Haaga-Helia
ammattikorkeakoulu Oy

Web-kehittäjän arki pienyrityksessä

Daryoush Farsimadan

Opinnäytetyö

Tietojenkäsittelyn koulutusohjelma

2020

Haaga-Helia ammattikorkeakoulu Oy

Tiivistelmä

Päiväys

Tekijä(t) Daryoush Farsimadan	
Koulutusohjelma Tietojenkäsittelyn koulutusohjelma	
Opinnäytetyön otsikko Web kehittäjän arki pienyrityksessä	**Sivu- ja liite-sivumäärä** 56 + 1
Opinnäytetyön otsikko englanniksi Daily life of a web developer in a small company	

Työ kuvaa tekijän päivittäistä arkea web-kehittäjänä pienessä ICT-koulutusalan yrityksessä, jossa pääasiallinen yrityksen toiminta on toteuttaa koulutuksia yritysasiakkaille. Koska tekijä työskentelee yksin yrityksessä, on tärkeää pystyä taustatutkimaan sekä ratkaisemaan ongelmia itsenäisesti.

Päiväkirjaa pidetään 20.1–20.4 Välisenä aikana, jolloin tekijä raportoi työtehtäviään sekä päivittäistä työskentelyään kohdeyrityksessä. Jokaisen viikon yhteenvetona analysoidaan omia työskentelymalleja ja ratkaisuja vedoten tieteellisiin artikkeleihin sekä teoksiin. Keskeisinä kehityksen tavoitteina on parantaa tekijän henkilökohtaisia työskentelytapoja sekä kehittää omat ammatilliset taidot asetettujen tavoitteiden tasolle.

Asiasanat
Moodle, ohjelmistokehitys, kehittyminen

Sisällys

1 Johdanto

Päiväkirjatyyppisen opinnäytetyön aloitus ja lopetuspäivämääräksi on määritelty 20.1–20.4.

Päiväkirjatyyppisen opinnäytetyön raportointi tapahtuu päivittäisellä työtehtävien kuvaamisella ja viikoittaisella analyysillä.

Työskentely ohjautuu tekijän osaamisella, joka perustuu ajankohtaiseen ammattikirjallisuuteen. Teokset ovat osittain standardisoineet alan tapaa kehittää ohjelmistoa, ja niiden tarjoama teoreettinen perusta on tekijän kehityksen kannalta arvokasta. Alla määritellään muutama teos, joita tekijä noudattaa työn aikana sekä referoi aina tarvittaessa niihin.

Koska työpaikalla työskennellään pääasiallisesti Moodle-pohjaisen järjestelmän kanssa, joka voidaan todeta olla vanha, pitää siihen liittyvien käytänteiden käsittelyyn kirjallisuutta, joka ohjeistaa päivittäistä työskentelyä sekä käsittelee vanhan koodin kanssa työskentelyyn liittyviä käsitteitä.

Tästä syystä päätin valita Miachel Feathersin kirjan ”Working with legacy code”, joka määrittelee vanhan ohjelmiston ja sen tunnistavat tekijät. Se käsittelee vanhan koodin työskentelyyn liittyvistä malleista ja koen sen olevan erittäin oleellinen oman työskentelyni kannalta.

Työskentely tapahtuu yksin ja siksi on tärkeää adoptoida työmenetelmiä, jotka varmistavat laadun ja koodin toimivuuden. Koska työskentely malleja pitää koko ajan kehittää koin, että Beck Kentin teos ”Test-driven development” olisi todella hyvä lisä omaan päivittäiseen työskentelyyn. Moodle laajennukset vaativat paljon testaamista ja siihen ei sisälly vain pienien osien testaaminen, mutta myös testaaminen Moodlen järjestelmän kanssa. Syy tähän on Moodlen järjestelmän luonne, jossa jokainen laajennus järjestelmään tekee kutsuja rajapintoihin, joita ei voida testata ilman testidataa.

Työtehtävien aikana tuli vastaan sellaisia asioita, joihin koen, että Robert C Martinin teos ”Clean Code” pystyy pureutumaan hyvin. Teoksessa käsitellään ohjelmoinnin hyviä käytänteiä. Koodia ei pitäisi kirjoittaa kirjoittamisen takia vaan myös sen laadun takia. Koen, että koodin tulisi olla aina kirjoitettu niin hyvin kuin mahdollista. Teoksen pääasialliset teemat ovat mm. koodin kommentointi, hyvät suunnittelu käytänteet sekä puhtaamman koodin säännöt.

Työtehtävät vaativat kykyä toteuttaa, suunnitella sekä jatko kehittää olemassa olevia web palveluita tai web applikaatioita. Sen lisäksi työ vaatii tekijältä vanhojen web arkkitehtuurien ymmärtämistä mm. niiden käyttämiä suunnittelumalleja.

Ohjelmointitaitojen tulee olla vähintään sellaisella tasolla, että kaikki ongelma tilanteet ja bugit pystytään ratkaisemaan ilman ulkopuolista apua. Tilanteen vaatiessa tekijän tulee pystyä tekemään tarvittava taustatutkimus liittyen teknologiaan tai kohdattuun ongelmaan. Käytössä oleva teknologia on vanhaa ja ongelmatilanteilta ei voida sataprosenttisesti vältyä.

Työn aikana seuraavat käsitteet olisivat hyvä olla tuttuja:

- UML – luokkakaaviot
- Olio-ohjelmointi
- Suunnittelumallit
- Palvelin arkkitehtuurit
- Yksikkötestit

Kirjoituksen hetken työpaikkani on Wistec Training Oy, jonka pääasiallinen liiketoiminta on koulutuksien järjestäminen ja myyminen. Lähikoulutusten kyljessä tarjotaan opiskelua myös sähköisesti elektronisella oppimisalustalla, johon on koottu aiheita eri aihealueilta, esim. tietotekniikkaa, tekstin- sekä kuvan käsittelyä ja sosiaalista mediaa.

Työhön liittyvät keskeiset käsitteet on määritelty auki liitteessä 1. Lukija voi tarvittaessa referoida siihen työn lukemisen aikana.

2 Lähtötilanteen kuvaus

2.1 Oman nykyisen työn analyysi

Työtehtäväni Wistecillä on yrityksen elektronisen oppimisalustan ja nettisivujen ylläpitäminen sekä kehittäminen.

Elektronisen oppimisalustan ylläpitäminen sekä kehittäminen pitää sisällään koodaus- ja suunnittelutyötä, joka tapahtuu pienien moduulien muodossa. Yrityksen alusta perustuu Moodleen, jonka arkkitehtuuri säännöstelee kaikki uudet toiminnallisuudet moduulien kautta. Kaikki uudet ominaisuudet kehitetään uusina moduuleina, jolloin alustaan ei ikinä tehdä muutoksia suoraan vaan ainoastaan uusien laajennuksien kautta.

Suunnittelutyö on suurimmilta osin ohjelmiston luokkarelaatioiden, arkkitehtuurin sekä logiikan määrittelyä, johon käytetään UML – luokkakaavio mallia sekä Tila diagrammeja. Laajennuksen määrittely tapahtuu iteratiivisesti koodaus työn kyljessä, jotta itse projektia saadaan myös eteenpäin.

Kehitystyön lisäksi sivutoimena minun tulee myös prosessoida uusia asiakkuuksia. Asiakkuuksien käsittely tapahtuu muun työn ohessa ja se tarkoittaa uusien käyttäjien lisäämistä palveluun. Käyttäjämäärät voivat vaihdella aina muutamasta kymmenestä jopa satoihin riippuen asiakkaasta. Tähän tehtävään on allokoitu muitakin it-henkilöitä aina tarpeen mukaan.

Edellisien määriteltyjen työtehtävien perusteella voidaan heti nähdä, että työtehtävät vaativat tekijältä vähintään hyvät ohjelmointitaidot sekä kyky oppia ja omaksua uusia asioita. Työssä keskeisintä tekijän osaamiselle on kyky ratkaista ongelmia ilman ulkopuolista apua. Projektit toteutetaan yksin, eikä tukea muilta kehittäjiltä ole saatavilla.

Uusien moduulien suunnitteleminen vaatii vähintään perusymmärryksen suunnittelumalleista sekä yleisistä standardeista liittyen luokka relaatioihin:

- Ymmärrystä ohjelmistojen rakenteesta ja toteutus tavasta
- Työmenetelmien kehittäminen ja parempien ratkaisujen löytäminen.
- Ongelman ratkaisu kyvyt. Bugeja ja bugeja kaikkialla.
- Raportointi ja dokumentointi kapasiteetti suhteessa koodaus työhön.

Olettaen, että osaaminen analysoidaan jokaisen yksinäisen tarvittavan taidon perusteella. Tämänhetkiset osaamisalueeni vastaavat tai ehkä jopa hiukan alittavat jossain määrin työtehtävien vaatimukset. Koska työn luonne on enimmäkseen jatkokehittämistä ja ylläpitämistä, työt eivät vaadi sellaista osaamista, joka luokittelisi vaatimukset liian korkealle.

Niin kauan kuin kykenen löytämään ja hyödyntämään uutta tietoa, työtehtävät eivät tule tuottamaan liian isoja vaikeuksia. Siitä huolimatta arvioin tämän hetken osaamistasoni olevan sellaisella tasolla, että pystyn suoriutumaan tehtävistä suurimmilta osin. Joskus tulee tilanteita vastaan, jossa kokemattomuuteni paistaa vahvasti. Esimerkiksi, jos vastaan tulee vaikeita ongelmia, ratkaisun löytämiseen voi kulua minulta enemmän aikaa kuin kokeneemmalta tekijältä, tai jos löydän ratkaisun, niin toteutukseni ovat todella naiiveja. Sen lisäksi kokemattomuuteni vaatii minulta enemmän panosta, jotta saisin aikaiseksi saman työpanoksen.

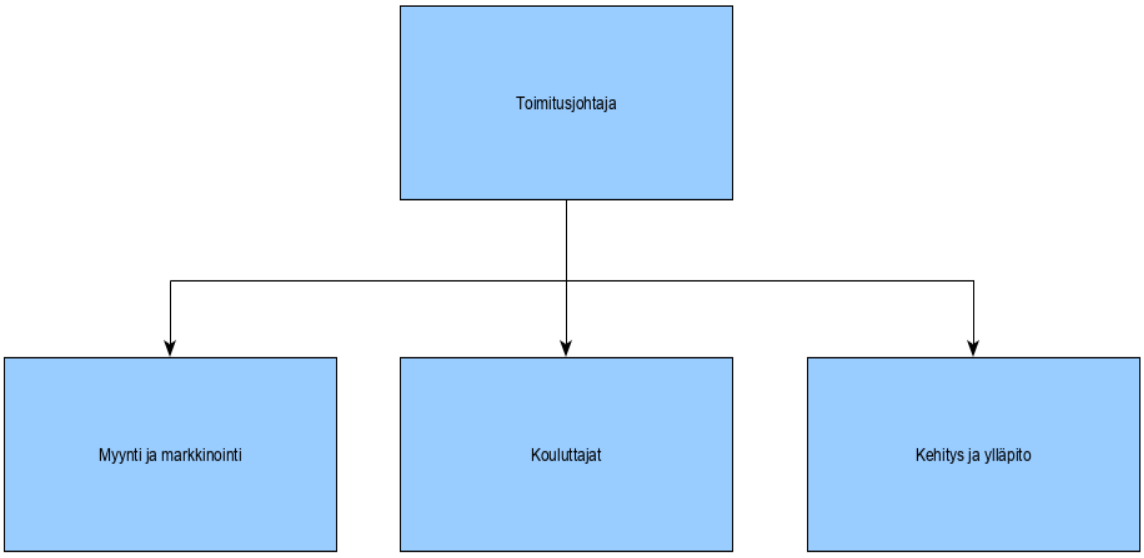
Tällä hetkellä ammatillinen kehittymiseni on noin junioritason loppupäässä. Pohjaa on rakennettu ja se on hyvällä mallilla. Työn vaatimukset myös pakottavat minua kehittymään nopeammalla tahdilla, kuin ehkä normaalisti olisi tarpeellista. Tämän lisäksi kaikki vapaa-ajan itsenäinen opiskelu sekä syventyminen työn vaatimiin asioihin on vahvistanut pohjaa, jolla tähän työhön on lähdetty.

Kehittyminen näkyy parantuneena työpanoksena sekä kykynä ratkaista ongelmia laatikon ulkopuolelta ja toteuttaa parempia lähestymistapoja tuleviin ongelmiin. Erityisesti sen vaikutukset näkyvät laadukkaampana koodina sekä tuotteina, jotka tulevat julkaisuun alustalle.

Päiväkirjan kirjoittamisen aikana haluan painottaa oman ammatillisen kehittymisen enemmän arkkitehtuurien ymmärtämiseen sekä bugittomamman koodin kirjoittamiseen.

2.2 Sidosryhmät työpaikalla

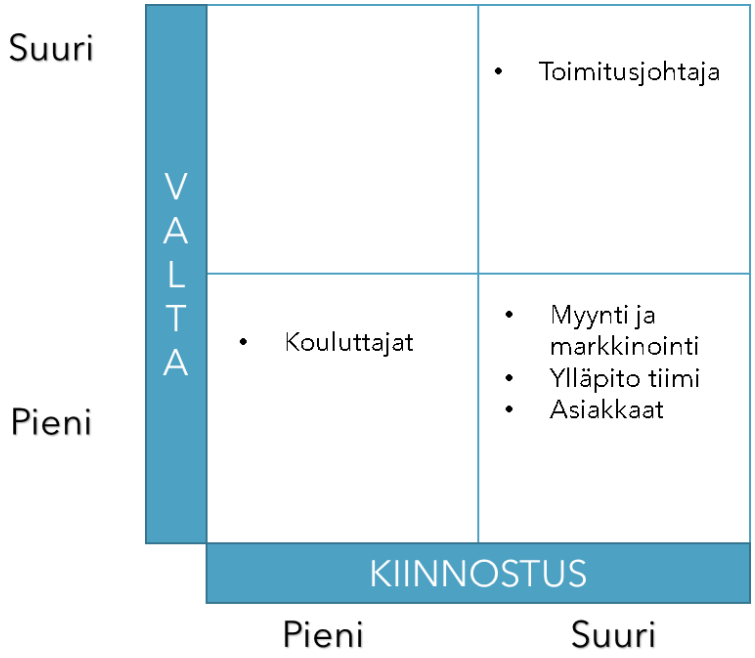
Nykyisen työpaikkani sidosryhmät on luokiteltu kuvion (ks. Kuvio 1) mukaisesti ja tärkeänä huomiona haluaisin mainita, että kaikki yrityksen roolit ovat joustavia. Suurimmalla osalla yrityksen sisäisillä sidosryhmien jäsenillä on yhdistettyjä työnimikkeitä ja näin ollen voivat kuulua muihin sidosryhmiin samanaikaisesti. Esimerkiksi kouluttajat voivat tehdä koulutuksien lisäksi oman erikoistumisalansa tehtäviä yrityksen sisällä.



Kuvio 1. Yrityksen sisäiset sidosryhmät

Omaan työhöni vaikuttaa eniten (ks. Kuvio 2) Myynti- ja markkinointi tiimi, jonka pääasiallinen vastuu on verkko ja etäkoulutusten myynti yritysasiakkaalle. Hyvin usein asiakkailta voi tulla suoria toiveita tai kehitys ehdotuksia, jotka tulevat aina suoralla viestillä kehitystiimin

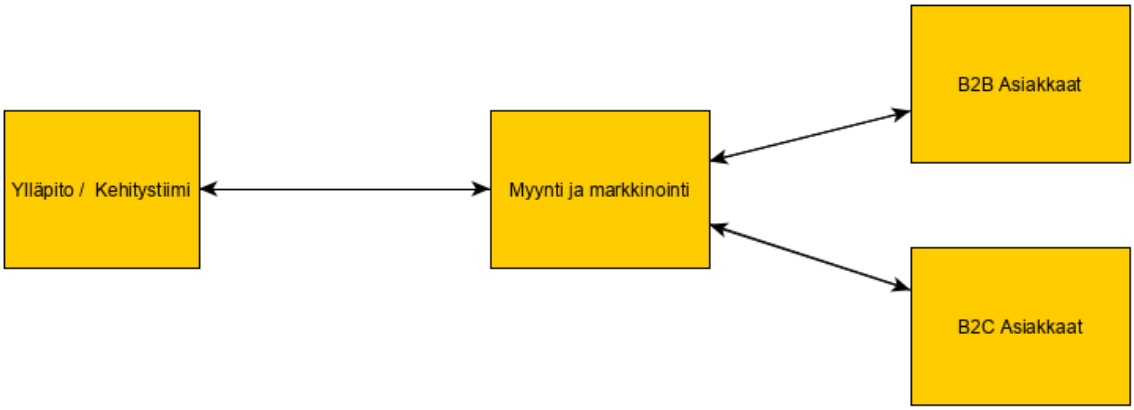
tietoon. Koska suurimmat päätökset tehdään yleensä kehitysjohtajan kautta, päätin määritellä myynti ja markkinointi tiimin vallan vähäiseksi.



Kuvio 2 Sidosryhmien valta sekä kiinnostus tekijän työtä kohtaan

2.3 Vuorovaikutustaidot työpaikalla

Työ tehtävien suorittaminen Wistec:in ylläpito tiimissä on pitkälti itsenäistä ja siksi kaikki kommunikaatio sekä vuorovaikutus kollegoiden kanssa tapahtuu yleensä silloin, kun joku tarvitsee toiselta konsultointia omaan projektiinsa. Projektien vetäminen ja edistäminen on jokaisen yksilön omalla vastuulla ja jokainen projekti määräytyy yrityksen hetkellisten tavoitteiden mukaan. Yleensä kehitettävät ominaisuudet tai projektit otetaan yrityksen kokonaisvaltaisesta tiekartasta, joka ohjaa yrityksen kehitystoimintaa.



Kuvio 3. Sidosryhmien kommunikaatio

Yrityksen toimintatapojen sekä dynamiikan takia kommunikaatio sekä vuorovaikutus työpaikalla tulee olemaan tapahtumaan suurimmilta osin sähköposti viestittelynä sekä ajoittain kokouksina, joissa työn edistymistä seurataan. Tähän vaikuttaa myös osaltaan se, että työntekijät voivat halutessaan tehdä töitä etänä.

Tiimien välinen kommunikaatio väylä tapahtuu vain muutamaan suuntaan ja tätä on yritetty kuvastaa esimerkissä (ks. Kuvio 3) kuviolla, jossa näkyy sidosryhmien välinen kommunikaatio. Ison huomion voi kiinnittää se, että ylläpito tiimi ei kommunikoi asiakkaan kanssa suoraan vaan sen välissä toimii myynti ja markkinointi, joka hallitsee kaikki relaatiot sekä yritysasiakkaisiin, että myös yksityisiin henkilöihin.

Mainitsemieni seikkojen vuoksi viestiminen ja vuorovaikutus sidosryhmien välillä voi olla ajoittain takkuavaa ja voi aiheuttaa viiveitä asioiden käsittelyssä. Erityisesti silloin, kun ylläpito tiimiltä halutaan raportteja, joita asiakkaalle halutaan välittää.

3 Päiväkirjaraportointi

3.1 Seurantaviikko 1 20.1–24.1

Maanantai 20.1.2020

Normaalilla aikataululla tänään olisi ollut yksikkötestausten kirjoittamista, mutta testejä kirjoittaessani koodista löytyi bugi, joka ei ilmennyt testien ajon aikana vaan testi ympäristössä. Tämä on ehkä harvoja tilanteita, jossa yksikkötesti ei pystynyt ottamaan saatua virhettä huomioon koska sen mukaan kaikki toimii niin kuin pitää ja ongelmia ei pitäisi olla. En löytänyt ratkaisua tähän ongelmaan tänään, mutta aikeena olisi keskittää resurssit sen

korjaamiseen, sillä laajennusta ei missään nimessä voida julkaista niin kauan, kun äskettäin mainittu bugi on ohjelmassa.

Kyseinen bugi aiheuttaa duplikaatti elementtejä array listaan, joka näyttäisi asiakkaalle siltä, että tietokannassa on kopioita samasta tiedosta. Yritin saada ongelmaa selvitettyä seuraamalla omaa koodin jalanjälkeä alusta loppuun niin, että käyn läpi kirjoittamani koodin järjestyksessä. Toivottavia tuloksia tämä ei kuitenkaan tuottanut sillä koodi näyttää ajavan niin kuin sen kuuluisi.

