Sanna-Mari Äyrämö

Ohjelmistotekniikan työkalujen tarjoamat näkökulmat kehitystyön tavoitteeseen ja mahdollisiin ratkaisuihin

Tietotekniikan pro gradu -tutkielma

7. joulukuuta 2020

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Sanna-Mari Äyrämö

Yhteystiedot: sanna-mari.s-m.ayramo@jyu.fi

Ohjaajat: Ilkka Pölönen

Työn nimi: Ohjelmistotekniikan työkalujen tarjoamat näkökulmat kehitystyön tavoitteeseen ja mahdollisiin ratkaisuihin

Title in English: Perspectives provided by software engineering tools on the goal and possible solutions of development work

Työ: Pro gradu -tutkielma

Opintosuunta: Ohjelmisto- ja tietoliikennetekniikka

Sivumäärä: 60+15

Tiivistelmä: Wernick ja Hall (2004) esittävät, että ohjelmistosuunnittelun työkalut toimivat eräänlaisina kognitiivisina linsseinä vaikuttaen suoraan siihen, miten suunnittelija hahmottaa käsillä olevan tehtävän tavoitteen ja sen mahdolliset ratkaisut. Kuhnilaisia käsitteitä soveltaen voidaan sanoa, että metodit ja työkalut sekä niiden käyttöönotto itsessään ilmentävät tietyllä alalla vallitsevan paradigman, eli tutkimusalakohtaisen matriisin, piirteitä. Tässä epistemologisesta näkökulmasta ohjelmistotekniikkaa tarkastelevassa opinnäytetyössä selvitettiin tapaustutkimuksen avulla sitä, miten ohjelmistokehittäjä kehitysprosessin mittaan eri työkaluja käyttäessään ymmärtää työskentelyn tavoitteen ja mahdolliset suunnittelu- ja toteutusratkaisut. Lisäksi tarkasteltiin sitä, kuinka eri työkalujen tarjoamien näkökulmien kanssa työskentely suunnittelu- ja toteutusprosessin myötä eteni. Tutkimuksessa eri työkalujen havaittiin asettuvan erityyppisiin, laajempaa kokonaiskuvaa täydentäviin rooleihin siten, että niiden voidaan tulkita osaltaan ilmentävän eheää paradigmaattista perustaa, jonka puitteissa ohjelmistoteknisiä suunnitteluongelmia voidaan tunnistaa ja ratkaista. Tulokset tukevat Wernickin ja Hallin (2004) tuloksia ja tulkintaa siltä osin, että alan käytännönhajoituksessa voitaisiin jo tunnistaa yhtenäisen paradigman, ja siten myös alan kypsymisestä kertovia normaalitieteen piirteitä.

Avainsanat: epistemologia, ohjelmistotekniikka, paradigma, tutkimusalakohtainen matriisi

Abstract: Wernick and Hall (2004) suggest that software design tools act as a kind of cognitive lenses, influencing how the designer perceives the goal of the task at hand and its possible solutions. Applying the Kuhnian concepts, the methods and tools, as well as their deployment in a project reflect the features of the paradigm (also known as the disciplinary matrix) prevailing in a particular field. In this thesis, which examines software engineering from an epistemological perspective, a case study was conducted to find out how a software developer understands the goal of the task and the possible design and implementation solutions when using different tools during a software development process. In addition, it was examined, how working with the perspectives offered by the different tools progressed along the development process. According to the results, different tools fit into different types of roles that complement the broader overall picture, so that they can be interpreted as contributing to a coherent paradigmatic basis within which software engineering design problems can be identified and solved. The results of the study support the findings and conclusions presented by Wernick and Hall (2004) insofar as some characteristics of unified paradigm basis could already be identified regarding the practical area of the software engineering field, and thus, despite diverse competing perspectives and schools of software engineering, there are observable signs of the phase of normal science in software engineering indicating the maturation of the field.

Keywords: disciplinary matrix, epistemology, paradigm, software engineering

Esipuhe

Lämmin kiitos ohjaajalleni Ilkka Pölöselle, jonka ystävällisessä ja asiantuntevassa ohjauksessa opinnäytetyön tekeminen oli antoisa oppimiskokemus. Kiitokset myös apelleni Kari Äyrämölle, jonka puoleen saatoin kääntyä sähkötekniikkaan liittyneiden kysymysteni kanssa työn empiirisen vaiheen alkutaipaleella.

Lopuksi haluan kiittää perhettäni. Lapset Milo ja Alisa kannustivat kotona toteutettua empiirisen vaiheen työskentelyä erityisesti elektronisiin rakennelmiin kohdistuvalla uteliaisuudellaan. Mieheni Sami kannusti ja tuki minua läpi opinnäytetyön tekemisen, kuten vain hän yhteisten vuosiemme kokemuksella osaa. Kiitokset teille kaikille!

Jyväskylässä 14.11.2020

Sanna-Mari Äyrämö

Termiluettelo

Arduino, arduino	Arduino on avoimeen laitteistoon ja ohjelmistoon perustuva mikrokontrolleri-elektroniikka-alusta ja ohjelmointiympäristö, jossa käytetään arduino-ohjelmointikieltä. Arduinon-laitteessa on kaksi kokonaisuutta: Arduino-mikrokontrolleriin liitetty ulkoinen kytkentä ja Arduinoon Arduino IDE:n avulla syötetty arduino-kielinen ohjelma.
hyperspektrikamera	Kamera, joka pystyy kuvaamaan useita eri valon aallonpituuksia erillisille kanaville. Hyperspektrikamerat toimivat yleensä 400-1000 nm ja 1000-2500 nm aallonpituusalueilla.
inkrementaalinen malli	Ohjelmistonkehitysmalli, jonka vaiheet voivat olla samoja kuin vesiputousmallissa, mutta prosessin vaiheiden toteutus tapahtuu päällekkäin, ja ennen kaikkea kokonaisprosessi sisältää joidenkin tai kaikkien vaiheiden toistoa. Ks. ’vesiputousmalli’.
lineaariskanneri	Automaatiojärjestelmä, jossa kelkka/pidike/alusta kulkee kiskoa pitkin edestakaisin.
rajakytkin	Kytkin, joka pysäyttää moottorin, kun kelkka/pidike/alusta koskettaa sitä.
viivaskanneri	Kuvantava fotodiodi, joka kuvantaa yhtä spatiaalista pikseliriviä kerrallaan.
vesiputousmalli	Ohjelmistonkehitys- tai -tuotantomalli, jonka vaiheiksi nimetään usein esitutkimus, vaatimusmäärittely, suunnittelu-, toteutus-, testaus-, käyttöönotto- ja ylläpitovaiheet. Mallille on leimallista, että nimetyt vaiheet suoritetaan mainitussa järjestyksessä. Työvaiheet voidaan toteuttaa osittain

limittäisinä, mutta prosessi ei sisällä toistoa, tai toistoa esiintyy vain hyvin vähän. Vaiheet pyritään toteuttamaan kerralla valmiiksi ennen seuraavaan vaiheeseen siirtymistä.

Kuviot

Taulukot

Sisältö

1 Johdanto

1.1 Työn tausta ja tavoitteet

Opinnäytetyön alkuunpanevana kimmokkeena toimi Paul Wernickin ja Tracy Hallin artikkelissaan ”*Can Thomas Kuhn's paradigms help us understand software engineering?*” (2004) esittämät ajatukset ohjelmistotekniikasta (*Software Engineering*) tieteenalana. Erityisen tärkeä tämän työn tutkimusaiheen löytämiselle oli Wernickin ja Hallin huomio siitä, miten ohjelmistoteknikon käyttämät metodit ja työkalut rajaavat kognitiivisen filtterin tavoin sitä, kuinka suunnittelija/tutkija kunakin hetkenä hahmottaa käsillä olevaa probleemaa ja sen mahdollisia ratkaisuja. Mainitussa artikkelissa tutkijat soveltavat Thomas Kuhnin (1922-1996) tieteenalojen kehitystä jäsentävää viitekehystä, ja erityisesti siihen sisältyvää paradigma-käsitettä hyödyntäen, ja selvittävät, missä määrin ohjelmistotekniikka voidaan tunnistaa kypsäksi ja yhtenäiseksi tieteenalaksi. (ibid.)

Wernickin ja Hallin kokoamien tulosten perusteella ohjelmistotekniikan teoriankehityksen kentällä vallitseva tilanne vastaa Kuhnin *esiparadigmaattisessa vaiheessa* olevan tieteenalan tilannetta, kun taas alan käytännönhajoituksessa voidaan jo nähdä kypsemmän alan piirteitä, nk. *normaalitieteen* piirteitä. Esipragmaattisessa vaiheessa tieteellisen yhteisön maailmankuva ja toiminta eivät vielä perustu yhteisössä jaetulle uskomuksien ja olettamuksien muodostamalle johdonmukaiselle kokonaisuudelle ja avainkäsitteiden määritelmille. Sen sijaan esiparadigmaattista vaihetta elävällä alalla ajatteluun ja toimintaan voivat vaikuttaa useat keskenään kilpailevat paradigmat. Wernick ja Hall kuitenkin huomauttavat, että kuhnilainen teoria on tarkoitettu kuvailevaksi, ei normatiiviseksi reseptiksi, jolla tieteenaloja voitaisiin ”parantaa”. Tutkijat ehdottavat, että ohjelmistotekniikan paradigman sisällöllinen monimuotoisuus saattaa olla jopa toivottavaa, mikäli se tukee alan käytänteiden moninaisuutta. (ibid.)

Tämä opinnäytetyö tarkastelee ohjelmistoteknisen suunnittelu- ja toteutustyön luonnetta tieteenfilosofian, erityisesti tieteen tieto-opin, eli *epistemologian*, näkökulmasta. Tutkimus jakautuu teoreettiseen ja empiiriseen osaan. Työn teoreettisessa osassa tarkastellaan, miten

ohjelmistotekniikkaa on käsitelty epistemologisesta näkökulmasta ja erityisesti Kuhnilaisen paradigma-käsitteen suhteen. Lisäksi kartoitetaan myös sitä, mitä tutkimus esittää ohjelmistotekniikassa käytettyjen työkalujen ja metodien vaikutuksesta ohjelmistosuunnittelijan maailmankuvaan, erityisesti tapoihin jäsentää käsillä olevia ongelmia ja niiden mahdollisia ratkaisuja.

Työn empiirisessä osassa toteutettiin ohjelmistotekninen kehitysprojekti, jonka myötä toteutettiin hyperspektrikameran lineaariskannerin ohjainohjelmiston ensimmäinen prototyyppi. Opinnäytetyön empiirisen vaiheen alkuvaiheessa kehitettävän järjestelmän toteutuksen suhteen päätettiin, että ohjelmointikielenä käytettäisiin Python-ohjelmointikieltä, ohjelmointiympäristöksi valikoitui Spyder ja askelmoottoreiden ohjailussa mikrokontrollerina päätettiin käyttää Arduinoa, jolla myöskin on oma ohjelmointikielensä ja ohjelmointiympäristönsä. Suunnittelu ja toteutustyö päätettiin aloittaa vaatimusmäärittelyllä. Nämä neljä työkalua – vaatimusmäärittely, ohjelmointiympäristöt eli IDE:t (*integrated development environment*), ohjelmointikielet ja Arduino-mikrokontrolleri oheislaitteistoineen – valittiin tutkimuksessa tarkasteltaviksi muuttujiksi.

