Iikka Konola

OHJELMISTOJULKAISUN AUTOMAATIOTESTAUS

Katsaus ohjelmistotestauksen käytäntöihin

OHJELMISTOJULKAISUN AUTOMAATIOTESTAUS

Katsaus ohjelmistotestauksen käytäntöihin

Iikka Konola
Opinnäytetyö
Kevät 2023
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Insinööri (AMK), Tietotekniikan tutkinto-ohjelma, ohjelmistokehitys

Tekijä: Iikka Konola
Opinnäytetyön nimi: Ohjelmistojulkaisun automaatiotestaus. Katsaus ohjelmistotestauksen käytäntöihin.

Työn ohjaaja: Eino Niemi
Työn valmistumislukukausi ja -vuosi: Kevät 2023 Sivumäärä: 41

Opinnäytetyössä kerrotaan päiväkirjatyyppisesti ohjelmistojulkaisun automaatiotestauksen kehityksestä testaajan näkökulmasta. Työ raportoitiin päivittäin ajanjaksolla 12.9.2022 – 4.11.2022. Työn suorituspaikka oli nelihenkinen ohjelmistokehitystiimi suomalaisessa pörssiyhtiössä.

Tarkastelujakson aikana Qt valmisti monenlaisia ohjelmistoja erilaisiin käyttötarkoituksiin, joista suurin osa yrityksen tuoteportfoliosta asentui asiakkaiden koneille Qt Installer -ohjelmiston kautta. Qt Installer oli valmistettu Qt Installer Framework -ohjelmistolla, jota kehitti Oulussa sijaitseva ohjelmointitiimi. Tiimin uutena jäsenenä perehdyin olemassa olevaan automaatiotestausprojektiin ja aloin kehittää uusia testejä sekä ylläpitää jo olemassa olevia.

Installer framework -projektin tarkoitus oli kehittää Qt Install Framework -ohjelmistolle automaatiotestausta Squish-testaustyökalulla. Pääasiallinen ohjelmointikieli oli Python. Raportoinnin aiheina olivat erilaiset näkökulmat ohjelmistotestaamiseen ja ohjelmistotestaajan roolissa toimimiseen sekä erilaiset liitännäisteknologiat testien kehittämisen rinnalla.

Johtopäätöksiä omien toimintatapojen muutoksista sekä prioriteettien asettamisesta saatiin päivittäisestä työstä. Prioriteettien asettaminen selkeni jakson aikana oman viikkontason työskentelyn osalta. Lisäksi saatiin erilaisia näkökulmia ohjelmistotestauksen kehittämiseen projektissa käytännön kokemuksien kautta.

Asiasanat: tietokoneohjelmat, tietokoneohjelmien validointi, testaus, ohjelmistokehitys,

päiväkirjamuotoinen opinnäytetyö

ABSTRACT

Oulu University of Applied Sciences
Bachelor of engineering, information technology, option of Software Development

---

Author: Iikka Konola
Title of thesis: Automation testing of software release
Supervisor: Eino Niemi
Term and year when the thesis was submitted: Spring 2023
Number of pages: 41

During the reporting period, Qt Company was building software for many different purposes, but most of its software portfolio is Installed via Installer Framework. We were working on software automation for this software and I recorded my thoughts and findings in this thesis.

This bachelor's thesis consists of diary-style records from daily work of a software test engineer's first months starting his work in software development team. Work is documented from 8 weeks in the period of 12.9.2022 - 4.11.2022.

Keywords: Software, software validation

SISÄLLYS

1 JOHDANTO

1.1 Opinnäytetyön aihe ja muoto

Tämä opinnäytetyö on päiväkirjamuotoinen, eli se on kertomus päivittäisestä työstäni ohjelmistotestaajana suomalaisessa suuryrityksessä. Tekstissä seuraan päivittäistä työskentelyäni sekä uudessa työtehtävässä kehittymistä. Seurannan alussa olen ollut töissä yrityksessä noin 4 viikkoa. Teksti seuraa työviikkojen tapahtumia ajalla 12.9.2022–4.11.2022. Opinnäytetyössä kuvaan työni suunnittelua sekä käytettyjä työmenetelmiä. Lisäksi kuvaan sidosryhmien vaikutusta tiimin toimintaan.

Raportointi tapahtuu päivittäin ja koostuu työtehtävien kuvauksesta sekä tulevan työn suunnittelusta. Viikoittain tarkastelen erilaisia viikon aikana ratkaistuja ongelmia sekä arvioin toteumaa suhteessa tavoitteisiin. Pohdin tiimien välistä dynamiikkaa, sillä työskentelen tiiviisti kahdessa tiimissä, joista toinen tuottaa asennusohjelmaa ja toinen on varsinainen testaustiimi. Hyvin hallittu ohjelmistojulkaisu vaatii myös muiden kehitystiimien projektien tilanteesta hyvän ymmärryksen.

1.2 Työn suorittamispaikka ja työympäristö

Työskentelen The Qt Company Oy:lla, joka on suomalainen ohjelmistoalan suuryritys. Qt on perustettu vuonna 1994 Norjassa ja yrityksen alkuperäinen nimi oli Trolltech. Qt työllistää kansainvälisesti yli 600 henkilöä ja pelkästään Oulun toimipisteellämme työntekijöitä on viidestätoista eri valtiosta. Yrityksen liiketoiminta perustuu alustariippumattomien ohjelmistojen ja graafisten käyttöliittymien ohjelmointiympäristön sekä käyttöliittymäkirjastojen kehittämiseen ja markkinointiin (Wikipedia 2022).

Ohjelmistoja itsessään kehittää laaja kokonaisuus eri tiimejä, mutta ohjelmistojen releasetestaukseen eli julkaisutestaukseen keskittyy oma testaustiimi. Releasetestauksella tarkoitetaan ohjelmistoversion julkaisua edeltävää automaattista ja manuaalista testaamista uusien ominaisuuksien toiminnan varmistamiseksi. Releasetestaustiimi koostuu seitsemästä henkilöstä. Tiimin testaajista osa työskentelee tiiviisti releasetiimissä ja osa on jaettu työskentelemään eri tuotteita, kuten Qt Design Studiota, Installer Frameworkia ja Boot to Qt:ta kehittävien tiimien kanssa. Oma työnkuvani

koostuu Installertiimin kanssa yhteisistä palavereista ja Installeria varten kehitettävien automaatiotestien kehittämisestä.

1.3 Osaamisvaatimukset

Release test engineerin roolissa työskentelemisen vaatimuksiksi voitaneen listata ainakin yhden käyttöjärjestelmän laaja-alainen osaaminen sekä ainakin yhdestä toisesta riittävä osaaminen. Installer framework sijoittuu toiminnallisuuksiensa vuoksi tietynlaiseen solmukohtaan erilaisten ohjelmistojen välille, joten työyhteisötaitojen tulee olla hyvällä tasolla testaajan joutuessa lukuisia kertoja viikossa hakemaan tietoa eri kehitystiimeistä. Englannin kielen taito on vaatimus yrityksessä toimimiseen. Kaikissa IT-alan yrityksissä ovat omat toimintatavat ja järjestelmät, joiden käyttöön ja ylläpitoon uusi työntekijä perehdytetään työpaikalla.

Automaatiotestit kehitetään Squish-kehitysympäristössä, jossa on hyvällä tasolla oleva integraatio Qt:n eri ohjelmistojen kanssa. Squish on alunperin Froglogic GmbH:n kehittämä testikehitystyökalu. Froglogic integroitui osaksi Qt Companya vuonna 2021. Pääohjelmointikieli on Python. Lisäksi projektissa on hyvä osata käsitellä XML-, JavaScript- sekä Qml-kielillä luotuja resurssitiedostoja.

Automaatiotestit suoritetaan automaatiopalvelimella käynnistettävillä profiileilla, jotka taas käynnistävät virtualisointialustalta lukuisilla erilaisilla prosessoriarkkitehtuureilla ajettavia erilaisia käyttöjärjestelmiä ja niiden eri versioita. Qt toimii alustariippumattomasti kaikilla suosituimmilla käyttöjärjestelmillä. Qt toimii myös erilaisissa sulautetuissa järjestelmissä, joten ainakin korkean tason ymmärrys ohjelmistotestauksesta sulautetussa ympäristössä vaaditaan.

Kahdessa eri tiimissä työskentely asettaa omat haasteensa työyhteisötaidoille. Moni asia tulee välittää tiimistä toiseen ja työntekijän tulee olla hyvin perillä molempien tiimien suunnitelmista ja aikatauluista. Hyvä ja selkeä ulosanti puhutussa ja kirjoitetussa viestinnässä varmistaa pitävien aikataulujen ja suunnitelmien luomisen. Tiimissä on palavereissa käytössä kielenä englanti, joten puhutun ja kirjoitetun englannin kielen taidot olivat myös yrityksessä työskentelyn vaatimus.

1.4 Tietoperusta

Oulun ammattikorkeakoulun (Oamk) opetussuunnitelman mukaisen koulutuksen sisällöissä on sivuttu, mutta ei tarkalleen ottaen juuri käyty tarkemmin läpi ohjelmistotestauksen periaatteita. Ohjelmistotestaaja on erillinen nimike verrattuna ohjelmistokehittäjään (engl. Software test engineer vs. Software engineer) ja ohjelmistotestaajaksi valmistavaa koulutusta ei varsinaisesti ole tarjolla oppilaitoksissa. Ohjelmistotestauksen tarkoitus on kohottaa kehitettävän tuotteen laatua, laskea kehityskuluja vähentämällä julkaisuihin eksyviä ohjelmointivirheitä, parantaa tietoturvaa sekä parantaa asiakastyytyväisyyttä (Hamilton 2022).

1.5 Tavoitteet

Opinnäytetyön tavoitteena on yhtäältä tarkkailla työskentelyäni kahden kuukauden ajan erilaisista näkökulmista ja toisaalta avata ohjelmistotestaajan käytännön arkea yhden ammattilaisen henkilökohtaisesta näkökulmasta katsottuna. Erilaisia näkökulmia käyttäen toivon löytäväni uusia tapoja tutkia ja kehittää omaa työntekoani. Aikaisemmin itse itselleni asetettuja erilaisia kehittymishaasteita on jo valmiiksi suunnitelmissani useampia. Haasteet liittyvät rooliini liittyvien eri järjestelmien käyttämiseen ja käytön sujuvuuden opetteluun sekä laadukkaiden muistiinpanojen laatimiseen näihin. Toivon ajanjakson ja pohdintojen tuovan selkeyttä näihin suunnitelmiin sen, mihin osa-alueisiin paneutumiseen käytän aikaani jatkossa.

1.6 Keskeiset käsitteet

Anti-pattern: suunnittelumalli, joka johtaa huonoon lopputulokseen (Wikipedia, 2022a).
Black-box–testaus: testausmalli, jossa testaaja ei pääse näkemään järjestelmän sisäistä toimintaa. (Wikipedia, 2022b)
Commit: lähdekoodin lisäys vertaisarviointijärjestelmään, josta hyväksytty lähdekoodi siirtyy pakettivarastoon. (Wikipedia, 2022d)
Corner case: testitilanne, jossa testataan äärimmäisiä vaihtoehtoja jostain asiasta, esimerkiksi asennusohjelmassa tilanteita, joissa asennetaan joko minimi- tai maksimimäärä komponentteja. (Wikipedia, 2022e)
Debugger: ohjelmointityökalu, jolla voidaan seurata ohjelmiston sisäistä tilaa. (Wikipedia, 2022f)

Testausautomaatio: ohjelmistoa testaavien testien kokonaisuus, joka ajetaan eri käyttöjärjestelmillä sekä niiden eri versioilla.