Huomenna yritän pilkkoa ongelman pienempiin osiin ja toteuttaa lisää testejä sekä testi tilanteita, jotka on erityisesti tähdätty ohjelman niihin osiin, joiden voidaan todeta tuottaneen tämän bugin. Näin varmistetaan vielä pienempien osien logiikka ja toiminta.

Tiistai 21.1.2020

Jatkoin tänään ongelman selvittelyä eilisestä. Kirjoitettuani lisää testejä huomasin ajon aikana, että olen käyttänyt JavaScript splice funktiota slice ominaisuuden sijaan, joka tarkoittaa, sitä että lista jaetaan aina kahtia konstruktoria kutsuessa. Konstruktori kutsutaan useammin kuin kerran ajon aikana, joten sen vuoksi array lista näytti duplikaatti elementtejä. Jatkossa olisi hyvä kiinnittää huomiota enemmän funktioihin, joiden toiminnallisuudesta en ole sataprosenttisesti varma. Teen tästä kattavan yhteenvedon vielä analyysissa.

Bugi metsästyksen lisäksi päätin uudelleen järjestää koodipätkän, joka käytti mainitsemaani JavaScript funktiota. Vaikka koodi oli kirjoitettu vain muutamaa viikkoa aiemmin, huomasin heti löytäneeni paremman ratkaisun, kun lähestyin samaa ongelmaa uudelleen. Koodin määrä pieneni kolmestakymmenestä rivistä vain kymmeneen.

Aikaisempi ratkaisu käytti hyväkseen tuplasilmukoita sekä array funktioita, jotka lisäävät prosessointi aikaa vain entisestään. Kaikki nämä operaatiot tehtiin, jotta listasta saataisiin ulos sisältöä sivutettuna.

Uudella ratkaisulla sisällön sivutus tehdään slice funktiolla, jolle annetaan ensin listan indeksi, josta sivu alkaa ja sen loppu. Jotta tiedetään aloitus- ja lopetuskohta, indeksi luku lasketaan seuraavalla kaaviolla: sivunumero * sisällön määrä per sivu. Lopetus indeksi lasketaan ensimmäiseen operaation jälkeen kaaviolla aloitus indeksi + sisällön määrä per sivu. Jälkimmäisen operaation ympärille on vielä sidottu clamp funktio, joka varmistaa sen, ettei lopetus indeksi mene listan koon yli.

Keskiviikko 22.1.2020

Tämän päivän teemana jatkui edellispäivän implementoitu ratkaisu, jonka toiminnan varmistamiseksi kirjoitin vielä muutaman yksikkötestin. Varmistettuani laajennuksen käyttöliittymä käyttäytyvän niin kuin se on määritelty, siirryin pikaisesti poistamaan vanhan koodin ja kommentoin vielä yhden luokan kokonaan.

Seuraavana tehtävä oli testata laajennus kehitysympäristössä vielä muutamalla käyttötilanteella, jossa pyrin simuloimaan oikeaa tapausta niin tarkasti kuin mahdollista. Tässä tapauksessa aikeena oli testata miten laajennuksen eri osat käyttäytyvät suhteessa Moodlen järjestelmään, kun tiettyjä muutoksia tehdään kurssien tai kurssi suoritusten asetuksiin.

Tämän jouduin tekemään manuaalisesti, koska tämän hetken ymmärtämykseni pohjalta Moodlen omat yksikkötestit eivät pysty simuloimaan esimerkiksi kurssi aktiviteettien merkintää suoritetuksi ilman, että sen koodia ruvettaisiin sorkkimaan. Moodlella on yksikkötestaukseen rajapinta, joka tarjoaa tyypillisen PHP yksikkötestien päälle funktioita. Joiden avulla laajennukset voidaan testata Moodlen järjestelmän eri rajapintojen tai järjestelmän osilla. Eli näiden testi tapausten toteuttamiseksi minun pitäisi etsiä koodinpätkä, joka merkitsee kurssin aktiviteetin suoritetuksi. En kuitenkaan saanut suorituksien seurantaa toimimaan Moodlen yksikkötestaamisen rajapinnalla, joten päätin tehdä testauksen manuaalisesti.

Torstai 23.1.2020

Tänään oli aikeena aloittaa seuraavan ominaisuuden työstäminen laajennukseen. Tiivistettynä ominaisuuden ideana olisi lisätä puhekupla, jokaisen listan elementin kortin kylkeen, kun käyttäjä vie hiirensä sen päälle.

Ennen kuin siirryin kirjoittamaan logiikkaa backend puolelle, päätin tutkia mahdollisia olemassa olevia ratkaisuja kyseiselle ominaisuudelle. Löysin hyvin nopeasti html esimerkin, joka näytti toimivan erinomaisesti.

Seuraavaksi siirryin puhekupla ominaisuuden työstämiseen. Puhekupla näyttää halutusta kurssista lisää tietoa ja kyseisten tietojen haku operaatiota ei ollut vielä kirjoitettu. Kaikki uudet laajennuksen operaatiot kirjoitetaan joko julkiseen kirjastoon tai lokaali-kirjasto

tiedostoon riippuen siitä halutaanko funktiosta tehdä julkista muille Moodlen osille. Tässä tapauksessa päätin kirjoittaa suoraan laajennuksen julkiseen kirjastoon, koska oletettavasti funktionaalisuutta käytetään muissakin tulevissa laajennuksissa.

Kirjoitin vielä lopuksi yksikkötestit, jotka testaavat aktiviteettien laskemista ja varmistaa, että operaatio palauttaa odotetut tulokset. Yksikkötestien aikana ilmeni ongelmia, jotka juontavat juurensa kirjasto luokaan. Joudun tutkimaan asiaa tarkemmin huomenna.

Perjantai 24.1.2020

Jatkan tänään eilen aloitettua ominaisuutta, joka otti takapakkia yksikkötestien epäonnistumisen takia. Tavoitteena on saada tänään kyseinen ominaisuus tehtyä kokonaan loppuun. Tämän tehtävän lisäksi pidän palaverin nimettömäksi jätettävän kollegan kanssa, joka on erikoistunut käyttöliittymien suunnitteluun.

Ominaisuuden kehittäminen eteni hyvin ja se on suurimmilta osin valmis muutamia viilauksia lukuunottamatta. Rakensin työkaluvihjeen ulkonäön loppuun ja lisäsin kurssinkuvauksen siihen. Kurssi kuvaukset voivat olla niin pitkiä, että ne eivät mahdu määriteltyyn kokoon, joten tämän ympärille oli tarve kirjoittaa tukifunktio, joka palauttaa lyhennetyn tekstin.

Pyysin hänen mielipidettään laajennuksen käyttöliittymän tämän hetken tilaan sekä demonstroin sitä. Demonstraation aikana tuli ilmi bugeja. Mm. yksi bugeista, rikkoo kurssilistauksen sivutuksen. Sen pitäisi palauttaa käyttäjän takaisin edelliselle sivulle, mikäli kyseistä ei ole olemassa, mutta nyt se näyttää vain tyhjää. Kaikki löydetyt bugit on otettu ylös ja lisätty projektihallinta tauluun.

Viikkoanalyysi

Oppimisen kannalta viikko oli erittäin tuottava, sillä työnaikana kehityin muutamalla alueella: Ongelman ratkaisemisessa sekä paremman koodin kirjoittamisessa. Kykenin tunnistamaan vanhan koodin tekemät virheet sekä naiivit ratkaisut.

Työhön liittyvien asioiden selvittelyyn ei mennyt paljon aikaa, ja aina kun jokin tarvitsi tausta tutkimusta, pystyin lukemaan dokumentaatioita. Esimerkiksi tällä viikolla etsin tietoa mm.

JavaScript funktioista sekä mahdollisista html ratkaisuista liittyen puhekupla-ominaisuuteen.

Viikon aikana ilmenneet ongelmat liittyivät vahvasti huonosti toteutettuun koodiin, sekä puuttuviin yksikkötesteihin käyttöliittymässä. Kaikki ongelmat saatiin ratkaistua kunnon yksikkötesteillä.

Koska viikon teema pyöri yksikkötestien ympärillä vertailen tässä omaa tekemistäni teoksen teoriaan. Beck. K kuvailee testi lähtöisen prosessin olevan työskentely tapa, jossa testit kirjoitetaan ennen, kuin koodia kirjoitetaan riviäkään applikaatioon.

Hänen kolmen askeleen prosessin ideana, on että kehittäjä havainnollistaa testin kirjoittamisen aikana kaikki toivotut rajapinnat ja operaatiot. Tavoitteena ei ole testin läpäisy niin nopeasti, kuin mahdollista, vaan koodin laatu. Testit kirjoitetaan mahdollisimman naiivilla ratkaisulla, joka läpäisee testin. Tämän jälkeen koodia uudelleen järjestetään järkevämpään ratkaisuun, joka läpäisee testit.

(Beck.K, “Test-Driven Development by example”, 2002, 11.)

Oma tekemiseni verrattuna teoksen kuvaamaan prosessiin on ollut hyvin naiivia. Sen sijaan että olisin kirjoittanut testit ensin ennen uuden ominaisuuden kehittämistä, hyppäsin suoraan koodiin ja testasin vasta kun ongelmia ilmeni työn aikana. Jatkossa pyrin kirjoittamaan haluamani koodin pätkän testeihin ennen sen kirjoittamista.

Beck. K kuvailee teoksessaan myös yksikkötestien väitteitä ja miten niiden tulisi kertoa tarinaa ohjelman operaatioista ja miten niiden tulisi toimia. (Beck.K, “Test-Driven Development by example”, Luku 25. “Patterns for test driven development”, 2002).

Tätä teemaa en noudattanut työssä ollenkaan vaan lähestyin testaamista ominaisuudet sekä funktionalisuus mielessä. Eli testasin aina, jokaisen funktion sekä luokan kerrallaan. Tulkitsen Beckin sanomaa tarinoista myös käyttäjätarinoina, joka antaisivat heti työn alussa jonkinlaisen osviitan, jonka mukaan koodia voidaan testata.

Jos jotain positiivista voidaan tekemisestäni ottaa se olisi tapa, jolla olen kirjoittanut testidataa. Beck mainitsi, että testidatan tulee olla kirjoitettu ihmiselle luettavaksi ja kaikkia abstraktioita tulee välttää niin paljon kuin mahdollista.

(Beck.K, “Test-Driven Development by example”, Luku 25. “Patterns for test driven development”, 2002).

Datan lisäksi olen onnistuneesti pitänyt kaikki testit toisistaan riippumattomina. En ollut ollenkaan edes miettinyt asiaa vaan oletin automaattisesti, että mitään liitännäisyyksiä ei tulisi olla testien välillä. Ideana on, että testit eivät riipu toisistaan ja kehittäjä pystyy halutessaan jättämään testejä ajamatta, joilla ei ole merkitystä sen hetken työhön.

(Beck.K, “Test-Driven Development by example”, Luku 25. “Patterns for test driven development”, 2002).

3.2 Seurantaviikko 2 27.1–31.1

Maanantai 27.1.2020

Tänään kurssi laajennuksen kehittäminen jatkuu bugi korjauksilla sekä hakuominaisuuden ratkaisun kehittämisellä. Yritän hyödyntää viime viikolla oppimiani asioita työskentelyn aikana.

Kertaan vielä tässä viime viikon palaverin aikana ilmennyttä bugia, jota olen lähtemässä korjaamaan. Kyseessä on sivutus bugi, jossa käyttöliittymä yrittää näyttää olemassa olemattomien sivujen sisältöä ja oletuksena näyttää käyttäjälle virheviestin. Päätin silmäillä koodia nähdäkseni mikä logiikassa saattaisi aiheuttaa kyseiset ongelmat. Ei mennyt kuin hetki niin tajusin, että clamp funktio, jonka olin kirjoittanut otti huomioon vain tilanteet, jossa sivun määrä ylitettiin korkeimmasta mahdollisesta. Ideana oli, että tukifunktio tarkistaisi parametrinä annetun arvon ja vertaisi sitä korkeimpaan mahdolliseen sivunnumeroon palauttaen, joko tuloksen tai parametrin riippuen tuloksesta.

Ongelman korjaaminen alkoi kahden yksikkötestin kirjoittamisesta. Molemmat toteuttavat naiivit ratkaisut kahdesta clamp funktion versioista, joista toinen tarkistaa vertaa minimaaliseen arvoon ja jälkimmäinen korkeimpaan arvoon. Kun yksikkötestit menivät läpi, korjasin naiivin ratkaisun puhtaampaan muotoon. Tämän jälkeen poistin koodin testeistä ja paikkasin ne funktiokutsuilla tukitiedostosta. Ongelma korjaantui ja pääsin seuraavaksi kehittämään hakuominaisuutta.

Hakuominaisuus on ollut hiukan haastavampi kuin olin ensiksi luullut. Aikeena on saada aikaan logiikka, joka pystyy suodattamaan kurssit asteittain aina lisäämällä uuden filtterin,

kun käyttäjä lisää ne käyttöliittymästä. Vaikka tällä hetkellä suodatus tapahtuu vain hakusanan sekä kategorian muodossa haluaisin koodin tukevan tulevaisuuden ekstensiota oletuksena.

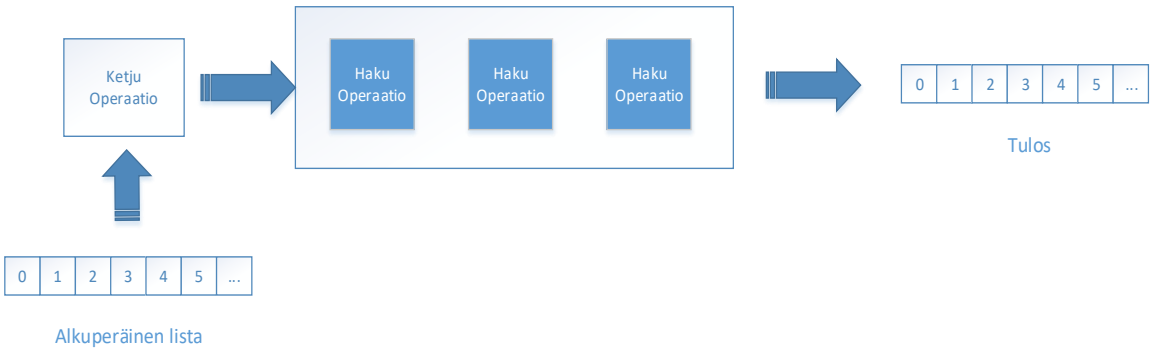
Ehdin tänään aloittamaan uuden ratkaisun suunnittelua. Ideoin mallia, jossa kaikki suodatus tehdään dictionary listan perusteella, jossa hakufiltteri on avain ja funktio sen arvo. Suodatus aloitettaisiin yhdellä funktiolla, joka kutsuu filtteri dictionarya ja toteuttaa kaikki parametrinä annetut filtteri operaatiot.

Tänään koen ottaneeni hiukan takapakkia henkilökohtaisessa kehityksessäni. Olen toistanut viime viikon virheitä, jotka johtivat bugeihin. Jatkossa minun tulee olla tarkempi, kun teen ajatustyötä., jotta turhilta bugeilta vältyttäisiin.

Tiistai 28.1.2020

Tänään jatkan eilen aloitettua suodatus- sekä hakuominaisuutta. Suunnittelen ja kehitän ratkaisun, joka tekee haku operaatioita array listaan kerroksittain filtteri tyypin mukaan. Filtteri tyypit on määritelty niiden kohteiden perusteella, esimerkkinä Hakusana ja Kategoria. Ideana on, että operaatiot tehdään ikään kuin ketjuina (Ks. Kuvio 4) ja jokainen operaatio tekee haun edellisen operaation tulokseen, kunnes viimeinen operaatio on saavutettu.

Päivän tähtäimenä olisi kehittää kykyäni kirjoittaa parempaa koodia, sekä hyödyntää aikaisemmin oppimiani yksikkötestaamisen malleja työskentelyn aikana.



Kuvio. 4 Ketjutetun hakuoperaation logiikka

Ominaisuuden kehittäminen eteni suhteellisen nopeasti ja sainkin sen toimimaan täysin ennen päivän päättymistä. Kirjoitin yksinkertaisen Luokan, joka määrittelee eilen mainitsemani

haku operaatiot sekä erillisen funktion, joka ajaa kaikki operaatiot. Hyödynsin luokan kirjoittamisessa viime viikolla käsittelemääni yksikkötestausta, eli testasin funktionaalisuuden testeillä, ennen kuin integroin sen laajennukseen.

Samalla kun yksikkötestasin hakuoperaatio luokkaa, päätin laajentaa testidata generaattori luokkaa muutamalla funktiolla, jotka mahdollistavat tekstimuotoisen testidatan generoinnin. Tämä on toteutettu siten, että generaattorille on määritelty teksti arvo, joka sisältää kaikki aakkoset.

Tänään positiivisesti en tuottanut yhtäkään bugia, huomasin yksikkötestien tarinoiden selventämisen auttaneen varmistamaan koodin toiminnallisuuden. Työtehtävätkin saatiin vietyä tänään päätökseen.

Keskiviikko 29.1.2020

Tänään tehtävänä on dokumentoida ohjelmiston muutokset olemassa olevaan tekniseen dokumenttiin, johon on määritelty laajennuksen luokka rakenteet, sekä sovellus logiikkaa. Dokumentoinnin lisäksi minun pitää vielä testata laajennusta testiympäristössä ja löytää mahdollisia bugeja, jotka eivät ole tulleet ilmi kehityksen aikana tai nopeiden yksikkötestien yhteydessä. Jos ehdin, niin kirjoitan lokalisaatio tiedostoja päivän loppuun mennessä.

Dokumentoin laajennuksen luokka relaatioiden muutokset tekniseen dokumenttiin. Tällä hetkellä olen käyttänyt UML- luokkamerkintöjen tekemiseen vain muutamia merkintöjä, joista tulee ilmi relaatiot luokkien välillä, sekä niiden väliset riippuvuudet. Kaikki mahdolliset merkinnät ja päivitykset tehtyäni siirryin seuraavaan tehtävään: Lokalisaation kirjoittaminen.

Laajennuksen lokalisaatio on toteutettu yksinkertaisessa dictionary muodossa. Oletuksena Moodlen järjestelmä etsii kielitiedostot laajennusten hakemisto rakenteesta, joka on seuraavanlainen: ”root/blocks/moduulinimi/lang/en/moduulinimi.php”. Huomaa että kielen nimi on määritelty lyhenteenä kansiona. Jokainen erillinen kieli tiedosto määrittelee globaaliin dictionary listaan omat käännöksensä, joita moduuli käyttää.

Varsinaisesti kehitystä ei tänään tapahtunut, mutta opin hiukan CSS animaatioista etsiessäni tapaa tehdä animaatioita ilman JavaScriptiä.

Torstai 30.1.2020

Lokalisaation kirjoittaminen laajennukseen jatkuu tänään. Sen rinnalla minun pitää vielä puhdistaa laajennus kaikesta turhasta koodista, jota on jäänyt aikaisempien testien jäljiltä. Softa on muuten valmis julkaistavaksi, mutta ennen sitä päätin pitää vielä yhden kokouksen kollegan kanssa. Kokous pidetään huomenna ja mikäli kaikki on ok laajennuksen paketointi ja asennus voidaan aloittaa.

Päätin myös käyttää mahdollisuudet hyväkseni ja uudelleen järjestää koodia hieman. Huomasin, että nimeämistavat eivät olleet johdonmukaisia, Käsittelen tätä varmasti hiukan viikkoanalyysissa.

Sain lokalisaatiot kirjoitettua tänään loppuun ja sen tiimoilta mitään ongelmia ei ilmennyt.

Perjantai 31.1.2020

Tänään ohjelmassa on kuvan hakuoperaation korjaaminen. Ideana on, että laajennuksen asetuksista voidaan määritellä oletus kuva kurssille, joka sitten näytetään listauksessa, jos kurssille ei ole määritelty kuvaa. Olin yrittänyt toteuttaa tätä aikaisemmin, mutta sen kanssa oli ilmennyt ongelma, jotka liittyvät Moodlen järjestelmän aiheuttamiin vaikeuksiin. Ensinnäkin Moodlen järjestelmästä on olemassa todella vähäiset dokumentaatiot ja kun halutaan kirjoittaa uusia ominaisuuksia tai käyttää sen rajapintoja, tulee minun lukea koodia.

