistan tyhjennys Zero-Bugs-konseptilla. (Sundman 5.2.2015.)

Siihen taas osasyynä on luultavasti valtava virhelista, jossa virheet saattavat koskea yksittäisen asiakkaan monimutkaista rajatapausta. Tällaisessa tapauksessa usein on kyse

myös tilanteesta, jossa kyseisen virhetilanteen uudelleen luominen ei ole onnistunut testiympäristöissämme ja ongelmaa on miltei mahdoton paikantaa. Vanhan virhelistan käsittelyyn Sundman (5.2.2015) ehdottaa kuvan 6 mukaista ratkaisua. Mikäli virhe on jo julkaistussa versiossa, poistetaan virheilmoitus. Mikäli virhe kaataa järjestelmän tai sen ominaisuuden, se korjataan heti. Samalla uudesta versiosta ilmenevät virheet arvioidaan ja joko priorisoidaan korjattavaksi välittömästi tai poistetaan.

Omaan työskentelyyni kaipaan virheiden korjauksessa vielä itsevarmuutta. Osa virheilmoituksista saattaa koskea minulle vielä tuntematonta järjestelmän osaa, jolloin tehtävä voi olla minulle joko hyvin opettavainen tai mahdoton selvittää itsenäisesti. Tämäkin tietysti riippuu tehtävästä, sillä huomaan joka päivä ymmärtäväni enemmän järjestelmästämme, sen rakentamisessa tehdyistä päätöksistä, sekä siihen sovellettavista ohjelmistokehityksen periaatteista.

3.9 Seurantaviikko 9

Maanantai 19.4.2021

Keskustelin aamupäivästä kollegani kanssa integraatioasetuksien GET-reittiin liittyvien metodien sijainnista, sekä reitin varrella olevien luokkien vastuista. Omassa ratkaisussani ohjain käytti suoraan liiketoimintalogiikassa sijaitsevaa *palveluluokkaa* tarvittavien tietokokonaisuuksien hakemiseen tietokannasta. Ohjain kutsui sen jälkeen *mallikääntäjää*, jonka tarkoituksena on rakentaa REST-kutsusta ulospäin näytettävät tiedot. Olin valinnut kyseisen rakenteen, koska se on yleisesti käytössä tällaisten reittien logiikassa. Kollegani kuitenkin ilmaisi haluavansa kaikenlaisen logiikan käyttäjäoikeuksien todentamista lukuun ottamatta pois ohjaimesta, jolloin ohjain pysyy mahdollisimman kevyenä. Kävimme keskustelua aiheesta ja päädyimme siirtämään metodit integraatioasetuksia varten rakennettuun *auttajaluokkaan* ja näin käsitellä kaikki varsinainen logiikka ohjainta alemmalla tasolla. Vaikka olin itse alun perin päätynyt erilaiseen rakenteeseen perustellusti, koin oppineeni keskustelustamme. Päädyimme mielestäni myös yhdessä parempaan ratkaisuun.

Loppupäivästä pääsin rakentamaan edellisellä viikolla aloittamaani Kata-tehtävää keilauspelin pisteenlaskulogiikasta. Tällä kertaa tarkoituksena oli kehittää ratkaisu mahdollisimman puhtaalla TDD:llä, jonka takia aloitin rakentamalla testiprojektin raamit ja jopa ensimmäisen testin ennen kuin yhtään tuotanto-ohjelmaa oli kirjoitettu. Aikaraja tehtävälle oli tunti, enkä saanut tehtävää rakennettua kokonaiseksi testilähtöisesti. Käydessämme mentoriryhmäni kanssa tehtävää läpi ennen työpäivän loppua ymmärsin kuitenkin osanneeni TDD:n ansiosta lähestyä ongelmaa pienissä paloissa, sen lisäksi että kaikki tekemäni logiikka oli nyt täysin testattu. Näin ollen toimin TDD:n raamien mukaisesti (Martin 2009, 122–125). Koska tehtävä osoittautui mentoriryhmässämme yllättävän haastavaksi, päätimme jatkaa kyseistä Kata-harjoitusta vielä tämän viikon ajan.

Tiistai 20.4.2021

Tänään miltei koko päivä meni sprintin päättämisessä. Uudistettu sprintin päätöksemme tarkoittaa sitä, että aamun ensimmäistä tuntia sekä lounastaukoa lukuun ottamatta sprintin päättävä päivä menee kokonaisuudessaan siihen liittyvissä palavereissa. Oman tiimini aikaan saannokset demosi tällä kertaa yksi Core-kehittäjistämme. Muutos vaikutti lupaavalta, sillä kyseessä olevat muutokset aiheuttivat kysymyksiä, johon normaalisti demomme vetävä käyttöliittymäohjelmoija ei välttämättä olisi osannut vastata. Omalta

osaltani jäin katselmuksessa pitkälti kuuntelijan rooliin, sillä omat muutokseni eivät olleet vielä käyttöliittymän puolesta valmiita.

Sprintin retrospektiivissä saimme aikaan paljon hyvää keskustelua. Kun osa lähitiimini kehittäjistä keskittyy retrospektiivissä usein nostamaan teknisiä aiheita, pyrin itse nostamaan aiheita projekti- ja tiimityöskentelyn näkökulmista. Tällä kertaa nostin esiin hyvinä työskentelytapoinamme aiempaa perusteellisemmin kuvatut muokkauspyynnöt sekä niiden palautteet. Aiemmin tiimissämme on osittain syyllistytty siihen oletukseen, että kehittäjät ymmärtävät pelkistä ohjelmariveistä muokkauspyynnön sisällön ja samalla muokkauspyynnön niukka tai olematon kuvaus vaikeuttaa laadunvarmistuksen työtä.

Kehittämiskohteina toin esiin jo aiemmin mainitun palaveri-isännän puutteen, jonka lisäksi keskustelimme paljon siitä, miten uudistettuun sprinttirytmiin saisi sisällytettyä muutakin kuin uusien ominaisuuksien rakentamista. Järjestelmässämme on paljon parannettavaa, kun esimerkiksi .NET runko halutaan vaihtaa. Tiimimme kävi keskustelua siitä, miten näitä kehittäjien huomaamia ja innovoivia parannuksia saisi tasaisesti mukaan sprintteihin. Päiväni sisälsi paljon hedelmällistä keskustelua ja koin antaneeni paljon sprintin päätökseemme.

Keskiviikko 21.4.2021

Keskustelin tänään kollegani kanssa suunnitelmistamme rakentaa integraatioasetuksien logiikkaa. Kollegani on työskennellyt paljon REST-rajapintamme parissa ja huomasin, kuinka paljon ajatuksemme siitä mikä on tärkeää, poikkesi toisistaan. Kollegani perusteli oman ratkaisunsa jättää kaikki yleensä ohjaimessa tehdyt metodikutsut alemmalle tasolle sillä, että ohjaimen vastuu ei ole ymmärtää mitään logiikasta vaan vain välittää tietoa käyttöliittymän ja mallin välillä. Vaikka perustelu pitää yleisesti paikkansa (CodeAcademy 2021), on ohjain yleensä pitänyt järjestelmässämme huolta mallin ja tietokokonaisuuden välisien tietojen välityksestä. Oma ratkaisuni noudattaakin yleisesti järjestelmässä käytettyä tyyliä vähentäen esimerkiksi erillisiä tietokantakutsuja, kun ohjain pitää huolta oikeiden metodien kutsumisesta oikeilla tiedoilla. Myönnyin rakentamaan ohjaimen kollegani suunnittelemalla tavalla oppiakseni uusia tapoja ratkaista sama ongelma.

Asetustehtävän lisäksi osallistuin palaveriin, jonka olimme integraatiotuotteen kehitystiimin kanssa sopineet. Palaverissa kävimme läpi integraatiosta esiin nousseita haasteita ja suunnittelimme Invoicyn ja oman tiimimme keskinäistä ohjelmointisessiota. Loppupäivästä tein keilailuun liittyneen Kata-tehtävän uudestaan, onnistumatta kuitenkaan ratkaisemaan

tehtävää kokonaan. Sain silti paljon uusia ideoita tehtävän ratkaisemiseen ja päätin tehdä tehtävän uudestaan seuraavana päivänä.

Torstai 22.4.2021

Osallistuin aamulla yksikkötestauksesta pidettävään koulutukseen. Koulutuksen ensimmäinen osio oli teoriapainotteinen ja aloitteleville kehittäjille suunnattu. Vaikka olen mentoriohjelman kautta tutustunut yksikkötestaukseen aiempaa lähemmin, uskon koulutuksen auttaneen minua hahmottamaan yksikkötestauksen periaatteita aiempaa paremmin. Kouluttajamme rohkaisi avoimeen keskusteluun koulutuksen aikana ja opimme sisartuotteiden kehittäjien kesken lisää toistemme testimenetelmistä. Koulutus jatkuu ensi viikolla käytännönläheisemmin, jolloin osallistujien on tarkoitus päästä itsekin rakentamaan testejä. Jään odottamaan innolla uusia tehtäviä, joiden avulla pääsen oppimaan konkreettisia menetelmiä testilähtöiseen kehittämiseen.

Loppupäivän käytin yksikkötestien testaamiseen integraatioasetuksien PATCH-reittiin liittyen. Huomasin pohtivani testien kirjoittamista uudelta kantilta ja testaavani paremmin metodin logiikkaa sen konkreettisen implementaation sijasta. Aiemmin koin vaikeaksi hahmottaa, mitä metodissa on tärkeää testata ja kirjoitin testejä, jotka riippuivat vahvasti implementaatiosta.

Riippuvuudet ovat niin tuotantokoodissa, kuin testeissäkin ongelmallisia. sen lisäksi että ne vaikeuttavat muutoksien tekoa, riippuvuuksilla on myös taipumus tuottaa duplikaattikoodia (Beck 2002, luku 1.1). Olenkin mielestäni oppinut testaamaan pikemminkin testin rajatapauksia, sekä metodin liiketoimintalogiikkaa kuin sen implementaatiota. Olen mielestäni samalla onnistunut rajaamaan metodin käyttötarkoituksen, jolloin yksikkötesti dokumentoi itse metodin toimintaa.