False positive: testitulos, joka näyttää, että testattava asia on tullut toivottuun lopputulokseen mutta väärällä tavalla. (Wikipedia, 2022g)

Installertiimi: Kolmen hengen tiimi, joka kehittää Qt:n tuotteita asentavaa ohjelmistoa.

Kovakoodattu: staattisesti luotu koodinpätkä, joka esimerkiksi etsii tiettyä tekstiä. Vastakohta voisi olla dynaamisesti koodattu koodinpätkä, joka etsii tekstiä, joka luodaan ajettavan koodin muuttujista. (Wikipedia, 2022)

Ohjelmiston julkaisu: ohjelmiston toimittaminen asiakkaalle tai yleisölle saataville.

Ongelmanseurantajärjestelmä: tietokonejärjestelmä, jossa listataan ja ylläpidetään erilaisia toteuttamista tai korjausta vaativia kehityksen kohteena olevan ohjelmiston osia tai osa-alueita. (Wikipedia, 2022)

Product owner: tuotteen arvon ja kehitystiimin työn arvon maksimoinnista vastaava projektin jäsen. (Wikipedia, 2022)

Refaktorointi: lähdekoodin sisäisen rakenteen muutos siten, että toiminnallisuus pysyy samana. (Wikipedia, 2022c)

Releasetestaus: ohjelmiston testaus, joka ajoitetaan julkaisua edeltävälle ajalle. Testaus kattaa mahdollisimman laajasti kaikki ohjelmiston tärkeimmät toiminnallisuudet sekä uudet ominaisuudet, jotka kyseiseen ohjelmistoversioon ollaan tuomassa. Releasetestaus suoritetaan ennen ohjelmiston julkaisua ja sen onnistuneesti läpäisy on vaatimus ohjelmiston julkaisulle. (Wikipedia, 2022)

Releasetiimi: Seitsemän hengen tiimi, joka varmistaa Qt:n tuotteiden hallitun ja laadukkaan julkaisun.

Testisuite: yhden ominaisuuden testaamiseen liittyvistä testeistä koostuva kokonaisuus. (Wikipedia, 2022)

Versionhallinta: ohjelmisto johon ohjelmoijat toimittavat lähdekoodia tai lähdekoodin muutoksia. Versionhallinta tekee koodaamisesta turvallista, sillä esimerkiksi samaan tiedostoon tai samaan toiminnallisuuteen kohdistuvat yhtäaikaiset muutokset eivät pääse sotkemaan toinen toistaan. (Wikipedia, 2022)

Versionumero: ohjelmiston julkaisun yhteydessä nostettava numeroarvo, joka koostuu kolmesta kokonaisluvusta pisteillä erotettuna, esimerkiksi 1.2.3. Ensimmäinen numero kuvaa majorversiota, keskimmäinen minorversiota ja viimeinen numero patchversiota. Pieni muutos, kuten ohjelmointivirheen korjaus kasvattaa patch-versionumeroa. Uuden ominaisuuden lisääminen kasvattaa minor-versionumeroa. Ohjelmiston uuden toimintatavan tai ohjelmiston kokonaan uudelleen järjestely tai muu suuri muutos aiheuttaa major-versionumeron kasvattamisen. (Wikipedia, 2022)

White box –testaus: testausmalli, jossa testaaja pääsee näkemään järjestelmän sisäistä toimintaa. (Wikipedia, 2022)

2 LÄHTÖTILANNE

2.1 Työtehtävä

Nykyinen työtehtäväni pitää sisällään Qt Installer –ohjelmiston automaatiotestauksen suunnittelun, toteutuksen ja raportoinnin Installeria kehittävässä tiimissä. Suunnitelmissa on myös ottaa selvää muiden automaatiotestaustiimiläisten vastuualueista, jotta sairastumisen tai muun estymisen vuoksi kaikilla tiimiläisillä olisi jonkinlainen ymmärrys toisten tiimiläisten vastuualueista.

2.2 Osaamistaso

Olen aikaisemmin tehnyt työkseni web-kehitystä sekä PHP:llä ja JavaScriptilla että Node.js:llä ja Vue.js:llä. Testauspuolella olen tehnyt tutkivaa manuaalitestausta ja automaatiotestausta Robot Frameworkilla Seleniumilla, sekä mobiililaitepuolella Nb-IoT modeemin RF- ja protokollatestausta. Vaikka ohjelmistoalan uraani on takana kohta kuusi vuotta, ohjelmistotestaajan urani ei ole varsinaisen pitkä vuosissa mitattuna. Ohjelmistotestaajan työtehtäviini on mahtunut jo useammalta alalta erilaisia rooleja ja teknologioita – sekä laitteita että piirisarjoja kehittävistä yrityksistä nyt puhtaasti ohjelmistoyritykseen.

Olen aikaisemmissa työpaikoissa sekä harrastuksen myötä kehittynyt Python skriptauskielen käytössä. JavaScript on tullut tutuksi web-kehityksen kautta ja sitä voi tarvita välillä Qt Installerin script enginen käytössä. Luontainen uteliaisuuteni erilaisia teknologioita ja teknisiä laitteita kohtaan auttaa suunnittelemaan testitapauksia, jotka voivat potentiaalisesti löytää virheitä laitteesta tai ohjelmistosta.

2.3 Kehittymishaasteet

Vuosia kehitettyyn projektiin mukaan tuleminen sisältää jo itsessään haasteen. Valmiiksi olemassa olevaan projektiin tutustumiseen menee aina oma aikansa, samalla tavalla kuin alusta saakka jonkin projektin aloittaminen vaatii pohjatöineen aikaa. Lopulta tavoitteenani on päästä ymmärrykseen Installerin toiminnoista sekä sille jo kehitetyistä automaatiotesteistä. En ole aiemmin työskennellyt

puhtaasti ohjelmistoa tuottavassa yrityksessä, joten ohjelmiston julkaisuprosessi on mielenkiintoinen tutkimusaihe.

3 PÄIVÄKOHTAINEN RAPORTOINTI

3.1 Viikko 1

Maanantai 12.9.2022

Seurannan ensimmäinen viikko alkoi tilanteessa, jossa olen perehtymässä ohjelmistojulkaisun testausautomaatioon Installertiimissä. Perehtyminen liittyy Installer framework -ohjemiston testaukseen. Installer frameworkin päälle rakentuu Qt:n kaikkien tuotteiden asennusohjelma. Myös asiakkaat voivat itse räätälöidä omiin tarpeisiinsa sopivan asennusohjelman Installer frameworkilla.

Installertiimissä työtä tehdään kahden viikon mittaisissa jaksoissa, sprinteissä, joille jaetaan tietty määrä työtehtäviä ongelmienseurantajärjestelmästä suoritettavaksi. Tällä sprintillä alkava Qt Design Studioon liittyvä työtehtävä odottaa toisen tiimiläisen toteutusta tuotteelle. Installer Framework -testit testaavat vain itse Installerin toimintaa ja testatessa Design studion asennusta ei tehtävänäni ole tuottaa testiä muulle kuin itse Installer Frameworkille. Toiminnan verifiointi siis vaatii toisen henkilön tekemän testin käyttämistä omassani.

Installerin product owner lisäsi tulevalle sprintille myös tutkittavaksi, onko meillä mahdollisuuksia testata MacOS binääreitä joissakin Mac ARM arkkitehtuurin prosessoreilla varustetuilla virtuaalikoneilla. Selvittelin continuous integration tiimistä mahdollisuuksia tuollaiseen, mutta tämänhetkinen ohjelmisto ei vielä sisältänyt kaikkia toivottuja ominaisuuksia. Toistaiseksi käytössä olivat ainoastaan fyysiset koneet, mutta virtualisointialustan käyttö tulisi mahdolliseksi vasta lähikuukausina.

Tiistai 13.9.2022

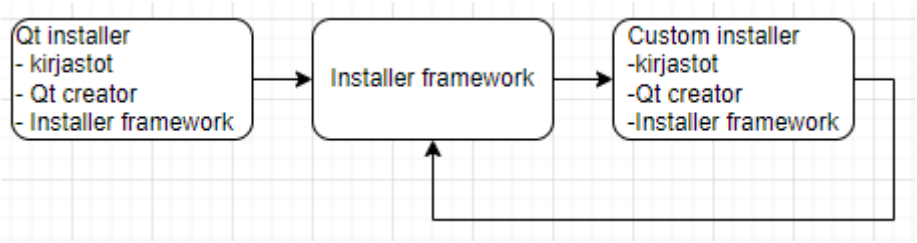
Installeritiimin aamupalaverissa keskusteltiin suuren, kaksi vuotta tekeillä olleen paketinhallintajärjestelmän integraatiokehityksen karsimisesta pois. Tämä päätös oli syntynyt käyttäjätilastojen perusteella sekä huolellisen arviointiprosessin tuloksena. Tiimin resurssit näiltä osin aletaan jatkossa kohdistaa toisen paketinhallintakanavaintegraation kehittämiseen. Kävimme myös keskustelua tähän asiaan liittyvästä inhimillisestä puolesta tekijän kanssa, kun tätä järjestelmää oli työstetty tosiaan jo kaksi vuotta. Hänen mielestään kanavan pois jättäminen oli oikea päätös ja ei olisi ollut hyötyä jatkaa muutaman innokkaan käyttäjän vuoksi suuren määrän aikaa vievää projektia.

Tällä hetkellä omana tehtävänäni oleva työ on ongelmallinen siitä syystä, että oma toteutukseni vaatii alleen toisen tiimiläisen tekemän toteutuksen. Testauksessa käytetään dependency injection tyylisesti luokkia, jotka alustetaan antamalla parametriksi toinen luokka. Tällä hetkellä toisen tiimiläisen toteutus on hieman kesken ja johtuen asian monimutkaisesta luonteesta voin joutua odottamaan tätä tovin. Omaa toteutustani en voi tällaisessa tilanteessa kuin hahmotella, sillä oman koodin testaaminen on käytännössä mahdotonta tästä syystä.

Keskiviikko 14.9.2022

Installer framework on Qt companyn tuote, jota käytetään luomaan asennusohjelma, jolla asennetaan kaikki Qt Companyn tuotteet. Asiakas lataa Qt Installerin, jolla voidaan asentaa muiden ohjelmien muassa Installer Framework. Tässä vaiheessa voidaan jo todeta, että kyseessä on jonkintasoinen rekursio.

Kuvan 1 mukaisen ohjelmistomallin ymmärtäminen kuulosti aluksi selkeältä, mutta on tässä vaiheessa jo paljastunut hieman odotettua monimutkaisemmaksi kuvioksi hahmottaa, varsinkin itse testejä kirjoittaessa. Käyttäjä siis lataa asennusohjelman, jolla voi asentaa ohjelmistokirjaston, jolla itse asennusohjelma oli alun perin luotu.



Kuva 1. Installerin rekursiivisuus

Testatessa jotain muuta Qt -tuotetta kuin Installer frameworkkia, esimerkiksi Qt Creatoria, suoritetaan testaaminen ensin lataamalla viimeisin julkaistu asennusohjelma, josta valitaan asennettavaksi Design studio. Tämän jälkeen voidaan ajaa testejä ja tarkistetaan, toimiiko Qt creator oikealla tavalla. Jos taas testataan IFW:lla Qt creatorin asentamista, ensin asennetaan viimeisin kehityksen alla oleva IFW:n versio, jolla sitten vasta asennetaan Qt creator ja testataan sen jälkeen, onko Qt creatorin asennus asentunut oikealla tavalla. Asentamisen onnistumisen näkökulmasta on oleellista testata itse ohjelman toimivuus, mutta kattavaa toiminnallisuuksien testaamista ei ole hyötyä tehdä, sillä sitä tekee toisessa tiimissä toinen henkilö.