Asiaa ei auta se, että Moodlen koodi on aivan hirveää spagettia, jonka ikuisen ehtolausekkeiden meren seasta pitäisi löytää logiikkaa, joka auttaisi ymmärtämään sen toimintaperiaatetta. Selailtuani koodia hetken päätin, että kyseisen ominaisuuden toteuttaminen jäisi julkaisun ulkopuolelle. Merkitsin tehtävän työlistaan, ja se suoritetaan vasta julkaisun jälkeen muiden projektien kanssa samaan aikaan.

Kollegani kanssa pidetyn kokouksen aikana käytiin läpi kaikki laajennuksen lokalisaatioon sekä ulkonäköön liittyvät asiat. Keskustelumme aikana päätimme viimeisistä muutoksista, jotka käyttöliittymän tyyleihin tehdään ennen julkaisua. Kokouksen päätteeksi sovimme julkaisuajankohdaksi seuraavan viikon keskiviikon illan jälkeen, jolloin käyttäjiä on kirjautuneena paljon vähemmän palvelussamme. Pyrimme tekemään ylläpito toimenpiteet aina arki työaikojen ulkopuolella.

Kehitystä ei tänäänkään ollut havaittavissa työtehtävien luonteen sekä muiden prioriteettien takia.

Viikkoanalyysi

Tällä viikolla kehittymiseni on jäänyt hiukan vähemmälle, mutta koen silti saaneeni paljon irti kehittäessäni ratkaisua hakuoperaatioiden kerrostamiseen. Kirjoittamani ratkaisu pystyy suorittamaan operaatiot kirjoittamalla silmukan, joka vertaa avaimia saatuihin arvoihin. Ainoa kriittinen piste ratkaisulla on se, että filtterit täytyy määritellä staattisesti sekä lomakkeeseen, että filtteri luokkaan. Opin sen kirjoittamisen aikana uusia tapoja ajatella ongelmia sekä ratkaista ne ilman, että operaatioista tulee liian kalliita järjestelmälle.

Toteuttamani hakuoperaatiot kuitenkin tuntuivat naiiveilta, joten haluan verrata ratkaisuani olemassaoleviin tapoihin toteuttaa hakuoperaatioita. Michael McMillanin teos “Datastructures and algorhitms with javascript” käsittelee tyypillisimpiä datarakenteita sekä algoritmeja, joilla operaatioita tehdään. Kirjan esimerkit ovat JavaScript kielellä, jota myös käytetään käyttöliittymien kirjoittamiseen työssäni.

McMillan määrittelee tyypillisimmeksi hakualgoritmeiksi lineaarisen sekä binaarisen puun, joista jälkimmäistä käytetään aina kun etsittävä lista on järjestetty jossain määrin suurimmasta pienimpään. Sen ideana on vähentää operaatioiden tarvetta tekemällä haun aina sinne, minne arvon mukaan olisi järkevintä. Lineaarista hakua käytetään silloin, kun listan sisältö on täysin satunnaisessa järjestyksessä.

(McMillan M, “Datastructures and algorhitms with javascript”, Luku 13. 2004)

Vertaillen omaa tekemistäni huomaan tässä kohti, että toteuttamani hakuoperaatio vastaa teoksessa käsiteltyä lineaarista hakuoperaatiota ja sen pohjalta olisikin hyvä pohtia onko meidän tapauksessamme järkevää toteuttaa binääristä hakualgoritmia, kun kursseja voi yhdellä käyttäjällä olla korkeintaan 30–40 kappaletta. Tulevaisuutta miettien tehokkaamman hakuoperaation toteutus varmistaisi palvelun skaalautumisen aina kurssien määrän kasvamisen kanssa.

Hakuoperaatioiden kehittämisen aikana oppimani malli Beck. K:n teoksesta auttoi sen jouhevuudessa, mutta teos ei siltikään käsitellyt asioita vanhan koodin näkökulmasta. Päätin ottaa käsittelyyn Michael Feathersin teoksen “Working Effectively with Legacy Code”, jossa käsitellään miten vanhan koodin kanssa tulee työskennellä ja mitä askelia siihen kuuluu.

Koska meidän ei tarvitse työssämme varsinaisesti muokata olemassa olevaa arkkitehtuuria sen toimivuuden takia. Käsittelemme varsinaisesti vain sitä osaa Feathers:in kirjasta, jossa käsitellään uusien ominaisuuksien kehittämistä.

Huomasin kuitenkin, että Feathers määrittelee, että vanhentunut koodi on yleensä testaamatonta ja jää muhimaan tiedostojen syvyyksiin, kunnes ohjelma menee rikki. Hänen mukaansa on parempi kohdata testaamaton koodi ennemmin kuin myöhemmin ja tämä onkin hyvä vertailu kohde, sillä olen työssäni yksikkötestannut kaikki mahdolliset osat sekä testauttanut vielä erikseen järjestelmää vasten.

(Westby Emma “Git for Teams”, Luku 8. “Patterns for test driven development”, 2004)

Hän myös määrittelee parhaaksi lähestymistavaksi testi lähtöisen testaamisen mallin, jota käsiteltiin jo viime viikolla, mutta olin harmikseni missannut muutaman tärkeän seikan, joka oli jäänyt Beckin abstraktioiden alle. Feathers määritteli, että testilähtöisessä testaamisessa uudet ominaisuudet toteutetaan kirjoittamalla testi ensin ja vasta sitten itse funktio tai ominaisuus.

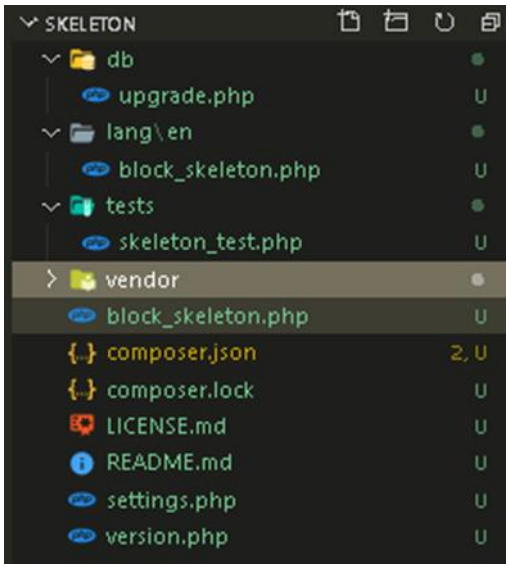
(Feathers M “Working Effectively with legacy code”, Luku 8. “Patterns for test driven development”, 2004)

3.3 Seurantaviikko 3 3.2–7.2

Maanantai 3.2.2020

Tämän päivän tehtävinä ovat luuranko projektin rakentaminen uusia Moodle laajennuksia varten. Uusia projekteja pystytään aloittamaan paljon nopeammin sekä vähemmällä vaivalla. Tehtävän suorittamiseksi minun pitää ensin luoda tyhjä repositorio ja rakentaa kaikki vaadittavat kansiot sekä tiedosto rakenteet, joita tyypillinen Moodle laajennusprojekti tarvitsee.

Sain toteutettua luuranko projektin, jota voidaan muokata tarpeiden mukaan mihin tahansa laajennuksen kehittämiseen. Päätin oletuksena käyttää palikka laajennus tyyppiä, sillä se on yleisin laajennus tyyppi, jota työtehtävissäni kehitetään. Kuvassa (ks. Kuva 1) näkyy esimerkki laajennuksen kansio rakenteesta, jossa on kaikki vaaditut osan sen toimiseen. Jouduin vielä erikseen konfiguroimaan phpunitin laajennuksen omien osien testaamiseen.



Kuva. 1 Laajennuksen minimaalinen kansiorakenne.

Tiistai 4.2.2020

Tänään työtehtävänä on vanhemman projektin tilanteen kartoitus. Kyseessä on todistus laajennus, jonka avulla opiskelijan tulisi pystyä tulostamaan tai lataamaan pdf tiedosto opinnoistaan. Sen suunniteltu julkaisu ajankohta oli vähän ennen uuden kurssilistaus-projektin aloittamista, mutta monien ongelmien ilmennettyä jouduin kuitenkin keskeyttämään sen toisen projektin vuoksi.

Työtehtävien aikana ilmeni monia ihmeellisiä ongelmia, kuten esimerkiksi oman koodin huono luettavuus, jonka takia tilanteen kartoituksessa meni oletettua pidempään ja sen päälle todella huono testauskäytäntö, joka juontaa juurensa profiili laajennukseen, joka on todistus laajennuksen ainoa vaadittu liitännäisyys.

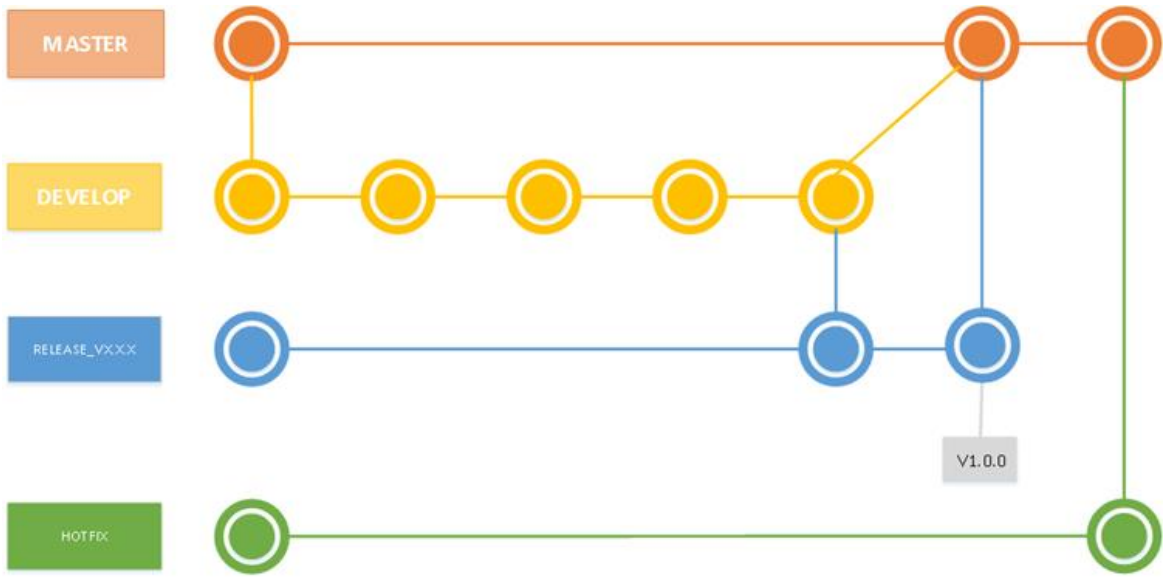
Seuraavaksi pitäisi kirjoittaa testit profiili laajennukseen, joka varmistaisi ensin kaikkien operaatioiden varman toiminnan, ennen kuin sertifikaattiin voidaan tehdä yhtäkään muutosta ilman, että kaikki räjähtäisi jauheeksi.

Keskiviikko 5.2.2020

Tänään työtehtävänä oli julkaisun valmistelu sekä laajennuksen asentaminen tuotantopalvelimelle testattavaksi. Asennus prosessi koostuu muutamasta askeleesta, joka alkaa julkaisun tekemisellä GitHubiin, josta se sitten ladataan pakettina ja asennetaan selaimen kautta Moodle alustalle.

Asennusta tehtiin arjen työpäivän ulkopuolella yhden kollegan kanssa yhdessä. Klo 16 jälkeen palvelussa on vähän liikennettä, jolloin kaikki mahdolliset muutokset eivät häiritse asiakkaita niin pahasti. Asennuksen aikana ilmeni kuitenkin ongelmia, jotka aiheuttivat todella pitkän validoinnin johon meillä ei ollut ihan heti vastausta, kunnes tajusin tarkistaa projektin kansio rakenteen ja mitä näinkään. Node moduuli kansio ei ollut merkittynä gitignore tiedostoon. Moodlen järjestelmä yritti siis tarkistaa 17 tuhatta tiedostoa.

Kun korjasimme node moduulin pois hotfix haaraan, testasimme laajennusta tuotantopalvelimella, josta löysimme muutaman pienen bugin. Nämä merkittiin heti korjattaviksi ennen laajennuksen käyttöönottoa. Näiden lisäksi löytyi yksi iso bugi, jossa Moodlen järjestelmä lisää tyhjän välilyönnin kurssin alakategorian alkuun. Nopea ratkaisu, joka ei vaatisi paljoa työtä olisi alakategorioiden eliminointi. Päätimme siis vielä varmistaa, että kyseinen ratkaisu on ok seniori kollegan kanssa.



Kuvio 5. Git workflow

Git repositoriin tehtiin hotfix haara, johon kaikki muutokset tehdään nopeasti. Kun muutokset olivat valmiita squashashin kaikki commitit yhteen ja tein yhteenvedon korjauksesta. Ylläoleva kuvio (Ks. Kuvio 5) kuvastaa työpaikalla käytettävää Git mallia, jonka mukaan softan muutokset tehdään. Perinteisestä mallista poiketen, jossa jokaisella ominaisuudella on oma haaransa. Tämä malli kevennetty sen alkuperäisestä versiosta (ks. Kuvio 6) sopimaan paremmin yhdelle kehittäjälle.

Torstai 6.2.2020

Tänään työpäiväni kesti vain noin 3 tuntia johtuen siitä, että eilinen työpäivä venyi 9 iltaan asti. Sain tänään vastauksen eiliseen meiliin liittyen kategorioiden aiheuttamaan ongelmaan. Tulimme siihen päätökseen, että laajennus julkaistaan perjantaina ja ennen sitä kaikki mahdolliset korjaukset tulisi tehdä.

Loppukädessä laajennus vaati vain yhden korjauksen ja se oli kurssienmäärän korjaaminen kahdeksasta kahteentoista. Korjaus toteutettiin hotfix haaralla niin kuin eilenkin ja kun työ oli saatu päätökseen tein rebasen ja mergesin korjauksen suoraan mestari haaraan. Tämän jälkeen palikka pudotettiin päivitettäväksi Moodleen.

Laajennus on aikataulutettu otettavaksi käyttöön kun asiakkaat ovat saaneet viestin sähköpostitse, joka tapahtuu myyntineuvottelijamme kautta. Hänelle on lähetetty spesifikaatiot uudesta laajennuksesta.

Perjantai 7.2.2020

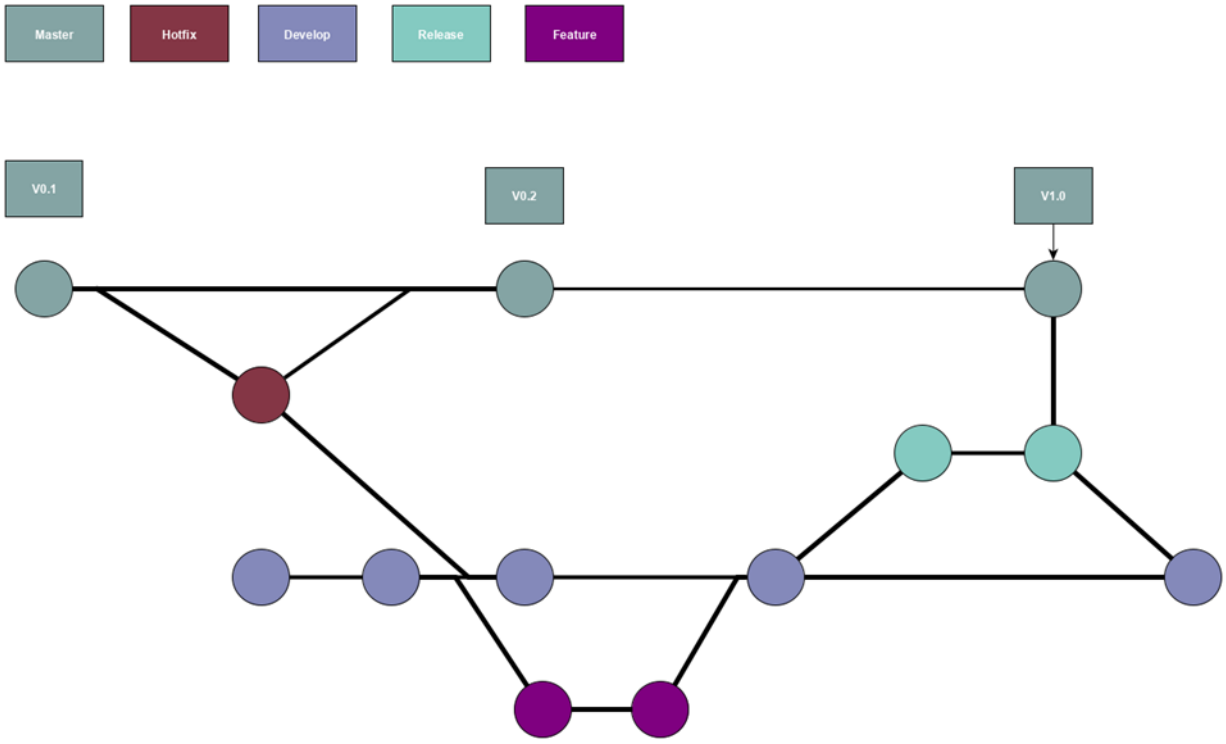
Tänään palasin takaisin työstämään korjauksia profiili laajennukseen, sekä toteuttamaan asetukset sivun, johon voidaan tehdä määrityksiä palikalle. Erityisesti tätä tarvitaan tietokanta osoitteiden sekä tunnuksien määrittelemiseen sen sijaan, että olisi erillinen php tai json tiedosto. Tämä mahdollistaa muutoksien tekemisen käyttöliittymästä suoraan.

Moodlen järjestelmä lukee laajennusten asetukset aina settings.php tiedostosta, johon määritellään ensin pääsyoikeudet ja sitten luodaan asetuspuu, jolle voidaan määritellä erilaisia palautus arvoja tyypin mukaan.

Tehtävän kanssa ilmeni ongelmia, joihin en saanut selkoa tänään. Palikan asetukset palauttavat jostain syystä NULL pointteria, joka viittaa siihen, että asetuksissa ei olisi mitään. Kaikille asetuksille on kuitenkin määritelty oletusarvot, joihin sen pitäisi mennä, mikäli mitään ei ole kirjoitettu. Meidän tapauksessamme kyseessä on jostain ihan muusta.

Viikkoanalyysi

Tämän viikon päätteeksi haluaisin käsitellä hiukan Git versionhallintaa, ja verrata työskentelyäni Westby Emman teokseen “Git for teams”, jossa käsitellään haarautumisstrategioita sekä Git version hallinnan hyviä käytänteitä tiimi ympäristössä.

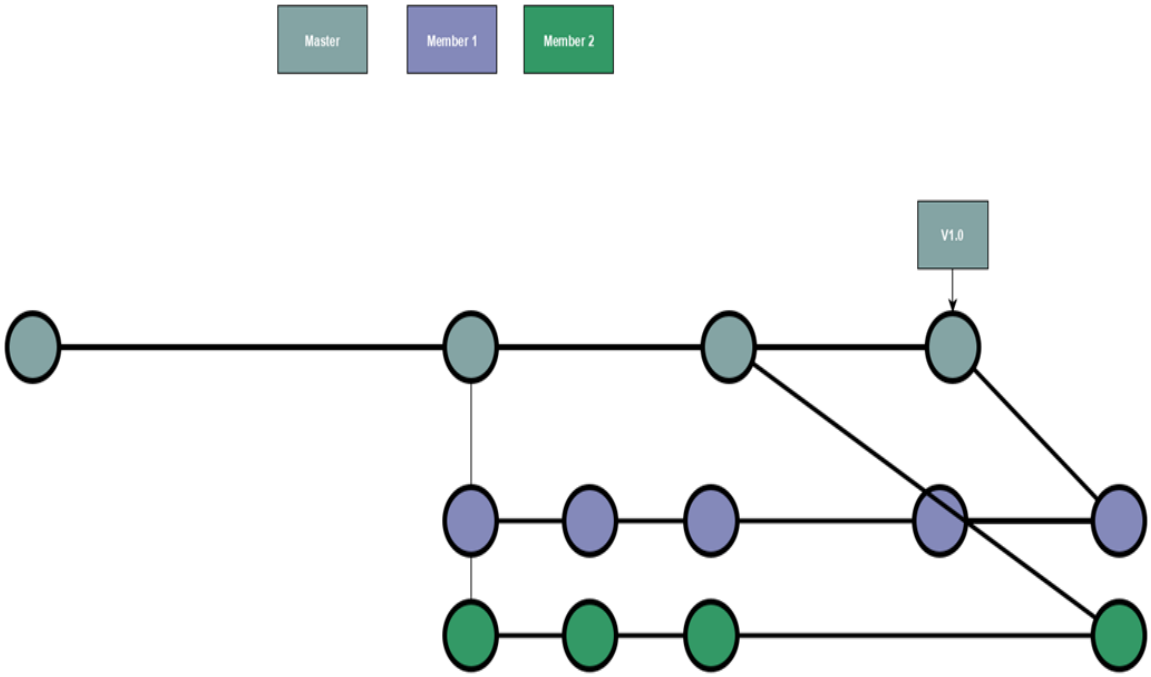


Kuvio 6. Git work malli

Kuten jo hiukan kerroinkin päivämerkinnöissäni käytän työssäni yksinkertaistettua versiota gitflow mallista (ks. Kuvio 6), jossa feature haarojen sijasta kaikki kehitystyö tehdään suoraan kehityshaaraan. Tyypillisesti tätä mallia käytetään keskikokoisista isoihin ohjelmistokehitys projekteissa ja koska itsekin pidän sen ideasta, päätin ottaa se omaankin käyttöön.