Perjantai 23.4.2021

Aamuni alkoi mentoriryhmäni tapaamisella, jossa kävimme läpi Kata-tehtävän tulokset. Keskustelimme siitä, miten kukakin meistä oli lähtenyt ratkaisemaan tehtävää TDD:n periaatteita noudattaen. Sen jälkeen mentorimme näytti, miten itse lähtisi ratkaisemaan kyseistä tehtävää ja suunnittelimme yhdessä sopivat testit sykli kerrallaan. Testaamiseen liittyen mentorimme näytti myös, miten yksikkötestattavan luokan voi alustaa tehokkaasti, joustavasti ja luotettavasti Builder-suunnittelumallia käyttäen (Freeman ym. 2004, 620–621). Malli oli minulle uusi ja päätin tutkia aihetta lisää viikonlopun aikana.

Kata-tehtävän läpikäymisen lisäksi keskustelimme myös tulevasta TDD-koulutuksestamme ja pohdimme, miten haluamme sen rakentaa. Koska lupasimme koulutuskutsussa pitää koulutuksen käytännönläheisenä, päätimme käyttää oppimisen tukena FizzBuzz-nimistä ohjelmointipulmaa. FizzBuzz-pulmassa ohjelman tulee tulostaa konsoliin numero tai sana riippuen siitä, onko numero jaollinen annetuilla numeroilla (Coding Dojo s.a.). Pulma ei itsessään ole vaikea, mutta koska ohjelmointisyklin kääntäminen testilähtöiseksi otti ryhmältämmekin hetken aikaa, sopii pulma loistavasti koulutuksen tehtäväksi.

Sen lisäksi, että kirjoitin FizzBuzz-tehtävän muutaman kerran – myös juuri löytämääni Builder-suunnittelumallia noudattaen – ehdin myös osallistumaan myynnin ja kehityksen väliseen palaveriin. Palaverin tarkoitus on jatkossa käydä keskustelua tiimiemme välillä yhteisistä tavoitteista ja jakaa informaatiota. Palaverin jälkeen viimeistelin vielä integraatioasetuksien PATCH-reittiin liittyviä muutoksia. Lähdin viikonlopun viettoon hyvillä mielin päivän aikaansaannoksistani.

Viikkoanalyysi

Mielenkiintoisina aiheina tältä viikolta poimin tutkittavakseni projektityöskentelyyn liittyvä ja retrospektiivissä esiin noussut aihe: muokkauspyyntöjen arvioinnin merkitys tiimissä. Teknisenä aiheena analysoin mentoriryhmäni keskusteluissa esiin noussut Builder-suunnittelumallin hyödyntäminen yksikkötestien alustamisessa.

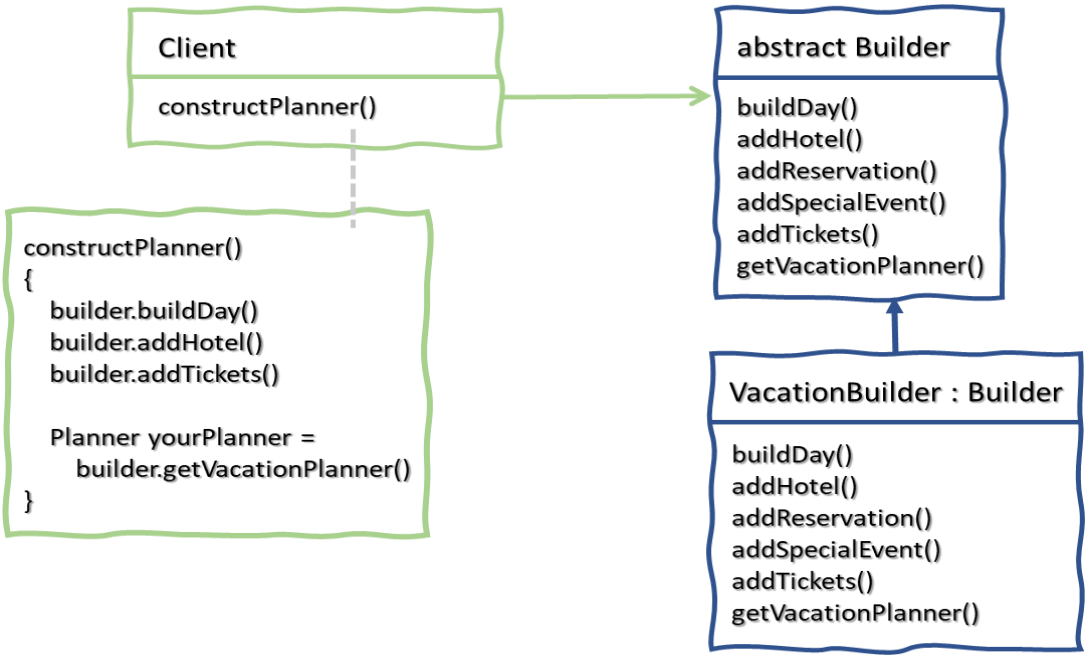
Keskustelimme tiimimme retrospektiivissä tällä viikolla aiempaa avuliaammista muokkauspyyntöjen arvioinneista. Osa tiimissämme toteutettavista muokkauspyynnön arvioinneista ovat lyhykäisyydessään hyvinkin niukkoja. On hyvin mahdollista saada hyväksytty arviointi ilman yhtäkään kommenttia tai kysymystä. Itse valitsenkin usein arviointiini kehittäjiä, joiden tiedän tarkastavan tekemäni muutokset suhteellisen tarkasti ja uskaltavan antaa palautetta. Junior-kehittäjänä koen hyötyväni huomattavasti enemmän muokkauksiini annettavista palautteista, joissa kuvataan sekä tämänhetkisen muokkauksen ongelma että sen ratkaiseva konsepti. Ajatuksella kirjoitettu kommentti tai esimerkiksi puhelu keskustelua herättäneestä muokkauksesta ei pelkästään avaa kommentoijan parannusehdotuksia vaan saattaa myös oikaista väärinkäsityksen ja jakaa lähes poikkeuksetta informaatiota tiimiläisten kesken (Bacchelli & Bird 2013, 1–2).

Vuonna 2013 julkaisemassaan tutkimuksessa ”Expectations, Outcomes and Challenges of Modern Code Review” Bacchelli ja Bird (1–10) määrittelivät modernia koodiarviointia yhä käytettävän pitkälti virheiden korjaamiseen. Tutkimus toteaa virheiden korjaamiseen

liittyvien kommenttien kuitenkin olevan vain pieni osa arvioinnin hyötyjä ja pääasiassa kattavan pitkälti matalan tason ongelmia. Sen sijaan tutkimuksen mukaan kehittäjien motivaatio koodiarvioinnin toteuttamiseen ovat myös koodin parantelu, vaihtoehtoisten ratkaisujen tarkastelu, tiedon jakaminen ja kehittäjien taitojen kehittyminen.

Omat arviointini jäävät usein kuitenkin lyhyiksi, johon osasyynä on mielestäni itsevarmuuteni kehittäjänä. Luen tehdyt muokkaukset kyllä läpi, mutta usein kontekstin ymmärtäminen on minulle kokeneempaa kehittäjää vaikeampaa. Lisäksi vaikka näkisin muokkauksissa jotain kummallista, luotan enemmän kokeneen kehittäjän tekemiseen kuin omaan ymmärrykseeni. Itsevarmuuden, sekä ymmärryksen kasvaessa uskon kehittyväni tulevaisuudessa myös arvioijana. Sen lisäksi arvioinnin voi nähdä opiskeluna ja muokkauspyynnön sisällön opiskelumateriaalina. (Obala 22.1.2020.)

Kävin tällä viikolla keskustelua Builder-mallin hyödyntämisestä yksikkötestitiedostojen rakentamisessa. Builder-malli on hyödyllisin tilanteessa, jossa monimutkaisen luokan esiintymiä halutaan rakentaa tilanteesta riippuen erilaisin ehdoin. Hyvänä esimerkkinä voidaan käyttää loman suunnitteluohjelmaa, jonka tehtävänä on erotella jokainen päivä sisältöineen omiksi kokonaisuuksiksi. Suunnittelijan alustaminen asiakaskohtaisesti voi näin käydä hyvinkin monimutkaiseksi, jolloin instanssien rakentamisen siirtäminen omaan luokkaansa ei pelkästään tuo ratkaisuun joustavuutta, vaan mahdollistaa yksittäisten ominaisuuksien lisäämisen tarpeen mukaan (Kuva 7). (Freeman ym. 2004, 260–621.)



Kuva 7. Luonnostelma loman suunnitteluohjelman sisältämästä Builder-suunnittelumallista (Freeman ym. 2004, 261).

Omassa järjestelmässämme käytämme tällä hetkellä NUnit-ohjelmakirjaston tarjoamaa [SetUp]-attribuuttia, jonka mukaisesti testattava luokka sekä muut siihen liittyvät luokat ja metodit alustetaan jokaisen testin aluksi käyttäen tätä yhteistä alustusmetodia (NUnit 2018b). Tällaisessa alustuksessa hyvä puoli on mielestäni se, että luokan alustus ja muu siihen liittyvä voidaan helposti lisätä tähän yhteen metodiin ja näin hyödyntää valmiiksi kirjoitettua alustusta nopeuttamaan yksikkötestien lisäämistä. Kyseinen lähestyminen tuo kuitenkin myös haasteita. Yhteinen alustusmetodi tarkoittaa, että jokaisen testin tulee erikseen erotella testiin liittyvät vaatimukset ja samalla alustusmetodi hukkuu helposti isoon testitiedostoon. Yhteinen alustus haittaa myös testien luettavuutta. (Truyers 15.7.2013.)