Suunnittelu- ja toteutusprojektin mittaan tuotettiin muistiinpanoaineisto, jossa tutkija sanallisti sitä, kuinka hän kunakin hetkenä hahmotti työskentelyn kohteena olevan ohjelmistoteknisen haasteen ja sen mahdolliset ratkaisut. Aineistoa analysoimalla selvitettiin, millaisia näkökulmia edellä mainitut neljä muuttujaa tarjosivat suunnittelijalle käsillä olevaan ohjelmistotekniseen ongelmaan ja sen mahdollisiin ratkaisuihin, ja kuinka näkökulmien kanssa työskentely suunnittelu- ja toteutusprosessin myötä eteni.

1.2 Tutkimusaihe, tutkimuskysymykset, työhypoteesi

Tutkimuksen aiheena on ohjelmistotekniikassa esiintyvien paradigmapiirteiden ilmeneminen käytännön tilanteessa konkreettisten ja abstraktien työkalujen kautta. Työn tutkimuskysymykset ovat:

TK1: Millaisia näkökulmia suunnittelu- ja toteutusprojektin mittaan käsiteltäviksi tulevat ohjelmointikielet, työkalut ja työskentelymetodit suunnittelijalle tarjoavat työskentelyn kohteena olevaan ongelmaan ja sen mahdollisiin ratkaisuihin?
TK2: Kuinka näkökulmien kanssa työskentely suunnittelu- ja toteutusprosessin myötä etenee?

Analyysia edeltäväksi työhypoteeseiksi asetettiin seuraavat kaksi olettamusta:

”Jokainen neljästä tarkasteltavasta muuttujasta täsmentää ja konkretisoi suunnittelijan näkemystä tavoitteesta ja kaventaa hänen käsitystään mahdollisista ratkaisuista.”
”Tämän täsmentämisen tavan taustalta on tunnistettavissa tietty ohjelmistotekniikan alalla vaikuttava paradigma.”

Analysointivaiheen jälkeen saatuja tuloksia arvioitiin vertaamalla niitä työhypoteeseihin.

1.3 Tutkimusstrategia ja sen valintaperusteet

Työn tutkimusstrategiaa voidaan luonnehtia empiiriseksi laadulliseksi tutkimukseksi, joka toteutetaan tapaustutkimuksen muodossa, hermeneuttisen tutkimusperinteen fenomenologis-hermeneuttista strategiaa (Sarajärvi & Tuomi 2017, luku 1.3.3) soveltaen. Tämä näkyy työn toteutuksessa esimerkiksi siten, että tutkimuksen toteuttaja asettuu samanaikaisesti sekä tutkittavan että tutkijan rooliin, mikä mahdollistaa tiedon muodostumistapahtumaan liittyvien kysymyksien tarkastelun. Fenomenologisen tutkimuksen piirteistöstä tässä opinnäytetyössä korostuvat nimenomaan tiedonkäsityksien tarkasteluun suuntautuneet puolet. Tällöin kiinnostavaa on, miten tietystä kohteesta voidaan saada tietoa ja millaista tuo tieto on luonteeltaan. Fenomenologinen analyysi voi koostua sekä tutkimuskohteen tarkastelusta että tutkijan omien kohdetta koskevien kokemusten ja ymmärrysprosessin tarkastelusta.

Strategian hermeneuttinen puoli sen sijaan nostaa etualalle inhimillisessä ajattelussa tuotetut subjektiiviset merkitykset, eli tulkinnat, ja pyrkimyksen ymmärtää näitä merkityksiä syvemmin. Tähän prosessiin liittyvät hermeneuttisen tutkimuksen

ydinkäsitteet *esiymmärrys* ja *hermeneuttinen kehä*. Ymmärtäminen lähtee aina liikkeelle esiymmärryksestä, eli kohteesta aikaisemmin muodostetusta ymmärryksestä. Varsinainen ymmärtäminen merkitsee tulkintaprosessia, jossa ymmärrys syvenee vähitellen hermeneuttisen kehän avulla kuvattujen sisäänpäin kiertyvien kerroksien tapaan. (Sarajärvi & Tuomi 2017, luku 1.3.3)

Fenomenologis-hermeneuttisen tutkimuksen tavoitetta on luonnehdittu pyrkimykseksi ”nostaa tietoiseksi ja näkyväksi se, minkä tottumus on häivyttänyt huomaamattomaksi ja itsestäänselväksi, tai se, mikä on koettu, mutta ei vielä tietoisesti ajateltu” (Sarajärvi & Tuomi 2017, luku 1.3.3). Lainaus kuvaa osuvasti myös tämän opinnäytetyön yleisempää tavoitetta.

Tutkimuksen ensimmäisen tutkimuskysymyksen kautta tarkastelun kohteena ovat ne ohjelmistosuunnittelijan työskentelynsä päätavoitteesta muodostamat tulkinnat, joiden muostostumista ohjaavat ohjelmistosuunnittelun ja kehitystyön yhteydessä käytettävät käsitteelliset ja konkreettiset työkalut ja työskentelymenetelmät. Tarkastellut tulkinnat koskevat suunnittelu- ja toteutustyön kohteena olevaa pääongelmaa ja sen mahdollisia ratkaisuja. Niinpä ensimmäisen tutkimuskysymyksen yhteydessä fenomenologis-hermeneuttisen strategian hermeneuttinen puoli korostuu. Fenomenologinen orientaatio sen sijaan korostuu työn toisen tutkimuskysymyksen yhteydessä, jolloin pyritään kuvailemaan sitä prosessia, jonka mittaan ohjelmistoteknisen rajapinnan suunnittelija-toteuttaja pyrkii hahmottamaan työskentelyn tavoitetta eri työkalujen ja työskentelymetodeiden kautta ja ymmärtämään niitä vaihtoehtoja, joiden kautta tavoitteen voisi saavuttaa.

1.4 Aineistonkeruu ja -analyysi

Tutkimuksen empiirisessä osassa kerätty tutkimusaineisto kerättiin vuoden 2020 kesä-heinä- ja elokuun aikana toteutetun tapaustutkimuksen yhteydessä. Tapaustutkimus koostui tutkijan toteuttaman ohjelmistoteknisen suunnittelu- ja toteutusprojektin ensimmäisen syklin läpiviennistä. Tehtävän toimeksianto saatiin Jyväskylän yliopistolta. Projektin parissa työskennellessään tutkija kirjoitti ja kokosi digitaalisessa muodossa olevan laadullisen muistiinpanoaineiston. Tässä aineistossa tutkija kuvailee suunnitteluprojektiin

liittyvien ohjelmointikielien, työskentelymenetelmien, työkalujen ja laitteistokomponentteihin tutustumisen kautta muodostuvaa käsitystään suunnittelu- ja toteutustyön päätavoitteesta ja sen mahdollisista ratkaisuista.

Muistiinpanoilla ei erityisesti pyritty kattamaan varsinaisen suunnitteludokumentin (ks. liite) sisältöjä. Sen sijaan muistiinpanot vastaavat muodoltaan ennemminkin vapaamuotoista kuvailua ja oppimispäiväkirjaa. Aineisto sisältää myös eri lähteistä peräisin olevia kuvia, kuten esimerkiksi askelmoottoreiden Arduino-kytkentöjä havainnollistavia esimerkkikuvia. Tutkimuksenteon ajan aineistoa säilytettiin tutkijan omalla tietokoneella. Tutkimuksen julkaisemisen jälkeen aineisto hävitettiin.

Muistiinpanoaineistoa analysoitiin teoria- ja aineistolähtöisen analyysin välimaastoon sijoittuvalla *teoriasidonnaisen sisällönanalyysin analyysimenetelmällä*. Fenomenologian vaikutteiden seurauksen analyysin taustalla voidaan katsoa olevan pyrkimys kohti aineistolähtöistä analyysia, mutta aineistolähtöisen menetelmän toteuttaminen ideansa puhtaassa muodossa, siis analysointi täysin vailla teoreettisen taustatiedon vaikutusta, oletetaan hermeneuttisen esitiedon käsitteen valossa epärealistiseksi: Tutkijan ei voi olettaa pystyvän sulkemaan työn kontekstiin liittyviä teorioita koskevia tietoja analysointitilanteen ulkopuolelle, toisin sanoen, objektiivisten ajatusten esittäminen kohteesta ei ole mahdollista (ibid.). Teoriasidonnaisessa sisällönanalyysissä mikään yksittäinen teoria ei ohjaa analyysia. Sen sijaan analyysissa tehtyjä havaintoja *kytketään* teoriaan, mikäli aineisto antaa siihen mahdollisuuden. Tämä voi tapahtua siten, että teoria tukee analyysin mittaan tehtyjä havaintoja, tai siten, että tutkijan tekemät havainnot osoittavat aikaisemman teorian jossain suhteessa paikkansapitämättömäksi tai riittämättömäksi (ibid.).

1.5 Työn rakenne

Tässä luvussa esittelen opinnäytetyön rakenteen johdantoluvusta (luku 1) eteenpäin. Luku 2 esittelee tutkimuksen kirjallisuuskatsauksen. Luku 2.1 johdattaa lukijan ohjelmistotekniikan epistemologisen tarkastelun piiriin. Luvussa 2.2.2 keskitytään tarkastelemaan sitä, mitä ohjelmistotekniikan alan tutkimuksessa on sanottu välineiden ja menetelmien vaikutuksesta ohjelmistosuunnittelijan ajatteluun.

Luku 3 käsittelee opinnäytetyön empiirisenä osana toteutettua tutkimusta. Tutkimuksen puitteissa suoritettiin pienimuotoinen ohjelmistotekninen suunnittelu- ja toteutusprojekti, joka esitellään luvussa 3.1. Suunnittelu-ja toteutusprosessin mittaan tuotettua tutkimusaineistoa esitellään luvussa 3.2. Aineistolle suoritetun analyysin toimintatapoja kuvaillaan luvussa 3.3, ja luku 3.4 kuvailee analyysitulosten muodostamisprosessia. Luvussa 3.5 tarkastellaan havaintoja, jotka nousivat esiin aineistoa tutkimuksen neljän muuttujan kautta läpi käydessä. Näitä havaintoja voidaan pitää tutkimuksen välituloksina.

Luvussa 4 käydään läpi tutkimuksen tulokset tutkimuskysymyksittäin. Luvussa 4.1 käsitellään ensimmäistä tutkimuskysymystä (TK1) koskevat tulokset. Luvussa 4.1.1 eritellään yksittäiset aineistosta tunnistetut tavoitekuvaukset ja niihin liittyvät ratkaisut. Luvussa 4.1.2 tavoitekuvaukset jaotellaan kategorioihin niitä yhdistävien näkökulmien perusteella. Näin päästään käsittelemään tavoitemuotoilujen osajoukkoja muodostavia piirteitä. Luvussa 4.1.3 tarkastellaan, millaisia tavoitekuvauksia eri työkalujen yhteydessä esiintyi. Luvussa 4.2 läpikäydään toista tutkimuskysymystä (TK2) koskevat tulokset, jotka kuvailevat sitä, miten eri näkökulmien välillä työskentely suunnittelu- ja toteutusprosessin mittaan eteni.

Luvussa 5 saaduista tuloksista ja suoritetusta tutkimuksesta esitetään pohdintaa ja luku 6 tarjoaa tutkimuksesta yhteenvedon.