Installertestejä kirjoittaessa joutuu tarkistamaan kyseisen toiminnallisuuden testaamisen nykyisen tilan testausdokumentaatioista, jotta vältettäisiin päällekkäistä testaamista. Asiakkaalla olevilla erilaisilla lisensointivaihtoehdoilla on saatavilla asennettavaksi eri sisällöt ja huomasimme päivän aikana, että tällä hetkellä minulla työn alla olevat testit on jo toteutettu käytännössä täysin samalla sisällöllä toisessa testisuitessa. Työ kirjattiin duplikaatiksi, mutta silti työ ei mennyt hukkaan. Huomasin, että testejä kehittäessä näkökulmaa pitää nostaa ylemmäs pystyäkseni hahmottamaan projektia paremmin kokonaisuutena.

Installertiimin tiekarttapalaverissa kävimme läpi Installerin seuraavaan versioon mukaan otettavia muutoksia, joista tietoturvapäivityksien mukaan saaminen oli yksi tärkeä asia. Tiekarttapalaveri pidetään product ownerin toimesta muutaman kerran vuodessa tai tarpeen mukaan. Tiimin testaaja on mukana tuomassa testauksen näkökulman mukaan suunnitelmiin.

Teimme päätöksen alkaa testata 4.4.2 -versiota etukäteen, jotta selviäisimme varsinaisesta releasetestauksen tekemisestä vähemmällä kiireellä. Releasetestauksen yhteydessä voi tulla esiin ohjelmointivirheitä, joita on tarkoitus korjata sitä mukaa, kun niitä ilmenee. Testaukselle määriteltiin mielestäni mukava määrä aikaa ja palaverista jäi mukava tunne sen puolesta, että kiirettä ei olla tekemässä keinotekoisesti.

Torstai 15.9.2022

Tänään ehdin ottaa hetkeksi työn alle uuden Installerversion testitapauksen, jossa tehdään Qt Creatorin asennus ja käynnistys erilaisilla lisenssivaihtoehdoilla. Asennus voidaan tehdä tiettynä ajanhetkenä, kun asiakkaalla on lisenssi voimassa. Creator voidaan jossain vaiheessa käynnistää sellaisessa tilanteessa, että asiakkaan lisenssi on umpeutunut tai sen tyyppi on muuttunut erilaiseksi.

Jatkuvan integraation järjestelmässä alettiin eilen ajaa testejä ja versionumeromuutoksen aiheuttamia epäonnistuneita testejä korjattiin triviaalisti vaihtamalla versionumero uudempaan. Tässä vaiheessa testaus on niin sanottuna kenraaliharjoitusta virallisen testaamisen aloittamiseen saakka.

Perjantai 16.9.2022

Päivällä päätimme alkaa viralliseen julkaisutestaukseen uutta Installerin versiota 4.4.2 varten töiden edettyä odotettua nopeammalla tahdilla. Alkuperäistä julkaisupäivää suunniteltiin aikaistettavan kymmenellä päivällä. Ohjelmistojulkaisun edellä testaus on koko tiimin suurin prioriteetti ja kaikki muu työ tähtää siihen, että ohjelmistojulkaisu tapahtuu ajallaan ja ohjelmistossa ei julkaisuhetkellä olisi ollenkaan ohjelmointivirheitä.

Testiautomaation ajon aikana ehdin paneutua hetkeksi uuden testikokonaisuuden kehittämiseen. Sain perjantaipäivän lopuksi hyvän flow-tilan päälle koodaillessani Qt Creatorin asennusta ja käynnistystä erilaisilla parametreilla. Pidän Don't Repeat Yourself ajattelutavasta koodauksessa ja keksinkin, miten kokonaisen viiden testin kokonaisuuden sai optimoitua ja ajettua nopeasti peräjälkeen. Työviikon sai lopettaa hyvällä mielellä saatuani muutosehdotukset versionhallintaan.

Viikkoanalyysi

Ohjelmistoprojektin aloituksessa mukana oleminen on harvinainen tilaisuus. Yleensä tyypillisesti projektiin tutustuminen tapahtuu siinä vaiheessa, kun projekti on ollut olemassa jo pidemmän aikaa. Projekti, joka sisältää kymmeniä tuhansia rivejä olemassa olevaa koodia, ottaa aikansa, että siihen ehtii perehtyä riittävästi. Projektiin tutustuessa huomaa monesti aiempien työntekijöiden tavan ratkaista erilaisia ongelmia toistuvan paikasta toiseen, mikä auttaa uusiin alueisiin tutustumista.

Suunnitellessa erilaisia toteutuksia on hyvä välillä pysähtyä ajattelemaan korkeammalta tasolta mitä sidonnaisuuksia toteutuksilla on, esimerkiksi tällä viikolla havaittu rekursiivisuus IFW:ssa. Viikon aikana korostui myös hyvän kommunikoinnin merkitys yrityksen sisällä sekä erilaiset roolit ja vastuut tiimissä.

3.2 Viikko 2

Maanantai 19.9.2022

Kovasta innosta huolimatta en ehtinyt jatkamaan perjantaina kesken jäänyttä koodausta, koska uuden testikokonaisuuden toteuttaminen jäi vähälle huomiolle suunnitellessamme varsinaisen ohjelmiston julkaisutestauksen aloittamista. Installertiimin releasemanageri päättikin, että ohjelmiston julkaisuaikataulu pidetään alkuperäisessä päivämäärässä. Varsinaista kiirettä tai tiukkaa kalenterointia julkaisupäivälle ei ollut, mutta hän halusi vain varmistaa, että julkaisu ei tule liian pian. Arvostin tätä päätöstä paljon sen vuoksi, että sen syynä oli ainoastaan varmistaa uuden tiimiläisen selviäminen tehtävästä ja kiitin tästä huomaavaisuudesta palaverissamme. Tällaiset asiat eivät ole itsestään selviä kaikissa paikoissa ja ovat varmasti omiaan sitouttamaan työntekijää työpaikkaansa, kun hyvä työyhteisö ei jää pelkän myyntipuheen tasolle.

Päivän aikana selvittelin automaatiotestien raportteja läpi menneistä ja epäonnistuneista testeistä. Osa epäonnistumisista testeissä johtui testissä itsessään olevista tarkistuksista, esimerkiksi tiedostopolun muuttumisesta versionumeron muuttuessa. Yhdessä tapauksessa paljastui, että eräs tutkimustasolla ollut serveri oli saavuttanut maturiteettinsa ja päivitetty tuotantoserveriksi. Osa datarakenteista tuli uusia serverin puolella, että testejä ajavilla virtuaalikoneilla oli pääsy noutamaan tarvittava data. Päivällä keskustelimme myös siitä, että olisi hyvä välillä käydä läpi myös läpi menneitä testejä ja niiden tarkkoja lokeja. On välillä hyvä kyseenalaistaa testien toiminta, sillä periaatteessa mikä tahansa testi voi mennä hyväksytysti läpi, vaikka ohjelma alkaisikin toimia väärin.

Tiistai 20.9.2022

Tämä päivä kului kaivaessa epäonnistuneiden testien raportteja raportointityökalun avulla eri testeistä. Prosessi on haastava siinä mielessä, että epäonnistuneet testit voivat olla ajettu eri käyttöjärjestelmällä, kuin mitä omalla henkilökohtaisella työkoneella on asennettuna. Eri käyttöjärjestelmissä voi olla omia käyttöjärjestelmäspesifisiä ongelmia, joita ei voi saada toistumaan toisessa käyttöjärjestelmässä. Jokin testi voi toimia omalla fyysisellä koneella ja sitten taas virtuaalikoneella ei. Tällaisia merkillisiä tilanteita varten joutuu joskus konsultoimaan servereistä vastaavaa tiimiä.

Tänään tuntui, että työtehtävät eivät edisty ja punaisesta langasta ei saa kiinni. Tällaisen tilanteen tunnistaminen on tärkeää ja vaatii henkilöltä alaistaitoja. Töiden paikalleen jumittuminen ei ole työnantajan puolelta toivottu tilanne ja suurin osa työnantajista kannustaa hakemaan teknistä apua yrityksen sisältä tai koulutuksia ulkopuolelta. Työntekijän näkökulmasta töiden etenemisen estyminen voi aiheuttaa ahdistusta ja stressiä (Hannonen & Koivisto 2022). Suomalaisessa työkulttuurissa avun hakeminen voi tuntua vaikealta ja onkin tärkeää työnantajan puolelta viestiä yrityksen sisäisestä kulttuurista, varsinkin jos se on avun hakemiseen kannustava.

Keskiviikko 21.9.2022

Releasetiimin viikkopalaverissa käydään läpi lyhyen ja keskipitkän tähtäimen tavoitteita sekä niiden toteutumista. Tässä palaverissa on mukana myös ylempää esimiestaholta tulleet toiveet tiimin osaamisen kehittämiselle tai tiettyihin osa-alueisiin keskittymiselle. Tiimimme on erittäin suurelta osin itseohjautuva johtuen varmasti suurelta osin siitä, että sen jäsenet kasvattavat testauskattavuutta oman ydintiiminsä ulkopuolisissa tiimeissä. Tiedot testaustarpeista tulevat muiden tiimien jäsenten kautta tiimille ja koko ajan riittää töitä miltei automaattisesti, koska testaustarve seuraa kehitettyjä ominaisuuksia tarkasti.

Kävimme läpi releasetestituloksia ja korjasimme testiympäristössä olleita ongelmia. Ongelmien läpikäynnissä ei löytynyt mitään vakavia löydöksiä. Testikattavuus saatiin erinomaiselle tasolle ja seuraava askel ennen testitulosten jakamista releasemanagerille on tarkistaa pari prosenttiyksikköä testien tuloksista ja varmistaa, että ne ovat testien omia ohjelmointivirheitä testattavan ohjelman ohjelmointivirheiden sijasta.

Torstai 22.9.2022

Jatkoin releasetestitulosten tutkimista ja testiympäristön erilaisten muuttujien korjausta vastaamaan uuden version ajamista. Tällä viikolla itse koodaaminen on ollut miltei olematonta. Olen saanut tehtyä vain oman toteutukseni virheiden etsintää ja se aika, mitä korjaamiselle ja vianetsinnälle on ollut aikaa, on käytännössä tuottanut nolla riviä koodia. Aikaisemmissa paikoissa olen ilmeisesti projektipäällikön johdolla sekä alitajuisesti kehittänyt töissä suoriutumisen itsearviointia varten psykologisen mittarin, joka kasvattaa lukemiaan sitä mukaan, kun git commit -komentoja saa syöttää

terminaaliin. Commitien määrä ei ole nykyisessä työroolissa niin merkittävä varsinkaan julkaisua edeltävällä ajanjaksolla, jolloin kaikista tärkeintä on keskittyä laadukkaan ohjelmistojulkaisun tukemiseen automaatiotestien tulosten valmiiksi saamiseksi.