Martinin (2009, 124–126) yksikkötestien tärkein ominaisuus on sen hyvä luettavuus. Martin havainnollistaa luettavuutta siivoamalla roskaantuneen metodin siirtämällä testin alustuksen sille tarkoitettuun metodiin, jolloin testi on kokonaisuudessaan ymmärrettävistä yhdellä vilkaisulla. Alustavan metodin nimi kertoo, miten testi on alustettu. Myös Truyers (15.7.2013) kirjoittaa hyvän testin olevan sekä joustava että ennen kaikkea itseään havainnollistava. Kuten Martinin, myös Truyersin mukaan testidatan alustaminen testin alussa testattavan luokan alustusmetodilla haittaa molempia testin tärkeistä ominaisuuksista. Truyers käyttää mentorini tavalla Builder-mallia näiden ongelmien ratkaisemiseen.

Mentorini havainnollisti Builder-mallia yksikkötestitiedostoa rakennettaessa. Hän hyödynsi mallia irrottaen testattavan luokan alustamisen Builder-luokkaan, jolloin testattavan luokan alustus jokaisen yksikkötestin vaatimusten mukaisesti on yhtä yhteistä alustusmetodia joustavampi. Näin ollen testattavan metodin muuttuessa voidaan siihen liittyvät testitkin nopeasti päivittää sopiviksi. Tällöin myös uusien yksikkötestien lisääminen muuttuvalle luokalle helpottuu, kun testeihin voidaan kirjoittaa oma Builder-luokan kautta käytettävä alustusmetodi. (Freeman ym. 2004, 260–261; Truyers 15.7.2013.)

Harjoittelin Builder-mallin rakentamista tehdessäni FizzBuzz-tehtävää. Tehtävän tarpeet vaikuttivat kuitenkin liian pieniltä tämänkaltaiseen rakenteeseen suhteutettuna, mutta voisin hyvin kuvitella sen tuovan joustavuutta Newbien testitiedostoihin. Mallin rakenne tuntuu suhteellisen yksinkertaiselta toteuttaa ja sen hyötyjen ylittävän sen aikavaativuuden Newbien kaltaisessa isossa järjestelmässä.

3.10 Seurantaviikko 10

Maanantai 26.4.2021

Asetin tavoitteikseni tällä viikolla viimeisteleväni integraatioasetuksiin liittyvät PATCH-reitit ja niiden liiketoimintalogiikan, sekä suunnittelevani mentorointiryhmäni kanssa tulevan koulutuksemme sisältöä. Koulutuksen sisältöön liittyen käytin aiemmalla viikolla hetken aikaa tehdessäni FizzBuzz-ohjelmointipulman testilähtöisesti soveltaen myös Builder-suunnittelumallia. Testilähtöisen kehittämisen periaatteet ovat minulle jo tuttuja, mutta koin kuitenkin teorian opiskelusta olevan minulle hyötyä. Käytin siis aamustani hetken testilähtöisen kehittämisen tutkimiseen ja päätin jatkaa opiskelua tiistaina.

Integraatioasetuksiin liittyen rakensin loppupäivän aikana tietokantaan tallennettavien tietojen validointilogiikkaa. Validointia helpottaa huomattavasti xml-muotoisten asetuksien irrottaminen omaan kääreluokkaansa. Tämänkaltainen ratkaisu olisi ongelmallisempi, mikäli asetuksia pitäisi käsitellä kokonaisena xml-tiedostona ja olinkin tyytyväinen löytämääni ratkaisuun. Kävimme aiheesta keskustelua kollegoideni kanssa ja koska olin työskennellyt samanlaisen ratkaisun kanssa toiseen integraatioon liittyen, tuntui logiikan rakentaminen helpolta. Vaikka koin vaikeaksi motivoitua työpäivääni, sain päivän aikana suhteellisen paljon aikaan.

Tiistai 27.4.2021

Aamuni alkoi koko organisaation yhteisessä Town Hall -palaverissa, jossa sisartuotteemme kehitystiimi esitteli tulevia järjestelmäuudistuksiaan. Tämän päiväinen palaveri oli erityisen mielenkiintoinen, koska samanlaiset uudistuksen on implementoitu omaan tuotteeseemme muutamia vuosia aikaisemmin. Sen lisäksi esittäjä oli tekninen asiantuntija, jolloin esityskin oli rakennettu itselleni mielenkiintoisesta näkökulmasta. Ymmärsin palaverin ansiosta paremmin sen, kuinka isosta uudistuksesta tällaisessa projektissa on kyse. Kun uudistusten motivaatio on usein vanhentuva järjestelmä ja sen ylläpidon haasteet, tulee uutta järjestelmää rakennettaessa kiire vastata kaikkiin vanhan järjestelmän tarjoamiin ominaisuuksiin (Martin 2009, 4–5). Seuratessani esitystä koin ymmärtäväni nyt paremmin myös Newbien rakentamiseen vuosia sitten kohdistuneita paineita.

Integraatioasetuksiin liittyen kävimme keskustelua laadunvarmistajan, sekä toisen kehittäjän kanssa ohjelman tulevista vaatimuksista. Muutama osa ominaisuudesta on vielä miettimättä, jonka lisäksi siitä puuttuu käyttöliittymä kokonaan. Lisää haastetta

tehtäviimme on tuonut lennossa muuttuneiden liiketoimintavaatimuksien lisäksi myös se, miten päätimme alun perin jakaa tehtävät. Koska tehtävät jaettiin eri ohjelmistologiikan tasojen sijasta REST-reittien mukaisesti, on seuraavan tehtävän suorittaminen aina riippuvainen edellisten muutoksien läpiviemisestä. Näin ollen omat muutokseni eivät välttämättä odota pelkästään omien aiempien muutoksieni yhdistämistä pääratkaisuumme, vaan riippuvat mahdollisesti myös kollegani työtahdista ja päinvastoin.

Suunnittelin tänään mentorointiryhmäni kanssa alustavaa aikataulua tulevaan koulutukseemme. Saimme mielestäni suunniteltua hyvin käytännönläheisen koulutuksen, jonka tarkistutamme vielä mentoriltamme huomenna. Kirjoitimme ylös keskustelustamme esiin nousseet ideat, jonka lisäksi päätimme jatkaa suunnittelua seuraavalla viikolla. Vaikka pelkäsin aluksi ajatusta pitää koulutusta muille, innostuin palaverin aikana ajatuksesta kehittyä esiintyjänä ja samalla mahdollisuudesta opettaa.

Keskiviikko 28.4.2021

Iso osa päivästäni kului erinäisissä palavereissa, jonka takia en asettanut ohjelmoinnilleni suuria tavoitteita. Integraatioasetuksien saralla kävin keskustelua kollegoideni kanssa hyvästä tavasta kiertää eräs validointiin liittynyt ongelma, jonka takia toista rakentamaani PATCH-reittiä kautta tulevan tiedon tallentaminen kantaan estyi. Kyseessä on ennen kantaan tallentamista tapahtuva validointi, joka ei ottanut huomioon nykyisillä vaatimuksilla tallennettavaa dataa.

Vaihtoehdoiksi muodostui samojen asetuksien kanssa työskentelevän kollegani tapa ohittaa kyseinen validointi tai uudenlaisen validoinnin lisääminen. Vaikka ensimmäinen vaihtoehto olisi ollut helppo ja nopea tapa ratkaista ongelma, soti se mielestäni järjestelmän yleistä rakennetta vastaan kutsuessaan tietokannan käsittelyä epäloogisesta paikasta, samalla ohittaen viimeisen validoinnin kokonaan. Päätin kirjoittaa tarkistukseen liittyneen metodin uudestaan lisäten siihen tarkistuksen myös kyseistä tilannetta vastaan. Kaiken kaikkiaan olin tyytyväinen aikaan saamaani ratkaisuun.

Pidimme tänään ensimmäistä kertaa vuosiin lähitiimissämme sprintin *uudelleenjärjestämisen*, jossa kävimme läpi tehtävälokimme tehtäväkokonaisuuksia arvioiden niiden vaatimuksia ja irrottaen isoja kokonaisuuksia pienemmiksi tehtäviksi. Sprintin uudelleenjärjestämisen ei ole tarkoitus olla joka viikkoinen Scrum-käytäntö, mutta helpottaa huomattavasti tiimiä ymmärtämään käsillä olevista kokonaisuuksista enemmän mahdollistaen samalla häilyvien tehtäväkuvauksien tarkentamisen. Osallistuin itse suurimmaksi osaksi kuuntelevana osapuolena palaveriin, kun kokeneemmat kollegani

esittivät syventäviä mielipiteitä teknisten tehtävien rakentamisesta. Lopetin päiväni hilpeässä Great Place To Work -etägaalassa, jossa yrityksemme sijoittui tänäkin vuonna Suomen 10 parhaan työpaikan joukkoon. (Great Place To Work 2021; Rogers 12.4.2019; Schwaber & Sutherland 2020, 10–11.)

Torstai 29.4.2021

Aamuni alkoi mentoriryhmäni kesken pidetyn palaverin merkeissä, jossa esittelimme alustavan agendan tulevalle koulutuksellemme. Mentorimme antoi agendasta palautetta ja nosti esiin muutamia ajatuksia siitä, miten valitsemiamme lähestymistapoja kannattaa ajoittaa koulutuksen aikana. Keskustelimme erityisesti siitä, miten saamme osallistutettua koulutettaviamme mahdollisimman paljon ja näin sidottua esitetyn teorian käytäntöön. Jaoimme lopuksi agendan osiin, jotta jokainen ryhmäläinen voi työstää niihin liittyviä kalvoja itsekseen. Sovimme tapaavamme koulutuksen suunnittelun merkeissä uudestaan seuraavalla viikolla.

Osallistuin aamupäivästä myös yksikkötestaukseen perehtyvään koulutukseen, jossa tällä kertaa kirjoitettiin valmiiseen ohjelmaan testejä edellisen kerran teorian pohjalta. Muut koulutuksen osallistujat olivat kokemattomia testaamisen kanssa ja huomasinkin nopeasti ymmärtäväni jo yllättävän paljon yksikkötestaamisesta .NET-ympäristössä. Ennen työsuhteeni alkua en ymmärtänyt yksikkötestaamisesta juuri mitään ja nyt tein koulutuksen tehtäviä lähinnä kokemattomampia avustaen. Kirjoitin yksikkötestejä myös PATCH-reittiini, sekä selvitin käyttöliittymäohjelmoijan kanssa erästä aiemmin korjaamani umpikujatilanteeseen liittyvää ongelmaa. Kaiken kaikkiaan päiväni oli hyvin kiireinen, enkä kokenut saaneeni keskityttyä mihinkään kunnolla.