2 Kirjallisuuskatsaus

Tämä luku esittelee tutkimuksen kirjallisuuskatsauksen. Luvussa 2.1 ohjelmistotekniikan alaa ja sillä tuotettua tietoa käsitellään epistemologian näkökulmasta. Luvussa 2.2 keskitytään tarkastelemaan, sitä miten ohjelmistotekniikan alan tutkimuksessa välineiden ja menetelmien vaikutusta käyttäjänsä ajatteluun on tarkasteltu.

2.1 Ohjelmistotekniikan ala epistemologian näkökulmasta

Ohjelmistotekniikan ala tarjoaa mielenkiintoisen tarkastelukohteen tieteenfilosofiselle ja epistemologiselle keskustelulle. Tieteenfilosofinen näkökulma nostaa tarkastelun kohteeksi ohjelmistotekniikan ominaispiirteet ja näiden suhteen muihin, esimerkiksi luonnontieteellisen tutkimuksen ominaispiirteisiin nähden. Epistemologisen näkökulman myötä esiin nousee myös kysymys siitä, millaisille olettamuksille ’tieto’ ja ’totuus’ ohjelmistotekniikassa perustuvat.

Ohjelmistotekniikassa tuotetun tiedon luonnetta on tarkastellut esimerkiksi Holloway (1994; 1995). Holloway (1994) esittää yleiskatsauksen kolmeen tieto-opin piirissä tunnistettuun tiedonlähde-tyyppiin ja käsittelee eri tiedonlähteistä peräisin olevan tiedon suhdetta totuuteen. Hollowayn (ibid.) mukaan tieto voi olla kaikkitietävän tai inhimillisen toimijan *antamaa*. Tieto voi olla jotain mikä on *todistettavissa deduktiivisen logiikan lainalaisuuksiin perustuen*. Lisäksi tieto voi myös olla jotain mikä perustuu aistein vahvistettuun kokemukseen. Viimeksi mainittuun tyyppiin kuuluvat tapaukset *anekdootinomainen kokemus* ja *kokeilemalla saatu todiste*.

Tiedonlähteestä riippuen tiedolla on ’totuuden’ suhteen erilainen status. Vain kaikkitietävä tekijä voi tarjota *absoluuttisen totuuden*. Kaikkitietävä tekijä voi olla esimerkiksi uskonnollisessa kontekstissa Jumala. Sen sijaan ihmistekijän antaman tiedon totuusstatus on ainoastaan *näkemys*. Deduktiiviseen logiikkaan perustuvalla päättelyllä voidaan saavuttaa *ehdollinen absoluuttinen totuus, joka suhde totuuteen perustuu asetelmaan:* ”jos tietty ehto toteutuu, asia on näin”. Anekdootinomaisen kokemuksen tarjoama totuus on *mahdollinen totuus,* jonka suhde totuuteen perustuu lähtökohtaan: ”jos tietty asia tapahtui

kerran, se voi tapahtua myös uudestaan". Toistettavissa olevan kokeilemisen tuottama todiste sen sijaan tarjoaa tietoa, jonka totuusstatuksena voidaan pitää *todennäköistä totuutta,* jolloin taustalla on olettamus: "jos koe oli suunniteltu oikein, asia on näin". (Holloway 1995.)

Holloway (1995) tarjoaa useita esimerkkejä ohjelmistotekniikan alalle tyypillisistä tavoista ilmaista tietoa, ja esittää, että ohjelmistotekniikassa esitetty tieto perustuu pääasiassa anekdootinomaisiin kokemuksiin (*mahdollinen totuus*) ja inhimillisen toimijan tarjoamiin *näkemyksiin*. Hollowayn mukaan ohjelmistoteknisen tiedon yhteydessä loogisia perusteluita esitetään vain harvoin, eikä empiirisiin todisteisiinkaan juuri viittailla (Holloway 1994, 577). Niinpä tutkija vaatii, että ohjelmistotekniikassa tehtävien päätösten perusteisiin tulisi paneutua huolellisemmin (Holloway 1995) ja että sekä ohjelmistotekniikan tuotteita että prosesseja tutkivia valideja näkökulmia tulisi kehittää (Holloway 1994, 570).

Esimerkiksi Smith, Wernick ja Veneziano (2011) ovatkin esittäneet, että ohjelmistotiede asettaa tarpeen, johon voitaisiin vastata sosiaalitieteen paradigmaa soveltamalla. Tutkijat kuvaavat ohjelmistotekniikkaa ristiriitojen sovitteluna, missä toisaalla ovat ihmisen muodostamat subjektiiviset kokemukset ohjelmistopohjaisista teknologioista, ja toisaalla ohjelmistot, joiden kehitys perustuu objektiivisuuden kattavalle arkkitehtuurille. Ohjelmistokehitys sisältää artefaktien suunnittelun lisäksi ohjelmistotekniikan ennustustyyppisiä ongelmia, joihin erityisesti vaatimusmäärittely pyrkii tarttumaan. Smith, Wernick ja Veneziano (2011) painottavat, että luonnollisessa ympäristössä toimiviin tekniikan aloihin nähden ohjelmistotekniikan tilanne on oleellisesti erilainen. Ihmiskäyttäjien vuorovaikuttaessa ohjelmisto-pohjaisen teknologian kanssa syntyy "keinotekoisesti" ilmeneviä ilmiöitä. Näillä ilmiöillä tutkijat voivat tarkoittaa esimerkiksi teknologian käytettävyyteen liittyviä ilmiöitä. Niinpä siinä missä luonnontieteellisten lakien puitteissa toimivat tekniikan alat voivat ennustaa ja korjata toteutettujen artefaktien puutteita, ohjelmistotekniikassa vaatimusten toteutumista ei välttämättä pystytä ennustamaan pelkästään luonnontieteeseen nojautuen.

Artikkelissaan ”*Can Thomas Kuhn's paradigms help us understand software engineering?*” Wernick ja Hall (2004) tarkastelevat ohjelmistotekniikan tieteenalan nykytilaa ja kypsyysastetta Thomas Kuhnin (1922–1996) tieteenalojen vaiheita jäsentävää viitekehystä soveltaen. Tieteenalan kypsyysaste ilmenee ennen kaikkea siinä, millaisen perustan ala pystyy tarjoamaan tieteen tekemiselle ja tieteellisen tiedon tuottamiselle. Kun pyritään selvittämään, täyttääkö tietty ala Kuhnin käsitteistön valossa kypsän tieteenalan tunnusmerkit, keskeiseksi kysymykseksi nousee, omaako tarkasteltava ala eheän *paradigman*.

Kuhn täsmentää paradigma-käsitteen merkitystä ja asemaa tieteenalan kehitysprosessissa teoksensa *Tieteellisten vallankumousten rakenne* (*The Structure of Scientific Revolutions*, 1962) toisessa painoksessa, joka on julkaistu vuonna 1969 (suomennos, 1994). Paradigman merkitys voidaan kiteyttää Wernickin ja Hallin (2004, 237) tapaan tietyn tieteellisen yhteisön jakamien olettamusten, uskomusten, mallien ja lähestymistapojen muodostamaksi kokonaisuudeksi.

Koska paradigma-termiä käytetään myös suppeammassa merkityksessä spesifimpien toimintamallien nimeämiseen, tieteenalaa määrittävän (laajemman) paradigman voidaan katsoa kattavan useita paradimoja (Kuhn 2012, 181). Myös ohjelmistotekniikan kentällä tapahtuneita muutoksia ja syntyneitä uusia virtauksia on useaan kertaan nimetty meneillään olevaksi paradigman muutokseksi (ks. esim. Frakes 1994; Garlan 2010; Gronogo & Shearing 2008). Muutoksien ilmaantumisen on selitetty johtuvan esimerkiksi laitteiston muistin ja suorituskyvyn kehityksestä, ohjelmointikielten monimutkaistumisesta, ohjelmistoihin kohdistuvien odotusten kasvamisesta ja ohjelmistojen elinkaareen kohdistuvan ajattelun kehittymisestä (Gronogo & Shearing 2008).

Northover, Kourie, Boake, Gruner, ja Northover kärjistävät paradigm-käsitteen käyttöön liittyvää ongelmallisuutta havainnollisesti: ”*Applying Kuhnian terminology to SE [Software Engineering] naively, we could thus consider AM [Agile Methods, Agile Movement] and XP [Extreme Programming] to be a new paradigm of SE, whereas ‘Waterfall’ or other traditional SE methodologies would be considered part of the old paradigm*” (2008, 101). Koska paradigman suppeampi merkitys on kuitenkin syytä pystyä

erottamaan koko tieteenalan ajattelun perusteista vastaavasta paradigmasta, Kuhn (1994) ehdottikin teoksensa toisen painoksen jälkisanoissa ilmaisua *tutkimusalakohtainen matriisi* (*disciplinary matrix*) käytettäväksi tästä tieteenalaa kokonaisuutena määrittävästä paradigmasta.

Esimerkiksi Wernick ja Hall (2004) pitäytyvät artikkelissaan Kuhnin (ibid.) suosittelemassa tutkimusalakohtaisen matriisin käsitteessä. Tässä tutkimuksessa käytän kuitenkin paradigma-termiä lähtökohtaisesti käsitteen laajemmassa, tieteenalakohtaisen matriisin merkityksessä. Paradigman suppeampaan merkitykseen viitatessa tämä tehdään tiettäväksi erikseen. Termivalinta perustuu lyhyemmän yksiosaisen ilmaisun kielelliseen notkeuteen, sekä siihen, ettei tieteenalakohtainen matriisi käsitteenä näytä yleistyneen varsinakaan suomenkielisen akateemisen keskustelun käyttöön.

Kun tietyllä tieteenalalla hallitsee tietty paradigma, siirtyy se uusille tieteentekijöiden sukupolville akateemisessa koulutuksessa, jossa tulevat tutkijat perehdytetään suhteellisen yhtenevämuotoisen koulutuksen kautta alan kanonisoituihin ideoihin ja lähdeteoksiin. Paradigma varustaa yhteisönsä sellaisella olettamuksien ja uskomusten joukolla, joka mahdollistaa ryhmän sisäisen, johdonmukaisen kommunikaation. Eheän paradigman avulla tieteellinen yhteisö kykenee tunnistamaan ja ratkaisemaan sisäiset erimielisyytensä tieteellisten ongelmien ja hyväksyttävien toimintatapojen muodossa. (Kuhn 1994.)

Kuhnilaisen näkemyksen mukaan tieteenalan erilaisia kehitysvaiheita tai tiloja ovat esiparadigmaattinen vaihe, normaalitieteen vaihe sekä mullistusten ajat, joista alat pyrkivät palaamaan uuteen normaalitieteen vaiheeseen. Ennen kuin tieteenalalla on muodostunut sen ensimmäinen eheä paradigma, ovat alalla tehdyt havainnot hajanaisia, sillä ne perustuvat erillisten koulukuntien ajatteluun. Eri koulukuntien väliset ajattelutavat ja tapa tehdä tiedettä poikkeavat toisistaan, sillä ne ovat keskenään *yhteismitattomia* (Kuhn 1994, 18). Alan kypsymättömyys näkyy myös alan tieteellisissä julkaisuissa. Jos yhteisesti hyväksyttyä paradigmaa ei ole, jokaisen kirjoittajan täytyy ”rakentaa alansa perusteista lähtien uudelleen” (Kuhn 1994, 26). Käytännössä tämä johtaa siihen, että julkaisut ovat yleistajuisempia ja ”koko oppineelle maailmalle” (Kuhn 1994, 34) suunnattuja. Sitä vastoin eheän, yhteisesti hyväksytyn paradigman piirissä tietyt alaan liittyvät perusteet

voidaan olettaa julkaisun erityisen kohdejoukon tuntemiksi, joten itse julkaisussa voidaan keskittyä huomattavasti erikoistuneempien aiheiden käsittelyyn (ibid.).