Testien ajamisen lomassa ehdin tänään paneutua silti uuden testikokonaisuuden ensimmäisiin testeihin. Testeissä ongelma on ollut se, että vaikka ne toimivat tässä tilanteessa oikein ja tein oman toteutukseni ohjelman asennusta varten, olisi järkevää käyttää jo valmista kirjastosta löytyvää wrapper-funktiota, joka hoitaa ohjelman asennuksen ja siihen liittyvän testauksen valmiiksi. Erillinen yleisesti käytetty funktio on hyvä idea siinä tapauksessa, jos testattavaan ohjelmaan tulee muutoksia, on korjaukset tehtävä ainoastaan yhteen paikkaan. Omassa aiemmassa ratkaisussani korjaukset olisi pitänyt lopulta tehdä joka paikkaan, jossa olisin käyttänyt omaa ratkaisuani.

Uutena työntekijänä projektiin tutustuessa ongelmana on monesti se, että valmista toimivaa koodia löytyy runsaasti ja sen käytön opettelu ei ole yhtä helppoa kuin tyhjästä Pythonin skriptien rakentaminen samalla lukien hyvien ohjelmointikäytäntöjen opasta ja Pythonin manuaalia. Tänään silti onnistuin löytämään omasta toteutuksestani bugin ja sain kolme viidestä testitilanteesta, jotka kehitin viime perjantaina, refaktoroitua toimimaan jo aiemmin toisen tiimiläisen kehittämällä funktiowrapperilla, joka käyttää dependency injectionilla tuoteluokkia.

Testissä on tarkoitus testata ohjelman käynnistymistä erilaisilla lisenssiskenaarioilla. Aluksi poistetaan asennus. Jos asennus on jo tehty, asennetaan ohjelma uudelleen tietyillä parametreilla ja suoritetaan yksi testi. Ajattelin tekeväni kaikki viisi tapausta yhteen testiin, sillä ajatuksella, että optimoin testin suorittamisen siten, että ohjelma tarvitsee asentaa vain kerran, jonka jälkeen vain muokataan Pythonilla lisenssitiedostot aina vastaamaan kutakin viidestä tilanteesta. Tämä säästäisi aikaa arviolta noin 80 % verrattuna siihen, että joka testin alussa suoritetaan uninstall ja install. Tämän mallin huono puoli olisi testien raportoinnin kannalta siinä, että yhden testitapauksen painoarvo testisuitessa vääristyisi testiraportin tasolla. Yhden testicasen ajo, joka sisältää viisi erilaista testiä, epäonnistuessaan aiheuttaisi suhteettoman pienen loven raportissa, joten päätin vielä kerran refaktoroida oman tuotokseni viideksi eri testicaseksi, joissa silti suoritettaisiin asennus vain kerran aluksi ja sen jälkeen luodaan erilaiset lisensointitilanteet muokkaamalla tiedostojärjestelmän tiedostot ja niiden sisällöt vastaamaan testitapausta.

Perjantai 23.9.2022

Koko päivä meni releasetestitulosten viimeisissä tarkastuksissa. Mitään varsinaisesti eilisestä poikkeavaa ei tapahtunut sprinttiarviointipalaveria lukuun ottamatta. Olin saanut valmiiksi yhden tehtävistä, jossa testataan uutta lisenssityyppiä tehtyä ja toisesta tehtävästä noin 66 %. Loput tehtävästä siirrettiin seuraavalle sprintille.

Viikkoanalyysi

Oman tekemisen priorisoiminen on tärkeää, varsinkin itseohjautuvuuden näkökulmasta. Ohjelmistokehittäjän työ on useimmiten parin viikon mittaisten sprinttien loppuessa arvioitua työtä, joten niin sanottu palaute tehdystä tai tekemättömästä työstä tulee vasta sprintin lopussa. Palautteen saaminen auttaa omien prioriteettien asettamista, mutta parin viikon sprinteissä palautetta voi joutua odottamaan, jos sitä ei aktiivisesti itse hae kesken sprintin.

Välillä uutta toiminnallisuutta voi löytää itsensä niin sanotusta writer's blockista. Mielikuvituksen puute ei ole tässä tilanteessa kyseessä, vaan koodaillessaan jotain toteutusta usein joutuu kesken kaiken alkaa tiedustella lisätietoja muilta tiimiläisiltä asiasta ja joissain tapauksissa löytääkin itsensä kyselemästä toteutuksesta toisen tiimin henkilöiltä. Tällaisen katkonaisen koodin kirjoittamisen sivutuote voi olla ajatuksen katkeaminen kesken kaiken. Kuten viime viikolla huomasin, olisi hyvä yrittää ottaa näkemys tekemisestään korkeammasta perspektiivistä ja koettaa hahmottaa kokonaisuudesta asiat, joka vaikuttavat omaan sen hetkiseen työskentelyyn.

Jokaisella tiimillä on prioriteettinsa ja releasetestauksessa se on ohjelmistojulkaisun tukeminen. Ohjelmistojulkaisun alla tekeminen keskittyi suurimmaksi osaksi testien ajamiseen ja korjaamiseen, mutta välillä kun tekemistä riitti, piti muistuttaa itseään testitulosten tarkastamisesta, vaikka sillä hetkellä olisi ollut mukavaa koodattavaa tarjolla.

3.3 Viikko 3

Maanantai 26.9.2022

Seurannan kolmas viikko alkaa julkaisun arvioinnilla. Viimeiset testiarviot olivat valmistuneet perjantaina työpäivän jo loputtua, joten tulokset olivat saatavilla vasta maanantaina töiden alettua.

Viikonlopun aikana voi joskus joutua tarkastamaan, että testiajot ovat varmasti päässeet loppuun saakka mutta tällaisia ongelmia on harvemmin minulla ollut.

Testaajan tehtäviin kuuluu rakentaa erilaisiin käyttötapauksiin liittyviä niin sanottuja polkuja, joita käyttäjä tulee kulkemaan ohjelmaa käyttäessään. Nämä peruskäyttötapaukset on hyvä olla testattuna, jotta voidaan varmistua, ettei mikään muutos ole rikkonut mitään komponentteja, joita polkua kulkiessa tarvitaan.

Exploratory testing taas terminä poikkeaa edellä kuvatusta testausmallista. Exploratory testingissä käyttäjä pyrkii toimimaan vapaasti ja useasti erikoisella tavalla, esimerkkinä vaikka “avaa asetukset-välilehti”, “muuta asetuksia”, “älä tallenna muutoksia”, “avaa Viestit-välilehti”. Tekemällä asioita kummallisessa järjestyksessä voi tuottaa hyviä löydöksiä järjestelmän toimiessa ennalta ajattelemattomalla tavalla. Tällaisen testin luominen on ohjelmallisesti silti vaikeaa, varsinkin testatessa käyttöliittymiä. Eräs tapa testata käyttöliittymän luotettavuutta on käyttää kaikki ylimääräinen aika testiservereillä siihen, että käyttöliittymää aletaan klikkailla pari kertaa sekunnissa satunnaisissa x- ja y-koordinaateissa. Jos satunnaistetussa klikkailussa saadaan ajaksi jonkinlainen virhe tai varoitus, se otetaan talteen lokeihin ja jatketaan klikkailua.

Pohdin tänään mahdollisuuksia toteuttaa jonkinlaista resurssien optimointia Installerissa, optimointia siinä mielessä, jos virtuaalikoneresursseja on käyttämättä. Yleensä ongelma on päinvastainen, eli kaikki testitapaukset tulisi tehdä mahdollisimman tehokkaasti aikaa ja resursseja käyttäviksi, että testitapaukset saataisiin mahdollisimman nopeasti valmiiksi. Yhden testin ajaminen nopeasti ei tuota paljoa säästettyä aikaa, mutta jos sama testi on ajettava yli kymmenellä erilaisella käyttöjärjestelmäkokoonpanolla, pienikin säästö nopeasti kymmenkertaistuu

Tiistai 27.9.2022

Sain tälle sprintille tehtäväkseni alkaa kehittää toteutusta uudelle testitapaukselle, jossa asennetaan ohjelma, jonka jälkeen ajetaan päivitys ja testataan päivityksen toimivuus. Testissä varsinkin asennuksen osuus kestää suhteellisen kauan, joten erilaisia optimointeja olisi testissä tehtävä. Asennuksen toistamisen välttäminen olisi avainasemassa.

Installertiimin kanssa käytiin keskustelua siitä, että ohjelmoija ei ole paras henkilö luomaan kuvausta uudelle testitapaukselle. Jos testitapauksen kuvauksen luo ohjelmoija, testi varmasti menee

läpi koska sen suunnittelee henkilö, joka on suunnitellut toiminnallisuuden. Testi tulisi laatia uusin silmin, jolloin tulee tehneeksi havaintoja omasta näkökulmasta ongelmaan, sekä myös ohjelmisto-testausinsinöörin osaamisen avuin yrittäen löytää mahdollisia ongelmakohtia ohjelmasta itsestään.

Keskiviikko 28.9.2022

Uusien testitapausten kirjoittaminen ei yksistään riitä, vaan koska toimitaan testausautomaation kanssa, on testit myös saatettava automaatioserverille saakka. Testien ajaminen automaattisesti vaatii automaatioserverillä tehtävät määrittelyt uudelle ajettavalle testisuitelle, käyttöjärjestelmäympäristölle missä suite halutaan ajaa sekä halutut ohjelmistoversiot, joita testissä aiotaan käyttää.

Testattavasta ohjelmasta on monessa tapauksessa olemassa uusimman tuotantoversion lisäksi niin sanottu tai niin sanottuja pitkäaikaisen tuen versioita. Pitkäaikaisen tuen Long term support tai "LTS" ohjelmistoja käytetään sellaisissa käyttötapauksissa, jossa halutaan lukita kehitystyö tiettyyn versioon, esimerkiksi kehitettäessä sulautettuja järjestelmiä. Pitkäaikaisen tuen ohjelmistoihin lisätään uusia ominaisuuksia erittäin korkealla kynnyksellä ja pääasiassa kaikki päivitykset ovatkin tietoturvaan liittyviä tai bugikorjauksia.

Torstai 29.9.2022

Työn alla oleva testitapaus vaati perusteellista tutkimustyötä ja Installertiimin jäsenen haastattelua. Lisäksi huomasin, että en vielä ymmärrä tarpeeksi laajemmasta releaseprosessista, varsinkaan niin sanotusta housekeepingista eli tiedostopalvelimien tietorakenteiden ylläpidosta ja siivoamisesta julkaisua edeltävästi ja sen jälkeen.

Toteutus testitapaukselle olisi ollut helppoa tehdä niin sanotusti kovakoodattuna testitapauksena, mutta arvaukseni testitapauksen kertakäyttöisyydestä paljastui todeksi haastateltuani myös releasetestaustiimin jäseniä asiasta. Toteutus olisi parasta parametrisoida siten, että tällä kertaa käytettäisiin tiettyjä parametreja testaamiseen ja jatkossa tarpeen mukaan uusilla parametreilla voitaisiin ajaa sama asennus- ja päivitystesti.

Perjantai 30.9.2022

Sain viimeinkin toteutuksesta selvän kuvan ja päivä meni koodaillessa testitapausta, jota ajettaisiin kulloinkin uutta kokonaisuutta testatessa silloisten päivitettyjen ohjelmistopakettien mukaan parametrisoidusti. Testiä voitaisiin käyttää jatkossa uusien pakettikokonaisuuksien mukaan ottamiseen asennusohjelmalle testissä. Mielenkiintoista toteutuksesta teki se, että halusin voida jatkossa toimittaa string -muotoisen listan ohjelmistopaketeista, jotka viittaavat funktiossa oleviin saman nimisiin muuttujiin. Muuttujiin itseensä viittaaminen tiedostosta toiseen Pythonissa ei onnistu globaalien muuttujien käytön ollessa huono käytäntö. Tutustuin vars() function avulla string-tyyppisten muuttujien avulla samassa nimiavaruudessa sijaitsevien muuttujien asettamiseen. Tämä paljastui mainioksi ja käyttäjälleen ergonomiseksi tavaksi kutsua ohjelmistopaketteja. Lisäksi tämä tapa mahdollistaa helpon ja siistin laajennettavuuden sitä mukaa kun uutta testaustarvetta ilmenee.