Perjantai 30.4.2021

Käytin aamupäiväni käymällä läpi integraatioasetuksiin liittyviä muokkauksiani. Tätä viikkoa rytmitti mentorointiryhmän tapaamisten lisäksi myös yksikkötestauskoulutus ja muut palaverit. Näin ollen en ollut päässyt pureutumaan Invoicyyn viikon aikana mielestäni tarpeeksi ja tunsin siksi hieman painetta saattaa loppuun siihen liittyviä tehtäviä.

Julkaistessani muokkauspyyntöni asetuksien validointiin liittyen kävin myös keskustelua kollegani kanssa siitä, miksi olen päätynyt tekemään kyseisenlaisen rakenteen asetuksien validoinnille. Keskustelussa kävi ilmi, että siinä missä itse olen ymmärtänyt, että kaikki tietokantaan tallennettava tieto validoidaan järjestelmässämme aina noudattamallani

tavalla, on kyseinen kollegani aina ratkaissut validoinnin muilla tavoin. Kävin ratkaisuni läpi kollegani kanssa ja perustelin päätökseni muun muassa sillä, että muualla järjestelmässä kyseinen ratkaisu on yleistynyt käytäntö. Koska ohjelmointikäytännöt ovat erittäin tärkeä pitää mielessä järjestelmämme kaltaisessa valtavassa kokonaisuudessa (Sengayire 25.11.2019), piti kollega perustelujani pätevinä ja hyväksyi muokkauspyyntöni.

Iltapäivästä tutkin käyttöliittymäohjelmoijan esiin nostamaa ongelmaa aiemmin tekemässäni umpikujatilanteen ratkaisemisessa (Luku 3.3 Seurantaviikko 3). Aiempi versio teki viiden eri REST-kutsun kautta työpäivän kopioinnin seuraavalle päivälle ja aiheutti näin umpikujatilanteen. Muutama näistä kutsuista aiheutti samalla samojen raskaiden laskutoimitusoperaatioiden tekemisen useamman kerran. Koska osaa laskutoimituksista ei tarvitsisi tehdä kuin kerran, pyrin optimoimaan tekemääni ratkaisua ohittamalla turhat laskutoimitukset.

Julkaistessani aiemman ratkaisuni ei käyttöliittymä vielä käyttänyt tekemääni reittiä ja näin ollen työtuntien laskutoimitusten pätevyyttä oli vaikea testata kunnolla. Käyttöliittymän muutokset paljastivat laskutoimituksiin liittyvän ongelman ja muutaman muokkausyrityksen jälkeen korjasin sen lopulta poistamalla aiemmin tekemäni optimointiyrityksen. Koska umpikujatilanne on huomattavasti järjestelmää mahdollisesti hidastavia – mutta täysin toimivia – laskutoimituksia vakavampi ongelma, koin päätökseni olleen hyvä. Näin ollen muutokset saadaan nopeasti eteenpäin ja umpikujatilanteet estettyä tulevaisuudessa. Laskutoimituksen optimointiin voidaan palata myöhemmin. Koin olleeni tehokas työpäivän aikana ja lähdin hyvillä mielin viikonlopun viettoon.

Viikkoanalyysi

Mielenkiintoisena aiheena tällä viikolla pohdin paljon tiimimme ohjelmointikäytäntöjä. Ohjelmointikäytännöt koskettavat tiimiä mielestäni sekä teknisestä, että projektihallinnollisesta näkökulmasta. Aihe tuli esiin, kun huomasin isoja eroja tiimiläisteni ohjelmointikäytännöissä työskennellessäni ensimmäistä kertaa erään kollegani kanssa. Olen juuri sisäistänyt hieman sitä, miten alemman logiikan tärkeyttä korostava kollegani haluaa asiat ratkaista. Tällä viikolla pääsinkin perustelemaan ratkaisujani oppimistani käytännöistä kollegalle, jolle taas API:n saumattomuus käytännönläheisesti on turhia optimointeja tärkeämpää.

Olen löytänyt sisäisiltä verkkosivuiltamme jonkinlaiset vuosia vanhat ohjelmistokäytännöt, mutten ole kertaakaan kuullut tiimissämme puhuttavan kyseisistä säännöistä. Ohjelmointikäytäntöjen tärkeys korostuu isossa projektissa, jossa valtavan ohjelmapohjan

käsittely muuttuu nopeasti sotkuiseksi ja jopa hallitsemattomaksi. Sengayire (25.11.2019) kirjoittaa artikkelissaan ”Coding Standards and Conventions in Software Development Team” siitä, miten tärkeää tiimille on asettaa yhteiset säännöt, joita jokainen tiimin jäsen noudattaa. Sengayire pitää hyvin rakennetun järjestelmän ja puhtaan ohjelmoinnin perustana tiimin yhteisiä standardeja, joiden avulla myös tiimin keskinäinen ymmärrys järjestelmästä helpottuu. Ilman standardeja jokainen ohjelmoija saa rakentaa järjestelmää mielivaltaisesti, jolloin yksittäisen kehittäjän ymmärrys järjestelmästä vaikeutuu ajan myötä. Vaikealukuinen järjestelmä vaikeuttaa puolestaan uusien kehittäjien työtä, monimutkaistaa järjestelmää ja pidentää kehitykseen käytettyä aikaa.

Sengayire (25.11.2019) kirjoittaa, kuinka yhteisesti sovitut ohjelmointikäytännöt voivat liittyä esimerkiksi nimeämiseen, formatointiin, dokumentointiin, testaukseen tai luokkien rakentamiseen. Yksinkertainen tapa merkitä esimerkiksi julkinen tai yksityinen muuttuja alaviivalla helpottaa huomattavasti ohjelman luettavuutta kehittäjälle, joka katselee uutta järjestelmätiedostoa ensimmäisen kerran tietäen alaviivan merkityksen. Jaswinder (26.9.2017) taas kirjoittaa, että ohjelmointikäytännöillä voidaan vähentää järjestelmän aikavaativuutta ja virhealttiutta, jolloin myös sen ylläpidettävyys paranee.

Jaswinder (26.9.2017) nostaa esiin hyviä käytäntöjä siitä, miten järjestelmän osia voidaan kehittää sen luettavuus edellä. Verraten Martinin (2009) kirjoittamaan käytännöistä löytyy paljon samankaltaisuuksia, mutta myös muutamia eroja:

1. Koodin hyvä dokumentointi ja kommentointi tekee Jaswinderin mukaan järjestelmästä erittäin luettavan. Jaswinder toki myös muistuttaa, kuinka itsestään selvien asioiden kommentointi ei tuo lisäarvoa koodille. Martin (55) toisaalta neuvoo välttämään turhan kommentoinnin ja korostaa, ettei kommentti korvaa hyvää koodia.

2. Sisennys tuo syvyyttä koodiin ja erottelee tiedoston osat niin, että siitä on helppo silmämääräisesti tunnistaa esimerkiksi metodit luokan määrittelystä. Myös Martin (88–89) kirjoittaa sisennyksen tärkeydestä koodin luettavuuden määreenä. Hän muistuttaa, kuinka lyhyt ehtolauseke halutaan usein jättää yhdeksi riviksi. Usealle riville sisennettynä lauseketta on kuitenkin huomattavasti helpompi tulkita (Kuva 8).

```
if (ehto == tosi) { this.TeeJotain(); }
else { this.TeeJotainMuuta(); }

if (ehto == tosi)
{
    this.TeeJotain();
}
else
{
    this.TeeJotainMuuta();
}
```

Kuva 8. Ehtolauseke yhdelle riville kirjoitettuna verrattuna sama lauseke syvennettynä usealle riville.

3. Koodiosioiden erottaminen toisistaan rivin vaihdolla helpottaa luettavuutta. Martin (86) puhuu myös rivin vaihdoista koodin erottelijana ja havainnollistaa, kuinka välilyönneillä voidaan erottaa toisiinsa tiukasti liittyvät asiat kätevästi.

4. Selvästi sovitut nimeämiskäytännöt helpottavat huomattavasti koodin luettavuutta. Siinä missä Jaswinder kirjoittaa isojen ja pienien kirjaimien käytöstä, Martin (90) nostaa esiin, kuinka nimeämisessä tärkeintä on muuttujan, metodin tai luokan kuvaaminen yksiselitteisesti.

5. Koodin toistamista on tärkeä pyrkiä välttämään. Jaswinder kehottaa karttamaan saman koodin kopioimista useaan kertaan ja neuvoo sen sijaan automatisoimaan toistuvat prosessit. Martin (48) mainitsee samaisen periaatteen lisäten, kuinka vaikea toistuvaa koodia on välillä tunnistaa. Martin kehottaa siirtämään toistuvan koodin aina omaan metodiinsa, josta sitä voi hyödyntää aina tarvittaessa.

6. Useiden ehtolausekkeiden *pesiytyessä* sisäkkäisiin rajauksiin järjestelmän kulkua voi olla hyvin vaikea seurata. Hyvä vaihtoehto sisäkkäisille ehtolausekkeille on Exit-malli, jolloin ehto käännetään ja sen täyttyessä sisäkkäiseen ehtoon pudottamisen sijaan metodin suoritus päättyy (Menaia, 5.10.2020).

Toisaalta ohjelmointikäytännöt eivät ole yksinkertainen vastaus kaikkiin kehitystiimin ongelmiin. Ohjelmointikäytännöistä on käyty keskustelua myös siitä näkökulmasta, kuinka ne voivat vähentää yksittäisen kehittäjän motivaatiota ohjelmoida, kun yhteiset sopimukset riitelevät kehittäjän omien periaatteiden kanssa. Ohjelmointikäytäntöjen ylläpitoon työkaluja tarjoavan SubMainin tuottamassa selvityksessä suurin syy ohjelmointikäytäntöjen käyttöönoton epäonnistumiseen onkin kehitystiimin vastahakoisuus. Tehdyn selvityksen mukaan 26% prosenttia käyttöönotoista epäonnistui

koska kehitystiimi ei päässyt yhteisymmärrykseen käytännöistä, 23% prosenttia kehitystiimin vastahakoisuudesta ja jopa 35% koska käytännöistä ei pidetty kiinni. (SubMain 2014, 2–3.)