Kun ala on kehittynyt kypsäksi tieteenalaksi, ja sen tila on vakaa, on se kuhnilaisen käsitteistön mukaan normaalitieteen vaiheessa. Normaalitieteen vaiheelle on siis määritelmällistä se, että on olemassa vallitseva paradigma, joka otetaan annettuna sitä liiemmin kyseenalaistamatta. Paradigman kyseenalaistamisen sijaan keskitytään ongelmien ratkaisemiseen *vallitsevan paradigman mukaisesti*. Tieteenalan suuriin linjoihin vaikuttavia paradigmoja voi kuitenkin samanaikaisesti olla olemassa enemmän kuin yksi. Kuhnilaisen näkökulman mukaan tieteessä tällainen tilanne voi esiintyä tieteenalan varhaisessa, esi-paradigmaattisessa vaiheessa sekä tieteenalan kumouksellisissa uudistumisvaiheissa, joissa jokin uusi paradigma nousee kumoamaan vanhan, alaa tähän asti hallinneen paradigman. (Kuhn 1994.)

Wernick ja Hall (2004) arvioivat aikaisempien osatutkimustensa perusteella, että vaikka ohjelmistosuunnittelun käytännön ja teoreettisen alan toimijat hyväksyvät tiettyjä yhteisesti jaettuja uskomuksia, vallitsee alalla kuitenkin oikeastaan useampi merkittävä keskenään kilpaileva paradigma. Kilpailevat näkemykset voivat poiketa toisistaan esimerkiksi siinä, miten ne suhtautuvat loppukäyttäjien rooliin ohjelmistokehitysprosessissa, tai siinä, tarkastellaanko käyttäjiä ihmisinä vai ennemminkin yhtenä mekaanisena ’järjestelmän osana’. Niinpä tutkijat päätyvät tulkitsemaan ohjelmistotekniikan tilan vastaavan Kuhnin käsitteistön esiparadigmaattista vaihetta, eli vaihetta, jossa ei vielä ole muodostunut yhtä hallitsevaa paradigmaa, jolle tieteenharjoitus perustuisi. (ibid.)

Tieteenalan kypsää, perustaltaan yhtenäisen ja vahvan ajattelun vaihetta kutsutaan normaalitieteen vaiheeksi. Myös normaalitieteen vaiheen alettua alalla voi tapahtua mullistuksia, nk. paradiman muutoksia, jolloin jokin uusi paradigma nousee kumoamaan aikaisemmin vallinneen paradigman johtoasemaa. Uuden valtaparadiman otettua asemansa (tai vanhan paradigman palatessa takaisin valta-asemaan) seuraa alalla jälleen uusi normaalitieteen vaihe. (Kuhn 1994.)

Mielenkiintoista kylläkin, Wernickin ja Hallin (2004) mukaan ei välttämättä ole mielekästä ajatella, että ohjelmistotekniikka alana olisi tulevaisuudessakaan kulkemassa kohti yhden

vallitsevan paradiman tilannetta. Tutkijat perustelevat näkemystään alan käsittelemien *ilmiöiden ja artefaktien monimuotoisuudella*.

Wernick ja Hall (2004) pitävät perusteltuna olettaa, että mikä tahansa ennalta muotoiltu ja määritelty mekanismi, jota sovelletaan tiettyyn prosessiin, vaikuttaa myös kyseisen prosessin tuloksiin ja käyttäjänsä ajattelutapaan. Kuhnilaisessa ajattelussa tämä selittyy siten, että työkalut ilmentävät aina jotain näkökulmaa, ja eri metodien tai työkalujen käyttöönotto on juuri sitä, miten tietty vaikuttava paradigma *tulee esiin* käytännön tilanteessa (Kuhn 2012). Ohjelmistotekniikassa tällaisiin mekanismeihin lukeutuvat erilaiset käsitteelliset ja konkreettiset työkalut, esimerkiksi prosessimallit, ohjelmointityökalut, -kielet ja -metodit.

The DM [disciplinary matrix] elements explicit and implicit in the tools used by a software developer influence how he or she interprets the circumstances of a situation, sees the problem in that situation, and defines a valid solution to the problem. The DM of an SE [software engineering] method or tool can thus be viewed as a cognitive filter, modifying a software developer's view of the world (cf. Episkopou & Wood-Harper, 1986; Petre, 1989). (Wernick & Hall 2004, 239).

Tutkijoiden käsittelyssä rinnastuvat siis tieteenalojen ajattelua ja toimintaa määrittävät paradigmat ja toisaalta ohjelmistoteknikon käyttämät konkreettiset työkalut. Työkalut ilmentävät aina jotain näkökulmaa – ja eri metodien tai työkalujen käyttöönotto on juuri sitä miten tietty vaikuttava paradigma tulee esiin käytännön tilanteessa (Kuhn 2012). Wernick ja Hall (2004) vaikuttavat nostavan esiin vaikutussuhteen myös päinvastaiseen suuntaan, jolloin (mahdollisesti tutkimustoiminnan ulkopuolella kehitetyn) työkalun käyttöönotto voisi periaatteessa myös ikään kuin syöttää edustamaansa (vierasta) paradigmaa. Tällä tavoin eriävien paradimojen samanaikaisen olemassaolon tilanne olisi ylipäätään mahdollinen, ja se juontuisi nimenomaan ohjelmistotekniikan piirissä käytetyistä moninaisista menetelmistä, työkaluista ja ohjelmointikielistä.

Tästä lähtökohdasta Wernick ja Hall (2004) pääsevät erittelemään ohjelmistotekniikalle kiinnostavien tutkimuskohteiden kentää huomattavasti yksittäisiä menetelmiä, työkaluja, ratkaisuja ja lopputuotteena syntyviä artefakteja laajemmaksi tietyssä historiallisessa ja

kulttuurisessa kontekstissa tapahtuvaksi toiminnaksi: "*When examining the practice of SE, the mindset of a developer when making a decision can be considered in the context of the influences arising from the tools and techniques in use at that time.*" (Wernick & Hall 2004, 236). Tällöin on vain luonnollista, että samaan ongelmaan eri aikoina tai eri ympäristössä "parasta mahdollista" ratkaisua etsittäessä päädytään todennäköisesti erilaisiin lopputuloksiin: "*different SE tools and techniques, operationalising differing sets of beliefs, may produce different results when applied in practice*" (Wernick & Hall 2004, 237). Niinpä tutkijat toivovat voivansa rohkaista näkökulman avartamiseen ja tutkimuskohteiden toisenlaiseen arvottamiseen ohjelmistotekniikan alalla: "*The application of the analogy described in this paper and the analysis of the results obtained is capable of changing the nature of the discourse within SE, in particular that relating to the relative merits of methods and tools*" (Wernick & Hall 2004, 236).

Lázaro ja Marcos (2005) korostavat Wernickin ja Hallin tapaan ohjelmistotekniikan alan tutkimuskohteesta juontuvaa alan erikoislaatua. Heidän mukaansa ohjelmistotekniikan alalla on samankaltainen kaksoisluonne, kuin esimerkiksi elektroniikan ja kemiantekniika aloilla, joskin viimeksimainitut ovat jo kehittyneet aloina kypsemmiksi. Ohjelmistotekniikan tutkimuskohteiden voidaan katsoa jakautuvan kahteen tyyppiin: luonteeltaan tieteellisiin tai teknisiin (Lázaro & Marcos 2005).

Lázaro ja Marcos (2005) kuvailevat, miten tieteellisten aiheiden yhteydessä tarkastellaan teoriaorientoituneesti jo olemassa olevia kohteita, kuten esimerkiksi koodia, suunnittelumallia tai dokumentaatiota, kun taas tekniset aiheet ovat sellaisia, joissa pyrkimys on rakentaa uusi artefakti. Kahden tutkimuskohdetyyppin erilaisuudesta seuraa myös, että ohjelmistotekniikassa vallitsevat rinnakkain kaksi erillistä paradigmaa, jotka vastaavat näiden perimmäiseltä luonteeltaan erilaisten pyrkimysten asettamiin tarpeisiin. (ibid.)

Wernickin ja Hallin kanssa hieman eri linjoilla näyttäisi olevan esimerkiksi Ogundare (2017), joka käsittelee teknistä paradigmaa tieteellisestä paradigmasta tyystin erillisenä juonteena. Tutkija tarkastelee rinnakkain teknisen paradigman (*engineering paradigm*)

mukaisen toteutusprosessin yleisiä etenemisvaiheita sekä tieteelliseen paradigmaan (*the scientific paradigm*) perustuvan tiedonhankinnan etenemisvaiheita (ks. Kuvio 1.).

	Tieteellinen paradigma	**Tekniikan paradigma**
1.	Määrittele ongelma	Määrittele ongelma
2.	Tunnista muuttujat	Löydä faktat
3.	Muotoile hypoteesi	Määritä parametrit
4.	Määrittele metodologia	Määritä hyväksymiskriteerit
5.	Suunnittele koe	Määritä järjestelmän rajat
6.	Suorita koe	Suunnittele prototyyppi
7.	Testaa hypoteesi	Rakenna prototyyppi
8.	Analysoi tulokset	Testaa
9.	Muodosta johtopäätökset	Verifioi
10.	Esittele tulokset	Esittele tulokset

Kuvio 1. Tieteellisen paradigman ja teknisen paradigman kymmenen vaihetta. Kuva on mukaelma Ogundaren (2017, 169) esittämästä rinnastuksesta.

Tieteellisen ja teknisen paradigman tiedonhankintaprosessin vaiheet eroavat toisistaan sisällöllisesti ensimmäistä (”määrittele ongelma”) ja viimeistä (”esittele tulokset”) askelta lukuunottamatta (Kuvio 1). Yksittäisiä vaihe-eroja oleellisempi huomio on kuitenkin se, että prosessien kokonaisuutena tuottama tieto on luonteeltaan erilaista. Ogundare (2017) päätyykin tiivistämään Hollowayn (1994, 1995) kritiikin kanssa samansuuntaisesti, että tieteellisten teorioiden esittämä tieto ja paradigmat (tässä yhteydessä tulkitsen kirjoittajan viittaavan paradigman suppeampaan merkitykseen) perustuvat nimenomaan kontrolloituihin empiirisiin prosesseihin, kun taas tekniset metodit johdetaan ensisijassa kokemuksesta, jonka mukaan jonkin jo ennestään *tiedetään toimivan*.