Viikkoanalyysi

Kuluneella viikolla pohdin tuottavuuden kausivaihteluita eri ilmiöissä nykyisellä ja edellisillä työpaikoillani it-alalla. Kiireen määrä, tai kiireellisten korjausten määrää voi yrittää ennustaa ja niihin valmistautua nykyisessä työtehtävässä julkaisupäivämäärän lähestyessä. Erilaisen dokumentaation tarpeen määrää voi olla vaikea ennakoida, dokumentaatioiden tekeminen on yleensä tarpeen tullen tehtävää työtä. Itse koodaamisen määrä taas on merkillinen ilmiö.

Kuten jo aiemmin mainittu, pyrin olemaan mieltämättä koodirivien määrää tuottavuuden mittarina mutta vaikeaa se silti on. Koodaaminen ja pakettivarastoon saatujen koodirivien määrä vaihtelee valtavasti, mutta olen havainnut tiettyjä säännönmukaisuuksia omassa työssäni. Tuottavuus menee suunnilleen kaavalla 5-10-85, eli heti työtehtävän alussa saa jotain pohjustuksia tehtyä, jonka jälkeen ajallisesti seuraavan kymmenen prosentin tekeminen kestää valtavan kauan, jopa useita päiviä. Tämä kymmenen prosentin vaihe on joskus erittäin turhauttavaa ja liittyy toteutuksen tutkimiseen sekä lisätietojen hakemiseen dokumentaatioista ja muilta tiimiläisiltä. Toteutuksen hakeminen on välillä todellista käärmeen piippuun ajamista, kun yksi toteutusyritys toisensa jälkeen joudutaan poistamaan tai uudelleenjärjestelemään erilaiseksi. Kun kaikki tieto on löytynyt, jossain vaiheessa palaset loksahtavat paikoilleen ja rivejä alkaa tulla kuin tulvaportit olisivat yhtäkkiä avautuneet. Jostain syystä tämä miltei flow-tila, joka kestää yleensä tunnin pari, tapahtuu pitkällä iltapäivästä ja yleensä perjantaina kello 15:n aikoihin. Paras arvaus tähän on, että viikon aikana hiljalleen kehittynyt ymmärrys on rakentunut siihen pisteeseen, että kokonaisuuden hahmottaa tarpeeksi

kirkkaasti, jolloin myös toteutuksen koodaaminen käy luonnollisesti. Se, miksi maanantaina aamulla koodaaminen ei jatku yhtä liukkaana kuin perjantaina työt lopettaessa säilyy yhä mysteerinä minulle.

3.4 Viikko 4

Maanantai 3.10.2022

Toteutus, joka tällä hetkellä on työn alla, tekee ensin asennuksen, jonka jälkeen asennukselle ajetaan päivitys. Näitä testejä on ajettava monella eri yhdistelmällä erilaisia asennettavia paketteja. Tarkoitus olisi käyttää Gitiä tallentamaan asennuksen jälkeinen tila, jonka jälkeen testataan päivitys ja tämän jälkeen Git repo resetoidaan, jotta voidaan aloittaa nopeasti asennuksen jälkeisestä tilasta. Git -toiminnallisuudelle oli tehty skriptipohjainen toteutus, joka toimi hienosti mutta jätti toiminnallisuuden piilottelemaan skriptitiedostojen syövereihin. Huomasin, että päivitykset toiminnallisuudelle olivat jokseenkin epäergonomisia, joten päätin kirjoittaa nopeasti luokan Git-toiminnallisuudelle. Ensimmäinen 90 % tehdystä työstä kesti noin neljä tuntia, jonka jälkeen testailujen ja viilailujen jälkeen totesin, että tähän vierähtikin koko päivä.

Tiistai 4.10.2022

Git luokka sai lisää pari helper-funktiota ja aloin kirjoittaa tätä luokkaa vasten testin toteutusta. Lisäksi pohdin päivän aikana paljon dokumentaatiota ja sitä minkälaisen perinnön jätän jälkeeni firman repoon. Tutustuin Pythonin type hintingiin eli funktioissa parametrien tyypin ilmaisuun. Type hintingistä esimerkkinä voisi toimia funktio string_to_list(input:str) jossa kaksoispisteen jälkeen ilmaistaan muuttujan tyypiksi string. Type hinting ei pakota käyttämään stringiä, eli funktio syö sisälleen tyytyväisenä kokonaislukuja, mutta linter viimeistään huomauttaa virheestä. Type hinting ei vaadi kirjoittajalta juurikaan ajatusta tai aikaa, mutta auttaa hahmottamaan funktiota ensimmäistä kertaa lukevalle selkeästi. Kaikki aika, joka saadaan leikattua pois turhasta ihmettelystä ja koodissa harhailusta on lopulta firman taseessa positiivisena asiana.

Keskiviikko 5.10.2022

Toimistolla tänään jatkoin monimutkaiseksi paljastuneen toteutuksen tekemistä. Pohdin toteutusta aikasemmasta näkökulmasta, Selenium frameworkilla web-testausta tehneenä. Seleniumia käytetään web-sivujen automaattitestaukseen ja sille on mahdollista “nauhoittaa” toimintoja, eli laitetaan nauhoitus päälle ja klikkaillaan tietyssä järjestyksessä elementtejä. Yleensä lopputulos on miltei jopa toimiva kasa koodirivejä, mutta useimmiten klikkausten välille pitää lisätä verifiointi, että nappia painamalla odotettu asia tapahtui.
Squish tukee samaa toiminnallisuutta eri sovelluksille, mutta tuki ei ole aina ollut olemassa. Aiempi toteutus käyttää skriptitiedostoja, jotka on kirjoitettu JavaScriptillä, jossa listataan elementtien klikkaukset ja valintanappien valinnat niiden ilmestyessä näkyviin. Toisin sanoen koko ohjelmiston asennus tai päivitys sekä komponenttien lisäys tai poisto voidaan automatisoida skriptin avulla. Jäin päivän päätteeksi miettimään tulisiko minun alkaa tehdä uutta toiminnallisuutta nauhoittamalla tehtävälle asennukselle, vaiko opetella tekemään vanhemmalla, mutta hyväksi havaitulla sekä todennäköisesti testiajon keston kannalta nopeammalla tavalla.

Torstai 6.10.2022

Toimistopäivä, jonka aikana oli useampi palaveri, yksi koulutus ja yksi vierailija käymässä toimistolla. Espoon pääkonttorilta on välillä Oulussa käymässä firman henkilöitä, joita on mukava nähdä kasvotusten. Tänään oli myös erään tuotteen ohjelmistojulkaisun kunniaksi kahvihetki koko toimiston henkilökunnalle.

Tällaisten yhteisten hetkien merkitys nousi puheenaiheeksi vierailijankin esityksessä. Etätyön yleistyminen on muuttanut it-alalla työskentelyn luonnetta erittäin paljon lyhyessä ajassa. Yksilön tuottavuus etätöissä on selkeästi parempi yksin kotona töitä tehdessä mutta siiloutuminen, eli työntekijöiden jakautuminen tietoa harvemmin toistensa kanssa jakaviin ryhmiin, sekä tiedon kulun heikkeneminen on selkeä ongelma etätöissä. Toimistolla on helppo käydä kysymässä kollegalta jotain asiaa kasvotusten, etätöissä taas esimerkiksi teknisten asioiden esittely käy helposti – vaikkapa jakamalla koodinpätkiä tai ruutukaappauksia. Silti yhteiset hetket toimistolla ovat tarpeellisia yhteishenkeä kehittäviä tapahtumia.

Perjantai 7.10.2022

Uuden toteutuksen tekemisen yhteydessä tutustuin taas lisää lähdekoodiin ja huomasin tehneeni taas kerran toteutusta eri tavalla kuin sen kuuluisi toimia. Olen pari kertaa uudessa projektissa ajautunut tilanteeseen, jossa luettuani testin ohjeistukset eli spesifikaatiot, olen saanut käsityksen vaaditusta työtehtävästä ja olen alkanut toimeen. Toteutusta tehdessä olen saanut lisää ymmärrystä itse vaatimuksista, ja testin spesifikaatiota uudelleen lukemalla olen ymmärtänyt mitä toteutukselta oikeasti haluttiin. Ensimmäinen käsitys on siis ollut väärä ja osa työstä on mennyt hukkaan tai vaatii refaktorointia.

Viikkoanalyysi

Erilaiset tyylit kirjoittaa lähdekoodia on rikkaus ja siihen tulisikin suhtautua rikkautena. Ohjelmointikielissä on olemassa niin sanottuja parhaita toimintatapoja koodata, sekä antipatterneita joilla tarkoitetaan toimintatapaa, jota tulisi välttää viimeiseen saakka. Yhdelle tehtävälle voi siis olla monta erilaista tapaa toteuttaa koodaaminen, monta oikeaa ja monta väärää tapaa.

Tätä kaikkea yrittää pitää hyppysissään ohjelmistoarkkitehti, joka luo toimintatavat ja muut määrittelyt projektille. Olemme keskustelleet arkkitehdin tarpeesta projektissamme, joka on ajan myötä paisunut kohtuullisen kokoiseksi. Ohjelmistoarkkitehti kuulostaa tittelinä mahtipontiselta ja ehkä myyttiseltä hahmolta, joka on noussut joko fyysisen tai kuvainnollisen vuoren laelle ja sieltä valaistuneena palattuaan jakaa viisauttaan projektissa. Ohjelmistoarkkitehdin työtehtävä ei varsinaisesti ole näinkään tarunhohtoista työtä, vaan enemmänkin juuri projektin rakenteen suurien suuntaviittojen piirtämistä, sekä käytettävien teknologioiden tehokkainta mahdollista soveltamista tukevien toimintaohjeiden luomista ja niiden käytön seuraamista.

3.5 Viikko 5

Maanantai 10.10.2022

Päivän suurin ponnistus oli tiedon hankinta erittäin monimutkaisen prosessin toiminnasta. Päivä meni suurimmaksi osaksi haastatellessa testattavan toteutuksen tekijöitä. Testaajan rooli vaatii tietynlaisen karisman, jottei testattavana olevan toteutuksen tekijälle välity vääränlainen kuva tehtävästä työstä. Testaajan tehtävä loppujen lopuksi on paljastaa toisten ihmisten tekemät virheet.

Tiistai 11.10.2022

Aiemmalla viikolla käsitelty exploratory testing on sekä suunniteltua että ennalta suunnittelematonta. Toistaessa erilaisia tyypillisiä toimenpiteitä, esimerkiksi menemällä ohjelmassa sivulta toiselle edestakaisin muutamia kertoja voi olla hyvä testi. Jossain tapauksissa suuri määrä toistoja voi aiheuttaa virheen, 255. kerta jotain asiaa voi tuottaa hallitsemattoman virheen. Myös kummallisessa järjestyksessä asioiden tekeminen voi tuottaa hyviä löydöksiä. Löysin erään piilevän virheen tekemällä asioita erikoisessa järjestyksessä, täysin tahattomasti.