Martin perustelee kuitenkin ohjelmointikäytäntöjen tärkeyden sillä, että järjestelmä on loppujen lopuksi valtava määrä luettavia tiedostoja, joiden halutaan näyttävän kuin ne olisi kirjoittanut yksi ja sama kehittäjä. Hän mainitsee, että vaikka sovitut säännöt eivät välttämättä ole linjassa kehittäjän omien mielipiteiden kanssa, on koodin luettavuus ja tiimin kollektiivinen ymmärrys yksittäisen kehittäjän näkemystä tärkeämpää. (Martin 2009, 90.)

Koen itse ohjelmointikäytännöt erityisen tärkeiksi, kun aloittelevana kehittäjänä navigoin järjestelmäämme etsien vastauksia sen hetkiseen tehtävääni. Koen ymmärtäneeni tällä viikolla aiempaa paremmin, kuinka haastavaksi eriävät tavat ratkaista samoja ongelmia samassa järjestelmässä voi käydä. Valtavassa järjestelmässä on uutena kehittäjänä yllättävän haastavaa soveltaa juuri opittua. Mitä johdonmukaisemmin järjestelmä on rakennettu, sitä helpompi sitä on oppia ymmärtämään. Omassa tiimissäni koen, että yhteiset pelisäännöt vielä uupuvat ja kehitämme yhdessä pitkälti luottaen toistemme ammattitaitoon ratkaisumalleista riippumatta. Uskon tämän kuitenkin tuovan vaikeuksia pidemmän päälle ja olemmekin suunnitelleet yhteisten käytäntöjen kehittämistä tulevaisuudessa.

4 Pohdinta ja päätelmät

Opinnäytetyötä kirjoittaessani huomasin, kuinka vaihtelevia työtehtäväni olivat. Seurantaviikkojen aikana minulle konkretisoitui se, miten ohjelmistokehittäjänä tarvitsen sekä teknisiä, että sosiaalisia taitoja ja kuinka tämä työ haastaa minut joka päivä oppimaan uutta. Haastavaa opinnäytetyön kirjoittamisessa olikin juuri työtehtävien vaihtelu. Kun toisina päivinä saatoin saada aikaan paljon sekä ohjelmoinnin että projektityöskentelynkin kannalta, toisina päivinä saatoin tehdä monia pieniä, toisiinsa liittymättömiä ja vaikeasti kuvattavia asioita, jolloin johdonmukainen raportointi oli haastavaa.

Toinen haaste työn kirjoittamisessa oli työkieleni, joka suomalaistenkin tiimiläisteni kanssa keskustellessa on parhaimmillaan vain osittain suomi. Vaikka olen tiennyt että loppujen lopuksi kaikki ohjelmointi nojaa englannin kieleen (McCulloch 4.8.2019), en ennen opinnäytetyön kirjoittamista hahmottanut kuinka vahvat vaikutukset sillä on myös tiimimme kommunikointiin. Monia työssämme yleistyneitä termejä käytetään suomeksikin puhuttaessa vain englanniksi, eikä niille välttämättä löydy suomenkielistä sanaa, jonka alalla työskentelevä ohjelmistokehittäjä ymmärtäisi samaksi. Toisaalta, vaikka olenkin pitänyt englannin kielen osaamistani vahvana, koen englanninkielisen ammattisanastoni kehittyneen hurjasti työelämässä.

Seurantaviikkojen ja niihin perustuvien viikoittaisten analyysien tuella pystyin kuitenkin syventymään työtehtävistäni nousseisiin aiheisiin. Saatoin opiskella uuden konseptin perustuen muokkauspyyntööni annettuun parannusehdotukseen soveltaakseni sitä työtehtävissäni. Kun sitten työpäivän päätteeksi opiskelin siitä lisää pystyäkseni kertomaan aiheesta päiväraportissani, tuli aihe tutuksi monesta näkökulmasta ja jäi näin myös muistiini paremmin.

4.1 Projektityöskentely- ja pehmeiden taitojen kehittyminen

Seurantaviikkoja edeltäen analysoin projektityöskentely- ja pehmeiden taitojeni olevan suhteellisen hyvät asiakaspalvelutaustastani johtuen. Omiin tiimiläisiini verrattuna totesin myös seurantaviikkojen aikana ymmärtäväni paremmin ketteriin työkaluihin liittyviä hyötyjä. Ymmärsin viikkoanalyysien aikana kuitenkin paljon uusia näkökulmia ketterään projektityöskentelyyn liittyen ja koen löytäneeni lisää työkaluja sen tueksi. Seurantaviikkojen jälkeen koen ketterät työkalut aiempaa tärkeämmiksi tehtäviksi ohjelmistokehittäjän työnkuvassa. Opin myös, että sosiaaliset taidot ovat

ohjelmistokehittäjälle välttämätön työkalu ja kuinka etätyöskentelyn aikakaudella kommunikoinnin tärkeys tiimissä korostuu (West 2021).

Ymmärrykseni kommunikoinnista kehitystiimissä on syventynyt. Olen huomannut, kuinka tärkeässä asemassa kommunikointi ja tiimin kollektiivinen ymmärrys ovat. Keskustelut on useimmiten helpompi käydä suullisesti ja samalla pyrkiä varmistamaan, ettei väärinkäsityksiä synny. Tiimiläisten kesken vaihdetut ajatukset kehittävät mielestäni tiimin kollektiivista ymmärrystä kehitettävästä tuotteesta ja kannustavat avoimeen keskusteluun. Itsevarmuuteni kasvaessa olen oppinut ottamaan enemmän kantaa myös teknisiin keskusteluihin, jolloin olen kokenut olevani hyödyksi koko tiimille.

Olen oppinut ajattelemaan ohjelmistokehitystä koko tiimin näkökulmasta. Löysin tiimiläisteni ohjelmointiperiaatteista eroavaisuuksia ja ymmärsin, kuinka eri tavoin samat asiat voi ratkaista. Toisaalta on hienoa kehittää ohjelmistoa miettien järjestelmän optimaalisuutta, saumattomia operaatioita ja yhteisiä ohjelmistokäytäntöjä. Tällaisessa kehittämisessä mietitään mahdollisimman pitkälle tulevaisuuteen ja edetään hitaasti tarkasti viilattuun ratkaisuun. Toinen tapa kehittää tiimissämme on ominaisuuden rajaaminen sen realististen mahdollisuuksien ja vaatimuksien mukaan, jolloin ohjelmistoa voidaan kehittää nopeasti ja ketterästi reagoiden mahdollisiin ongelmiin ja muutoksiin sitten, kun niiden eteen tullaan. Järjestelmämme kehittyy jatkuvasti ja liiketoiminnalliset vaatimukset varsinkin uusissa ominaisuuksissa muuttuvat moneen kertaan, jolloin nopea ja ketterä kehittäminen pieni ongelma kerrallaan on myös suhteellisen pätevä.

4.2 Teknisten taitojen kehittyminen

Aloittelevana ohjelmistokehittäjänä arvioin teknisen osaamiseni olevan heikomman puoleinen seurantaviikkojen alussa. Määrittelin seurantaviikkojen aluksi työskentelyyni vaadittujen teknisien taitojen, kuten relaatiotietokannan ymmärtämisen, .NET-ympäristön ja sen kielien kanssa työskentelyn, yksikkötestaamisen sekä web-sovelluksen kehittämisen olleen vastavalmistuneen kehittäjän tasolla. Erityisesti yksikkötestaaminen on opinnoissani jäänyt hyvin vähälle ja jälkeenpäin olen kokenut olleeni seurantaviikkojen alussa siinä vasta aloittelija. Junior-tason ohjelmistokehittäjänä saatoin siis huoletta olettaa, että tekninen osaamiseni tulee kehittymään seurantaviikkojen aikana valtavasti.

Teknisen ymmärrykseni kasvamiseen on monia syitä. Olen seurantaviikkojen aikana huomannut, kuinka vaihtelevia työtehtäväni ovat ja kuinka tehtävien vaihtuvuus on opettanut minulle paljon ohjelmistokehityksestä käytännössä. Joinain päivinä saatan työskennellä tiukasti tietokantakyselyiden parissa, kun toisina päivinä pelkästään

työskentelen rajapintamme tuntumassa. Jonain päivänä saatan kirjoittaa koko päivän pelkkiä yksikkötestejä tai testata järjestelmäämme rakentamieni REST-kutsujen kautta. Päivittäin vaihtuvien teknisten tehtävien lisäksi olemme mentorointiryhmässäni käyneet läpi useita yleispäteviä konsepteja, suunnittelumalleja sekä ohjelmistokehityksen periaatteita, joiden ymmärtämistä on sitten konkretisoitu käytännön tehtävillä.

Tekninen osaamiseni ei ole kehittynyt pelkästään käytännön työllä. Saan päivittäisessä työskentelyssäni jatkuvasti palautetta, kehitysehdotuksia sekä ideoita kollegoiltani teknisestä näkökulmasta. Olen myös itse päättänyt käydä jatkuvaa keskustelua teknisistä aiheista ja pyytää parannusehdotuksia ratkaisuihini. Oma-aloitteisilla palautekeskusteluilla koen löytäneeni paljon asioita, jotka eivät muokkauspyynnön arvioinnin yhteydessä välttämättä olisi noussut esiin ollenkaan. Mentorointiryhmässä olemme käyneet läpi toistemme ratkaisuja ja keskustelleet rakentavasti niiden hyvistä sekä huonoista puolista. Mentorointiryhmässä toistemme ratkaisujen lukeminen on myös nostanut esiin monia konsepteja, joihin en olisi törmännyt omassa kehitystiimissämme. Muiden kehittäjien muokkauspyyntöjen lukeminen, kokeneiden kollegojen keskisen keskustelun kuunteleminen ja heidän työskentelynsä seuraaminen on myös ollut hyvin keskeinen osa oppimistani.