Tämä tiedon löytämis- tai muodostamistavan erilaisuus ilmenee myös kuvio 1.:ssä rinnastettujen prosessien vaihe-eroissa. Siinä missä tieteellisen menettelytavan

alkuvaiheessa tunnistetaan muuttujat – mikä edellyttää yleistämistä ja abstrahointia – ohjelmistoteknisen prosessin piirissä ”löydetään faktat”, eli tunnistetaan tietyt konkreettiset lähtökohdat. Tieteellisen paradigman piirissä tutkijan muotoilemaa hypoteesia päästään testaamaan sen jälkeen, kun on ensin suunniteltu ja suoritettu erityinen tieteen perinteelle perustuva koe, jonka avulla saadaan tuloksia. Tekniikassa määritetään parametrit, hyväksymiskriteerit ja järjestelmän rajat, minkä jälkeen voidaan suunnitella ja toteuttaa prototyyppi. Prosessien tuottaman tiedon kannalta oleelliselta vaikuttaa erityisesti se, miten hypoteesin testaaminen ja prototyypin toteuttaminen rinnastuvat toisiinsa (ks. Kuvio 1).

Kuvio 1:n perusteella voidaan nähdä, miten tieteellinen menettely koettelee hypoteesin muotoon rakennettuja ideoita ja tuottaa lopulta analysoituja tuloksia, jotka voidaan esittää esimerkiksi teorioiden muodossa. Tieteellinen koe itsessään tarjoaa löydetylle tiedolle perusteen. Tekniikassa sen sijaan prosessin tuloksena tarjotaan artefaktia, johon päädytään kokoamalla ja yhdistelemällä toisiinsa tarvittava määrä jo olemassaolevaa tietoa. Ogundare (ibid.) täsmentää artikkelissa myöhemmin, että ohjelmistotekniikassa tietoa saavutetaan ennen kaikkea *mallintamisen* kautta, ja mallille on ensisijaisen tärkeää olla sisäisesti yhtenäinen. Teknisen prosessin tuotteena synnytetyn ratkaisun oikeellisuus varmistetaan testaamalla ja verifioimalla artefaktia sille asetettuja vaatimuksia vasten (ks. Kuvio 1.).

Siitä, kuinka Kuhnin ja myös muiden filosofien ajatukset ja teoriat nousevat esiin ohjelmistotekniikan alalla käydyssä keskustelussa, löytyy runsaasti lisää kirjallisuutta esimerkiksi Northoverin, Kourien, Boakerin, Grunerin ja Northoverin (2008) kirjoittamasta ohjelmistotekniikkaa filosofian näkökulmasta tarkastelevasta laaja-alaisesta esseestä.

Ohjelmistotekniikkaan kohdistuva tieteenfilosofinen tutkimus on vielä verrattain vähäistä, mahdollisesti alan nuoresta iästä johtien (Northover, Kourie, Boake, Gruner, & Northover 2008). Lopulta vaikuttaa siltä, että epistemologisessa keskustelussa ohjelmistotekniikan alaan tiedon tuottajana kohdistuu kritiikkiä (Holloway 1994, 1995; Ogundare 2017) erityisesti silloin, kun alaan liittyvä tieto ja toimintatavat oletetaan lähtökohtaisesti yhdeksi yhtenäiseksi kokonaisuudeksi ja niitä tarkastellaan tieteelliselle tiedolle yleisesti asetettujen kriteerien näkökulmasta. Keskustelussa esiintyy myös näkemys, jonka puitteissa ohjelmistotekniikka voitaisiin nähdä tieteellis-teknisenä hybridi-toimintakenttänä, jonka

ratkaisujen etsimiseen omistautuvilla alueilla ei välttämättä esiinny tieteenaloille kuhnilaisittain tunnistettua pyrkimystä säilyttää sisäinen yhtenäisyytensä. Tällöin toiminnan ja tuotetun tiedon heterogeenista luonnetta on selitetty niin tarkasteltun kohteena olleiden kuin myös tutkimuksessa käytettyjen välineiden moninaisuudella (Wernick ja Hall 2004). Lisäksi alan heterogeenista luonnetta on selitetty toiminan taustalla olevien motiivien perimmäisillä eroilla (Lázaro ja Marcos 2005). Näkökulmien välillä esiintyy eroja myös sen suhteen, tulkitaanko ohjelmistotekniikan uutta luova tekninen alue tutkimuksesta irralliseksi vai tutkimuksen piiriin kuuluvaksi toimintakentäksi.

2.2 Ohjelmistotekniikan välineiden ja menetelmien vaikutus käyttäjänsä ajatteluun

Esimerkiksi Tahsiri, Hale ja Niblock (2017) sekä Gürel (2018) ovat tarkastelleet konkreettisten suunnittelutyökalujen vaikutuksia suunnittelijan työhön. Tahsiri, Hale ja Niblock (2017) tarkastelevat, miten suunnittelutyökalun valinta vaikuttaa siihen, missä vaiheessa suunnitteluprosessia tuotetaan tietoa ja milloin tätä tietoa sovelletaan. Tiedon merkitykseksi tässä yhteydessä määritellään varusteet tai tarvikkeet (*paraphernalia*), joilla tietyn ongelma-avaruuden aluksi määrittelemätön alue saadaan suoritetun prosessin myötä määritellyksi. Tutkimuksessa havaittiin, että välineiden vaikutus liittyi merkittävimmin siihen, miten suunnitteluväline vaikutti uuden tiedon esille tuomiseen suunnittelutyön ulkoisessa tilassa, eli artefaktin tuottamisessa työkalun avulla (vastakohtana sisäiselle tilalle, eli esimerkiksi sille, että suunnittelijan on aktiivisesti pidettävä muistissaan tietty tieto voidakseen työskennellä).

Gürel (2018) selvittää opinnäytetyössään parametrisen suunnittelutyökalu *Grasshopperin* käytön vaikutuksia käsin luonnostelemiseen verrattuna. Työkalua käytettiin arkkitehtuurisuunnittelun käsitteellisen suunnittelun vaiheessa (*Conceptual Design Phase*). Työssä käytettyä tiedonhankintakeinoa kuvaillaan kognitiiviseksi vertailuksi. Tutkimuksessa työskentelymenetelmän havaittiin vaikuttavan merkittävässä määrin käyttäjänsä kognitiiviseen toimintaan. Suunnittelutyökalun käyttö esimerkiksi tehosti

suunnitteluongelman havaitsemista ja ratkaisemista, ajankäyttöä, erilaisten vaihtoehtojen luomista ja suunnitteluelementtien suhteiden tarkastelua. (ibid.)

Hirschheim ja Klein (1989) esittelevät neljä erilaista tietojärjestelmäkehityksen kentällä vaikuttavaa kehitysparadigmaa. Esitellyt paradigmat asettavat joukon erilaisia lähtöoletuksia koskien todellisuuden luonnetta, sen mahdollistamaa tiedonhankintaa ja suunnittelutyön tavoitteita ja mahdollisuuksia. Paradigman asettamat alkuoletukset ohjaavat kehitysprosessiin osallistuvien toimijoiden työskentelyä, ja niiden soveltaminen johtaa myös erilaiseen konkreettiseen lopputulokseen. Näin käy riippumatta siitä, ovatko paradigman asettamat taustaoletukset ohjelmistosuunnittelijan/-kehittäjän eksplisiittisesti tai implisiittisesti omaksumia. Taustaoletusten voimakas vaikutus työn toteutukseen ja lopputulokseen johtuu esimerkiksi siitä, että kehitysnäkökulman valinta vaikuttaa suoraan siihen, miten järjestelmäkehityksen tavoitteet oikeutetaan. Lisäksi paradigmanvalinnalla on myös erilaisia sosiaalisia seurauksia. (ibid.)

Hirschheimin ja Kleinin (1989) käsittelemät kehitysparadigmat ovat *funktionalistinen paradigma*, *sosiaalinen relativismi*, *radikaali strukturalismi* ja *neohumanismi*. Funktionalismi perustuu positivistiselle epistemologiselle asenteelle: todellisuudesta voidaan hankkia tietoa tekemällä todellisuuden objekteihin, ominaisuuksiin tai prosesseihin kohdistuvia mittauksia. Kehitysparadigman piirissä tunnistettuja toimijoita ovat johtava taho, järjestelmäkehittäjät ja järjestelmän käyttäjät. Näistä johtava taho asettaa järjestelmän tavoitteet, kehittäjät muuttavat erityisosaamisellaan tavoitteet tuotteeksi ja käyttäjät käyttävät tuotetta organisatoristen tavoitteiden saavuttamiseksi. Koko kehitystoimintaa ohjaava perusajatus on tuottojen maksimointi. Funktionalistisen paradigman yhteydessä työskentely perusuu tyypillisesti erilaisten mallien ja mallintamistyökalujen hyödyntämiselle. (ibid.)

Sosiaalinen relativismi ja radikaali strukturalismi ovat molemmat reaktioita funktionalistisen paradigman puutteisiin (Hirschheim ja Klein 1989). Sosiaalisen relativismin puitteissa todellisuus näyttäytyy monimutkaisena ja tavoittamattomana. Todellisuudesta voidaan hankkia vain useita erilaisia *käsityksiä*, joten antipositivistisessa hengessä itse todellisuuden nähdään jäävän tiedonhakijan tavoittamattomiin. Sosiaalisen

relativismin mukaan ohjelmistoteknisen kehitysprosessin toimijoita ovat käyttäjät ja kehittäjät, jotka yhdessä osallistuvat järjestelmäkehityksen prosessiin, joka kääntyy eräänlaiseksi merkityksenantoprosessiksi tai jopa kokonaan uuden merkityksen tuottamisprosessiksi. Tässä prosessissa käyttäjien tehtävä on tulkita ja merkityksellistää ympäristöään, kun taas kehittäjät toimivat muutosagentteina, jotka auttavat käyttäjiä ymmärtämään uutta järjestelmää ja sen ympäristöä. Sosiaalisen relativismin mukaisen kehitysprosessin yhteydessä suositaan erityisesti työkaluja, jotka tukevat ryhmätyöskentelyä. (ibid.)

Radikaalin strukturalismin perusta on materialistisessa historiankäsityksessä, joten sen intressien kiintopisteeksi muodostuu yhteiskunta, ja erityisesti sen keskiöön oletettu omistavan luokan ja työvoiman välinen konflikti (Hirschheim ja Klein 1989). Ohjelmistoteknisen prosessin toimijoita ovat johtava taho ja kehittäjät, joista johtava taho toimii omistavan tahon agenttina, kun taas kehittäjän tehtäväksi tulee valinta, toimiako omistaja-johtaja-linjan mukaisesti, vaiko työvoiman agenttina. Ensin mainitussa tapauksessa kehitetyt tietojärjestelmät ennen kaikkea koventavat työskentelyn intensiteettiä, kun taas jälkimmäisessä tapauksessa tietojärjestelmän avulla tuetaan työntekijöiden ammattitaitoa, tehdään työstä entistä palkitsevampaa ja saadaan samalla aikaan parempia tuoteita. (ibid.)

Viimeisenä Hirschheim ja Klein (1989) esittelevät neohumanistisen paradigman, jonka lähtökohtaisia intressejä ovat työ, keskinäinen yhteisymmärrys ja emansipaatio, eli yksilöiden vapauttaminen holhouksen tai eriarvoisuuden alaisuudesta. Nämä ovat ne kentät, joiden ympärille yhteiskunnan ja sosiaalisten järjestelmien nähdään rakentuneen. Nämä ovat myös ne kentät, joilta kehitysprosesseissa täytyy hankkia tietoa. Eri kentillä tieto on luonteeltaan erilaista. Neohumanismin epistemologisessa perustassa yhdistyvät positivistinen kiinnostus teknisen kontrollin mahdollisuuteen (liittyen luontoon ja myös ihmiseen) ja antipositivistinen näkemys ihmisten välisestä keskinäisestä yhteisymmärryksestä ja emansipaatiosta. Tietojärjestelmäkehitys tähtää yksilöiden emansipatoriseen vapauttamiseen, joka toteutetaan kehittämällä järjestemiä, jotka tukevat rationaalista keskustelua. Kehitysprosessin toimijoiksi tunnistetaan monimuotoinen sidosryhmien edustajista koostuva joukko sekä järjestelmäkehittäjät. (ibid.)