Westby määrittelee teoksessaan muutaman strategian, joista silmää pistävä malli on “Mainline Branch Deployment”, jossa kaikki kehitys tehdään suoraan päähaaraan, jolloin kaikkien puskujen täytyy olla varmasti toimintakunnossa ja varmasti testattu. Tiimissä tätä mallia (ks. Kuva 7) käytetään siten, että jokaisella tiimin jäsenellä on oma haaransa, johon he kehittävät ominaisuudet. Kaikki muutokset pusketaan jäsenien haarasta suoraan päähaaraan. Yksi hyvä puoli kyseisessä mallissa on historian lineaarisuus sekä junaratamaisen historian eliminointi kokokaan.

(Westby Emma, “Git for teams”, Luku 3. 2015)



Kuvio 7. mainline branch deployment malli tiimissä.

Toinen malli, jota teos käsittelee, on hyvin samanlainen, kuin tällä hetkellä hyödyntämäni Git flow malli (ks. Kuvio 7). Olennaisesti se eriää siten, että sen mallissa on kaksi haaraa ja kaikki uudet ominaisuudet yhdistetään integraatio haaraan, josta se sitten työnnetään päähaaraan. Huomaa, että tässä mallissa kaikki jäsenet yhdistävät erikseen omat muutoksensa päähaaraan. Tällöin konflikteja voi syntyä todella herkästi, mikäli jäsenet työstävät samoja tiedostoja.

(Westby Emma, “Git for teams”, Luku 3. 2015)

Westby:n määrittelemät strategiat ovat oikein hyviä tapoja hallinnoida versioita, mutta uskon hyvin vahvasti, että tällä hetkellä käyttämäni malli sopii projektien luonteeseen paremmin, johtuen tarpeesta pystyä nopeasti toteuttamaan bugien korjauksia. Päähaara halutaan myös pitää erillään kaikesta muusta. Committeja squashataan pakostakin koko ajan, jotta muutos historiat pysyvät puhtaana. Muutoksien tekemiseen suoraan päähaaraan ei siis olisi kauhean järkevää.

3.4 Seurantaviikko 4 10.2–14.2

Maanantai 10.2.2020

Jatkan tänään viime viikolla aloittamaani tehtävää, jossa piti implementoida asetukset sivu palikka laajennukseen, joka integroi yrityksen luokkakoulutukset Moodlen järjestelmään. Viime viikolla minulla oli ongelmia asetuksien kanssa, sillä en tiennyt millä funktiolla tai luokalla minun kuuluisi kutsua laajennuksen omia asetuksia. Tänään kävi kuitenkin selväksi, että Moodlen kehittäjät eivät ole suoraan dokumentoineet kaikkia käytänteitä asetuksien ympärillä, kuten esimerkiksi sitä, että jokaisen asetuksen nimi tulee määritellä muodossa “laajennusnimi/asetusnimi”. Jouduin kaivamaan tämänkin tiedon lukemalla ensin Moodlen kirjastoja ja sitten kehitysympäristön tietokannan taulua, johon laajennusten asetukset oli määritelty.

Nopeasti alla olevassa kuvassa (ks. Kuva 2) on esimerkkinä laajennuksen asetukset, jossa näkyy miltä asetuksen tulisi oikeasti näyttää. Minulla oli siis merkitty nimeen vain asetuksen nimi, joka aiheuttaa sen, ettei asetuksen arvoa aseteta tietokantaan.

```
defined('MOODLE_INTERNAL') || die();

if ($ADMIN->fulltree) {

    $name = 'skeleton/setting_1';
    $title = 'Title of the settings';
    $description = 'Description';
    $setting = new admin_setting_configtext($name, $title, $description, 'defaultvalue', PARAM_TEXT);
    $settings->add($setting);


}
```

Kuva 2. Esimerkki laajennuksien asetuksista

Tiistai 11.2.2020

Tänään agendassa on bugi korjauksia, joista päätin ottaa eniten huomiota vaativan. Kyseisessä bugissa kurssin järjestäminen päivämäärä järjestyksessä ei toimi ollenkaan. Kyseinen laajennus oli kirjoitettu työharjoitteluni aikana, joten se selittikin jo hiukan miksi toteutukseni näyttää silmiini todella rumalta. Lähikoulutusten listaamiseen on käytetty datatables JavaScript moduulia, joka vaatii alamoduuleja, jotta tietyt ominaisuudet toimivat.

Kyseinen moduuli on toteutettu Jquerylla, joka voidaan todeta tänä päivänä olevan menneen aikojen teknologioita. Etsin tietoa datatablen funktioista sekä järjestämiseen liittyvää materiaalia, jossa päivämäärä formaatit ovat dokumentoitu. Harmikseni en tänään saanut ratkaisua ongelmaan, mutta olen askeleen lähempänä. Ymmärrän ongelman syyn ja sen tarvitseman ratkaisun.

Koska toteutus itsessään näyttää todella rumalta ja koodin luettavuus kärsii, pohdin vahvasti myös koko käyttöliittymäpuolen uudelleen kirjoittamista. Ratkaisuun tarvittava koodi olisi jo valmiiksi olemassa Edellisessä projektissa, jossa työstin paginointi ratkaisua, joten senkään puolesta ei vaadita hirveitä työpanoksia.

Keskiviikko 12.2.2020

Jatkoin tänään eilisen ongelman selvittelyä, jossa ongelmana oli lähiopetus kurssien järjestäminen päivämäärän mukaan. Yritin tänään selvittää ongelmaa lisää ja huomasinkin, että järjestelyoperaatio toimii, kun formatointia ei yritetä tehdä. Oletusformaatti on Yhdysvaltojen käyttämä “DD/MM/YYYY” formaatti, jossa päivämäärän attribuutteja ei ole eroteltu pisteillä. Yritin siis määritellä datatablen kolumneihin render callback funktion, joka formatoisi päivämäärän näkymään käyttöliittymään eri muodossa kuin missä se käsitellään. Tästä huolimatta ongelma ei ole ratkennut.

Torstai 13.2.2020

Sain tänään ratkaistua eilen työstämääni bugia, jossa kurssien järjestäminen formatoidun päivämäärän muodoss ei onnistunut ollenkaan. Luettuani dokumentaatioita läpi muutamaan otteeseen silmäni kiinnittivät huomion kirjastojen versionumeroihin, joita esimerkeissä oli käytetty. Ne eivät olleet samoja, mitkä olivat minulla käytössä. Vaihdoin siis kirjastojen versiointi numerot, jolloin päivämäärän mukaan järjestäminen lähti toimimaan niin kuin oli odotettu. Korjauksen tehtyäni tein puskun versionhallintaan ja merkkasin version ylös pienenä päivityksenä.

Seuraavaksi siirryin kurssilistaus bugin korjaamiseen, joka liittyi kategoria suodatukseen, jossa kategoria jostain syystä jakautuu kahtia kun se syötetään käyttöliittymään. Tämän syyksi paljastui kirjoitusvirhe, jossa minulta oli unohtunut kirjoittaa Hipsut passatun arvon ympärille. Niin hassulta kuin se kuulostaakin, ongelma korjaantui nopeasti tehdessäni nämä muutokset.

Tänään sain jälleen oppia tarkkaavaisuuden tärkeyden ja kuinka pienetkin bugit voivat päästä läpi huomaamattomana, kunnes ne löydetään manuaalisesti.

Perjantai 14.2.2020

Tänään agendassa oli profiili-laajennuksen uuden version julkaisu alustalla. Laajennukseen oli tehty korjauksia sekä uusi ominaisuus, jonka avulla ylläpito pystyy tekemään määrityksiä

asetus sivulta. Asennus aikataulutettiin työajan ulkopuolelle, jotta asiakkaalle koituisi mahdollisimman vähän haittaa ylläpito työstä. Prosessi alkoi varmennuskopion ottamisesta koko järjestelmästä, jonka jälkeen päivitin laajennuksen käyttämällä palvelun asennusohjelmaa, joka hoiti prosessin loppuun.

Kun sain varmistettua, että kaikki muutokset ovat ok, raportoin suoraan seniori kollegalle, sekä myyntineuvottelijallemme julkaisun onnistumisesta.

Pääsin tänään jatkamaan todistus-laajennuksen työstämistä, joka oli jäähdytetty hetkellisesti muiden prioriteettien takia. Rupesin tutkimaan parempaa tapaa toteuttaa PDF tulostus ominaisuus, jonka kanssa oli ilmennyt ongelmia, kun yritin toteutusta PHP:llä suoraan. Nyt kuitenkin viisaampana tiedän, että kyseinen prosessi olisi parempi jättää käyttöliittymän tehtäväksi ja antaa palvelimen vain siirtää tieto sen käsittelyyn. Muutama kandidaatti löytyi ja aloitinkin jo niiden testaamisen.

Viikkoanalyysi

Tällä viikolla kaikki työtehtävät keskittyivät bugi korjauksiin sekä uusien ominaisuuksien kehittämiseen todella nopealla syklillä, joten koen nyt olevan hyvä aika käsitellä ohjelmiston vianselvitystä, joka tällä hetkellä tarkoittaa minulle koodin ongelman selvitystä ja korjaamista. Jotta voidaan analysoida virheenjäljitys prosessia, jota käytän työssäni, on tärkeää ymmärtää ensin mitä kyseiseen prosessiin kuuluu. Käsittelen tämän viikon analyysissa Butcher, P. sekä Carter, J:n teosta Debug It, jossa käsitellään virheenjäljityksen prosessia aina löytämisestä korjaamiseen saakka.

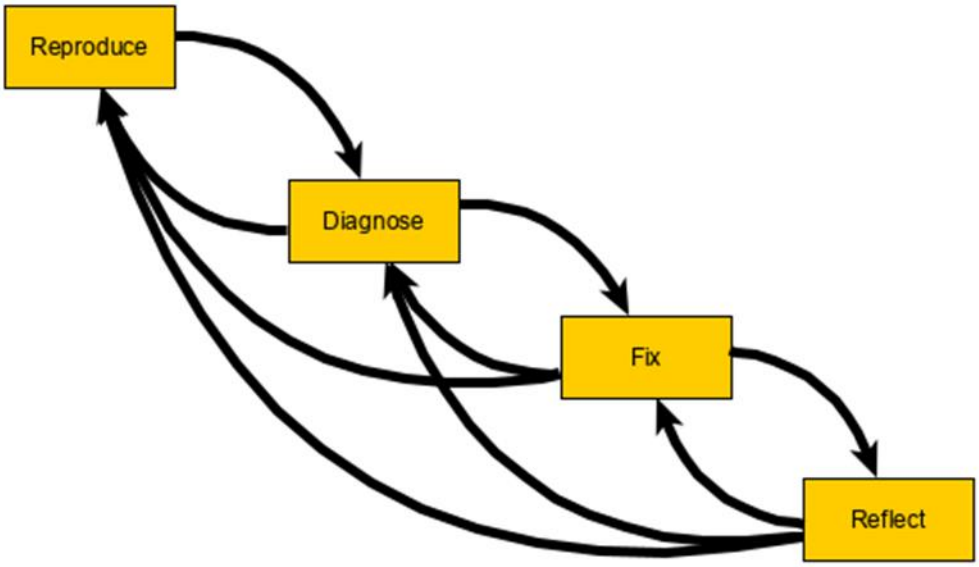
Butcher sekä Carter määrittelivät virheenjäljityksen koostuvan viidestä eri askeleesta, jotka kehittäjän tulee suorittaa löytäessään bugin ohjelmasta. Analysoi ohjelman käyttäytymisen syy, Korjaa ongelma, Vältä rikkomasta muita arkkitehtuurin osia, ylläpidä tai paranna koodin luettavuutta sekä laatua, ja varmista, että sama virhe ei toistu muualla koodissa.

(Butcher, P. & Carter, J. Luku 1.1 “Method in the madness”)

Tähän asti teos määritteli tyypillisen prosessin virheenjäljitykselle ja painotti, että lähestymistapoja on monia. Butcherin sekä Carterin mielestä empiirinen lähestymistapa virhettä jäljittäessä olisi tuottavin kaikista vaihtoehdoista. Toisin kuin sen muut vastakohdat, jotka perustuvat täysin teoriaan sekä logiikkaan, empiirinen lähestymistapa on vahvasti riippuvainen kokemuksista sekä tarkkailusta, jolla tarkoitetaan ohjelman käyttäytymisen tarkkailua tässä kontekstissa. Eli toisin sanoen ongelma voidaan jäljittää paljon helpommin

järjestämällä testejä, joita tarkkailen voidaan analysoida samalla myös omaa mentaali mallia softan käyttäytymisestä.

(Butcher, P. & Carter, J. Luku 1.1 “Method in the madness”)



Kuvio 8. Carterin ydin jäljitys malli.

Butcherin ja Carterin teos määrittelee empiirisen lähestymistavan ydinprosessille (ks. Kuvio 9) neljä vaihetta, jossa ongelman jäljitys aloitetaan virheen uudelleen tuottamisella. Seuraavassa prosessissa tehdään diagnoosi tuotetun virheen pohjalta tekemällä testejä erilaisten teorioiden pohjalta. Kun diagnoosi ongelmalle on tehty, voidaan siirtyä ongelman korjaamiseen. Tässä vaiheessa painotetaan, että softan arkkitehtuuria kannattaa myös muokata, jotta ongelmia voidaan välttää myös tulevaisuudessa. Prosessin viimeisessä askeleessa tehdään retrospektiivinen yhteenveto kaikesta opitusta. Ydin kysymykset ovat mm. Mitkä asiat menivät pieleen? Onko ongelmasta muita esimerkkejä, jotka voidaan korjata? Miten kyseinen ongelma voidaan välttää tulevaisuudessa? Prosessi ei kuitenkaan ole porrasmainen vaan, askeleissa voidaan aina mennä taaksepäin tarpeen mukaan.

(Butcher, P. & Carter, J. Luku 1.1 “Method in the madness”)

Verrattuna omaan virheenjäljitys prosessiin teoreettinen lähde määrittelee todella vakuuttavan mallin, jossa on otettu huomioon softan elinkaari myös pitkällä tähtäimellä. Tällä hetkellä bugien ilmestyessä korjaukset tehdään täysin mutu tuntumalla, joka tarkoittaa ei-manuaalisia testejä vaan korjaa sitä mukaan, kun kollega tai minä itse löydän ne

henkilökohtaisesti. Uskon, että kyseinen malli olisi jatkoa miettien todella hyvä vastaava menetelmä virheenjäljityksen avuksi.

3.5 Seurantaviikko 5 17.2–20.2

Maanantai 17.2.2020

Työ jatkuu todistus laajennuksen kanssa. Päätehtäväni oli tänään testata pdf tiedoston tulostaminen JavaScriptillä suoraan. Viimeviikolla olin tehnytkin jo taustatyötä mahdollisuuksista toteuttaa kyseinen ominaisuus JavaScript kirjastoilla. Päädyin valitsemaan jsPDF kirjaston, joka tarjoaa hyvän dokumentaation kehittäjälle ja itse esimerkki koodit näyttävät erittäin yksinkertaisilta.

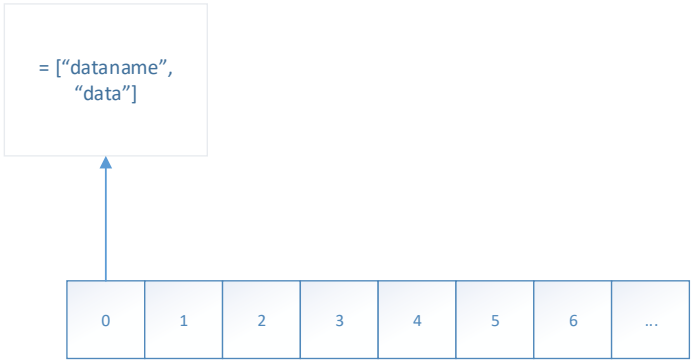
Kesken kokeilujani eteeni tuli ongelma, jossa en saanut kuvia tulostumaan pdf tiedostolle käyttäen normaalia osoitetta sen sijaintiin, vaan se käyttää minulle täysin uutta käsitettä data osoite. Data osoitetta käytetään tiedon en koodaamiseen muodosta toiseen. Data osoitteen etu on sen lyhyys koodieditorissa. Oikeasti sen osoite on todella pitkä ja Mozillan määrityksen mukaisesti rajoitettu 65535 merkkiin.

```
<img src="data:image/png,%89PNG%0D%0A..." />
```

Kuva 3. Data osoite esimerkki html:ssä

Tiistai 18.2.2020

Todistus-laajennuksen kehittäminen jatkuu edelleen. Päätin tänään, että projektin koodia täytyy uudelleen järjestää kovalla kädellä ja toteutuksia muuttaa, jotta laajennus saadaan toimimaan. Aloitin prosessin luonnostelemalla nopeasti yksinkertaisen mallin, joka visualisoi miten data siirtyy palvelimelta käyttöliittymään. Vaikka meidän tapauksessamme palvelin palauttaa näkymän suoraan, päätin silti määritellä ne omina osinaan. Visualisointi on toteutettu UML-notaatiolla.



Kuvio 9. PHP Array luettelo string esimerkillä

Määrittelemäni malli sisältää lähtökohtaisesti kolme osaa, joista tärkeimmässä osassa on PDF data luokka. Luokka määrittelee array luettelon (ks. Kuvio 9) ja ottaa sisäänsä avain “key/value” arvoja, jotka lähetetään sitten eteenpäin käyttöliittymään. Valitsin tämän lähestymistavan siksi, että minun ei tarvitsisi määritellä erillistä json serialisointi funktiota, joka prosessoisi datan json muotoon. Seuraavat osat ovat Moodlen standardin mukaiset renderöinti ja kirjasto luokat, joista ensimmäinen välittää datan suoraan käyttöliittymälle ja jälkimmäiset toteuttavat sertifikaatille rajapinta funktioita, johon sisältyy mm. Käyttäjä Datan hakeminen Kurssilistaus laajennuksen sekä profiili laajennuksen rajapinnasta.

Keskiviikko 19.2.2020

Tänään aikeena oli jatkaa todistus laajennuksen työstämistä, mutta eteen sattui korjaus keikka, sillä palvelun uuden kurssilistauksen käyttöönoton aikana ilmeni visuaalisia bugeja, jotka tarvitsivat korjaamista. Koska kyseiset bugit olivat tuotannossa käyttäjien näkyvillä, vaati tämä siis erityisen kiireellistä korjaamista. Korjaukset kohdistuivat siis pääasiallisesti Moodlen teeman CSS tyyleihin, jossa tietyt määritykset aiheuttivat ongelmia, kuten mm. tyhjiä tiloja, elementtien piiloutuminen elementtien alle. Tätä ei voitu mitenkään ennakoida kehityksen aikana, koska kehitys ympäristö käyttää oletus teemaa ja sen kanssa ei ilmennyt samoja ongelmia.