Olen ymmärtänyt testaamisen tärkeyden ohjelmistokehityksessä ja kuinka tärkeässä roolissa se on erityisesti jatkuvassa integraatiossa. Jopa useasti päivässä tehtävä julkaisu vie mukanaan kaikki järjestelmän muutokset ja huonosti testattuna myös uudet ohjelmavirheet. Koska järjestelmä on koko ajan asiakkaan käytössä, saattaa ohjelmavirhe pahimmillaan estää ohjelman käytön kokonaisuudessaan. Perusteellisella testaamisella voidaan varmistaa ohjelman toimivuus eri tilanteissa ja samalla huomata heti, mikäli jotain on mennyt rikki. Olen myös ymmärtänyt, mitä muita rajoitteita jatkuva integraatio asettaa ohjelmalle. Jotkut muutokset vaativat ohjelman tai tietokannan hetkittäisen uudelleen käynnistämisen, jolloin uusin integraatio pitää tehdä silloin, kun järjestelmässä on mahdollisimman vähän käyttäjiä. (Sacolick 17.1.2020.)

Testaamisen ja sen puutteen vaikutukset ovat hyvin verrattavissa Oldien ja Newbien välillä. Kohtasin myös muita rasitteita liittyen Oldiesta Newbieen siirryttäessä ja olen oppinut paljon järjestelmien vertailussa siitä, kuinka ohjelmistoala kehittyy jatkuvasti. Ymmärrän nyt entistä paremmin, kuinka helposti järjestelmän ylläpito vaikeutuu ajan kuluessa ja ohjelmistokäytäntöjen puuttuessa ja kuinka vaikeaa puhdas ohjelmointi voi olla.

Web-sovelluksen kehittämisestä olen ymmärtänyt sen, kuinka tärkeää on ajatella järjestelmään saapuvien kutsujen määrää ja niiden aiheuttamia vaikutuksia järjestelmässä. Mikäli joku kutsu tehdään käyttöliittymästä ja Public REST-rajapintaamme käyttävästä järjestelmästä miljoonia kertoja, voi järjestelmä huomattavasti hidastua esimerkiksi moninkertaisten tietokantakutsujen takia. Toisaalta alemman logiikan prosessit saattavat kutsujen myötä ajautua umpikujatilanteeseen. Tärkeä huomio on myös, ettei web-sovellusta käytetä pelkästään ennalta määriteltyjen käyttäjäpolkujen mukaisesti (Thoman 12.10.2020). Olen oppinut ajattelemaan kehittämistäni siltä kantilta, että kirjoittamani logiikan on tärkeä ottaa käyttäjätarinan lisäksi huomioon myös mahdolliset rajatapaukset ja väärinkäytökset.

4.3 Loppupäätelmät

Olen huomannut itsevarmuuteni ohjelmistokehittäjänä kehittyneen huomattavasti seurantaviikkojen aikana. Aloitin työskentelyni ajatellen kuuluvani vielä koulun penkille ja peloissani siitä, etten yltäisi minulta odotettuihin työsuorituksiin. Seurantaviikkojen aikana huomasin, kuinka aloin ottamaan kantaa tekniseen toteutukseemme ja siirtymään kuuntelijan roolista keskustelijan rooliin teknisissä keskusteluissa. Mentorointiohjelmasta löysin uusia teknisiä konsepteja, joista pääsin kertomaan muille kehittäjille ja keräämään kannatusta esiin nostamilleni kehitysehdotuksille.

Seurantaviikkojen päätteeksi en koe kuuluvani enää koulun penkille vaan löytäneeni itselleni sopivan syventävän jatkokoulutuksen mentorointiohjelmasta. Mentorointiohjelmassa tulemme syventymään web-sovelluskehityksen julkaisuputkeen, erilaisiin suunnittelumalleihin ja syventäviin opintoihin. Opintojen edetessä uskon itsevarmuuteni kehittäjänä vielä kasvavan. Lisäksi samalla kun alan ymmärtämään tarpeeksi ohjelmistokehityksen perustuksista uskon tulevaisuudessa pystyväni soveltamaan myös vaikeampia konsepteja.

Opinnäytetyön seurantaviikkojen raportointi tuntui työläältä, mutta hyödylliseltä. Työpäivän aikana nousseiden teemojen käsittely helpotti sisäistämään monia konsepteja, joita en kerennyt työpäivän aikana muuten tutkia. Jatkon kannalta opiskelun jatkaminen päiväraportoinnin sijaan tietojenkäsittelytieteen maisteriohjelmassa voisi olla otollista. Tällä hetkellä opin kuitenkin päivittäisessä työssäni joka päivä uutta. Lisäksi mentorointiryhmä mahdollistaa täysipäiväisen työskentelyn alalla ja tuntuu luonnolliselta tavalta jatkaa siirtymistäni ohjelmointikikkojen opiskelusta merkittävien järjestelmän suunnittelumallien rakentamiseen.

Lähteet

Addie, S. & Smith, S. 12.5.2019. Handle requests with controllers in ASP.NET Core MVC. Microsoft dokumentaatio. Luettavissa: https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/actions?view=aspnetcore-5.0#what-is-a-controller. Luettu: 25.2.2021.

Arene Ry, s.a. Opinnäytetyön eettiset ohjeet. Luettavissa: http://www.arene.fi/julkaisut/raportit/opinnaytetoiden-eettiset-suositukset/. Luettu: 10.10.2020.

Asiakastieto s.a. Yritystietopalvelu. Luettavissa: https://www.asiakastieto.fi/web/fi/etusivu.html. Luettu: 7.2.2021.

Bacchelli, A. & Bird, C. 2013. Expectations, Outcomes, and Challenges Of Modern Code Review. Proceedings of the International Conference on Software Engineering (IEEE), San Francisco, s.1–10.

Bates, B., Freeman, E., Robson, E. & Sierra, K. 2004. Head First Design Patterns. O‘Reilly Media, Inc. Kalifornia.

Beck, K. 2002. Test Driven Development: By Example. Addison-Wesley Professional. Boston. Luettavissa: https://learning.oreilly.com/library/view/test-driven-development/0321146530/. Luettu: 22.4.2021.

Belida, S. & Varanasi, B. 2015. Spring REST. Apress. New York. Luettavissa: https://haagahelia.finna.fi/Record/nelli21.3710000000433708. Luettu: 13.2.2021.

Bertucci, P., Gallelli, C., Rankins, R. & Silvertsein, A. 2013. Microsoft SQL Server 2021 Unleashed. Sams. Carmel. Luettavissa: https://learning.oreilly.com/library/view/microsoft-sql-server/9780133408539/. Luettu: 29.3.2021.

Bliley Technologies 4.2.2020. 5 Simple Steps for Truly Effective Design Reviews. Bliley Technologies. Luettavissa: https://blog.bliley.com/5-simple-steps-for-truly-effective-design-reviews. Luettu: 20.2.2021.

Blokland, K., Mengerink, J., Pol, M. 2013. Testing Cloud Services. Rocky Nook Inc. Kalifornia. Luettavissa: https://haagahelia.finna.fi/Record/nelli21.3460000000129496. Luettu: 10.2.2021.

Bogue, R. 20.7.2005. Anatomy of a Software Development Role: Quality Assurance. developer.com. Luettavissa: https://www.developer.com/java/other/article.php/3515426/Anatomy-of-a-Software-Development-Role-Quality-Assurance.htm. Luettu: 11.2.2021.

Bowes, J. 9.10.2014. Scrum in practice: the Sprint Demo. Manifesto. Luettavissa: https://manifesto.co.uk/scrum-practice-sprint-demo/. Luettu: 22.3.2021.

Bugayenko, Y. 13.5.2021. Why NULL is Bad? Asiantuntijablogi. Luettavissa: https://www.yegor256.com/2014/05/13/why-null-is-bad.html. Luettu: 3.4.2021.

Casacuberta, J. s.a. Perfect Your UX Design Process – A Guide to Prototype Design. Top tall. Luettavissa: https://www.toptal.com/designers/prototyping/guide-to-prototype-design. Luettu: 20.2.2021.

CodeAcademy 2021. MVC: Model, View, Controller. Luettavissa: https://www.codecademy.com/articles/mvc. Luettu: 21.4.2021.

Coding Dojo s.a. FizzBuzz. Luettavissa: https://codingdojo.org/kata/FizzBuzz/. Luettu: 23.4.2021.

Coline, 26.1.2017. History of REST APIs. Mobapi. Luettavissa: https://www.mobapi.com/history-of-rest-apis/. Luettu: 18.4.2021.

Concepta 2018. How To Define Stakeholders for Your Software Development Project. Concepta. Luettavissa: https://www.conceptatech.com/blog/how-to-define-stakeholders-for-your-software-development-project. Luettu: 13.2.2021.

Croft, C. 2.6.2014. Project Management – Key Drivers – Video 8 of 38. Youtube-video. Luettavissa: https://www.youtube.com/watch?v=AHvEeAA7_R8. Luettu: 13.4.2021.

Dalmjin, M. 7.10.2019. 6 common mistakes when doing Sprint Planning. Medium. Luettavissa: https://medium.com/serious-scrum/6-common-mistakes-when-planning-a-sprint-6fedbedf604e. Luettu: 7.4.2021.

Davičienė, V. & Raudeliuniene, J. 2010. ICT tacit knowledge preservation. Kuudes kansainvälinen tieteellinen konferenssi ”Business and Management 2010”, Vilnius, s. 822–824.

Dodds, K. 17.8.2020. Testing Implementation Details. Asiantuntijablogi. Luettavissa: https://kentcdodds.com/blog/testing-implementation-details. Luettu: 6.3.2021.

Elliot, E. 25.5.2019. Behavior Driven Development (BDD) and Functional Testing. Medium. Luettavissa: https://medium.com/javascript-scene/behavior-driven-development-bdd-and-functional-testing-62084ad7f1f2. Luettu: 15.2.2021.

Fielding, R. & Reschke, J. 1.7.2014. Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. Luettavissa: https://datatracker.ietf.org/doc/html/rfc7231#section-4.2.2. Luettu: 27.3.2021.