Episkopou ja Wood-Harper (1986) ehdottavat viitekehystä, jonka avulla voidaan valita parhaiten soveltuva työskentelynäkökulma systeemianalyyttistä ongelmanratkaisua ja suunnittelutyötä varten. Näkökulman valinta voidaan tehdä esimerkiksi seuraavien näkökulmien välillä: *The Human Activity Systems Approach, The Participative Approach* ja *The Data Analysis Approach*. Esitellään seuraavaksi mainitut työskentelynäkökulmat ja tämän jälkeen Episkopoun ja Wood-Harperin ehdottama viitekehys. Yksittäisen työskentelynäkökulman käyttöönotto ei välttämättä sulje pois jonkin toisen näkökulman soveltamismahdollisuutta. (ibid.)

The Human Activity Systems Approach -näkökulma perustuu ajatukselle, jonka mukaan ongelman todellisen ympäristön tuntemus ja analysointi johtavat mahdolliseen ratkaisuun. Ratkaisun nähdään virtaavan jossain määrin intuitionomaisesti tarkkaavaisen ja mahdollisimman laaja-alaisen havainnoinnin myötä suunnittelijaan ja ratkaisun kognitiivisen työstamisen vaihe seuraa vasta tämän jälkeen. Näkökulmaa voidaan nimittää myös *pehmeäksi* näkökulmaksi. (ibid.)

The Participative Approach -näkökulmassa korostetaan lähtökohtaa yksilön oikeudesta ja tarpeesta kontrolloida omaa kohtaloaan. Näin ollen käyttäjien sisällyttäminen suunnitteluprosessiin nähdään oleellisena oikeanlaisen ja hyväksytyn lopputuloksen saavuttamiseksi. Näkökulma perustuu idealle, jonka mukaan suunniteltava systeemi on jo olemassa, ja suunnittelijan tulee vain antaa sille rakenne ja käsittellistää se. Niinpä tämän näkökulman yhteydessä suunnitteluratkaisun nähdään virtaavan analyytikosta ympäristön suuntaan. Toista näkökulmaa voidaan nimittää myös *kyberneettiseksi* näkökulmaksi. (ibid.)

The Data Analysis Approach -näkökulmassa analyytikko tarkastelee ensimmäisen näkökulman tavoin ympäristöä, mutta tällä kertaa kyse on holistisemman tarkastelun sijaan *määrittämään pyrkivästä läpikäynnistä*, jossa erotellaan ja nimetään kokonaisuuksia, ominaisuuksia ja suhteita. *The Participative Approach* -näkökulmaa muistuttaen tässäkin näkökulmassa suunniteltava systeemi oletetaan jo ennalta olemassaolevaksi, ja suunnittelijan tehtävä on kognitiiviseen ajatteluun perustuen kuvailla se syntyvän datan ja järjestelmän toiminnan osalta. Näkökulmaa voidaan nimittää myös *rationaaliseksi* näkökulmaksi. (ibid.)

Episkopoun ja Wood-Harperin (1986, 224) ehdottama viitekehys, auttaa valitsemaan, mikä tarjolla olevista työskentelynäkökulmista olisi paras valinta kulloistakin tilannetta ajatellen. Viitekehys koostuu viidestä tehtäväalueesta:

1. probleeman omistajien (käyttäjä(t)) arviointi,
2. ongelman sisällön systeemin (ongelma, sen omistaja(t) ja näiden ympäristö) arviointi,
3. ongelman ratkaisijan (hän tai he jotka pyrkivät käymään käsiksi ongelmaan) arviointi,
4. tarjolla olevien ongelmanratkaisunäkökulmien luokitteleminen ja arviointi taustaideologian, konkreettisten työkalujen, tiedonhankinta- ja -esitystapojen sekä työvoima- ja ajankäyttö-tarpeiden suhteen ja
5. probleeman ratkaisijan toimintaympäristönä olevan ongelmanratkaisusysteemin muodostaminen.

Kuten edellisessä luvussa tuli esiin, painottavat Wernick ja Hall (2004) käytettyjen työkalujen vaikutusta prosessin tuloksiin ja käyttäjänsä ajattelutapaan. Kääntäen ilmaistuna työkalujen käyttöönotto voidaan nähdä tietyn paradigman vaikutuksien konkreettisena ilmauksena.

Vaikka Kuhnin kuvaamien tieteenalan kehitysvaiheiden tietyissä tilanteissa saattaakin olla useampi alalla samanaikaisesti vaikuttava paradigma, on paradigman vaikutus yksittäisen kannattajansa maailmankuvaan niin kaikenkattava, ettei kahden eri paradigman suora vertailu toisiaan vasten ole Kuhnin (1994) mukaan lainkaan mahdollista. Paradigmojen perustan muodostavat kokonaan toisistaan eriävät olettamusten joukot ja erilliset tai eri tavoin määritellyt maailmaa kuvailevat termit. Niinpä vertailu edellyttäisi molempiin vertailukohteisiin nähden ulkopuolista kolmatta tarkastelupistettä, josta käsin voitaisiin muodostaa vertailtavien paradigmojen käyttämien käsitteiden *käännökset*. (Kuhn 1970, Wernickin & Hallin 2004 mukaan.)

Wernick ja Hall (2004, 239) luonnehtivat ohjelmistosuunnittelijoiden käyttämiä työkaluja eräänlaisiksi kognitiivisiksi filttereiksi, jotka vaikuttavat siihen miten suunnittelijat ymmärtävät maailmaa: miten he hahmottavat käsillä olevaa tilannetta, ratkaistavana olevaa

ongelmaa sekä yksittäiseen tilanteeseen soveltuvia ratkaisuja. Kuten esimerkiksi Episkopoun ja Wood-Harperin (1986) työskentelynäkökulman valitsemisessa auttavaa viitekehyksestä koskeva ehdotus antaa ymmärtää, konkreettista suunnittelutyötä ja työkaluvalintoja tulisi edeltää työskentelyn lähestymistapojen vertailu. Lähestymistapojen välillä pitäisi tämän johdosta pystyä tekemään tietoisia, perusteltuja valintoja *sovellusalakohtaisiin piirteisiin* liittyen. Ehkäpä Episkopoun ja Wood-Harperin ehdottama viitekehys pyrkiikin juuri tarjoamaan Kuhnin edellyttämän käsitteellisen tarkkailupisteen, josta käsin paradigmavertailu tulee mahdolliseksi. Toinen vaihtoehto on, ettei Episkopoun ja Wood-Harperin käsittelemiä työskentelyn lähestymistapoja tulisi käsittää tieteenalaa määrittävän paradigman laajuisiksi vaihtoehdoiksi, vaan ainoastaan saman paradigman alle mahtuviksi lähestymistavoiksi.

Wernick ja Hall (2004, 236) huomauttavat, ettei ohjelmistotekniikan työkalujen vaikutusta käyttäjänsä näkökulmaan ole aikaisemmin tutkittu. Artikkeli on julkaistu kuusitoista vuotta sitten, mutta Tahsirin, Halen ja Niblockin (2017) tutkimusta lukuunottamatta aiheeseen liittyvä tutkimus vaikuttaa edelleen vähäiseltä. Gürelin (2018) opinnäytetyön tyyppiset, tietyn suunnittelutyökalun käytön etuja tarkastelevat tutkimukset liittyvät kyllä aihepiiriin, mutta tämän tyyppisissä tutkimuksissa näkökulma siirtyy epistemologiasta hyötynäkökulmaan ja tutkimuksen taustalla vaikuttava motivaatio vaikuttaakin liittyvän ennen kaikkea tietyn työkalun tarjoamien mahdollisten etujen todentamiseen.

3 Suunnittelijan käsitys työskentelyn tavoitteesta ja mahdollisista ratkaisuista sekä erilaisten käsitysten kanssa eteneminen

Tämä luku käsittelee opinnäytetyön empiirisenä osana toteutettua tutkimusta. Tutkimuksen puitteissa suoritettiin pienimuotoinen ohjelmistotekninen suunnittelu- ja toteutusprojekti, joka esitellään luvussa 3.1. Suunnittelu-ja toteutusprosessin mittaan tuotettua tutkimusaineistoa esitellään luvussa 3.2. Aineistolle suoritetun analyysin toimintatapoja kuvaillaan luvussa 3.3, ja luku 3.4 kuvailee analyysitulosten muodostamisprosessia. Luvussa 3.5 tarkastellaan havaintoja, jotka nousivat esiin aineistoa tutkimuksen neljän muuttujan avulla läpi käydessä. Näitä havaintoja voidaan pitää tutkimuksen välituloksina.

3.1 Hyperspektrikameran lineaariskannerin ohjainohjelmiston suunnittelu- ja toteutusprojekti

Opinnäytetyön empiiriseen osaan kuuluva ohjelmistotekninen suunnittelu- ja toteutusprojekti toteutettiin kesä-heinä-elokuun aikana 2020. Työskentely tapahtui koronapandemian edellyttämissä etätyöolosuhteissa, tutkijan kotoa käsin. Työn alkuunpanevan tehtävänannon mukaan projektissa oli määrä toteuttaa hyperspektrikameran lineaariskannerin ohjainohjelmisto, jolla ohjataan lineaariskannerin kolmea askelmoottoria koordinaatiston x-, y- ja z-ulottuvuuksia vastaavasti. Tehtävänannon mukaisesti ohjainohjelmistoa tulisi pystyä käskyttämään pythonkielisesti komentoriviltä käsin.

Työskentelyn perustaksi tutkija sai yliopistolta käyttöönsä arkullisen sekalaisia elektroniikkakomponentteja, mukaanlukien Arduino-aloituspakkauksen, askelmoottoreita ja moottoreiden ohjaimia. Ensimmäiset moottorien liikuttelukokeilut suoritettiin 9V:n neppariparistoa käyttäen, mutta projektin kuluessa laitteistoa täydennettiin vielä uusilla, lopullisen järjestelmän moottoreita vastaavilla moottoreilla ja laboratoriokäyttöön tarkoitetulla virtalähteellä. Ohjelmointityö tapahtui tutkijan omaa kannettavaa tietokonetta käyttäen.

Etätyöskentelyvaatimus näkyi projektin toteutetuksessa esimerkiksi siten, ettei kehitettävää ohjelmistoa suunniteltu tai testattu kehitystyön ensimmäisen syklin aikana varsinaista, yliopiston tiloissa sijaitsevaa laitteistokokoonpanoa käyttäen. Tähän kokoonpanoon olisi kuulunut mm. lineaariskannerin kiskot niiden päissä olevine rajakytkimineen sekä hyperspektrikamera. Tämän sijaan ohjelmistoa lähdettiin toteuttamaan itse kootun laitteiston avulla. Tämä laitteisto koostui kolmesta, varsinaisen laiitteiston moottoreita vastaavista askelmoottoreista, niiden ajureista, Arduino UNO -mikrokontrollerista ja ulkoisesta virtalähteestä. Lisäksi työn kuluessa tähän ydinkokonaisuuteen liitettiin myös väliaikainen komponentti, painonappiryhmä, jonka tehtävä oli demonstroida varsinaisen laitteiston lineaariskannerin rajakytkimien toimintaa.