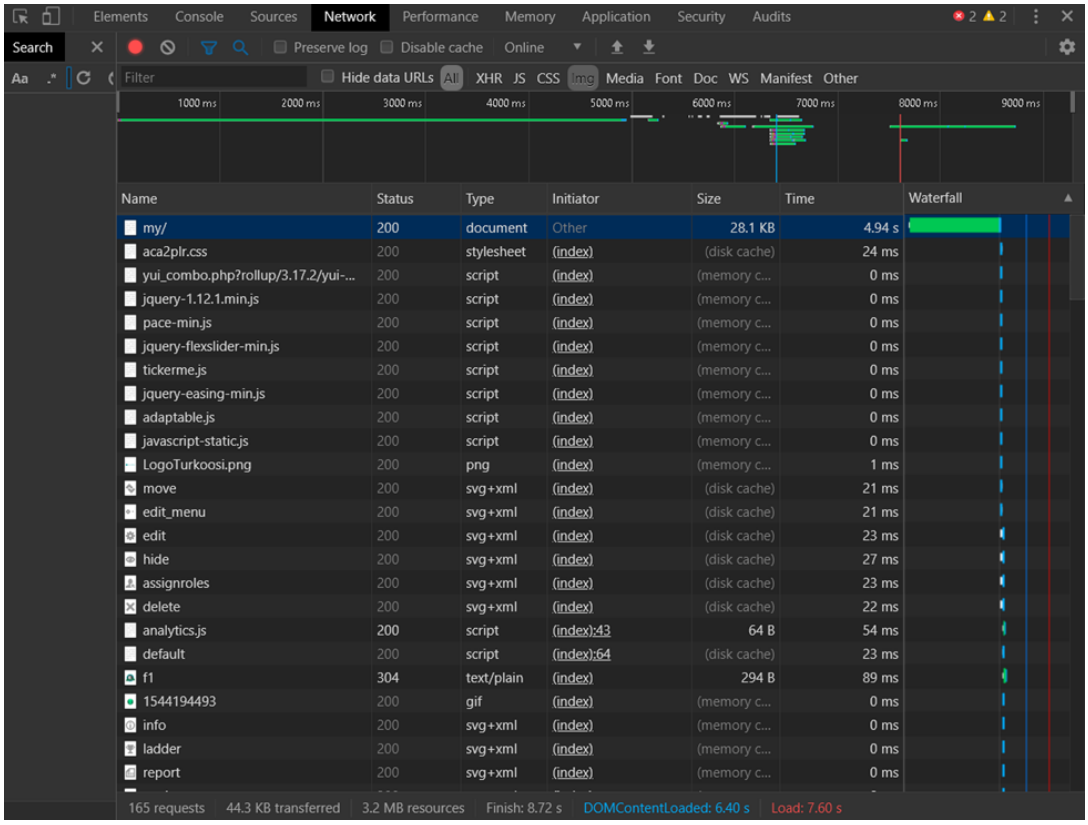
Tästä huolimatta tyylit saatiin korjattua Moodlen teeman asetuksiin ja kaikki lähti rullaamaan oikein hyvin. Seniori kollega pyysi minulta pienimuotoisen syy seuraus raportin, joka sitten välitettiin yritys asiakkaille. Perustoimenpide on aina raportoida asiakkaalle statusta, kun uusia ongelmia ilmestyy.

Päivän lopuksi seniori kollega siirsi minulle, tehtävän selvittää etusivun pitkiä lataus aikoja. Etusivun latausaika on tällä hetkellä välimuistin kanssa 8 sekuntia, mikä on johtanut huonoon käyttäjäkokemukseen. Missään nimessä sivun lataamisessa ei saisi mennä näin

kauan. Keskiverto käyttäjä jaksaa odottaa vain sekunnin ennen kuin selain suljetaan. Tehtävä on saanut korkeimman prioriteetin.

Torstai 20.2.2020

Aloitin tänään selvittämään palvelun suorituskyky ongelmia, jotka ovat aiheuttaneet käyttäjille ongelmia. Tässä olisi vielä hyvä huomioida, että pitkät latausajat ovat olleet ongelma ennen uuden kurssilistauksen käyttöönottamista, joten heti alkuun voidaan eliminoida yksi mahdollinen tekijä. Jotta pystyisin analysoimaan ongelmaa lisää, päätin ensin nauhoittaa etusivun lataamisaikaa. Selaimen kehitystyökalut (ks. Kuva 4) mahdollistavat suorituskyvyn katselmointia nauhoittamalla vesiputous aikajana, jossa visualisoidaan, jokaisen http kutsun kesto.



Kuva 4. Sivun aikajana sekä latausajat

Tutkiessani aikajanaa huomasin, että etusivun indeksillä menee 6 sekuntia palauttaa ensimmäinen vastaus palvelimelta, ennen kuin DOM eli käyttöliittymä elementit ladataan sisälle. Yritimme selvitellä ongelmaa kollegani kanssa, mutta emme päässeet ratkaisuun. Teimme ilta kahdeksaan asti töitä, jotta ongelma saataisiin kitkettyä pois.

Perjantai 21.2.2020

Tänään työpäivä on edellistä hiukan lyhyempi ja työtehtävät pysyvät samoina, eli selvitetään palvelun performanssi ongelmia. Päätimme kollegani kanssa asentaa palvelimelle monitorointi ratkaisun, joka antaa vielä tarkempaa dataa sekä tietoa kaikista palvelimelle menevistä ja ulos tulevista HTTP kutsuista, joiden avulla pyrimme tutkimaan miksi palvelun operaatioissa kestää niin kauan.

Sattumalta kollegani totesi minulle, että palvelumme sivun muokkaus oli jostain syystä mennyt epäkuntoon ja kaikki viittaukset osoittivat teemaan. Päätimme siis testata kurssilistaus laajennuksen käyttäytymistä tuotantopalvelimen teema-laajennuksella. Tuloksena oli sivun muokkauksen rikkoutuminen, sekä lataus ajan tuplaantuminen.

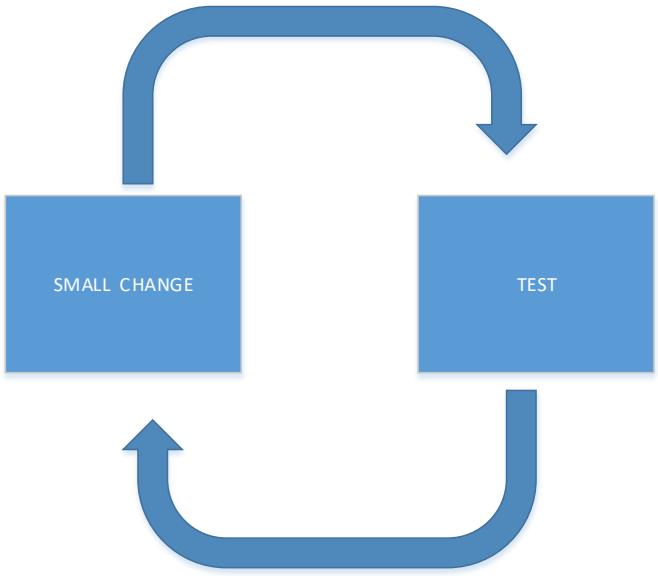
Raportoimme löydöksemme sekä mahdollisen korjaustoimenpiteen seniori kollegalle, joka otti työn vastuulleen.

Viikkoanalyysi

Huomasin, että työni aikana järjestin koodia uudelleen aina, kun koodin suorituskyky tai rakenne ei ollut ollenkaan mieleeni. En kuitenkaan koskaan pysähtynyt miettimään miksi olen sitä tehnyt tai millä kriteereillä uudelleen järjestäminen kannattaa tehdä. Tämän hetken ymmärtämykseni mukaan koodia järjestään uudelleen, kun ohjelman koodi halutaan kirjoittaa uudelleen muuttamatta sen alkuperäistä toiminnallista logiikkaa. Syitä tälle voivat olla tarve lisätä ominaisuuksia, jotka eivät ole tuettu ohjelmassa valmiiksi tai koodin hetkellisen tilan aiheuttamat riskit.

Martin Fowlerin ja Kent Becking teos “Refactoring: Improving design of existing code” teos määrittelee, että aina kun kehittäjällä on aikomus järjestää koodia uusiksi, tulee heidän kirjoittaa siihen liittyvät yksikkötestit. Mutta minä kirjoitan kaikki yksikkötestit, jo ennen kuin kirjoitan itse koodin logiikkaa, jotta koodin funktionaalisuus voidaan aina varmistaa. Prosessi vastaa tässä vaiheessa erehdyttävän paljon kirjassa käsiteltyä mallia.

(Fowler ja Beck. Luku 1. “Refactoring a first example”, 1996)



Kuvio 10. Uudelleenjärjestäminen pienessä syklissä.

Oleelliset erot minun työskentelytapani sekä kirjan käsittelemän mallin välillä on muutoksien tekeminen pienessä syklissä, niin että koodi yksikkö testataan, jokaisen muutoksen jälkeen. Lähestymistapa sopii sekä pienille ja suurille muutoksille.

(Fowler ja Beck. Luku 2. “Principles in refactoring”, 1996)

Vielä en ole kuitenkaan koskenut syihin, milloin koodia kannattaa uudelleen järjestetää. Se ei ole yksiselitteistä, sillä kehittäjä voi halutessaan keksiä kriteereitä loputtomiin. Projektin etenemisen kannalta Fowler ja Beck eivät ole määritelleet tiukkoja kriteereitä koodin järjestämiseen, mutta enemminkin vaaran merkit, jotka kehittäjän kannattaa ottaa huomioon työnsä aikana. Vaaran merkkejä on todella pitkä lista ja en koe tarpeelliseksi käydä kaikkia läpi tässä, mutta muutamat hyvät Fowlerin ja Beckin mainitsemat vaaran merkit ovat Duplikaatti koodi, Pitkät funktiot tai luokat ja liian suuri määrä arvoja luokalla.

(Fowler ja Beck. Luku 3. “Bad smells in code”, 1996)

Jatkossa koen tarpeelliseksi pitää mielessäni vaaran merkit ja uudelleen järjestää koodi aina kun, projektin etenemisen kannalta olisi mahdollista. Moodlen kaltaisessa ympäristössä pahan koodinhajut ovat melkein osa arkea, sillä jokainen iso luokka on lähestulkoon tuhat riviä.

3.6 Seurantaviikko 6 24.2–27.2

Maanantai 24.2.2020

Tänään oli aikeena jatkaa todistus-laajennuksen työstämistä, kunnes seniori kollegalta tuli viesti jossa todettiin, että uuden teema-laajennuksen asennus voitaisiin tehdä ja testata, vaikuttaako se ongelmiin joihin oli törmätty viime viikolla. Pidimme asiasta palaverin kollegan kanssa ja harmiksemme jouduimme asennuksen jälkeen toteamaan, että vika johtuu jostain muusta. Päätin tässä vaiheessa ottaa puheeksi, että palikan tekemät tietokanta- sekä käsittelyoperaatiot voivat mahdollisesti aiheuttaa suorituskykyongelmia, jos kursseja on yhdellä käyttäjällä suuri määrä. Tämä ei kuitenkaan selitä, miksi suorituskykyongelmaa ei havaittu testi ympäristössä.

Totesimme iltaan mennessä, että ongelman täytyy olla lähtöisin koodin huonosta optimoinnista, joka aiheuttaa pidemmät latausajat. Otin siis prioriteetiksi selvittää lähtökohtaiseksi mikä koodissa aiheuttaa pitkät lataamisen piikit ja miten sitä voidaan lieventää tai eliminoida kokonaan.

Keskiviikko 26.2.2020

Tänään työtehtävänä on koodin analysoiminen ja mahdollisien pullonkaulojen löytäminen koodista, jotka voivat aiheuttaa pitkiä latausaikoja. Tähän en ollut aikaisemmin kiinnittänyt niin suurta huomiota, koska oletuksena oli, että Moodlen arkkitehtuuri olisi toteutettu suorituskykyiseksi. Tein tämän olettamuksen pohjalta päätöksen toteuttaa jotkut laajennuksen ratkaisut todella naiivisti, kuten esimerkiksi lineaarinen tapa käsitellä tietoa, joka tarkoittaa jokaisen arrayn elementin käsittelyä erikseen tai sellaisien operaatioiden kirjoittaminen, joiden kompleksisuus aiheuttaa pidempiä käsittely aikoja.

Aloitin prosessin sillä, että tutkin ne funktiot läpi, jotka laajennus kutsuu kurssitiedon käsittelyn aikana. Yksi funktio ei voi yksinään aiheuttaa merkittäviä suorituskykyrasitteita, mutta, jos esimerkiksi kaikkialla koodissa tapahtuu samanlaisia yksittäisiä rasitteita, voi tulos olla erilainen. Huomasinkin, että olin tehnyt for silmukoita monessa paikassa. Erityisesti kurssidataa käsittelevät funktiot näyttivät erittäin painavilta näin jälkikäteen katsottuna. Yksi funktioista kutsui Moodlen tietokanta rajapintaa ja haki kurssit kannasta, jonka jälkeen Moodle toteuttaa omat prosessointioperaationsa. Tämän jälkeen minun kehittämäni laajennus käsittelee ne vielä uudelleen, jotta tieto voidaan näyttää halutussa muodossa. Meistä riippumattomista syistä Moodlen koodi on määritellyt kurssin datat kahteen eri objektiin. Eli kaksi tietokanta kyselyä sekä sen jälki prosessit.

Torstai 27.2.2020

Kurssilista-laajennuksen optimointi jatkuu tänään, samoilla kuvioilla kuin eilenkin. Yritin tänään ratkaista suorituskykyongelmaa käyttämällä hyväkseni muutamaa teosta, jossa kuvaillaan hyviä prosesseja koodin hyvän optimoinnin saavuttamiseksi. Käsittelen näitä lisää viikko analyysissa.

Yrittäessäni ratkaista suorituskykyongelmaa, huomasin, että osaamiseni sekä tietoni koodin optimoinnista oli selkeästi puutteellinen, joten tässä vaiheessa päätin, että tietoa tulee etsiä muualta ja joutuisin tässä tapauksessa opiskelemaan samalla kun ratkaisisin minulle haasteellista ongelmaa.

Päätin siis heittäytyä kunnolla algoritmien sekä koodin suorituskyvyn analysoinnin maailmaan, lukemalla muutamasta teoksesta avainkäsitteitä. Tiesin jo ennalta, että koodia analysoidaan käyttämällä matemaattista merkintää O(N) ja, että siitä on olemassa erilaisia variaatioita riippuen mitä koodin pätkää ollaan analysoimassa. Tässä kohti on hyvä huomioida, se että en ole edes varma onko algoritmien ajallisen monimutkaisuuden analysointi sama asia, kuin koodin suorituskyvyn analysointi, sillä algoritmien operaatiot ovat melkein aina monimutkikkaampia, sekä vaativampia.

Meidän tapauksessamme koodin ajoaika kasvaa suhteessa kurssienmäärään, koska kaikki operaatiot tehdään iteratiivisesti eli yksi kerrallaan ja yhden kurssin käsittelyyn voi sisältyä jopa kymmenen operaatiota. Yritin laskea toteuttamieni koodin ajallisen monimutkaisuuden käyttämällä edellä mainittua O(N) mallia, jossa O on operaatio ja N edustaa listan elementtien määrää. Koska suurimmaksi osaksi koodissani tehdään silmukoita, koen että tämä olisi hyvä lähtökohta.

Perjantai 27.2.2020

Koodin optimointi jatkuu tänään. Analysoin koodiani pidemmälle ja olen edelleen ihan pihalla siitä miksi koodi on edelleen niin intensiivinen ajonaikana. Näennäisesti koodin suorituskyky ottaa piikin heti kun kurssien määrät (Ks. Kuvio 11) kasvavat. Kuvassa havainnollistetaan koodin suorituskykyä kolmesta näkökulmasta, jossa sininen on tämän hetken tulos, vihreä viiva kuvastaa tavoitetta, johon halutaan pyrkiä ja keltainen ihanne tilannetta, jossa suorituskyky pysyy koko ajan tasaisena, eikä minkäänlaista vaikutusta olisi näkyvissä.

Kuvio 11. Suorituskyvyn havainnointi

Päätin tarkistaa koodini läpi vielä kerran, mutta en siltikään pysty hahmottamaan miksi ajoaika hyppää näin korkealle, vaikka koodini sisältää minimaalisen vähän looppeja refaktoroinnin jälkeen.

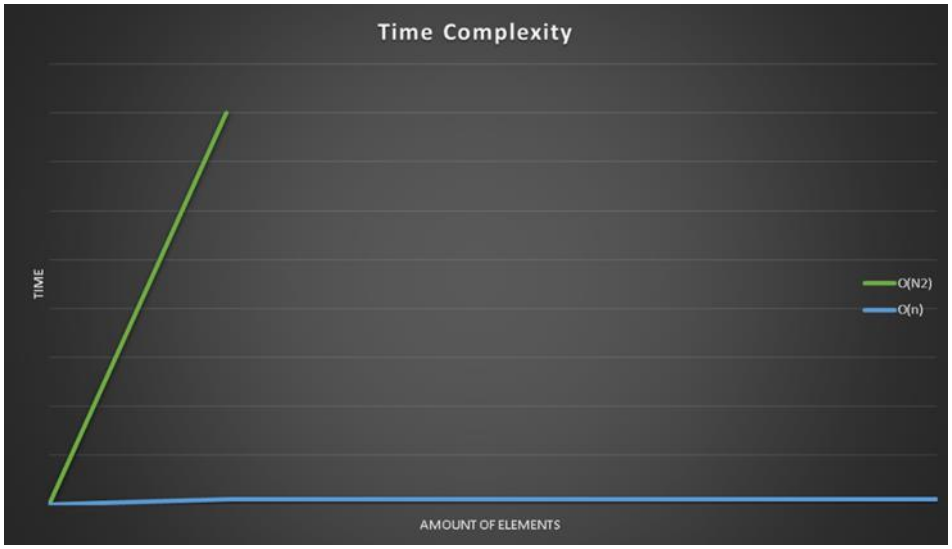
Viikkoanalyysi

Tällä viikolla keskeisenä aiheena työpaikalla on ollut koodin suorituskyky, sekä sen optimointi. Työtapani sekä mallit tämän aiheen ympärillä ovat tällä hetkellä hyvin alkeellisella tasolla, eikä minulla ole kuin perustiedot aiheesta. Jouduin siis työni aikana tekemään taustatutkimusta aiheesta, jotka liittyvät koodin suorituskyvyn parantamiseen. Ensimmäiset käsitteet, jotka tutkimukseni aikana tulivat vastaan, olivat koodin ajallinen monimutkaisuus, “Time Complexity” sekä päätöskeskeinen monimutkaisuus “Cyclomatic Complexity”.

Ajallinen monimutkaisuus on käsitelty suurimmilta osin algoritmien näkökulmasta, joten joudun siis sovittelemaan teoksien käsittelemiä malleja omaan työhöni sopiviksi. Päätin valita Jay Wengrown teoksen ”A Common-Sense Guide to Data Structures and Algorithms”, jossa käsitellään pääasiallisesti algoritmeja sekä data rakenteita yleisellä tasolla.

Wengrow käsittelee O(N) mallia ”big o notation”, jota käytetään listojen ajallisen monimutkaisuuden laskemiseen. O vastaa operaatiota ja n taas operaation käsittelemää arvoa saavuttamiseen. Työpaikalla kohtaamani ongelman tapauksessa me haluamme käsitellä lineaarisia monimutkaisuuksia, jossa voidaan sanoa, että operaatio kestää yhden operaation verran riippumatta N arvon koosta.

(Wengrown. Luku 3. “Oh yes! Big O Notation”, 2017)



Kuvio 12. Ajallinen monimutkaisuus Wengrown kuvaamana

Koodin monimutkaisuus Wengrownin mukaan halutaan siis rajoittaa mahdollisimman pieneksi, jotta koodin suorituskyky (Ks. Kuvio 12) pysyisi aina samana riippumatta siitä, kasvaako antamamme data parametri vai ei. Jotta ymmärrämme paremmin mitä edellä mainitut matemaattiset kaaviot tarkoittavat, päätin kirjoittaa vielä erikseen koodin pätkän (Ks. Kuva 5), joka kuvastaa tyypillistä lineaarista iteraatiota, jossa listan elementit tulostetaan konsoliin. Ideana tässä on huomata, että tupla iteraatiot aiheuttavat O(n2) tuloksen, joka taas kaavion (Ks. Kuvio 11) mukaan aiheuttaa suuren piikin, jossa suoritusaika nousee suhteessa annetun arvon kokoon.