Keskiviikko 12.10.2022

Bugiraportointi on ohjelmoijalle tehtävä raportti löytyneestä ohjelmointi- tai semanttisesta virheestä. Semanttinen virhe ei ole varsinainen bugi, vaan se voi johtua esimerkiksi väärinymmärryksestä tai muuttuneista tehtävän määrittelyistä.

Hyvä bugiraportti sisältää mahdollisimman tarkan kuvauksen siitä, miten ohjelmiston virheellinen toiminta voidaan toistaa. Virheen toistaminen voi olla joissain tilanteissa vaikeaa tai jopa erittäin vaikeaa, jos ohjelmointivirhe voi johtua monesta järjestelmästä ja niiden toiminnasta. Bugiraportissa on lisäksi kaikki mahdollinen lokitieto mitä ohjelmasta on saatu ulos, mahdollisesti ruutukaappauksia tai jopa ruudulta kaapattu video. Parhaimmassa tilanteessa bugiraportissa on arvio bugin sijainnista tai jopa korjausehdotus, mutta tämä menee jo ohjelmistotestaajan osaamisvaatimusten ulkopuolelle ja on hyvä osata rajata oma työnsä testaamisen puolelle.

Torstai 13.10.2022

Paketointiprosessi tuli tutuksi release managerin pitämässä infopalaverissa. Ohjelmiston julkaisuun kuuluu sekä paketointiin liittyviä teknisiä seikkoja, että muiden tiimien kanssa kommunikoimista. Tietyn ohjelmiston julkaisua edeltävän ajanhetken jälkeen tehdään ohjelmistolle feature freeze, eli julkaisuun mukaan otettavaan sisältöön ei tehdä enää minkäänlaisia uusia ominaisuuksia. Tämän ajanhetken jälkeen tehtävät ominaisuudet otetaan mukaan seuraavaan ohjelmistoversioon, jonka julkaisuajankohta voi olla parin viikon tai jopa puolen vuoden päässä.

Joskus release manager joutuu tekemään ohjelmistokehittäjien kanssa syvällisempää analyysiä kehitteillä olevan ominaisuuden tilasta. Jos kyseessä on merkittävä uudistus, joka ehdottomasti

halutaan mukaan julkaisuun, voidaan julkaisupäivää tarvittaessa jopa lykätä. Jos kyseisen ominaisuuden toteutuksen valmistuminen on päivän tai parin päässä, tällaiseen voidaan ryhtyä.

Viikkoanalyysi

Testaajan työssä on tärkeä löytää ohjelmistoprojektin rakenteen ja toteutustapojen lisäksi sovelluskehittäjien kanssa yhteiset toimintatavat testauksen suorittamisen kanssa. Tehokkaat ja selkeät testitapausten määrittelyt sovelluskehittäjiltä testaajille vähentävät selkeästi lisäkysymyksien määrä tehtävää aloittaessa. Lisäksi testaajan tulisi oppia raportoimaan mahdollisista löydöksistä ohjelmointitiimille mahdollisimman tehokkaalla tavalla ja olla valmis tekemään laajempia tai jopa myös suppeampia bugiraportteja.

Kaikki edellä mainittu vaatii paljon aktiivista keskustelua projektiin tutustuessa ja erilaisten raporttien laatimisen yhteydessä. Jo aiemmin mainitut työyhteisötaidot korostuvat tässä tilanteessa. Jatkuva mahdollisen kevyeen mutta tehokkaaseen työskentelyyn tähtäävä kehitysprosessi takaa tehokkuuden tiimin toiminnassa.

3.6 Viikko 6

Maanantai 17.10.2022

Erilaisten toimintojen testaus vaihtelee yksinkertaisista “klikkaa nappia, tarkista vaihtuiko sivu” -testeistä erittäin kompleksisiin kokonaisuuksiin. Monimutkaisten testien tekemisessä varsinaisten tarkistusten suorittaminen voi olla vaikeaa tai joissain tilanteissa jopa mahdotonta black box -testauksessa. Black box -testauksessa testaaja ei tiedä mitä järjestelmän sisällä tapahtuu, sillä testaaminen tapahtuu kokonaisuudessaan loppukäyttäjän tasolla. White box -testauksessa taas testaajalla on täysi pääsy lähdekoodeihin sekä esimerkiksi ohjelman output -lokiin, josta voi seurata ohjelman toimintaa hyvin tarkalla tasolla.

Tämänhetkinen testitoteutus on vaikea siinä mielessä, että testattava kokonaisuus liittyy pieneen muutokseen suuressa kokonaisuudessa. Tilannetta auttaa huomattavasti Squishin tarjoama tuki seurata Qt Installerin toimintaa erittäin tarkalla tasolla, mutta se ei auta kuin tiettyyn pisteeseen saakka siinä tilanteessa, jos testaaja on vielä tutustumassa testattavaan järjestelmään.

Tiistai 18.10.2022

Tämänhetkisessä UI testauksessa käyttöliittymän kautta tapahtuva testaus, eli automaattisesti hiirtä tai näppäimistöä käyttämällä kohdistuu eri kohtaan ohjelmistopinoa kuin automaatioskriptillä asioiden tekeminen. Klikkaaminen hiirellä vaatii siis päälleen enemmän toiminnallisuutta kuin skriptin käyttö. Automaatioskriptissä käytetään JavaScriptiä, jolla tehdään valintoja ja painetaan nappeja, kun ne ovat näkyvissä Installerissa. Skriptillä voi onnistua painamaan nappeja, jotka ovat olemassa mutta piilotettuja. Käytännössä hiirellä tätä nappia ei siis voi painaa mutta skripti löytää ne elementtien seasta.

Keskiviikko 19.10.2022

Testauksen ohessa tietotyöläisen arkeen kuuluu todella laaja ohjelmistopino, mukaan lukien tietoliikenneyhteydet, siihen liittyvät ohjelmistot sekä itsessään tietoliikenneprotokollapino. Tietoliikenneyhteyksiin voi liittyä erittäin laaja kirjo erilaisia ongelmia, joiden selvittelyä varten olisi erittäin hyvä ymmärtää minkälaisiin eri tasoihin verkkoliikenne on jakaantunut. Tcpdumpista lähtien, pingin kautta Wiresharkkiin olisi hyvä osata hakea eri tasoisia ongelmia.

Verkkoliikenteen ongelmat voivat naamioida bugeja ja tämä tulisi huomioida myös testejä rakentaessa. False positive testit tulisi voida eristää oikeista virheistä, esimerkiksi testatessa tiedoston latauksen epäonnistumista. Tilanteessa, jossa tiedoston lataus yrityksestä huolimatta ei toimi, ei riitä tarkastus, että tiedostoa ei löydy kansiosta, vaan olisi tärkeää seurata ennemmin lataustapahtuman outputtia.

Torstai 20.10.2022

Testaustiimissämme on lukuisia erilaisia rooleja ja jokaiselle testaajalle on muodostunut oma osaamis- ja vastuualueensa. Nämä alueet pitävät sisällään toisistaan paljonkin poikkeavia toimintoja, sekä toteutustapoja. Designer studion lukuisat toiminnot on toteutettu erilaisilla käyttäjätarinoilla siinä missä asennusohjelmassa testataan tietyn komponentin binäärien oikeanlaista asentumista. Paketointipuolen testaaminen taas vaatii verkkolevyn ja verkkoyhteyksien testaamista.

Näitä rooleja kierrätetään säännöllisesti tiimiläiseltä toiselle siten että alueesta vastaava perehdyttää vuorollaan kaikki muut tiimiläiset yksi kerrallaan aihealueeseen pinnallisesti. Tämän toiminnon

tarkoitus on lisätä tietoisuutta projektin eri osien toiminnoista sekä luoda varmaa pohjaa tiimin toiminnalle poikkeusolosuhteissa, esimerkiksi lukuisten yhtäaikaisten sairastumisien aikana.

Perjantai 21.10.2022

Testejä rakentaessa ja uuteen toiminnallisuuteen perehtyessä tein tänään paljon virtuaalikoneiden kanssa töitä. Sain muodostettua pari parannusehdotusta prosessiin, jossa testaaja testituloksia selatessa kaivaa esille lukuisten virtuaalikoneiden seasta kyseisen koneen, jolla testit on varsinaisesti ajettu, mahdollisesti tehdäkseen lisätestejä tai lukeakseen tarkempia lokitietoja.

Aiempi kommenttini laajan ohjelmistopinon hallitsemisesta tuli taas ajankohtaiseksi, sillä virtuaalikoneiden verkkoyhteyksien toiminta voi olla täysin erilaista verrattuna fyysisiin koneisiin. Virtuaalikone voi olla täysin avoin ulkopuoliseen verkkoon sekä internettiin, täysin suljettu tai kaikkea siltä väliltä. Virtuaalikoneen verkkorajapinta on täysin virtualisointialustan hallussa, siinä missä fyysisen koneen verkkokortti harvemmin on laajasti hallittavissa. Insinöörinä tulisi hallita sekä fyysisten että virtualisoitujen käyttöjärjestelmien käyttö. Aikaisempi taustani kodinautomaation ja erilaisten tietokonepeliserverien ylläpidosta on osoittautunut useita kertoja hyödylliseksi.

Viikkoanalyysi

Erilaisia testaustapoja on lukematon määrä ja saman toiminnon voi testata riittävän kattavasti monella eri tavalla. Riittävän kattavan testaamisen määrittäminen on tärkeä taito, jonka kehittyminen ei tapahdu hetkessä. Testattavaan ominaisuuteen liittyvien teknologioiden ymmärrys pitäisi olla vähintään riittävällä tasolla, kuten viikolla pohditut verkkoliikenteeseen liittyvät ilmiöt. Erilaisiin lisenssivaihtoehtoihin liittyvä testaus taas pakottaa testaajan tutustumaan lisensointimalleihin, sekä niiden kuriositeetteihin.

Rajanveto riittävän kattavan ja liiallisen kattavan välille on tehtävä, täydellistä testikattavuutta on joillekin ominaisuuksille mahdotonta luoda. Tilanne tulee vastaan esimerkiksi kompleksista käyttäjätarinaa seuratessa, jossa voidaan tehdä variaatiota siihen, kuinka käyttäjä pääsee vaikka viisivaiheisesta lomakkeesta viimeiselle sivulle, jossa lomake palautetaan. Esimerkkinä käytettäköön lomakesivuilla vapaata liikkumista ja valinnaisten kenttien mielivaltaisessa järjestyksessä täyttämistä. Pelkästään viiden sivun kertoma antaa 120 erilaista permutaatiota siitä missä järjestyksessä

sivuilla käydään ja jos joka sivulla on esimerkiksi kolme vapaavalintaisessa järjestyksessä täytettävää kenttää, nousee erilaisten permutaatioiden määrä 1.3 miljardiin erilaiseen vaihtoehtoon. Tämä absurdilta kuulostava määrä erilaisia käyttäjän polkuja hahmottaa hyvin sitä useasti vastaan tulevaa testaajan tunnetta siitä, että testikattavuus on riittävä mutta sitä voidaan laajentaa sitä mukaa kun sille tulee tarve tai joku sitä vaatii.

3.7 Viikko 7

Maanantai 24.10.2022

Ohjelmistokehitys on yksilösuoritus, jota silti tehdään ryhmässä. Ohjelmiston laatuun vaikuttaa valtavan paljon versiohallinta, eli yhden henkilön ohjelmakoodi ei pääse tuotantoon saakka ilman että se katselmoidaan vähintään yhden, mielellään useamman aiheen ymmärtävän henkilön toimesta.