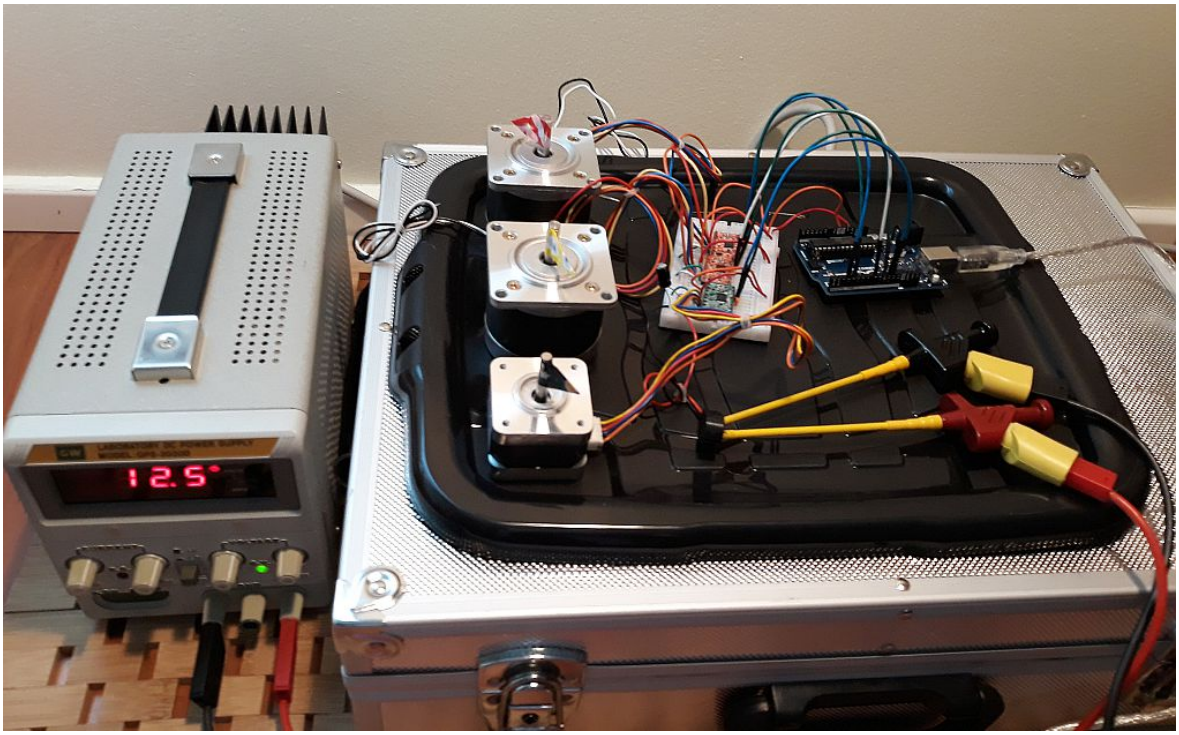
Laitteistonäkökulmasta työskentelyn lähtökohtana olivat siis jo valitut askelmoottorit, ja niihin liittyvien muiden laitteisto- ja ohjelmointitarpeiden selvittäminen kuului käsillä olevan suunnittelu- ja toteutusprojektin tehtäviin. Se, että päätettiin lähteä rakentamaan nimenomaan Arduino-mikroprosessorilla toimivaa kokonaisuutta, päätettiin yhteistuumin projektin alussa. Tässä vaiheessa vaihtoehtona harkittiin myös Wago-automaatiokomponentteja ja CoDeSys-perustaisen ohjelmiston rakentamista. Näistä vaihtoehdoista päätettiin kuitenkin luopua, sillä Arduino-kokoonpanoon näytti löytyvän enemmän niin erilaisia tutoriaaleja kuin ohjelmointikirjastojakin.

Projektin tuloksena rakennetun prototyypin toteutus koostui lopulta seuraavanlaisesta laitteistojen, ohjelmistojen ja ohjelmointikirjastojen muodostamasta kokoonpanosta:

- iso askelmoottori Nema23, 2 kpl (x- ja y-suunnista vastaavat moottorit)
- askelmoottori Nema17, 1 kpl (z-suunnasta vastaava moottori)
- Pololu A4988 -ohjaimet, 3 kpl
- painonapit, 6kpl
- Arduino UNO -mikroprosessori, 1 kpl
- Arduino IDE
- StandardFirmataPlus-sketch
- pyFirmata-kirjasto
- laboratoriokäyttöön tarkoitettu virtalähde GW GPS-3030D

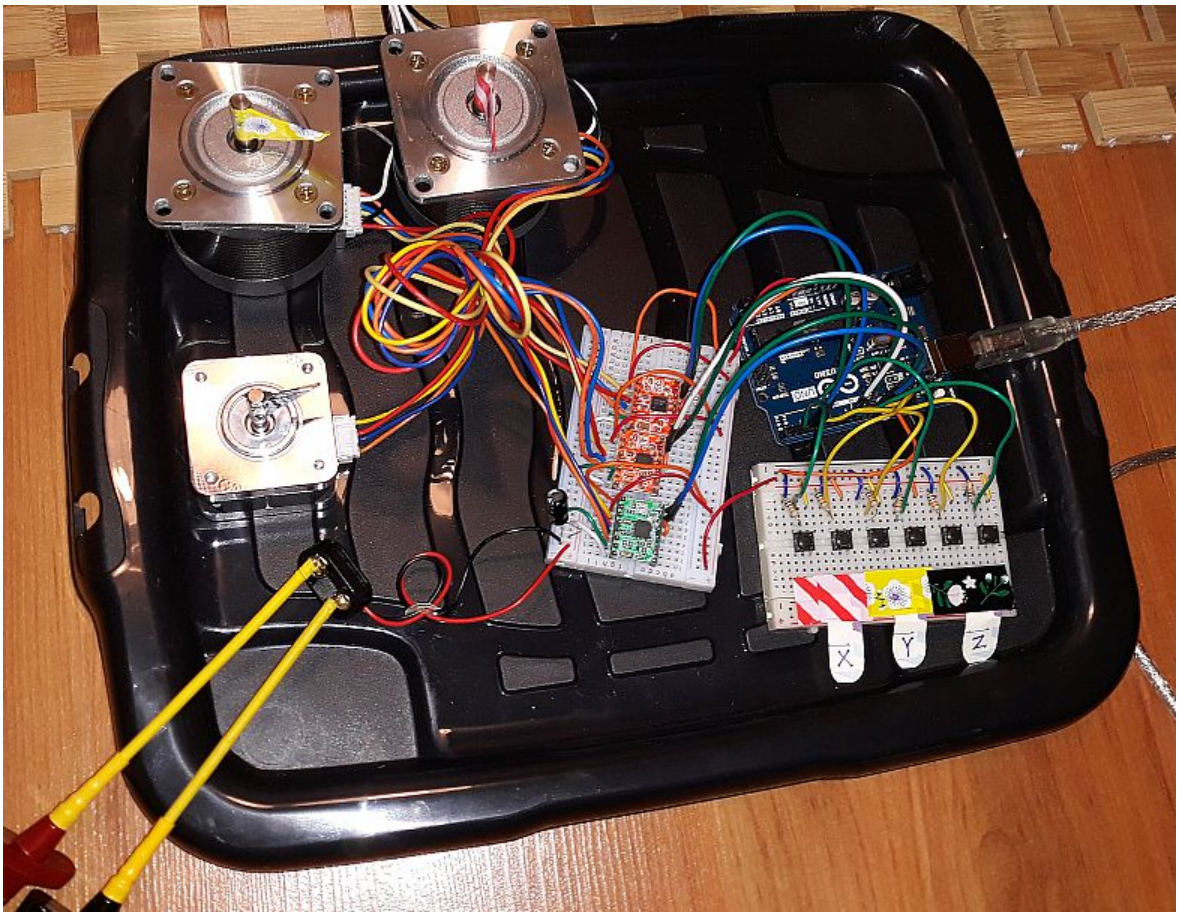
Lisäksi kehitysympäristöön kuuluivat:

- Spyder 4 -IDE,
- Python 3.7.6
- ThinkPad -kannettava tietokone
- Windows 10 -käyttöjärjestelmä



Kuvio 2. Prototyypin laitteistoa: virtalähde, askelmoottorit ajureineen sekä Arduino UNO.

Kuvio 2:ssa näemme prototyypin laitteistosta (vasemmalta) askelmoottoreiden oman ulkoisen virtalähteen, askelmoottorit, ajurit sekä Arduino UNO:n. Arduino UNOsta lähtevä USB-kaapeli yhdistää mikroprosessorin tietokoneeseen, joka toimii prototyyppi-vaiheessa myös mikroprosessorin virtalähteenä.



Kuvio 3. Prototyypin elektroniikkakomponenttien kokonaisuus, jossa painonapit korvaavat todellisen laitteiston rajakytkimiä.

Kuvio 3:ssa näkyy, miten prototyypin laitteistoa on täydennetty painonappiryhmällä, jolla korvattiin varsinaisen lineaariskannerin x-, y- ja z-suuntaisten kiskojen päissä olevia rajakytkimiä. Työskentelyn helpottamiseksi eri liikuttelusuunnista vastaavat moottorit on erotettu toisistaan erivärisillä, teipistä taitelluilla lipuilla, ja kuhunkin suuntaan liittyvät painonapit (”rajakytkimet”) on merkitty koekytkentälevyn reunaan moottorin teippiä vastaavilla teipinpaloilla.

3.2 Tutkimusaineisto

Tutkimuksen empiirisen osion suunnittelu- ja toteutusprojektin mittaan koottiin muistiinpanoaineisto, johon kirjattiin ylös toistuvasti niitä ajatuksia, joilla suunnittelu- ja toteutustyön tavoitetta ja tuon tavoitteen mahdollisia toteuttamistapoja työskentelyn eri

vaiheissa hahmotettiin. Lisäksi muistiinpanot sisältävät kuvausta työskentelyn etenemisestä sekä niistä seikoista, joita työn mittaan täytyi selvittää, jotka tuottivat epävarmuutta tai jotka estivät työskentelyn etenemistä.

Työskentelyn alkuvaiheessa muistiinpanojen kirjoittaminen päätettiin toteuttaa kahtena erillisenä dokumenttina. Toinen dokumenteista keskittyi suunnittelu- ja toteutustyön käsittelemiseen edelläkuvatulla tavalla. Toisessa muistiinpanodokumentissa kuvattiin laitteistoon ja työkaluihin liittyvän tutustumisen, tiedonhaun ja käytön opettelun mittaan muuttuvia käsityksiä työn tavoiteesta ja ratkaisumahdollisuuksista. Varsinkin työskentelyn alkuvaiheessa suunnittelutyö ja laitteistoon ja työkaluihin tutustuminen näyttäytyivät selkeämmin erillisinä aihioina. Tiedon karttuessa nämä kaksi juonnetta alkoivat kuitenkin lähentyä toisiaan. Työskentelyn loppuvaiheessa (esimerkiksi ohjelmointikirjastoja koskevat) työkaluvalinnat näyttäytyivät ennemminkin erilaisina vaihtoehtoisina ratkaisumahdollisuuksina.