(Wengrown. Luku 3. “Oh yes! Big O Notation”, 2017)

```
//O(n)
function LoopLog(value){
    for(var i = 0; i < value.length; i++){
        console.log(value[i]);
    }
}

//O(n2)
function LoopLogNested(items, prices){
    for(var i = 0; i < items.length; i++){
        console.log(items[i]);
        for(var x = 0; x < prices.length; x++){
            console.log(prices[i]);
        }
    }
}
```

Kuva 5. Esimerkki koodi lineaarisista kompleksisuuksista

Edellä käsitellyn teoksen perusteella pystyn jo heti sanomaan, että koodissani on todella paljon kohtia, jossa on toteutettu tupla iterointia ja niiden sisällä on vielä lisää operaatioita, joista minulla ei ole kirjoituksen hetkellä täydellistä tietoa siitä, miten kyseiset funktiot toteuttavat ominaisuutensa. Jatkossa minun tulee kiinnittää huomiota koodiin jota kirjoitan. Tämä ennaltaehkäisee suorituskyvyllisiä ongelmia, joita voi tulla tuotannossa huonon optimoinnin seurauksena.

3.7 Seurantaviikko 7 4.3–12.3

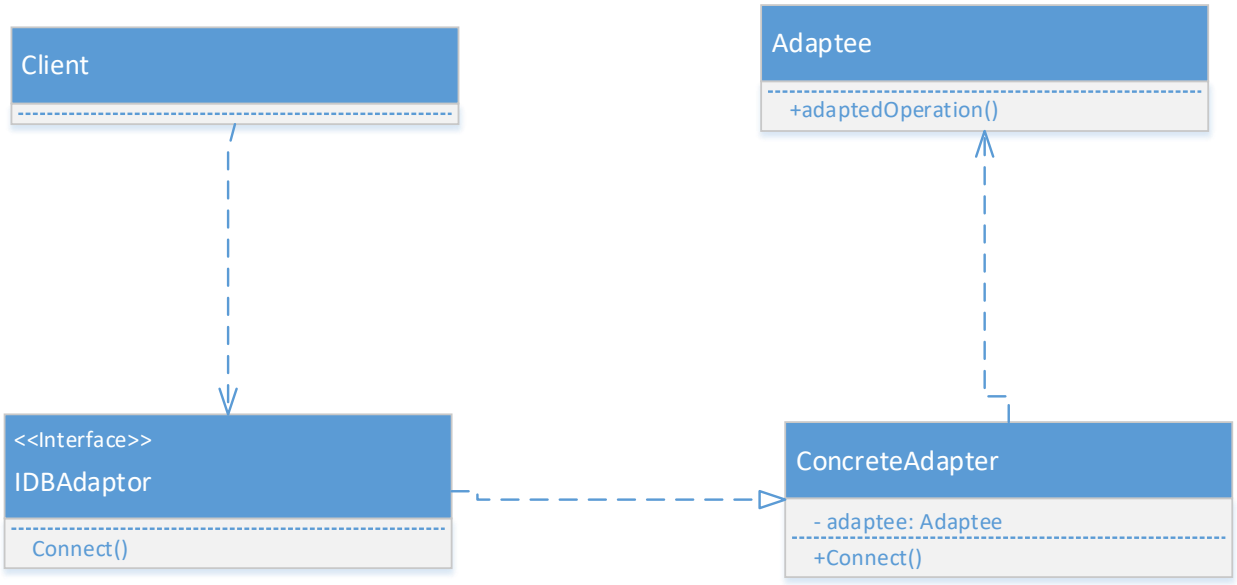
HUOM. Tämän viikon merkinnät on hajautettu kahdelle viikolle työkiireiden vuoksi.

Keskiviikko 4.3.2020

Tämä viikko on hiukan erilainen, koska olemme kollegani kanssa sopineet tekevämme yhdessä ison päivityksen viikonloppuna, jossa Moodle päivitetään versioon 3.5 ja PHP 7.3+ versioon. Tämän kautta toivomme, että palvelun suorituskyky paranee PHP:n päivityksen sekä koodin optimoinnin jälkeen. Tästä syystä päätin kirjoittaa merkinnät keskiviikon ja sunnuntain välisenä aikana.

Tänään teimme esisuunnittelua sekä ajurien asentelua palvelimelle, jotta päivitykset saadaan toimimaan normaalisti. Palvelut ajetaan virtuaalisilla palvelimilla, joista otetaan aina snapshotit, kun halutaan tehdä muutoksia, päivityksiä tai uusia asennuksia. Jouduimme

päivän aikana selvittelemään asennusongelmia, johon lukeutui mm. Tietokanta ajurien toimivuus uudemmalla PHP:llä, jossa kirjasto oli laitettu ihan uusiksi.



Kuva. 6 Adapteri malli

Tässä kohti todennäköisesti joudumme refaktoroimaan palikan uudelleen tukemaan uutta kirjastoa, koska uuden kirjaston mukana tulee muutoksia, joiden takia meidän tarvitsisi muuttaa koodia paljon. Jotta tulevaisuuden ongelmat voidaan välttää, päätin, että sitä varten olisi hyvä hyödyntää adapter suunnittelumallia (ks. Kuva 6), jossa ideana on kirjoittaa välipala kirjaston sekä sitä käyttävän luokan välille, jolloin jos kirjasto muuttuu uudelleen funktionaalisuutta, tarvitsisi vain muuttaa yhdestä paikasta.

Torstai 5.3.2020

Tänään tein esisuunnittelua wistec profiili palikan kehittämiselle, sillä viikonloppuna alkaa kolmen päivän mittainen päivitys prosessi, jossa Moodleen perustuva palvelumme päivitetään versiosta 3.1 versioon 3.5 ja samalla päivitetään PHP versiosta 5.5 versioon 7.2. Tämän mukana tulee liuta asioita, joita käsittelin jo eilen palikan näkökulmasta, mutta koska muut prioriteetit estivät palikan seuraavan version feature määrittelyä, jouduin sen tekemään tänään.

Seuraava versio olisi siis 1.2, johon lukeutuu yleinen koodin refaktorointi sopimaan muutoksiin, joita mahdollisesti tulee uuden PHP:n myötä. Kovimmalla prioriteetilla on tietokanta toiminallisuuden päivittäminen. Unohdin eilen mainita, että 7.2:n myötä vanha tietokantakirjasto poistuu käytöstä ja täten joudumme käyttämään uutta kirjastoa.

Perjantai 6.3.2020

Aloitimme yhdessä kollegoiden kanssa Palvelun suuren päivitysoperaation kello 16.00 työtuntien ulkopuolella, jolloin suurin osa käyttäjistä ei ole paikalla. Prosessi koostuu 13 askeleesta, joista ylipuolet ovat seniori kollegan vastuulla. Palvelun tietokanta päivitetään samassa prosessissa Moodlen kanssa, joten erillisiä tietokanta datan siirtoja ei tarvitse tehdä. Jokaisen prosessin välissä palvelimelta otetaan snapshot, joka antaa meille turvatyynyn, johon laskeutua ongelmien tai virheiden ilmetessä.

Minulle relevantti tehtävä on kaikkien laajennuksien päivittäminen uuteen Moodle versioon, joka tulee relevantiksi vasta, kun päivitysoperaatio on viety loppuun. Päivitys vaihe alkaa Wistec Profiilista, jota olen työstänyt jo muutaman päivän.

Keskiviikko 11.3.2020

Tänään on ensimmäinen normaali työpäivä suuren palvelun päivitys operaation jälkeen, jonka tuloksena elektroninen oppimisalusta päivitettiin Moodle versioon 3.5 ja PHP sai tuuppauksen versiosta 5.5, versioon 7.2. Ensimmäisenä asiana tajusin, että käyttämäni Kirjasto ei ollut mennyt umpeen vaan sitä oli jatkettu. Syystä tai toisesta en löytänyt tietoa heti dokumentaatiosta. Tämä tarkoittaa sitä, että suunniteltua muutosta Profiili laajennukseen ei ollut tarpeellista toteuttaa ja eteenpäin päästiin tavanomaisella päivitys operaatiolla, jossa tarkistin läpi koodin syntaksin ja luin läpi PHP 7.2 dokumentaation saadakseni hyvän ymmärryksen siitä, mitä pystyn tekemään uuden PHP:n kanssa.

Silmääni pisti erityisesti uusi ominaisuus, joka antaa kehittäjälle kyvyn määritellä palautettava tyyppi funktioon (ks. Kuva 7). Tämä ja monet muut uudet ominaisuudet ovat tervetullut lisä, jotka selventävät koodia huomattavasti.

```
//NEW SYNTAX
function exampleOperation($a, $b): float {
    return $a + $b;
    }
}

//OLD SYNTAX
function exampleOperation($a, $b) {
    return $a + $b;
    }
}
```

Kuva 7. Esimerkki syntaksi muutoksesta

Uuden syntaksiin tutumisen lisäksi jatkoin Todistus-laajennuksen kehittämistä tutkimalla jsPDF kirjastoa lisää ja lähdin implementoimaan prototyyppi toteutusta todistuksen tulostamisesta. Tuloksena oli toimiva ratkaisu, jonka ainoa ongelma oli se, että kuvat oli staattisesti tulostettu pdf tiedostoon. Huomenna tehtävänä olisi ratkaista miten kuvat saadaan Base64 Urli:na projektin hakemistosta sekä Moodlen asetuksista suoraan PDF kirjaston käytettäväksi.

Torstai 12.3.2020

Selvittelin tänään tapoja tuoda kuvat baseurlina suoraan PDF kirjaston käytettäväksi, ja pienen dokumentaation lukemisen jälkeen päätin rakentaa baseurlin manuaalisesti enkoodaamalla kuvan datan Base64 muotoon, josta sitten se käsitellään baseurl muotoon vielä kerran. jsPDF kuvan lisäys funktio osasi onnistuneesti tulostaa kuvan pdf tiedostoon käyttämällä baseurlia.

Selvitettyäni tämän ongelman siirryin implementoimaan taulukkoa PDF tiedostoon. Löysin siihen sopivan kirjaston, joka on jsPDF kirjastoa laajentava lisäosa. Testatessani taulukkoa huomasin, että jsPDF ei osaa tehdä sivutuksia automaattisesti eikä sillä myöskään ole funktiota, jota voitaisiin käyttää oman implementaation toteuttamiseen. Joudun siis selvittelemään, miten kyseinen ominaisuus saadaan toteutettua kirjastoon tai vaihtoehtoisesti ottamaan toisen kirjaston käyttöön.

Viikkoanalyysi

Kuten jo huomautin viikkomerkinnän alkupuolella, tämän viikon merkintä on hajautettu kahdelle viikolle. Suuri päivitysoperaatio kesti kauemmin kuin muutaman päivän ja vaati toimenpiteitä, joita ei voitu hoitaa allokoidussa ajassa, joten kollegojen kanssa päätimme tehdä ylitöitä normaalien tuntien ulkopuolella. Sis. Viikonloput.

Tämän viikon analyysin aiheeksi päätin ottaa Suunnittelumallit, sillä huomasin, että päädyin todella nopeasti käyttämään Suunnittelumalleja ongelmien tultua esiin. Ideana oli ymmärtää miksi päädyin tekemään tietyt ratkaisut suunnitellessani ratkaisua. Huomasin kuitenkin pohtiessani koodin ongelmia, että olin päätynyt melkein koko softan logiikan uudelleen kirjoittamiseen, koska aikaisemmin se oli sellaisessa muodossa, jossa sitä ei voitu helposti muuttaa toisesta kirjastosta toiseen ilman, että työaika venyy turhan pitkäksi.

Tähän kohti koin, että Robert. C. Martinin teos ”Clean Code” olisi oiva kirja käsiteltäväksi tämän viikon analyysissa ja ehkä jopa tulevillakin viikoilla, sillä kyseinen aihe on yksi uusia

sekä veteraani kehittäjiä eniten askarruttava aihe. Haluan erityisesti verrata oman projektini koodia sekä etenemistä verrattuna kirjan antamiin ohjeisiin.

Robert Martin haastatteli muutamia tunnettuja kehittäjiä ja kysyi heiltä syitä, miksi tietyt projektit epäonnistuivat pysymään elossa julkaisunsa jälkeen. Kaikkia yhdistävä tekijä oli selkeästi huonon koodin aiheuttamat työn määrän kasvaminen sekä tuottavuuden laskeminen. Huonon koodin välttäminen on kuitenkin hänen mukaansa vaikeaa, sillä hyvin useasti yrityksillä on deadlinet, joissa pitää pysyä ja usein tämä johtaa likaiseen koodiin.

(Robert C Martin. Luku 1. “Clean Code”, 2009)

Mutta miten puhdas koodi voidaan edes ylipäätään määritellä? Martin kysyi haastateltavilta kehittäjiltä heidän käsitystään puhtaasta koodista ja miten he ovat itse sen määritelleet. kehittäjien joukossa yksi yhteinen tekijä tuli ilmi ja se oli koodin luettavuus. Heidän mukaansa koodin tulisi olla elegantti ja tehokas. Sen logiikan pitäisi olla niin selkeää, että uusien kehittäjien olisi helppo lisätä uusia ominaisuuksia tai muokata olemassa olevaa arkkitehtuuria.

(Robert C Martin. Luku 1. “Clean Code”, 2009)

```
var tod; //time of the day

var timeOfTheDay;
```

Kuva 8. Nimeämis- Esimerkki

Seuraavaksi haluan käsitellä hiukan arvojen nimeämiskäytänteitä, sillä jokaisella kehittäjällä on omat mieltymyksensä sekä tapansa arvojen nimeämisen suhteen ja monilla on myös taipumusta tehdä arvojen nimistä mahdollisimman lyhyitä voidakseen kirjoittaa koodia nopeammin. Martin painottaa, että hyvin nimetyt arvot tuovat esiin funktion tai olion tarkoituksen, se ei jätä tulkintaa ollenkaan kehittäjän varalle. Martin näyttää teoksessaan esimerkin (ks. Kuva 8), joka havainnollistaa hyvin ja huonosti nimetyn olion eron. Alempi olio kuvailee itse itsensä, kun taas ensimmäinen on täysin abstraktiota.

(Robert C Martin. Luku 2. “Clean Code”, 2009)

```
function getThem(ar1){
    var rs = [];
    for(var x = 0; x < ar1.length; x++){
        if(ar1[x]==8)
            rs.push(ar1[x]);
    }
    return rs;
}



function getAmountOfDuplicates(numbers_list){
    var result = [];
    for(var x = 0; x < numbers_list.length; x++){
        if(numbers_list[x]==8)
            rs.push(numbers_list[x]);
    }
    return result;
}
```

Kuva 9. Esimerkki funktioiden kanssa

Yksinään pienet huonot nimeämiskäytänteet eivät tee vielä paljon tuhoja, mutta heti kun koodia on hiukankin enemmän, luettavuus kärsii todella paljon. Martin havainnollisti tätä esimerkillä (ks. Kuva 9), jossa kaksi funktiota verrattiin toisiinsa. Ylimmäinen koodi näyttää todella abstraktilta, ja koodin lukija ei pysty silmäyksellä sanomaan mitä koodi tekee. Sama koodi kirjoitettiin uudestaan, mutta tällä kerralla kaikki nimet on nimetty Martinin määrittelemän standardin mukaisesti. Koodin luettavuus parantui heti ja funktio dokumentoi itse itseensä vaatimatta kehittäjältä sen kummempia toimenpiteitä.

(Robert C Martin. Luku 2. “Meaningful names”, 2009)

Verrattuna omaan tekemiseeni sanoisin, että pyrin aina nimeämään mahdollisimman selkeästi kaikki olioiden sekä funktioiden nimet, mutta välillä saatan laiskasti kirjoittaa lyhenteitä, jotka minä tiedän henkilökohtaisesti, mutta muut kehittäjät eivät. Tästä tulee iso ongelma vähintään silloin, kun joku muu astuu kehittämään projekteja, joissa minun koodiani on. Jatkoa pohtien koen, että koodin laatu tulisi kulkea käsi kädessä ohjelmiston muiden aspektien kanssa. Jos olisin nimennyt kaikki oliot kunnolla, niin palvelun päivitys operaatiokin olisi sujunut paljon jouhevammin, eikä aikaa olisi mennyt hukkaan yrittäessäni ymmärtää omaa koodiani sekä ratkaista ongelmaa, jota ei edes ollut olemassakaan.

3.8 Seurantaviikko 8 17.3–20.3

Tiistai 17.3.2020

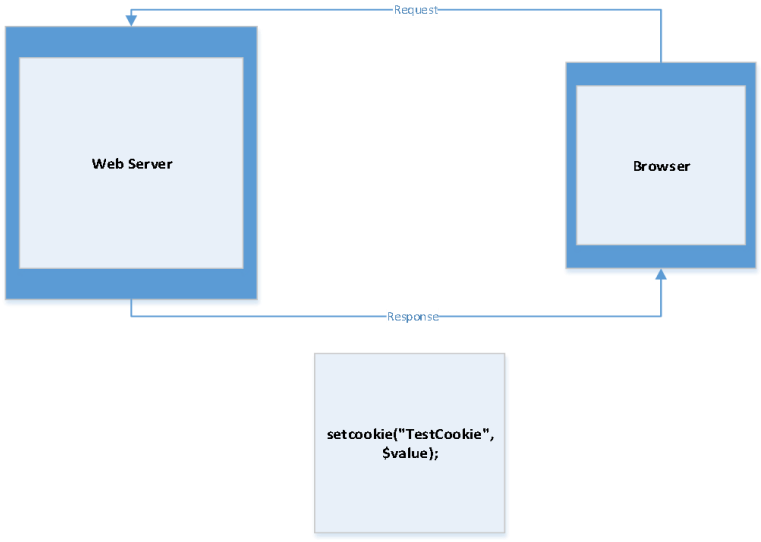
Työskentely todistus-laajennuksen parissa jatkuu. Tänään yritin selvitellä edelleen mahdollisuuksia PDF generoinnin suhteen. Ongelmana oli edelleen automaattinen sivutus sisällön määrän mukaisesti niin, että kirjasto pystyisi automaattisesti tekemään sivutukset. Alkuperäinen ratkaisu oli erillisten kirjastojen kokeilu, mutta yksikään ei tue kyseistä ominaisuutta ilman, että kaikki data tuodaan suoraan HTML:stä. Eli toisin sanoen vanha klassinen document.getbyelement skenaario, jota yritän minimoida niin paljon, kuin mahdollista, jotta käyttöliittymä koodi ei abstraktoidu liikaa.

Kokeiltuani muutamia mahdollisia kirjastoja tulin siihen päätökseen, että olisi parempi käyttää alkuperäistä jsPDF kirjastoa ja sen HTML pdf piirtämis- ominaisuutta, joka vaatii vain elementin sekä callback funktion, johon palautetaan piirroksen tulos.

Keskiviikko 18.3.2020

Työ jatkuu siitä mihin jäätiin eilen tulostus ominaisuuden kanssa. Tajusin tänään sattumalta lukiessani html2pdf kirjaston dokumentaatiota, että kyseinen kirjasto tarvitsee 4 muuta kirjastoa toimiakseen oikein. Tämä selitti miksei se toiminut ollenkaan. Virheenjäljitystä vaikeutti huomattavasti myös se, ettei virheviestejä tulostettu konsoliin vaan, kaikki virheet käsiteltiin softan puolesta ja ainoa tapa saada virhe viestit näkyville olisi console login kautta.

Seuraavaksi lähdin kokeilemaan PDF tulostamista html sisällöllä ja se lähti toimimaan oikein sutjakkaasti. Seuraavaksi pitäisi saada tuotua kaikki käyttäjän suoritustiedot kannasta suoraan tulostettavaksi. Ongelmana tällä hetkellä on se, että päätin toteuttaa erillisen esikatselu näkymän, josta käyttäjä voisi erikseen ladata tai tulostaa todistuksensa.



Kuvio. 14 Välimuisti ratkaisu

Huomenna tulisi siis selvittää, miten saan datan persistoimaan sivusta toiseen ilman backend operaatioita, jotka aiheuttaisivat turhaa lataamista. Mieleen juolahti heti ensimmäisenä yksinkertainen keksi skeema (ks. Kuvio 14), jossa data muunnetaan jsoniksi ja sitten tallennetaan tekstimuodossa käyttäjän selaimen välimuistiin.

Torstai 19.3.2020

Tänään kehittäminen eteni kovaa vauhtia eteenpäin. Jatkoin töitä eilisestä, jolloin yritin keksiä hyvän ratkaisun datan persistointiin sivujen välityksellä. Ensimmäinen ideani oli toteuttaa ratkaisu, joka tallentaisi kaikki tarvitsemamme tiedon välimuistiin. Kyseinen ratkaisu olisi ollut hiukan liioiteltu nähden käyttötarpeeseemme, jossa ainoa tarve oli käyttäjän kurssi suoritustietojen vieminen eteenpäin tulostettavaksi PDF tiedostoon.