Ohjelmakoodin lisääminen olemassa olevaan lähteeseen on tarkka prosessi, jossa auttaa myös ymmärrys yrityksen ohjelmointikonventioista, tiimin konventioista sekä alkuperäisen toteutuksen tehneen henkilön tyylin tuntemus. Lukiessa toisen henkilön koodia alkaa pikkuhiljaa ymmärtää kuinka hän on halunnut jäsentää monimutkaisen kokonaisuuden – joko pieninä paloina tai pitempänä selkeänä jatkumona.

Tiistai 25.10.2022

Installerin JavaScriptillä toteutettu skriptaustuki oli tänään työn alla. Eräs muutos aiheutti sen, että oli tehtävä testi, joka yrittää asentaa mutta asennusta ei sallita. Kaikki Installerin ikkunan elementit on script enginen saavutettavissa, mutta ongelmaksi muodostui se, miten komponenttien nimiin ja kaikkeen muuhun pääsee käsiksi. Pohdittuani asiaa hetken, huomasin että kaikki elementit sai yksinkertaisesti esille gui.pageWidgetByObjectName -funktion palauttaman objektin sisältämät jäsenet listaamalla. Pythonilla JavaScriptin generoiminen oli ensimmäinen kerta, kun ylitin ohjelmointikielien rajat tällä tavalla.

Keskiviikko 26.10.2022

Kaikessa ohjelmistokehityksessä yksinkertainen on kaunista. Yksinkertaista koodia on mukava ja selkeä lukea ja sen kanssa on vaikeampaa tehdä virheitä. Lähdekoodin pitäminen yksinkertaisena on tärkeää ja siihen on erilaisia keinoja yhtä paljon kuin on koodaajia.

Eräs tapa pitää koodi siistinä on eritellä muuttujien sisältö erilliseen määrittelytiedostoon. Esimerkiksi pitkät tekstityyppiset muuttujat saa siististi peräjälkeen yhteen tiedostoon, joka voidaan sitten import –komennolla tuoda tarvittaessa tiedostoon kuin tiedostoon.

Määrittelytiedoston eduksi voidaan myös lukea se, että jos muuttujan sisältö täytyy muuttaa esimerkiksi ulkoisen muutoksen vuoksi, riittää korjauksen tekeminen ainoastaan määrittelytiedostoon, jos siitä on tehty kaikkialle muualle vain viittaus.

Määrittelytiedoston käytön huonoksi puoleksi voidaan laskea se, että kaikki varsinainen muuttujien sisältö ei näy itse ohjelmakoodissa. Muuttujien sisällön näkeminen kehitettävän toiminnallisuuden vieressä auttaa monesti hahmottamaan erilaisia toimintoja tekstin muokkauksen tai yhdistelyn kanssa, tai esimerkiksi matemaattisissa toimenpiteissä.

Torstai 27.10.2022

Tämä päivä meni suurelta osin koodin tulkkauksessa debuggeri apuvälineenä. Monessa ohjelmointityökalussa on mahdollisuus asettaa merkki jollekin koodiriville, jonka kohdalla koodin suoritus pysähtyy ja debugger näyttää ohjelman sisäisen tilan. Ohjelman tilasta saadaan selville mitä muuttujia on määriteltynä tässä kohtaa koodin nimiavaruutta, mitä tyyppiä muuttujien sisältämä data on sekä mikä itse muuttujan sisältö on. Useasti ongelman ratkaisuun riittää pelkkä virhesanoma, mutta useasti silti koodissa pitää pysähtyä pitkästi ennen virhettä, jotta itse ongelman aiheuttajan löytää.

Perjantai 28.10.2022

Lähdekoodia kaivellessa erilaiset dokumentaatiot ovat kullanarvoisia selvitellessä erilaisten funktioiden toiminnallisuuksia ja logiikkaa. Pythonissa luokkafunkiosta käytetään nimitystä metodi ja luokkafunktioiden dokumentaatiot ovat docstring -tyyppisiä merkkijonoja, eli Python -kääntäjän dokumentaatioksi ymmärtämiä merkkijonoja (Goodger & Rossum 2001).

Kuvan 2 mukainen esimerkki yhdestä dokumentaatiotyylistä on Javadoc –tyylinen moduulin dokumentaatio. @param <parametrin tyyppi> <parametrin nimi> <kuvaus> syntaksilla kuvaillaan metodin parametrien tarkemmat tiedot ja vaatimukset. Erittäin yksiselitteisetkin funktiot on hyvä dokumentoida yksirivisellä kuvan 3 tapaisella dokumentaatiolla, jonka lisäksi parametreille on määritetty type hinting asettamalla sulkeissa oleviin parametreihin kaksoispisteellä ja välilyönnillä erotettuna.

```
"""
replaces template place holder with values

@param string timestamp     formatted date to display
@param string priority      priority number
@param string priority_name priority name
@param string message       message to display

@return string formatted string
"""
```

Kuva 2. Javadoc-tyylinen funktion kommentointi. (Dion, 2011)

```
def return_sum(first_number: int, second_number: int):
    """ Method returns the sum of two integers """
    return first_number + second_number
```

Kuva 3. Metodin dokumentaatio

Kaikki dokumentaatio, oli se kattavaa tai suppeaa, auttaa logiikan ymmärtämisessä valtavasti. Palatessa pitkän ajan kuluttua oman koodinsa äärelle ei sen logiikka enää ole välttämättä ollenkaan muistissa. Logiikan muistiin palautumisen nopeuttaminen keinolla millä hyvänsä säästää valtavasti aikaa ja parantaa työtehoa. Dokumentaatioiden tekeminen olisi hyvä toteuttaa lähdekoodin vertaisarviointijärjestelmän tasolle, jolloin dokumentoimatonta koodia ei edes voisi saada hyväksytyksi paketinhallintaan.

Viikkoanalyysi

Uuden koodin kirjoittaminen on useimmiten helpompaa ja nopeampaa kuin olemassa olevaan logiikkaan lisätoimintoina lisääminen. Olemassa olevien toimintojen toiminnan rikkoontumattomuuden varmistaminen on ainakin allekirjoittaneen kokemuksesta vaativampaa kuin kokonaan uuden toteutuksen tekeminen.

Kaikenlainen lähdekoodin yhtenäisen tyylin säilyttämisen sopiminen tiimitasolla tehtynä auttaa tekemään laadukkaampaa ja helpommin ylläpidettävää koodia. Kommentointi koodirivien sekaan sekä metodien ja luokkien dokumentoiminen on vaatimus laadukkaalle koodille. Jollain ajanhetkellä itsestään selvä logiikka voi olla vuoden päästä unohtunut tyystin ja muutama rivi dokumentaatiota voi pelastaa paljon työaikaa, mitä joutuisi muuten käyttämään tutkimustöissä.

3.8 Viikko 8

Maanantai 31.10.2022

Tämä päivä meni oman toteutukseni refaktoroinnissa, Kyseessä oli vars() -funktiolla paikallisesta nimiavaruudesta muuttujien arvojen kaivaminen nimellä. Ajattelin, että voi olla näppärää, jos funktiolle voidaan antaa parametrina listatyyppinen muuttuja, joka sisältää muuttujien nimet, jotka on määritelty samassa tiedostossa missä itse funktiokin. Tämä järjestely paljastui vähintäänkin huonoksi, sekä tietoturvamielessä että eri ympäristöjen erilaisten nimiavaruuksien vuoksi. Vaihtoehdoksi löysin Python 3.7 versiosta eteenpäin tarjolla olleen tietoluokan, eli dataclass-luokan. Dataclass-luokka on vapaasti kuvailtuna kevyt versio tavallisesta Python-luokasta ja määritellään yksinkertaisimmillaan kuvan kaltaisella tavalla:

```
@dataclass
class C:
    ...

@dataclass()
class C:
    ...

@dataclass(init=True, repr=True, eq=True, order=False, unsafe_hash=False, frozen=False,
class C:
    ...
```

Kuva 4

Poikkeuksena tavalliselle luokalle, kuvan 4 mukainen dataclass ei tarvitse init –funktion määrittelyä vaan se tekee ne itsestään automaattisesti tai parametrilla ‘init=True’. Dataclass -järjestelyn ansiosta sain toteutettua “näppärän” ajatukseni siististä funktiokutsusta, jonka taakse piilotetaan muuttujien nimillä pitkiä tiedostopolkuja ja funktiokutsuissa voidaan käyttää ainoastaan komponentin nimeä tai nimiä listamuuttujan sisällä.

Tiistai 1.11.2022

Toimistolla tänään käytiin läpi eilistä dataclass –luokan toteutusta ja kerroin uudesta Pythonin ominaisuudesta tiimissä. Pythoniin tulee aika taajaan uusia melko merkittäviä ominaisuuksia, jotka voivat helpottaa ohjelmointia merkittävästi. Tieto uusista ominaisuuksista tulee useasti kollegoilta tai ohjelmointia työkseen tai harrastuksekseen tekeviltä tuttavilta, tällainen tiedon jakaminen on mielestäni tärkeää.

Sain tänään lähitulevaisuudessa odottavaa seuraavaa ohjelmajulkaisua varten omat aiemmin tekemäni muutokset sisään paketinhallintaan ja uudet muutokset mukaan automaatiotesteihin. Käynnistin alustavan ajon, joka kattoi kaikki Installeria koskevat testit. Muistin aikaisemman julkaisun yhteydessä pohtineeni asioiden tärkeysjärjestystä uuden toteutuksen ja vanhojen testien ajamisen välillä. Huolehdin tässä julkaisussa siitä, että pidän releasemanagerin koko ajan kartalla siitä, mikä on automaatiotestien tila ja onko mahdollisesti jotain ongelmia testeissä tai löytyneitä bugeja itse Installerissa.

Keskiviikko 2.11.2022

Tänään kävimme viikkopalaverissa läpi sitä, että saatuani tämänhetkisen erittäin laajan toteutuksen valmiiksi voisin alkaa käymään läpi uutta, vaikkakin pientä testauksen vastuualuetta. Uuden vastuualueen opettelu aloitetaan pala palalta. Tutustumisen ensimmäinen vaihe olisi tutustua olemassa olevaan testikokonaisuuteen ja varmistaa testien oikeanlainen toimintatapa.

Lisäksi sain tänään miltei kuukauden puurtamisen jälkeen laajan yksittäisen testikokonaisuuden ajoon virtuaalikoneille ja kaikille kolmelle käyttöjärjestelmälle. Valtava määrä opiskelua ja tutkimista tuotti matkan varrella useamman aihetta sivunneen lähdekoodilisäyksen tai muutoksen sekä erinäisiä bugiraportteja kokeilevasta testauksesta.

Torstai 3.11.2022

Päivä kului julkaisutestien tuloksien raportoinnissa. Raportointityylini on aina mahdollisimman lyhyt, mutta kattava, toimitan epäonnistuneen testin tulokset kaikkine oleellisine lokeineen toteutuksen tehneelle kehittäjälle sekä pyydän kertomaan, jos testiympäristöistä tarvitaan yhtään mitään muita

lokitietoja. Näyttää siltä, että tuleva julkaisu saadaan testattua hyvin ennen julkaisua sekä korjaukset tehtyä ajoissa.

Uusia korjauksia odottaessa silmäilin olemassa olevia työtehtäväkuvauksia ja tutustuin tuleviin implementaatioihin hieman etukäteen. Samalla yritin saada selville sitä, laitetaanko koko testauskokonaisuus ajoon heti seuraavan version tullessa saataville vai odotanko kaikki mahdolliset korjaukset tähän väliin. Julkaisua edeltävänä aikana on seurattava vähän tiukemmin muun tiimin tilannetta, että osaa ajoittaa omat toimensa.