Ford, N., Kua, P. & Parsons, R. 2017. Building Evolutionary Architectures. O'Reilly Media, Inc. Kalifornia. Luettavissa: https://learning.oreilly.com/library/view/building-evolutionary-architectures/9781491986356/. Luettu: 9.4.2021.

Frankenfield, J. 29.8.2020. Business Logic. Investopedia. Luettavissa: https://www.investopedia.com/terms/b/businesslogic.asp. Luettu: 27.4.2021.

Fröschle, H. & Reinheimer, S. 2010. Cloud Computing & SaaS. Dpunkt. Heidelberg. Luettavissa: https://haagahelia.finna.fi/Record/nelli21.3230000000079175. Luettu: 7.2.2021.

Fowler, M. 1.5.2006. Continuous Integration. Luettavissa: https://martinfowler.com/articles/continuousIntegration.html. Luettu: 20.2.2021.

Fowler, M. 26.7.2005. InversionOfControl. Artikkeli henkilökohtaisella verkkosivulla. Luettavissa: https://martinfowler.com/bliki/InversionOfControl.html. Luettu: 24.2.2021.

Freeman, E., Robson E., Sierra, K. & Bates, B. 2014. Head First Design Patterns. O'Reilly Media, Inc. Kalifornia.

GeeksforGeeks, 6.8.2020. Introduction of Deadlock in Operating System. Luettavissa: https://www.geeksforgeeks.org/introduction-of-deadlock-in-operating-system/. Luettu: 3.3.2021.

GeeksforGeeks 13.10.2020. Red-Black Tree | Set 1 (Introduction). Luettavissa: https://www.geeksforgeeks.org/red-black-tree-set-1-introduction-2/. Luettu: 12.3.2021.

Great Place to Work 2020. Suomen parhaat työpaikat. Luettavissa: https://www.greatplacetowork.fi/parhaat-tyopaikat/suomen-parhaat-ty%C3%B6paikat/2020. Luettu: 7.2.2021.

Great Place to Work 2021. Suomen Parhaat Työpaikat. Luettavissa: https://greatplacetowork.fi/parhaat-tyopaikat/suomen-parhaat-tyopaikat-2021/. Luettu: 28.4.2021.

Habib, A. 6.1.2014. All about TransactionScope. Code Project. Luettavissa: https://www.codeproject.com/Articles/690136/All-About-TransactionScope. Luettu: 28.2.2021.

Halonen, H. 8.2.2019. 15 termiä, joiden ymmärtäminen helpottaa työtäsi koodareiden kanssa. Poplatek. Luettavissa: https://www.poplatek.fi/15-termia-joiden-ymmartaminen-helpottaa-tyotasi-koodareiden-kanssa/. Luettu: 28.2.2021.

Hilton, P. 5.1.2021. Implement a zero-bug policy. Writing by Peter Hilton. Luettavissa: https://hilton.org.uk/blog/zero-bug-policy. Luettu: 14.4.2021.

Hilyard, J. & Teilhet, S. 2004. C# Cookbook. O'Reilly Media, Inc. Luettavissa: https://learning.oreilly.com/library/view/c-cookbook/0596003390/ch04s06.html. Luettu: 19.2.2021.

Howard, M. & LeBlanc D. 2005. Writing Secure Code, Second Edition. Microsoft Press. Washington. Luettavissa: https://flylib.com/books/en/1.290.1/. Luettu: 3.4.2021.

IfM, Institute for Manufacturing. What is roadmapping and how can it benefit your organization? Cambridgen yliopisto. Youtube-video. Luettavissa: https://www.youtube.com/watch?v=0JxsSHjyJPc. Luettu: 13.4.2021.

Indeed 3.11.2021. Soft Skills: Definitions and Examples. Indeed. Luettavissa: https://www.indeed.com/career-advice/resumes-cover-letters/soft-skills. Luettu: 3.11.2021.

Jarrett, C. 4.5.2018. Learning by teaching others is extremely effective – a new way study tested a key reason why. Research Digest. Luettavissa: https://digest.bps.org.uk/2018/05/04/learning-by-teaching-others-is-extremely-effective-a-new-study-tested-a-key-reason-why/. Luettu: 14.3.2021.

Jaswinder 26.9.2017. The Significance of Coding Standards. WebGuruz. Luettavissa: https://webguruz.in/the-significance-of-coding-standards-2/. Luettu: 2.5.2021.

Jubi, R. & Sankar, U. 2015. Agile Software Develoopment Dynamics through TACIT Knowledge: An Attempt to Acquire and Share TACIT Knowledge. Research Journal of Management Sciences. 4, s. 8–14.

Kanjilal, J. 2013. ASP.NET Web API. Packt Publishing. Birmingham. Luettavissa: https://learning.oreilly.com/library/view/aspnet-web-api/9781849689748/. Luettu: 18.4.2021.

Kavis, M. 2014. Architecting the Cloud: Design Decisions for Cloud Computing Service Models. John Wiley & Sons. New Jersey. Luettavissa: https://haagahelia.finna.fi/Record/nelli21.2670000000520080. Luettu: 10.5.2021.

Korneeva, M. 23.10.2020. How to Handle Multiple Click Events in Angular & RxJs. Medium. Luettavissa: https://medium.com/ngconf/how-to-handle-double-click-events-in-angular-rxjs-e318697c9e26. Luettu: 7.3.2021.

Laaksonen, A. 2020. Tietorakenteet ja algoritmit. Kurssikirja Helsingin yliopiston avoimelle kurssille ”Tietorakenteet ja algoritmit”. Luettavissa: https://www.cs.helsinki.fi/u/ahslaaks/tirakirja/. Luettu: 18.2.2021.

Madan, S. 16.12.2019. DONE Understanding Of The Definition Of ”Done”. Scrum-dokumentaatio. Luettavissa: https://www.scrum.org/resources/blog/done-understanding-definition-done?gclid=CjwKCAiAm-2BBhANEiwAe7eyFO5Eutnjjofk9tUr-gzSWWigdGIq2zhJO3jiaEjXjm5WwPywPa3apBoCg78QAvD_BwE. Luettu: 28.2.2021.

Martin, R. 2009. Clean Code: A Handbook of Agile Software Craftmanship. Pearson Education, Inc. New Jersey.

Martin, R. 2011. The Clean Coder: A Code of Conduct for Professional Programmers. Pearson Education, Inc. New Jersey.

McCulloch, G. 4.8.2019. Coding is for Everyone – as Long as You Speak English. Wired. Luettavissa: https://www.wired.com/story/coding-is-for-everyoneas-long-as-you-speak-english/. Luettu: 9.5.2021.

Mehmood, M. 9.2.2019. Transaction Scope – A simple way to handle transactions in C#. Medium. Luettavissa: https://medium.com/@mahrukhmehmood/transaction-scope-a-simple-way-to-handle-transactions-in-c-491f42cdd3c9. 22.2.2021.

Menaia, L. 5.10.2020. Return Early Pattern. Luettavissa: https://medium.com/swlh/return-early-pattern-3d18a41bba8. Luettu: 2.5.2021.

Microsoft 2021a .NET dokumentaatio. Luettavissa: https://docs.microsoft.com/en-us/dotnet/. Luettu: 8.2.2021.

Microsoft 2021b List<T> Constructors. Luettavissa: https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1.-ctor?view=net-5.0. Luettu: 18.2.2021.

Microsoft 2021c TransactionScope Class. Luettavissa: https://docs.microsoft.com/en-us/dotnet/api/system.transactions.transactionscope?view=net-5.0. Luettu: 10.2.2021.

Microsoft 2021d. RangeAttribute Class. Luettavissa: https://docs.microsoft.com/en-us/dotnet/api/system.componentmodel.dataannotations.rangeattribute?view=net-5.0. Luettu: 4.3.2021.

Microsoft 2021e. Reference Source. Microsoftin .NET Framework -rungon julkiset lähdekoodit. Luettavissa: https://referencesource.microsoft.com/. Luettu: 5.3.2021.

Microsoft 2021f. IsolationLevel Enum. Luettavissa: https://docs.microsoft.com/en-us/dotnet/api/system.transactions.isolationlevel?view=net-5.0. Luettu: 22.3.2021.

Microsoft 2021g. Nullability reference types. Luettavissa: https://docs.microsoft.com/en-us/dotnet/csharp/nullable-references. Luettu: 29.3.2021.

Microsoft 2021h. Enumerable.Any Method. Luettavissa: https://docs.microsoft.com/en-us/dotnet/api/system.linq.enumerable.any?view=net-5.0. Luettu: 3.4.2021.

Microsoft 2021i. XML Data Type and Columns (SQL Server). Luettavissa: https://docs.microsoft.com/en-us/sql/relational-databases/xml/xml-data-type-and-columns-sql-server?view=sql-server-ver15. Luettu: 12.4.2021.

Microsoft 30.3.2017. Caching in .NET Framework Applications. Luettavissa: https://docs.microsoft.com/en-us/dotnet/framework/performance/caching-in-net-framework-applications. Luettu 11.3.2021.

Microsoft 16.10.2018. Run background tasks with WebJobs in Azure App Service. Luettavissa: https://docs.microsoft.com/en-us/azure/app-service/webjobs-create?pivots=webjob-type-continuous. Luettu: 10.3.2021.

Microsoft 8.12.2019. Understanding isolation levels. Luettavissa: https://docs.microsoft.com/en-us/sql/connect/jdbc/understanding-isolation-levels?view=sql-server-ver15. Luettu: 22.3.2021.

Miller, B. 7.8.2016. 6 Pros and Cons of Entity Framework. Green Garage. Luettavissa: https://greengarageblog.org/6-pros-and-cons-of-entity-framework. Luettu 16.2.2021.

Nestell, J. & Olson, D. 2018. Successful ERP Systems: A Guide for Businesses and Executives. Business Expert Press, LLC. New York. Luettavissa: https://haagahelia.finna.fi/Record/nelli21.4340000000238540. Luettu: 11.2.2021.

NUnit 2018a. TestCase. NUnit dokumentaatio. Luettavissa: https://docs.nunit.org/articles/nunit/writing-tests/attributes/testcase.html?q=TestCase. Luettu: 17.2.2021.