Päädyin lopulta käyttämään laajennuksen kirjastoa, josta tieto voidaan kutsua suoraan kohde luokassa, joka haluaa käyttää käyttäjään suoritustietoja. Käyttäjän suoritustiedot haetaan siis ensin funktion avulla, josta se sitten tulostetaan esikatselu näkymään käyttäjän näkyville. Html2Pdf kirjasto osaa lukea html suoraan lähteestä ja osaa rakentaa siitä PDF tiedoston, johon myös lukeutuu kaikki kehittäjän määrittelemät tyylitykset.

Perjantai 20.3.2020

Tänään pääsin suunnittelemaan todistuksen tulostus-ominaisuuden esikatselu näkymää sekä kokeilemaan CSS tyylityksiä ja niiden vaikutuksia PDF tiedoston tulostamiseen. Huomasin todella nopeasti, että taustavärin muuttaminen toimii, mutta se ei värjää kaikkia sivuja oletuksena vaan, sen alueen, jossa on määriteltyä sisältöä. Joudutaan siis tyytymään valkoiseen taustaväriin.

Tämän lisäksi löysin toisen ongelman, jossa taulu jakautuu kahtia sivun vaihdossa ja osa tekstistä jää ensimmäiselle sivulle. En ehtinyt tänään vielä ratkaista ongelmaa, mutta luulen, että JavaScriptin avulla pystytään laskemaan, missä kohti meidän tulee määritellä katkos taululle ja jatkaa sitä seuraavalla sivulla. Jokainen sivu on tietyn kokoinen ja tämä määritys voidaan staattisesti antaa JavaScript koodille suoraan käsiteltäväksi, jolloin kaikki välit generoitaisiin aina tietyn välein.

Viikkoanalyysi

Tämän viikon päätteeksi haluaisin edelleen käsitellä samaa aihetta kuin viime viikolla, jolloin työni aikana nousi esiin hyvin useasti kysymyksiä koodin laadusta sekä miten voisin itse

välttää kirjoittamasta koodia, joka hidastaisi työpanostani. Tällä viikolla hyvä aihe olisi koodin kommentointi ja miten välttää turhaa kommentointia luettavamman koodin kautta. Käsittelen siis edelleen Robert. C. Martin Kirjaa Clean Code, jossa käydään läpi puhtaan koodin kannalta kaikki olennaisimmat asiat.

Omassa työssäni olen useasti yrittänyt kirjoittaa mahdollisimman paljon kommentteja, jotta koodi olisi dokumentoitu mahdollisimman hyvin. Ideana oli, että tarvittaessa joku toinen kehittäjä pystyy työskentelemään koodin kanssa ilman ylimääräistä sähellystä. Luulin, että tämä toiminta tapa oli absoluuttisesti paras, kun puhutaan koodin dokumentoinnista.
Martinin mukaan kommentointia tulisi aina välttää ja koodin tulisi aina dokumentoida itseään, koska suuri kommenttien määrä tekee koodin kanssa työskentelystä vaikeaa, mutta samaan aikaan kommentointi on tarvittava pahe siksi, että koodi ei pysty ilmaisemaan itseään tarpeeksi niin, että koodi voitaisiin kirjoittaa itsensä selittäväksi. Siksi onkin hyvä ymmärtää, milloin kommenttien käyttäminen on tarpeellista ja milloin sitä voidaan välttää.

(Robert C Martin. Luku 4. “Comments”, 2009)

```
function addHuman() {
  // Check to see if the employee is eligible for full benefits
  if (employee.flags & HOURLY_FLAG && employee.age > 65) {
  }
}
```

Kuva 10. Esimerkki kommentoidusta koodista

Teoksessa visualisoidaan yksi käyttötapaus, jossa koodin kommentointi voitaisiin välttää yksinkertaisella refaktoroinnilla, joka samaan aikaan dokumentoi itseään ja tekee koodista luettavampaa. Esimerkissä näytetään kaksi koodi esimerkkiä (ks. Kuva 10 ja 11), jossa kummatkin näyttävät kommentoidun version koodista ja refaktoroidun version.

(Robert C Martin. Luku 4. “Comments”, 2009)

Ensimmäisessä versiossa (ks. Kuva 10) koodi on selkeästi abstraktia ja sen lukeminen ilman kommenttia olisi huomattavasti vaikeampaa. Koodi refaktoroitiin niin, että ehto lauseesta tehtiin oma funktio ”employee” objektiin, joka nimettiin sen toteuttaman logiikan mukaisesti. Huomaa, että korjatussa esimerkissä (ks. Kuva11). Koodi on paljon luettavampaa ja se dokumentoi itsensä selkeästi.

(Robert C Martin. Luku 4. “Comments”, 2009)

```
function addHuman() {
  if (employee.isEligibleForFullBenefits()) {
  }
}
```

Kuva 11. Esimerkki koodista, jossa ei ole kommenttia

Teoksessa on määritelty käyttötapauksia, jossa kommentointi on melkein välttämätöntä. Esimerkiksi lakisääteiset kommentit, jossa määritellään tekijänoikeudet. Riippuen projektin luonteesta sekä yrityksestä. Usein kehittäjät joutuvat myös kommentoimaan toistensa ratkaisuja, jotta toinen kehittäjä ymmärtää miksi kyseinen ratkaisu on tehty. Koska käyttötapauksia on monia ja kaikkia ei ehditä käsitellä viikko analyysissa päätin ottaa muutaman esimerkin, jotka vaikuttavat eniten omaan työhöni.

(Robert C Martin. Luku 4. “Comments”, 2009)

Todo kommentit ovat eniten käytettyjä kommentti tyyppejä omassa työssäni ja Martin määrittelee, että kyseinen kommentti tehdään, kun tiettyä koodin pätkää halutaan muuttaa tai sen nykyistä ratkaisua halutaan selittää auki. On välttämätöntä, että kehittäjät ymmärtävät mitä muutettavalle koodin pätkälle tapahtuu tulevaisuudessa, muutoin voidaan luulla, että koodin pätkä on virhe. Esimerkissä (ks. Kuva 12) visualisoidaan vielä tarkemmin, miltä todo kommentti näyttää käytännössä.

(Robert C Martin. Luku 4. “Comments”, 2009)

```
//TODO-MdM these are not needed
// We expect this to go away when we do the checkout model
protected VersionInfo makeVersion() throws Exception {
    return null;
}
```

Kuva 12. Esimerkki todo kommentista.

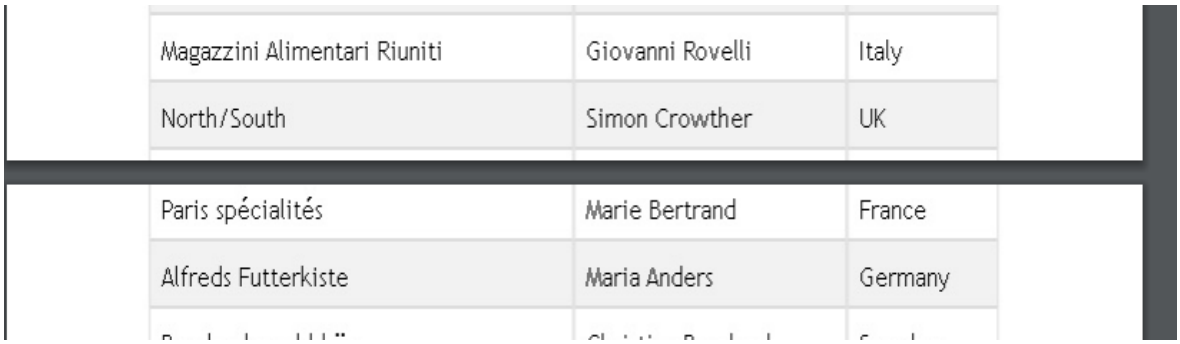
Verrattuna omiin käytänteisiin koen, että omassa koodissani kommentointia tapahtui liikaa ja olisi suotavaa, että sitä pyrittäisiin välttelemään niin kauan, kuin se on luettavuuden kannalta mahdollista. Koodin luettavuuden ylläpitäminen alkaa aina projektin alusta asti, kun sitä aletaan suunnittelemaan ja koenkin, että jo heti luokka relaatioiden suunnitteluvaiheessa voitaisiin vaikuttaa koodin kykyyn ilmaista omaa funktionaalisuuttaan. Jatkossa pyrin kirjoittamaan vain tarpeelliset kommentit ja jättämään kaiken ylimääräisen pois luettavuuden parantamiseksi.

3.9 Seurantaviikko 9 24.3–27.3

Tiistai 24.3.2020

Todistus ominaisuuden kehittäminen jatkuu ja tänään tehtäväni oli selvittää, miten kaikki sisältö saadaan loogisesti pysymään sivun sisällä niin, että se ei leikkaa itseään kahtia. Yritin ratkaista ongelmaa määrittelemällä staattisesti koon, johon kaiken sisällön tulisi mahtua mutta ääritapauksissa kyseinen ratkaisu saa osan kursseista näkymättömiksi, koska elementtien määrä menee staattisen koon yli.

Näytön kaappauksesta (ks. Kuva 13) näkyy vielä tarkemmin taulun rikkoutuminen sivun vaihtuessa, tähän pitäisi saada kehitettyä ratkaisu.



Kuva 13. Esimerkki Taulun rikkoutumisesta

Keskiviikko 25.3.2020

Tänään todistuksen kehittämisen piti jatkua saman tehtävän parissa, mutta sain kollegaltani viestin, että kurssilistauksen ominaisuuksilla on bugeja tietyillä selaimilla. Näistä ehkä kriittisin bugi on se, että kategoria filtteri ei toimi ollenkaan ja tämä persistoi tällä hetkellä vain edge selaimella. Tietyillä selaimilla on erilaiset tuet JavaScript ominaisuuksille, mutta yleisesti kaiken pitäisi toimia normaalisti.

Selvitysprosessini alkoi tiedonhaulla. Minun piti selvittää miksi ja mikä voi aiheuttaa kyseiset ongelmat. Löysin muutamasta stackoverflown foorumilta viesti ketjun, jossa todettiin, että edge ei tue kaikkia ES6 ominaisuuksia, jotka ovat käytössä kurssilistaus palikassa. Tämän ongelman ratkaisemiseksi, jouduin kirjoittamaan koodin osittain uudelleen, siten ettei se ole riippuvainen ES6 ominaisuuksista, joita ei edgessä tueta.

Ongelmana tässä on se, että koko laajennuksen käyttöliittymän hiearkia on kirjoitettu ES6 standardilla syntaksilla, niin, että uudelleen kirjoittaminen aiheuttaisi liian suuren työmäärän. Yritän siis selvitellä mahdollisuuksia, jotta liian pitkät korjaus ajat voitaisiin välttää.

Torstai 26.3.2020

Tänään jatkoin kurssilistauksen yhteensopivuus ongelmien korjaamista. Tänään kävi ilmi, että kurssilistaus ei toimi ollenkaan Safari sekä IE 11 selaimilla samoista syistä, kuin mainitsin eilenkin selvitellessäni filtteröinti ongelmaa. Internet Explorer 11 ja safari eivät tue ES ominaisuuksia ollenkaan ja näin ollen laajennus ei toimi kyseisillä selaimilla. Asiakkaita on ohjeistettu käyttämään muita selaimia, kunnes korjaus on saatu toteutettua.

Edelleen prioriteetti ykkösellä menee kuitenkin kategoria filtteröinti ongelma Edgellä, joka menee muiden ohi, koska käyttäjiä Edgellä on enemmän, kuin äskettäin mainituilla selaimilla, joten korjaus toimenpiteet olisi hyvä saada tehtyä niin nopeasti kuin mahdollista. Koodin korjaamisen aikana yksikkötestit ovat olleet paras ystäväni, sillä ne ovat tarjonneet suojaa ja varmuutta siitä, että koodi toimii niin kuin se on suunniteltu toimivan. Koska kirjoitin testejä koko ajan kehityksen aikana, pystyn nyt helpommin ja tehokkaammin testaamaan erilaisia ratkaisuja, ilman että ylimääräistä aikaa menisi koodin virheenjäljittämiseen.

Perjantai 27.3.2020

Selvittelin tänään kurssilistauksen suodatus ominaisuuden ongelmia dokumentoinnin yhteydessä ja luettuani hiukan selain yhteensopivuuksista lisää huomasin, että Edgestä on julkaistu uudempi versio, joka perustuu Chromium selaimeen. Jotta kaikki ES6+ ominaisuudet toimivat kaikilla käyttäjillämme oikein olen ohjeistanut kollegani kommunikoimaan, että on suositeltavaa päivittää Edge selain uusimpaan versioon.

Edgen versio 18 ei tue dynaamisia moduuli importteja, joka tarkoittaa sitä, että kaikki kategorian suodatukseen liittyvät moduulit eivät pysty ajamaan selaimessa, aiheuttaen näennäisesti suodatus ominaisuuden rikkoutumiseen.

Vaikka ehdinkin toteuttaa hotfix juureen jo korjauksia, päätin laittaa muutokset odottamaan kommunikaatiota kollegalta. En halua lähteä tekemään enempää muutoksia ennen kuin tilanne on arvioitu uudelleen löydetyn tiedon jälkeen.

Viikkoanalyysi

Jatkan edellisen viikon aihetta paremmasta koodista sekä Robert. C. Martinin teosta. Haluan tänään käsitellä hyviä käytänteitä, joita kehittäjä käyttäessään pystyy organisoimaan kirjoitetut luokat. Hyvin useasti luokat voivat paisua todella suuriksi, jolloin niistä tulee niin isoja, että on vaikeaa määritellä mitä kyseisen luokan pitäisi tehdä. Tämä johtaa myös vaikeaan testattavuuteen ja jossain vaiheessa uusia ominaisuuksia on vaikea lisätä luokkaan.

```
public class SuperDashboard extends JFrame implements MetaDataUser {

public Component getLastFocusedComponent()
public void setLastFocused(Component lastFocused)
public int getMajorVersionNumber()
public int getMinorVersionNumber()
public int getBuildNumber()
}
```

Kuva 14. Luokka esimerkki 1

Ensimmäinen ja ehkä tärkein sääntö on luokan pitäminen mahdollisimman pienenä. Robertin sanoin pienempääkin pienempänä, kun luokkia suunnitellaan. Hänen mukaansa luokan koko määritellään sen velvollisuuksien määrän mukaan. Velvollisuudella tarkoitetaan luokan jokaista loogista tehtävää, jota sen pitää suorittaa. Teoksessa painotetaan, että funktioiden tai rivien määrä ei määritä velvollisuuksien määrää. Hänen esimerkissään (ks. Kuva 14) havainnollistetaan superdashboard luokkaa, joka pienestä koostaan yrittää tehdä liian monta asiaa ja sen nimi jättää todella paljon tulkinnan varaa. Martinin mukaan luokan tehtävä on pystyttävä määrittelemään 25 sanalla ja sen nimi tulee pystyä kertomaan kehittäjälle mitä se yrittää saavuttaa. Esimerkin luokan tehtävänä on hallita ohjelmiston versiota sekä ohjelman JW komponentteja, joiden tehtävä on käyttöliittymän piirtäminen ja näyttäminen.

(Robert C Martin. Luku 10. “Classes”, 2009)

Parempi lähestymistapa Martinin mukaan olisi jakaa luokka kahtia niin, että toinen luokista hallitsee ohjelmiston versiota ja toinen ohjelman käyttöliittymää. Eli jokaiselle luokalle on määrätty vain yksi velvollisuus. Kyseinen toimintatapa tunnetaan paremmalla termillä yhden velvollisuuden periaate ”Single Responsibility Principle”. Se määrittelee, että luokalla tulisi olla vain yksi syy muuttua ja että velvollisuus antaa kyseisen luokan velvollisuuden määritelmän sekä yleispätevän ohjenuoran luokan koolle. Martin myös noteeraa velvollisuuksien tunnistamisen tärkeyden, koska se myös auttaa luomaan parempia abstraktioita koodille.

Hyvin usein kehittäjillä on pelko siitä, että luokkia on liikaa ja selkeää yleiskuvaa on vaikea hahmotella ja se havainnoidaan näyttävän siltä, että ohjelmalla liian monta liikkuvaa osaa.

Martin painottaa, että velvollisuuksien jaotteleminen luokille helpottaa jälkikädessä, kun jokaisella on yksi velvollisuus ja selkeästi nimetty kokonaisuus.

(Robert C Martin. Luku 10. “Classes”, 2009)

Luokkien velvollisuuksien lisäksi tärkeäksi tekijäksi luokkien jakamisessa Martin on määrittänyt ajon aikaiset oliot, joita luokan funktiot mahdollisesti käsittelevät tai manipuloivat jossain määrin. Luokka, jossa kaikki arvot ovat käytössä jokaisessa funktiossa voidaan määritellä olevan maksimaalisesti yhtenäinen. Yleisesti täysin yhtenäisten luokkien kirjoittaminen ei ole suositeltua tai edes mahdollista, koska on melkein mahdotonta kirjoittaa useita funktioita, joissa jokaista oliota käytetään. Luokka on siis yhtenäinen, kun olioiden ja funktioiden kollektiivinen riippuvaisuus toisistaan on korkea. Teoksessa painotetaan, että kyseisissä tapauksissa luokka olisi hyvä jakaa kahtia ja määritellä molemmille omat tehtävänsä ja siten rakentaa riippuvaisuudet niiden ympärille.

(Robert C Martin. Luku 10. “Classes”, 2009)

Kun itse kirjoitan luokkia huomaan, että esimerkiksi luokat, jotka kantavat dataa tekevät myös ylimääräisiä toimenpiteitä, kuten tarkistavat kyseiseen objektiin liittyvät ehdot tai tekee liian monta asiaa kerrallaan kuten esimerkiksi kantaa dataa ja antaa käyttäjälle tietoa näkyviin. Pahimmillaan tämä voi luoda ylimääräisiä tasoja, jotka monimutkistavat koodiani sellaiseen pisteeseen, että päädyn uudelleenjärjestämään sitä. Tämän hetken toimintaperiaatteeni on kirjoittaa uudelleen, jos koodi ei näytä loogiselta, joten siihen voi myös mennä paljon työpanosta, joka voitaisiin muuten käyttää tärkeämpiin asioihin.

3.10 Seurantaviikko 10 30.3–3.4

Maanantai 30.3.2020

Työtehtäväni on tänään korjata kurssilistaus-laajennuksen korkeimmalla prioriteetilla oleva bugi, joka tuli ilmi viime viikolla. Sain selvitettyä ongelman syyn sekä mahdollisen ratkaisun sen ympärille, mutten aloittanut toimenpiteitä ennen kuin sain seniori kollegalta mielipiteen siitä, miten meidän kannattaisi edetä asian kanssa. Päädyimme yhdessä siihen tulokseen, että bugi tulisi korjata välittömästi, koska uusi chromium edge on vasta hiljattain julkaistu ja kaikkien käyttäjien koneisiin ei ole vielä asennettu uutta selainta.

Tiistai 31.3.2020

Tänään työtehtäväni muuttuivat hetkellisesti, kun seniori kollega pyysi tiimiämme auttamaan uuden Moodle instanssin käyttöönotossa. Uusi instanssi tulee asiakasyrityksen käyttöön, joka haluaa Wistec:in elektronisen palvelun käyttöönsä niin, että palvelu olisi tyylitelty heidän brändinsä mukaisesti. Eli asiakkaalle tehtäisiin klooni alkuperäisestä alustasta.

Normaalisti kaikki päivitykset ja muutokset laajennuksiin synkronoitaisiin erillisten alustojen välillä käyttämällä deployment pipelinea, joka osaisi puskea muutokset jokaiselle asiakkaalle automaattisesti. Koska emme ole sellaista järjestelmää implementoineet ja ottaneet käyttöön yrityksessä, joudumme asentamaan ja poistamaan kaikki asiakkaalle ei tarkoitetut laajennukset manuaalisesti.

Keskiviikko 1.4.2020

Työtehtäväni jatkuvat tänään Kurssilistauksen korjaamisen parissa. Ongelman syy ja juuri oli aivan eri, kuin mitä olin aikaisemmin todennut ratkoessani kategoria filtteröintiin liittyvää ongelmaa. Filtteröinti ei toiminut oikein, koska käyttöliittymässä on hyödynnetty select elementtiä, jolla on muutama callback even funktio, joita kehittäjät voivat käyttää. Olin itse päättänyt käyttää input eventtiä, joka tuntui toimivan hyvin selaimilla, joilla olin sitä testannut kehittäessäni ominaisuutta.