Perjantai 4.11.2022

Seurantajakson viimeinen päivä kului releasekandidaattipakettien testien aloituksella, tuloksien seuraamisella ja testin parin korjauksen tekemisellä. Viikkopalaverissa keskustelimme ulkoisesta paineesta julkaisuajankohtaan liittyen, tai itse asiassa paineen puuttumisesta. Ulkoinen paine esimerkiksi asiakkaan suunnalta voi aiheuttaa hätiköityjä päätöksiä ja kiireen tuntua testaamisessa ja itse julkaisuprosessissa. Ohjelmiston viat olisi aina paras löytää ennen ohjelmiston julkaisua ja tarvittaessa julkaisun ajankohtaa lykätään edemmäs, jos korjaukset vaativat aikaa.

Ohjelmistojulkaisussa myös yrityksen sisäisten prosessien on tuettava julkaisuprosessia, esimerkiksi myynnin on ymmärrettävä se, että asiakkaalle ei kannata antaa tarkkaa julkaisuaikataulua ilman varmaan tietoa siitä, että se varmasti toteutuu.

Ohjelmistoalalla, kuten monella muullakin alalla kiire on monessa tapauksessa keksittyä eikä se liity mihinkään muuhun kuin huonosti ymmärrettyyn tai ylläpidettyyn prosessiin jossain vaiheessa ohjelmistotuotannon ketjua.

Viikkoanalyysi

Ohjelmistojulkaisun aikatauluttamisen tärkeys selkeästi tiedostetaan yrityksessämme, sillä releasemanagereita on useampi henkilö. Jokaisella managerilla on sopivan kokoinen projekti–yleensä pari – vastuualueellaan. Olen pohtinut joitakin pelialan julkaisuita, joiden julkaisupäivä tiedotetaan etukäteen hyvissä ajoin, esimerkiksi puolen vuoden päähän. Puoli vuotta ohjelmistokehityksessä on suuren tiimin kanssa todella pitkä aika ja puolen vuoden aikana voi tapahtua vaikka mitä onnistumisia tai epäonnistumisia projektissa. Pelkästään projektin sisäiset tekijät voivat aiheuttaa projektin

viivästymistä, puhumattakaan kolmansien osapuolien teknologioista, joista moni pelialan projekti on vahvasti riippuvainen.

Ohjelmistojulkaisun laadun varmistamiseksi olisi optimaalista, että ohjelmisto julkaistaan vasta, kun se on valmis. Edellinen lause itsestäänselvyyksien lisäksi sisältää sen huomion, että ohjelmiston julkaisupäivää voidaan muuttaa joustavasti. Testaamamme Qt Installer tuntuu pieneltä hippuselta suuressa projektissa, jota suuret yleisöt eivät kiukkuisine kommentteineen soimaa sosiaalisessa mediassa siirtyneen julkaisupäivän takia. Qt Installer on silti lukuisille ohjelmistokehittäjille tarpeellinen työkalu, jota ilman monelle yritykselle elintärkeä ohjelmisto ei asennu.

Tällaisen asian tiedostaminen on tärkeää, jotta hahmottaa oman paikkansa pitkässä ketjussa, jossa tuotetaan laatua eri paikoissa arvoketjua erilaisille ohjelmiston käyttäjille. Me olemme tiimimme kanssa pitkän ketjun alkupäässä, jossa huolehditaan Qt Frameworkin kunnollisesta asentumisesta ohjelmistokehittäjän tietokoneelle, joka kehittää bensapumpun käyttöliittymää öljy-yhtiön alihankkijalla, jonka tehtävä on huolehtia siitä, että tavaratoimitusta tekevä rekkakuski saa varmasti oikean määrän polttoainetta, jotta voi toimittaa taas tärkeän lastinsa perille saakka.

4 POHDINTA

Opinnäytetyön tavoitteena oli tarkkailla työntekoa kahden kuukauden ajan tarkastellen työskentelyä erilaisista näkökulmista. Lisäksi tavoitteena oli tutkia ja kehittää omaa työskentelyä. Tavoitteina oli myös järjestelmien käytön sujuvuus, sekä näiden käytön dokumentointi että työnkuvaan liittyvien eri osa-alueiden tärkeysjärjestyksen selvittäminen.

Kahden kuukauden aikana projektin eri osa-alueet ovat käyneet tutummiksi ja syy-seuraussuhteet eri toiminnallisuuksien välillä ovat selkeytyneet. Olen myös alkanut hahmottaa, kuinka laaja käyttäjäkunta Installer frameworkilla ja Qt Installerilla on. Installerilla on “asiakkaita” yrityksen sisäisesti, maksavia asiakkaita firman ulkopuolelta ja Installeria käyttää myös laaja open source yhteisö. Projektin teknisen sisällön oppimisen lisäksi olen alkanut hahmottaa omaan rooliini liittyvien henkilöiden ja sidosryhmien merkityksen. Tietystä asiasta tiedon hankkiminen onnistuu helpommin, kun tietää, kuka vastaa mistäkin osa-alueesta.

Eri tiimien välinen dynamiikka selkeni tarkastelujakson aikana. Installertiimissä syntynyt tarve muotoutui ominaisuudeksi, joka vietiin tiedoksi CI -tiimille. CI -tiimin jäsenien kanssa pienten palaverihetkien jälkeen oli selkeä kuva, mitä virtuaalikoneelle asetetut odotukset olivat. Prosessi, jonka lopputuloksena on verkossa oleva oikealla sisällöllä varustettu virtuaalikone, oli kevyt ja vähän byrokratiaa vaativa. Toinen hyvä esimerkki oli releasetiimin ja Installertiimin välisen tiedon kulun hoituminen releasemanagerin kautta. Release manager oli molempien tiimien palavereissa tarpeen mukaanjakamassa tietoa Installerin kehityksen uusista uutisista tai lähitulevaisuudessa tapahtuvista muutoksista, jotka oli otettava huomioon releasetiimissä.

Kahden kuukauden aikana opin jäsentämään omaa ajankäyttöäni vastaamaan paremmin projektin vaatimuksia. Oman tekemisen priorisoinnin tarkastelua jouduin tekemään pariin otteeseen. Julkaisuajankohdan lähestyessä testaaminen ja olemassa olevien testien toiminnan varmistaminen on tärkeämpää kuin uusien ominaisuuksien kehittäminen. Lisäksi testikattavuuden riittävyyden arvioiminen tuli tutuksi. Joihinkin testitapauksiin ei ole mahdollista saada riittävää kattavuutta. Joissakin tapauksissa muutama tyypillinen testitapaus ja corner caset voivat olla riittävä määrä testejä.

Kysyin palautetta kahdeksan viikon ajanjaksolta sekä releasetiimistä että Installertiimistä siitä, miten yhteistyössä kanssani ollut henkilö oli nähnyt allekirjoittaneen kehitysvaiheet projektiin tuloni

jälkeen. Palautteessa mainittiin, että prosessien ymmärtäminen ja tuntemus on lisääntynyt, samoin olemassa olevan lähdekoodin toimintojen hyödyntäminen. Sain myös palautetta hyvistä kehitysideoista projektiin. Kysyin myös, ovatko tiimiläiset oppineet jotain uutta minulta tai olenko tuonut jotain uutta projektiin. Tähän liittyvässä palautteessa mainittiin, että vanhojen toimintatapojen hyvällä tavalla kyseenalaistaminen on tuonut uutta innokkuutta projektiin, sekä myös Python–ohjelmointikielen tutuksi tekeminen C++ -ohjelmointikieltä käyttäneeseen Installertiimiin koettiin hyvänä.

Installertiimin kanssa tutuksi tuleminen on edesauttanut ohjelmoijan ja testaajan välisen yhteistyön muodostumista. Sen ymmärtäminen, miten Installertiimin ohjelmoijat testaavat itse omaa koodiaan, auttaa hahmottamaan automaatiotestien tuottaman lisäarvon tuotteen kehityksessä. Kokematto-muuteni testitapausten dokumentoinnin kanssa on antanut aihetta pohtia keinoja löytää paras tapa haastatella ohjelmoijaa uutta testitapausta luodessa. Selkeä ja yksinkertainen kertomus ohjelmistotestin etenemisestä antaa hyvän pohjan tehdylle työlle ja voisi jopa sellaisenaan toimia testikokonaisuuden dokumentaation osina. Tämä on selkeästi yksi kehityshaaste, jonka koen tarpeelliseksi osaksi urakehityksessäni.

LÄHTEET

Wikipedia 2022a. Anti-pattern. Hakupäivä 12.12.2022
https://en.wikipedia.org/wiki/Anti-pattern

Wikipedia 2022b. Black-box testing. Hakupäivä 12.12.2022
https://en.wikipedia.org/wiki/Black-box_testing

Wikipedia 2022c. Code refactoring. Hakupäivä 12.12.2022
https://en.wikipedia.org/wiki/Code_refactoring

Wikipedia 2022d. Commit (version control). Hakupäivä 12.12.2022
https://en.wikipedia.org/wiki/Commit_(version_control)

Wikipedia 2022e. Corner case. Hakupäivä 12.12.2022
https://en.wikipedia.org/wiki/Corner_case

Wikipedia 2022f. Debugger. Hakupäivä 12.12.2022
https://en.wikipedia.org/wiki/Debugger

Wikipedia 2022g. False positives and false negatives. Hakupäivä 12.12.2022
https://en.wikipedia.org/wiki/False_positives_and_false_negatives

Goodger, David & van Rossum, Guido 2001. PEP 257 – Docstring conventions. Hakupäivä 12.10.2022
https://peps.python.org/pep-0257/

Hamilton, Thomas 2022. What is Software Testing? Definition. Hakupäivä 29.10.2022.
https://www.guru99.com/software-testing-introduction-importance.html

Wikipedia 2022. Hard coding. Hakupäivä 12.12.2022
https://en.wikipedia.org/wiki/Hard_coding

Hannonen, Heli & Koivisto, Tiina. Näin ehkäiset työstressiä. Työterveyslaitos.
Hakupäivä 10.11.2022
https://www.ttl.fi/teemat/tyohyvinvointi-ja-tyokyky/stressi-ja-tyouupumus/nain-ehkaiset-tyostressia

Stackoverflow 2022. JF Dion. Using Javadoc for Python documentation.
Hakupäivä 13.10.2022
https://stackoverflow.com/questions/5334531/using-javadoc-for-python-documentation

Wikipedia 2022. Project management software
https://en.wikipedia.org/wiki/Project_management_software

Wikipedia 2022. Qt (kehitysympäristö).
Hakupäivä 11.12.2022
https://fi.wikipedia.org/wiki/Qt_(kehitysymp%C3%A4rist%C3%B6)

Wikipedia 2022. Scrum (software development). Product owner.
Hakupäivä 12.12.2022
https://en.wikipedia.org/wiki/Scrum_(software_development)

Wikipedia 2022. Software testing.
Hakupäivä 12.12.2022
https://en.wikipedia.org/wiki/Software_testing

Wikipedia 2022. Software versioning.
Hakupäivä 12.12.2022
https://en.wikipedia.org/wiki/Software_versioning

Wikipedia 2022. Test suite.
Hakupäivä 12.12.2022
https://en.wikipedia.org/wiki/Test_suite

Wikipedia 2022. Version control.
Hakupäivä 12.12.2022
https://en.wikipedia.org/wiki/Version_control

Wikipedia 2022. White box testing.
Hakupäivä 12.12.2022
https://en.wikipedia.org/wiki/White-box_testing