NUnit 2018b. SetUp. NUnit dokumentaatio. Luettavissa: https://docs.nunit.org/articles/nunit/writing-tests/attributes/setup.html. Luettu: 24.4.2021.

Obala, J. 22.1.2020. Meaningful Code Reviews. Linkedin-artikkeli. Luettavissa: https://www.linkedin.com/pulse/meaningful-code-reviews-japheth-obala/. Luettu: 24.4.2021.

Ola, S. 13.11.2018. RESTful API Design – PUT vs. PATCH. Medium. Luettavissa: https://medium.com/backticks-tildes/restful-api-design-put-vs-patch-4a061aa3ed0b. Luettu: 15.2.2021.

Paul, P. & Wang, R. 11.1.2019. Fitness function-driven development. ThroughtWorks. Luettavissa: https://www.thoughtworks.com/insights/articles/fitness-function-driven-development. Luettu: 16.4.2021.

Pauli, J. 2013. Most Common Web Vulnerabilities. The Basics of Web Hacking. Sciencedirect. Luettavissa: https://www.sciencedirect.com/topics/computer-science/malicious-input. Luettu: 20.2.2021.

Pride, J. 2018. Unicorn Tears. John Wiley & Sons Australia, Ltd. Milton. Luettavissa: https://haagahelia.finna.fi/Record/nelli21.3840000000330349. Luettu: 7.2.2021.

Pääkkö, T. 23.10.20218. Kaikki koodi on jossain vaiheessa legacya. Medium. Luettavissa: https://medium.com/@Glenf/kaikki-koodi-on-jossain-vaiheessa-legacya-35263b0c81ab. Luettu: 3.4.2021.

Radom, A. 21.3.2019. The ‘Design Review’ Between Design and Developers. Medium. UX Planet. Luettavissa: https://uxplanet.org/the-design-review-between-designers-and-developers-75152868f717. Luettu: 18.2.2021.

Rahman, T. 13.3.2019. Whats should and should not be tested in unit tests? Medium. Luettavissa: https://medium.com/@tasdikrahman/what-should-and-should-not-be-tested-in-unit-tests-7e6bdeb744e4. Luettu: 2.3.2021.

REST API Tutorial s.a. Using HTTP Methods for RESTful Services. Luettavissa: https://www.restapitutorial.com/lessons/httpmethods.html. Luettu: 22.2.2021.

Rogers, P. 12.4.2019. Backlog Refinement and Sprint Planning: Similarities and Differences. Medium. Luettavissa: https://medium.com/agile-outside-the-box/backlog-refinement-and-sprint-planning-similarities-and-differences-d08761aca3ae. Luettu: 28.4.2021.

Sacolick, I. 17.1.2020. What is CI/DC? Continuous integration and continuous delivery explained. Luettavissa: https://www.infoworld.com/article/3271126/what-is-cicd-continuous-integration-and-continuous-delivery-explained.html. Luettu: 9.5.2021.

Sambol, M. 25.8.2016. Red-black trees in 4 minutes – The basics. Youtube-video. Luettavissa: https://www.youtube.com/watch?v=qvZGUFHWChY. Luettu: 12.3.2021.

Schmidt, C. 2015. Agile Software Development Teams. Springer International Publishing. New York. Luettavissa: https://haagahelia.finna.fi/Record/3amk.267974. Luettu: 8.2.2021.

Schwaber, K. & Sutherland, J. 2020. The Scrum Guide. Luettavissa: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf#zoom=100. Luettu: 23.2.2020.

ScrumDesk s.a. Sprint Review. Luettavissa: https://www.scrumdesk.com/start/manual-for-scrumdesk-start/sprint-review/. Luettu: 9.3.2021.

ScrumInc, s.a. What is Timeboxing? Luettavissa: https://www.scruminc.com/what-is-timeboxing/. Luettu: 28.2.2021.

Sengayire, P. 25.11.2019. Coding Standards And Conventions In Software Development Team. Luettavissa: https://medium.com/@psengayire/the-importance-of-coding-standards-and-conventions-in-the-software-development-team-how-they-can-5d252556a05. Luettu: 2.5.2021.

Statista Research Department 2021. Global public SaaS market size 2008–2020. Statista. Luettavissa: https://www.statista.com/statistics/510333/worldwide-public-cloud-software-as-a-service/. Luettu: 8.2.20201.

SubMain, 2014. Coding Standards in the Real World. SubMain. Kirkland. Luettavissa: https://submain.com/ebooks/coding-standards-in-the-real-world/Coding%20Standards%20in%20the%20Real%20World.pdf. Luettu: 2.5.2021.

Sundman, Y. 5.2.2018. Stop Managing Bugs, Start Focusing on Quality! Crisp. -blogi. Luettavissa: https://blog.crisp.se/2018/02/05/yassalsundman/stop-managing-bugs-start-focusing-on-quality. Luettu: 18.4.2021.

Telerik JustMock s.a. Welcome to Telerik JustMock. Luettavissa: https://docs.telerik.com/devtools/justmock/introduction.html. Luettu: 11.3.2021.

Thakur, D. 17.11.2017. Group By Vs Distinct Difference In SQL Server. Luettavissa: https://www.c-sharpcorner.com/blogs/group-by-vs-distinct-difference-in-sql-server. Luettu: 12.4.2021.

Thoman, M. 12.10.2020. SQL Injections. Medium. Luettavissa: https://medium.com/faun/sql-injections-e8bc9a14c95. Luettu: 19.2.2021.

Thomas, D. 2016. CodeKata. Luettavissa: http://codekata.com/. Luettu: 16.4.2021.

Torstensson, J. 14.9.2020. Sailboat Retrospective by Johanna Torstensson. Miro. Luettavissa: https://miro.com/app/board/o9J_klDboNk=/?fromEmbed=1. Luettu: 16.2.2021.

Truyers, K. 15.7.2013. Flexible and expressive unit tests with the builder pattern. Asiantuntijablogi. Luettavissa: https://www.kenneth-truyers.net/2013/07/15/flexible-and-expressive-unit-tests-with-the-builder-pattern/. Luettu: 24.2.2021.

Tutkimuseettinen neuvottelukunta 2012. Hyvä tieteellinen käytäntö ja sen loukkausepäilyjen käsitteleminen Suomessa. Helsinki. Luettavissa: https://tenk.fi/fi/ohjeet-ja-aineistot. Luettu: 10.10.2021.

Tutorialspoint s.a.a. ASP.NET – Data Caching. Luettavissa: https://www.tutorialspoint.com/asp.net/asp.net_data_caching.htm. Luettu: 14.3.2021.

Tutorialspoint s.a.b. SQL - Indexes. Luettavissa: https://www.tutorialspoint.com/sql/sql-indexes.htm. Luettu: 25.3.2021.

Tutorialspoint s.a.c. SQL – Using Views. Luettavissa: https://www.tutorialspoint.com/sql/sql-using-views.htm. Luettu: 25.3.2021.

Upwork Staff, 19.4.2017. SOAP vs. REST: A Look at Two Different API Styles. Luettavissa: https://www.upwork.com/resources/soap-vs-rest-a-look-at-two-different-api-styles?utm_source=google&utm_medium=cpc&utm_campaign=11384804789&utm_content=113089129402&utm_term=&vt_cmp=11384804789&vt_adg=113089129402&vt_src=google&vt_kw=&vt_device=c&utm_source=google&utm_campaign=11384804789&utm_medium=paidsearch&gclid=Cj0KCQjwse-DBhC7ARIsAI8YcWKe5rN0GZPDA9nwk4f47xZqz5r07VesNHRj4uKzbcNGesVCOPMXlfAaAvCPEALw_wcB. Luettu: 18.4.2021.

Vadgama, J. 13.1.2021, Ling performance with Count() and Any(). DZone. Performance Zone. Luettu: https://dzone.com/articles/linq-performance-count-and-any. Luettu: 29.3.2021.

West, J. 2021. How to manage long-distance teams. The Economist. Luettavissa: https://execed.economist.com/career-advice/guest-post/how-manage-long-distance-teams. Luettu: 9.5.2021.

W3Schools 2021a SQL Injection. Luettavissa: https://www.w3schools.com/sql/sql_injection.asp. Luettu: 19.2.2021.

W3Schools 2021b. SQL IN Operator. Luettavissa: https://www.w3schools.com/sql/sql_in.asp. Luettu: 12.4.2021.

W3Schools 2021c. The SQL UNION Operator. Luettavissa: https://www.w3schools.com/sql/sql_union.asp. Luettu: 12.4.2021.

Young, G. 29.4.2017. Tietokirjailija, konsultti. Stop Over-Engenering. Videoitu seminaariesitys. Odessa. Katsottavissa: https://www.youtube.com/watch?v=GRr4xeMn1uU. Katsottu: 3.3.2021.

Liitteet

4.4 Liite 1. Vieraskieliset sanat suomennettuna

Työhön suomennettu sana	Vieraskielinen sana
Ajurit	Key Drivers
Atomisuus, johdonmukaisuus, eristäytyminen, kestävyys	Atomicity, Consistency, Isolation, Durability
Auttajaluokka	Helper (class)
Juuri	Root
Katselmus	(Sprint) Review
Kääreluokka	Wrapper (class)
Lueteltu (tietotyyppi)	Enumerable
Mallikääntäjä	Model converter (model to entity, entity to model)
Muokkauspyyntö	Pull request
Muotoilu	Design
Ohjain	Controller (Model-View-Controller)
Ohjelmointimalli	Design Pattern
Palveluluokka	Service (class)
Perintökoodi	Legacy code
Pesiytyä	Nesting
Pysäytyspiste	Breakpoint
Pääratkaisu	Master (branch)
Rajapinta	Interface
Solmu	Node
Strategiasuunnitelma	(Product / Project) Roadmap
Suunnittelumalli	Design Pattern
Tehtäväloki	Backlog
Tietokokonaisuus	Entity / Object
Tyhjennettävä	Nullable
Umpikuja	Deadlock
Uudelleenjärjestäminen (Sprintin)	Sprint Refinement
Viitetyyppi	Reference-type
Virheenkäsittelytila	Debug
Ylisuunnittelu	Over-Engineering