Kävi kuitenkin ilmi, että Edge selain ei tue input event ominaisuutta select elementin yhteydessä, joten koodi piti korjata käyttämään onchange eventtiä, joka laukaisee tapahtuman, kun elementin arvo muuttuu.

Torstai 2.4.2020

Tänään siirsin työpanokseni toisen projektin pariin kollegani pyynnöstä muokata profiili-laajennusta käyttämään välimuistia, jotta tietoa ei tarvitsisi ladata suoraan tietokannasta jokaisella kerralla, kun käyttäjä avaa työpöytänäkymän palvelussa. Hänen mukaansa välimuistin hyödyntäminen optimoisi palvelun suorituskykyä ja nopeuttaisi latausaikoja huomattavasti.

Moodlella on oma rajapintansa, jonka kautta jokainen laajennus pystyy tarvittaessa kirjoittamaan omaan kontekstiinsa liittyvät tiedot välimuistiin. Se on toteutettu niin, että jokainen laajennus määrittelee omalle muisti alueellensa nimen ja konfiguroi sen omiin asetuksiinsa, josta Moodle osaa aina viitata takaisin.

Tehtävänäni oli siis muokata profiili-laajennus tallentamaan välimuistiin kaikki tieto, joka näytetään käyttäjälle. Prosessi eteni sulavasti, sillä moodlella oli rajapinnasta olemasta hyvin kattavat esimerkit, joten pystyin helposti ymmärtämään, miten sen funktioita käytetään.

Torstai 3.4.2020

Työtehtäväni jatkuvat edelleen välimuisti ominaisuuden kehittämisen parissa. Sain kehitettyä ominaisuuden, mutta testauksen aikana ilmeni ongelma, joka edellisiin ongelma tilanteisiin verrattuna oli todella abstraktoitunut. Kun käyttäjän tiedot ladataan välimuistista, taulun filtteröinnit menevät täysin epäkuntoon ja konsoli viittaa tietyn kirjaston aiheuttamaan virheeseen. Virhe viittaa päivämäärä olioon, joka määrittää taulun aika formaatin suomen standardiin. Yritin selvittää, miksi taulu ei pysty ajamaan instantioivaa funktiotaan taulun alussa, kun käyttäjän data ladataan välimuistista.

En saanut tänään vielä minkäänlaista syytä sille, mutta tässä vaiheessa pitää vakavasti harkita laajennuksen renderöinti funktionaalisuuden uudelleen kirjoittamisen, sillä kuten viime viikon analyysissa jo käsiteltiinkin. Kehittäminen on nopeampaa, kun koodin arkkitehtuuri on puhdasta ja tässä tapauksessa tunnistin rikkoneeni Martinin sääntöjä liittyen luokkiin. Laajennuksen pääluokka on räjähtänyt täysin käsiin ja minun on vaikea lukea omaa koodiani, jota ei ole dokumentoitu kunnolla.

Viikkoanalyysi

Edellisillä viikoilla käsittelemäni aihe puhtaasta koodista ja hyvistä työskentely tavoista on pysynyt edelleen relevanttina heijastuen tämänkin viikon työtehtäviin. Siispä koen tärkeäksi käsitellä aihetta lisää ja olen päättänyt käsitellä tällä kerralla huonon koodin tunnistamiseen liittyviä tekijöitä, joita tulikin esille päiväkirjani aikana muutamaan otteeseen. Ero edellisiin kertoihin on se, että silloin ns. Koodi hajuja käsiteltiin pikaisesti uudelleenjärjestämisen näkökulmasta ja en ehtinyt uppoutua niihin niin paljon kuin olisin halunnut, mutta Robert C Martinin teos puhtaasta koodista käsittelee aihetta kattavasti ja koin siksi nyt olevan hyvä hetki ottaa aihe uudelleen käsittelyn alle.

Teos käsittelee Martin Fowlerin teoksessa “Refactoring”, määriteltyjä koodi hajuja ja lisää siihen omat muokkauksensa ja täydentää jo valmiiksi rakennettua listaa. Fowlerin teosta käsiteltiin viikolla 5 uudelleenjärjestämisen tarpeen näkökulmasta, jolloin koin, ettei koodi

hajuja tarvitse käsitellä liian pitkälle, sillä työssäni ei vielä esiintynyt sellaisia ongelmia, jotka olisin voinut mieltää olevan koodihajujen syytä.

Ensimmäinen kategoria koodi hajun tunnistamiselle on määritetty olevan huonot kommentit. Kommentteja käsiteltiin jo viikkoa aikaisemmin, joten pidän tämän osion hiukan lyhempänä. Kaikista määritetyistä hajuista, joita listattiin parhaiten kohdalleni osui koodin kommentointi. Ohjelmiston rakentamisen aikana tulee testattua kaikenlaisia ratkaisuja ongelmiin ja hyvin usein niistä jää kommentoituja ylijäämiä, joilla ei ole minkäänlaista funktiota koodissa. Martinin mukaan sellaiset jäämät ovat vaarallisia, kun ne saavat jäädä versionhallintaan pidemmäksi aikaa. Pääargumentti sille on annettu koodin vanhentuminen, eli kaikki oliot ja funktio kutsut ovat vanhentuneita tai eivät ole enää olemassa koodissa.

(Robert C Martin. Luku. 17 “Code Smells”, 2009)

Teoksessa käsiteltiin funktioita ja niiden aiheuttamia koodi hajuja. Valitsin kategorian käsiteltäväksi, koska huomasin omassa koodissani vastaavanlaisia virheitä, joita haluaisin aina välttää tulevaisuudessa. Martin määrittelee kategoriassa neljä virhettä, joiden pitäisi välittömästi hälyttää kehittäjän varoituskellot. Niistä parhaiten omalle kohdalleni osuvat ovat kuolleet funktiot sekä argumenttien määrä.

Kuolleilla funktioilla tarkoitetaan koodia, jota ei kutsuta missään, mutta ne kuitenkin ovat olemassa versionhallinnassa. Yleinen käytänne on Martinin mukaan poistaa kuolleet funktiot välittömästi, koska vaikka se olisikin tärkeä versionhallinta muistaa sen aina ja kehittäjät voivat tarvittaessa palata takaisin kyseiseen juureen tarkastelemaan muutoksia.

Yleisesti hyvänä käytänteenä pidetään Martinin mukaan pieni määrä argumentteja funktioissa. Suositeltu määrä on aina käyttötapauksen mukaan yhdestä kolmeen argumenttia ja kolmekin on teoksessa määritelty olevan kyseenalaista ja silloinkin argumentit kannattaa pitää yksinkertaisina. Esimerkiksi lippu argumentit eli Booleani ovat Martinin mukaan myös koodi hajuja, koska ne ovat selkeä merkki siitä, että funktio yrittää tehdä liian monta asiaa samaan aikaan.

(Robert C Martin. Luku. 17 “Code Smells”, 2009)

Vanhan koodin kanssa työskennellessäni olen huomannut todella useasti, että Moodlen koodissa yli puolet funktioista määrittelevät isoja määriä argumentteja ja kaikki argumentit on nimetty yksinkertaisesti vain lyhenteillä ja koska olin itse yrittänyt koodata Moodlen

asettamalla esimerkillä ja standardilla, olin itsekin päätynyt käyttämään liian montaa argumenttia kirjoittamissani funktioissa.

Yksi todella iso koodi haju, jonka tunnistin teoksen määrittelemästä listasta on enemmän kuin yhden koodikielen sijaitseminen yhdessä lähdetiedostossa. Tämä on todellakin raapinut omaa mieltäni ja olen aina ensimmäisestä päivästä lähtien työssäni yrittänyt ratkoa, miten pystyisin kirjoittamaan järkevämpää koodia Moodlessa, ilman, että sekoitan monta kieltä yhteen tiedostoon. Pahimmassa tilanteessa yhdessä tiedostossa saattoi olla kolme kieltä sekoitettuna, jotka kaikki ajavat eri logiikkaa. Martinin mukaan kehittäjän tulee aina pyrkiä minimoimaan kielien määrä yhdessä tiedostossa, jotta luettavuus ja luotettavuus pystytään ylläpitämään. Ideaalisesti parhaassa tilanteessa oltaisiin silloin, kun yhdessä tiedostossa käytetään vain yhtä kieltä logiikan ajamiseen.

Koen oppineeni paljon tämän viikon aiheesta ja yleisesti koodi hajujen välttäminen vaatisi todella paljon työprosessin muuttamista, johon pitäisi saada sisällettyä koodinkatsaussessioita, vaikka ne olisivatkin yksin toteutettuja. Pääpointtina on kuitenkin, se että koodia ei pitäisi kirjoittaa kirjoittamisen takia, mutta myös ylläpitää laatua, joka takaa aina jälkikädessä paremman ylläpidettävyyden.

Pohdinta ja päätelmät

Päiväkirjan kirjoittaminen työstäni Wistec Training:lla on ollut minulle täysin uusi kokemus. En ollut aikaisemmin tehnyt vastaavanlaista työtä, jossa omaa osaamista sekä työskentely tapoja analysoidaan jatkuvasti. Viikkoanalyysit ovat käsitelleet erilaisia aiheita ja jokainen niistä on antanut minulle mahdollisuuden kehittää omia sekä yrityksen työskentely prosesseja. Lähtötilanteeni oli se, että halusin näkyvästi parantaa omaa ammattitaitoisuuttani ohjelmistokehityksen alalla ja puskea juniori tason yli. Kehityksen painopisteeksi oli asetettu parempi koodin laatu sekä web arkkitehtuurien ymmärtäminen syvemmällä tasolla.

Päivittäisiä merkintöjä tehtäessä huomasin, että työssäni nousi esiin teemoja, jotka keskittyivät suurelta osin ohjelmiston kehittämisen prosessiin sekä ohjelmiston laadun ylläpitämiseen. Tällöin kehityksen kohteista jälkimmäinen jäi täysin pois päiväkirjan kirjoittamisen aikana. Tästä huolimatta olen oppinut paljon ohjelmiston kehittämisen prosessista ja miten se toimii oikeassa yritysympäristössä.

Kolme ensimmäistä viikkoa käsitteli omia sen hetken työskentelymalleja ja vertasi niitä kirjallisuudessa oleviin malleihin. Huomasin jo heti ensimmäisen viikon aikana, että omat

työskentelytavat eivät sopineet yhden henkilön kehitys prosessiin ja että niitä pitäisi muokata. Koska bugeja oli syntynyt ohjelmistoon huomattava määrä, minun piti analysoida omia yksikkötestaamisen menetelmiä, joiden piti teoriassa varmistaa, että koodin logiikka pätee haluttuihin tuloksiin. Oleellisimmat mallit, jotka sain toteutettua omaan päivittäiseen työskentelyyn, olivat testi lähtöinen kehittäminen, versionhallinta rebase mallilla sekä bugien jäljittäminen käyttämällä Carterin ydin mallia.

Testi lähtöinen kehittäminen muutti työskentelytapojani valtavasti, kun huomasin analysoidessani omia käytänteitäni viikoilla 1 ja 6. Ensimmäisen viikon analyysissa vertasin omia yksikkötestaamisen käytänteitäni testi lähtöisen testaamisen menetelmään, joka tuli esille etsiessäni parempia kehittämiseen käytettäviä malleja, jotka sopivat parhaiten soolo kehittäjälle. Käsittelin prosessejani vedoten Beckin teokseen (Beck, Kent. Test Driven Development: By Example). Huomasin että teoksen kuvaama testilähtöinen kehittäminen sopii hyvin omaan työhön, jossa työskentelyä tehdään pääasiallisesti yksin.

Uuden mallin myötä olen pystynyt kirjoittamaan parempia yksikkötestejä, jotka kuvaavat tiettyjen luokkien sekä funktioiden käyttötapauksia tarkemmalla tasolla. Isoin oppi oli käyttäjätarinoiden tärkeys testien kirjoittamisessa, jonka sisällyttäminen omaan prosessiin auttoi varmistamaan, että sovellus tekee sen minkä asiakas haluaa. Kirja sai minut ymmärtämään, että kaikkia trendikkäitä työmenetelmiä ei tarvitse sisällyttää omaan päivittäiseen työhön vain siksi, että ne ovat suosittuja suurten teknologiayritysten keskuudessa, vaan on paljon viisaampaa käyttää työmenetelmiä, jotka sopivat minulle yksilöllisenä kehittäjänä.

Yksikkötestejä kirjoittaessa piti myös ottaa huomioon koodin luonne, jonka kanssa työskentelin. Moodle on vanha ohjelmisto, joka kehitettiin ohjelmistotuotannon teollisuuden ollessa vielä nuori, ja monien nykyiset työmenetelmät eivät olleet vielä käytössä tai ne olivat erilaisia. Valitettavaa oli myös se, että ensimmäisten 7 viikon aikana työskentelin vanhan softa version vaatimana vanhentuneella PHP-versiolla (5.2), joka lisäsi työhöni omat haasteensa. Minun piti olla varovaisempi uusien ominaisuuksien luomisessa ja vaikka Moodlella onkin hyvä infrastruktuuri, jonka avulla kehittäjät voivat helposti luoda uusia toimintoja Moodlen ekosysteemiin, en pystynyt ylenkatsomaan sen tuomia haittoja.

Vanhan koodin kanssa työskenteleminen on ollut oleellisessa roolissa päivittäisessä työssäni ja on opettanut minulle monia asioita. Ennen kuin luin vanhan koodin määritelmää ja menetelmiä sen kanssa työskentelemiseen, henkilökohtaiset menetelmäni eivät olleet millään tavalla mukautettu tilanteen vaatimuksiin. En ymmärtänyt, kuinka helposti voi eksyä koodiin, jonka joku muu kirjoitti, ja kuinka tärkeätä oman työn dokumentointi on. Tulin tähän oivallukseen yrittäessäni löytää lisätietoja Moodlen infrastruktuurista ja tutkiessani heidän

lähdekoodissansa käytettyjä malleja. Vaikka sitä kehitetään edelleen, oli selvää, että monien vuosien takaa oli jäljellä paljon vanhoja koodinpätkiä, joiden käyttämiä koodin kirjoitus tyylejä voidaan pitää nykyään vanhentuneina.

Päivittäisessä työrutiinissani koodin laadun ylläpitäminen on ollut takaraivossani piilevänä ajatuksena. Aika ajoin tapani kirjoittaa koodia oli johtanut uudelleenjärjestämiseen. Uudelleenjärjestämisen tarve on yleensä tullut ilmi ohjelman hetkellisen rakenteen aiheuttamana taakkana, jonka takia joitain ominaisuuksia tai lisäyksiä ei pystytty implementoimaan niin kuin oli haluttu. Vaikka syynä olikin vain virheet, joita oli tehty projektin alussa.

Hyvän koodin kirjoittaminen on jokaisen kehittäjän tavoite ja usein näistä tavoitteista joudutaan joustamaan projekteissa, joissa aikataulut velvoittavat tiimin viemään tuotteensa tuotantoon määritetyssä ajassa. Vaikka itselläni ei ollut samankaltaisia rajoitteita huonoista koodin pätkistä ei voitu välttyä. Kysymys hyvän ja huonon koodin erosta nousi työssäni esiin päivittäin ja koska se oli oleellisessa osassa omaa kehitystäni, päätin keskittää 3 perättäistä viikkoa sen käsittelyyn. Jokaisena viikkona käsiteltiin eri aihetta liittyen parempaan koodiin kuten luokkien rakentaminen ja logiikan erittely luokkien välillä. Isoin oppi, jonka sain näiden viikkojen aikana, oli se, että koodin tulisi aina olla niin yksinkertainen kuin vain mahdollista. Käsittelin aiheeseen liittyen Robert Martinin teosta ”Clean Code” ja huomasin lukiessani sen käsittelemiä malleja, että Moodlen koodi tyyli sekä jotkin vanhemmat luokat vastasivat kirjoissa näytettyjä esimerkkejä huonosta koodista, jossa luettavuus oli kärsinyt. Tämä on mielenkiintoinen havainto siinä mielessä, että olin ennen viikkojen 8–10 analyyseja kirjoittanut koodia Moodlen ohjeiden mukaisesti ja koin, että kokonaiskuvan hahmottelu olisi helpompaa, kun kaikki koodi on johdonmukaista ja tyyli on suurin piirtein sama.

Koodin laadun ylläpitämisen lisäksi työssäni esiintyneet bugit olivat erittäin iso taakka, joka todisti minulle kokemattomuuteni ja kenties jopa heikkouksia omassa osaamisessa. Virheiden jäljittäminen tuotti huomattavia ongelmia, joiden selvittämiseen meni hukkaan niin paljon aikaa, että tarvitsin tueksi sellaisen työskentely mallin, jolla prosessia voitaisiin helpottaa. Viikolla 4 käsittelemäni Carterin ydin malli (ks. Kuvio 8) selkeytti virheenjäljittämisen prosessin neljään ydinkonseptiin. Mallin hyödyntäminen työssäni helpotti virheiden jäljitystä ja muutti lähestymistapaani uusiin virheisiin, joita tuli tätä seuraavilla viikoilla. Kaikki tämän mallin myötä oppimani tulee olemaan erittäin hyödyllinen, kun joudun selvittelemään virheitä tulevaisuuden projekteissa.

Jokaisen tehdyn viikon analyysin jälkeen pystyin hyödyntämään kaikkea oppimaani välittömästi työssäni ja mukauttamaan omia työ menetelmiä ja malleja vastaamaan lähteistä

oppimiani malleja. Uusien työskentelytapojen ja käytänteiden hyödyntäminen ei jäänyt pelkästään omalle yksilölliselle tasolle vaan pystyin tuomaan sitä myös yrityksen päivittäiseen kulttuuriin. Jatkossa kaikki Legacy ohjelmistoihin liittyvät projektit tulevat olemaan paljon helpompia toteuttaa nykyisellä osaamisella ja tiedoilla, jotka olen saanut tämän työn kirjoittamisen tuloksena.

Wistec Training Oy:n ylläpito tiimissä työskentely on aina haastanut osaamiseni absoluuttisiin rajoihin ja sen myötä koen osaamisessani tapahtuneen kehitystä koko ajan ilman minkäänlaisia hidasteita. Jossain vaiheessa kehittyminen yhdessä työpaikassa saavuttaa kattonsa, mutta henkilökohtaisesti oma osaamiseni on kehittynyt jatkuvasti ja uskon, että minulla on vielä tilaa oppia paljon ennen kuin oma kattoni on saavutettu. Seuraava kehittymiseni tavoitteet olisi realistisesti hyvä tähdätä uudempiin teknologioihin ja yrittää integroida niitä omaan työskentely-ympäristöön.

Lähteet

Shore, James. The Art of Agile Development, 2007.

McMillan, Miachel. Data Structures and Algorithms with JavaScript, 2014.

Westby, Emma Jane Hogbin. *Git for Teams*. O'Reilly Media, 2015.

Lasater, Christopher G. Design Patterns, 2010.

Martin, Robert Cecile. Clean Code, A Handbook of Agile Software Craftsmanship, 2009.

Fowler, Martin, ja Kent Beck. Refactoring: Improving the Design of Existing Code. Addison-Wesley Professional, 1999.

Beck, Kent. *Test Driven Development: By Example*. Addison-Wesley Professional, 2002.

Wengrow, Jay. A Common-Sense Guide to Data Structures and Algorithms, 2017.

Feathers, Michael C. Working Effectively With Legacy Code, First Edition. Prentice Hall, 2004.

Butcher, Paul, ja Jacquelyn Carter. Debug It! Pragmatic Bookshelf, 2009.

Liitteet

Liite 1. Ammattisanasto ja keskeiset käsitteet

Uudelleenjärjestäminen: ”refactoring” tarkoittaa koodin uudelleen kirjoittamista tai sen suunnitellun rakenteen muuttamista niin, ettei sen alkuperäinen logiikka muutu.

Koodinkatsaus: ”code review” on prosessi, jolla koodin laatu valvotaan toteuttamalla katselmointi sessioita, joiden aikana kehittäjät antavat toistensa koodista mielipiteensä.

Vanha ohjelmisto: ”legacy software” on nimensä mukaisesti vanha ohjelmisto.

Moduuli: Moodleen asennettava laajennus ohjelmisto, joka lisää tai muokkaa alustan ominaisuuksia.

Tiekartta: ”Roadmap” on suunnitelma, jolla määritellään tuotteen tai jonkin asian tavoitteet monella eri askeleella.

Backend: Termi web arkkitehtuurin palvelin logiikalle.

Repositori: GIT versionhallinta ympäristön hallinnoima kansio