3.3 Aineiston analysointi

Muistiinpanoaineistolle suoritettiin teoriasidonnainen sisällönanalyysi, jossa aineiston toistuvat teemat pyrittiin tunnistamaan useiden läpikäyntien avulla. Teemat koodattiin muistiinpanoaineistoon atlas.ti-sovellusta apuna käyttäen. Aineiston koodauksessa päädyttiin käyttämään kaikkiaan 29 asiasanaa, jotka olivat aakkosjärjestyksessä seuraavat:

algoritmi
arduino
ei totta
ennakkokäsitys
epävarmuus
esimerkkitoteutus
IDE
käyttöohjeet
kieli
kokeileminen

kulttuurinen konteksti
laitteisto
meta
muuntaminen
oivallus
oppiminen
outoa käytöstä
paradigma
python
ratkaisu
ratkaisu ”olemassa ennalta”
ratkaisuvaihtoehto
suunnitteluongelman havaitseminen
tavoite
tunnistamispyrkimys
toteuttaminen
työkalujen valinta
vaatimukset
valmiit ratkaisut

Useimpien asiasanojen merkitys lienee ilmeinen, mutta joidenkin käytön tarkoitus vaatinee lisätietoja. Esimerkiksi epävarmuuden tunteminen pitkin kehitysprosessia lienee suhteellisen yleistä ja hyödyllistä varsinkin, jos projektin aihepiiri sisältää runsaasti tekijälle ennalta uusia tekijöitä. ’Epävarmuus’-koodin merkitseminen lähes jokaisen muistiinpanojen kirjauksen yhteyteen ei kuitenkaan olisi ollut tarkoituksenmukaista. Sisällönanalyysiä tehtäessä ’epävarmuus’-asiasana liitettiinkin vain niiden otteiden yhteyteen, missä oli selkeästi pysähdytty kriittisesti uudelleenarvioimaan ja kyseenalaistamaan prosessissa jo tehtyä ohjelmistoteknistä ratkaisua. Asiasana ’outoa käytöstä’ puolestaan liittyi askelmoottoreiden toiminnassa ajoittain esiintyneisiin odottamattomiin piirteisiin, näiden ilmiöiden havainnointiin ja niiden syiden selvittelyyn liittyneisiin muistiinpano-otteisiin. ’Paradigma’-asiasanalla merkittiin otteita, joissa

käsiteltiin *ohjelmointiparadigmaan* liittyviä seikkoja. ’Toteuttaminen’-asiasanalla merkittiin ainoastaan ohjelmointityön tekemistä käsitelleisiin otteisiin silloin, kun jokin muu tilannetta tarkemmin erittelevä asiasana ei tullut kyseeseen.

Käytännössä aineiston koodauksessa ei automaattisesti liitetty esimerkiksi avainsanaa ’python’ tai ’toteuttaminen’ jokaiseen muistiinpanojen otteeseen, jossa kyseinen sana esiintyy, sillä analyysin tavoitteena ei ollut tarkastella tiettyjen sanojen esiintyvyyttä. Sen sijaan tutkijan termille tarkoittaman merkityksen tuli oleellisesti esiintyä avainsanalla koodatussa otteessa. Tämä tarkoittaa luonnollisesti myös sitä, ettei avainsanan tarvitse esiintyä sillä merkityssä muistiinpanojen otteessa (joka saattoi sisältää tekstiaineiston lisäksi myös kuvia).

Näin ollen analyysia toteuttaessa kaikki asiasanat eivät olleet samanarvoisia. Osa asiasanoista auttoi kokoamaan eri muuttujiin liittyvät otteet, osa (esimerkiksi ’ratkaisu’, ’tavoite’, suunnitteluongelman havaitseminen’, ’oivallus’, ’paradigma’) taas käsitteli suunnittelu- ja kehitysprosessiin liittyviä ilmiöitä tai muita tutkimuskysymysten kannalta keskeisiä ilmiöitä. Analyysin huomioita muodostettaessa tarkasteltiinkin esimerkiksi sitä, mitkä muut asiasanat esiintyivät tietyn asiasanan sisältävien otteiden muodostamissa muistiinpano-otteiden ryhmissä. Lisäksi muistiinpanojen otteita luettiin valittujen asiasanojen avulla koottuina kokonaisuuksina. Pyrkimys oli tällöin havaita, mikäli jokin muistiinpanoissa ilmaistu seikka näyttäytyisi uudessa valossa.

Teoriasidonnaisen sisällönanalyysin jälkeen työskentelyn etenemistä erilaisten näkökulmien kanssa tarkasteltiin fenomenologisen tutkimuksen otteella. Tässä käytettiin hyväksi sekä muistiinpanoaineistoa että tutkijan omaa kokemusta. Kyse on eräänalaisesta kokemuksen lähiluvusta, jossa pyrkimys on nähdä tarkasteltava kohde ’sellaisenaan’, ilman kohteeseen kuulumattomien seikkojen vaikutusta.

3.4 Analyysin tulosten muodostaminen

Analyysin tuloksia muodostettaessa aineistoa tarkasteltiin ensin neljän ennalta valitun muuttujan suhteen. Muuttujat koostuvat prosessissa käytetyistä päätyökaluista - Arduino-mikrokontrollerista oheislaitteistoineen, ohjelmointiympäristöistä (IDE:ista),

ohjelmointikielistä ja vaatimusmäärittelystä. Näiden lisäksi prosessin mittaan suunnittelutyökaluina päätettiin käyttää myös UML- ja vuokaavioita, jotka käsiteltiin osana vaatimusmäärittely-muuttujaa.

Analyysissä jokaisen muuttujan suhteen tarkasteltiin, esiintyykö niiden yhteydessä työskentelyn tavoitteen tai mahdollisten suunnittelu- tai toteutusratkaisujen kuvauksia. Lisäksi pidettiin silmällä myös asiasanoja ’uuden suunnitteluongelman havaitseminen’ ja ’oivallus’, sillä näillä merkittiin suunnittelu- ja toteutusprosessin kannalta merkittäviä käännekohtia tai edistymistä. Kunkin muuttujan suhteen pyrittiin tunnistamaan ne tutkimuskysymysten kannalta merkitykselliset suunnitteluprosessin tapahtumat (esimerkiksi uuden suunnitteluongelman havaitseminen, oivallus) ja aihepiirit (esimerkiksi ohjelmointiparadigma, algoritmi), jotka nousivat keskeisinä esiin yksittäistä muuttujaa koskevien muistiinpano-otteiden yhteydessä. Pelkkä avainsanojen esiintymien tarkastelu ei näin ollen vielä riittänyt, vaan asiasanan asemaa muuttujan yhteydessä täytyi tarkastella asiasanalla merkittyä muistiinpano-otetta tarkastelemalla. Jos yksittäisen muuttujan yhteydessä esiintyi suunnitteluratkaisuihin liittyviä havaintoja, tarkasteltiin ratkaisujen suhteen muita kyseisen muuttujan yhteydessä esiintyviä avainsanoja ja näiden merkitsemiä muistiinpanojen otteita sen hahmottamiseksi, millä tavoin ratkaisuja pyrittiin muodostamaan tai mihin liittyviä ratkaisuja muuttujan yhteydessä tuotettiin. Avainsanojen esiintymismäärät tai niiden suhteelliset osuudet tietyn muutujan yhteydessä eivät olleet tarkastelun kohteena.

Muuttujittain läpivedetyn tarkastelun jälkeen aineistosta koottiin ’tavoite’-asiasanaan liittyvät löydökset. Löydöksistä pyrittiin erottelemaan erillisiksi tunnistettavat tavoitekuvaukset (ts. mahdollista toistoa sisältävät otteet liittettiin yhteen). Tämän jälkeen tavoitekuvauksille hahmoteltiin niiden kronologinen syntyjärjestys.

Yksittäisen tavoitekuvauksen yhteydessä tarkasteltiin myös niihin liittyen esitettyjä ’ratkaisu’-asiasanalla merkittyjä otteita. Lisäksi tavoitekuvausten osalta tarkasteltiin, mitkä neljästä muuttujasta esiintyivät kyseisen kuvauksen yhteydessä. Lopuksi tavoitekuvauksista pyrittiin tunnistamaan useissa kuvauksissa toistuvat, yleistettävissä

olevat sisällöt, mikä mahdollisti tavoitekuvausten jakamisen niiden yleistä näkökulmaa luonnehtiviin kategorioihin.

3.5 Välitulos: työkalujen rooli suunnittelu- ja toteutusprosessissa

Tässä luvussa tarkastellaan välituloksia, joiden avulla hahmotetaan, mitä suunnittelu- ja toteutusprosessiin liittyviä teemoja aineistosta nousee esiin, kun aineistoa tarkastellaan eri muuttujien suhteen. Muuttujina toimivat työskentelylle etukäteen nimetyt työkalut: vaatimusmäärittely (sisältäen sekä määrittelyprosessin että tässä prosessissa tuotetun dokumentin), IDE:t, eli python-ohjelmointiin tarkoitettu Spyder ja Arduino IDE, ohjelmointikielten (python ja arduino) ja Arduino-mikrokontrollerin oheislaitteistoineen. Aineistosta tunnistettuihin suunnittelutyön tavoitteen kuvauksiin ei vielä paneuduta tarkemmin, sillä niiden erittely on seuraavan luvun aihe.

Arduino-laitteiston kanssa työskentely oli avainasemassa työskentelyn tavoitteen hahmottamisessa prosessin alkuvaiheessa. Tällöin tavoite ymmärrettiin laitteistoon liittyvinä konkreettisina työvaiheina ja työskentelyedellytyksinä (esimerkiksi ”rakenna piirit”, ”suunnittele protokolla”). Kun Arduino-laitteistoon liittyvissä aineisto-otteissa käsiteltiin suunnittelua edistäviä ratkaisuja, keskeisiksi nousivat erityisesti tiedonhakutyö ja siihen liittyvä yleistasoisten kysymysten jakaminen pienempiin aiheisiin ja konkreettisempiin kysymyksiin, sopivien käyttöohjeiden löytäminen ja esimerkkitoteutusten soveltaminen. Laitteistokysymyksiin vastauksia etsiessä työskentelyä leimasi vaikutelma valmiiksi olemassaolevista ratkaisuista ja niiden etsimisestä.

Vaatimusmäärittelyihin liittyvien otteiden yhteydessä esiintyi monenlaisia huomioita, jotka liittyivät sekä työskentelyn tavoitteen hahmottamiseen että ratkaisuehdotuksiin, mutta myös suunnitteluongelman havaitsemiseen ja prosessin mittaan koettuihin oivalluksiin. Vaatimuksia käsiteltiin useaan otteeseen yhdessä toisten työkalujen kanssa – sekä työn tavoitteita ja ratkaisuja Arduino-laitteiston äärellä hahmoteltaessa että ohjelmointikieliin liittyvien suunnitteluratkaisujen yhteydessä. Suunnitteluprosessin alussa vaatimusmäärittelyä muodostettaessa ja työskentelyn myöhemmissä vaiheissa vaatimusdokumentin sisältöihin uudelleen palattaessa suunnittelijan työstämät aiheet

liittyivät esimerkiksi askelmoottoreiden käyttöön Arduino-laiteiston yhteydessä, algoritmin suunnitteluun, ohjelmointiparadigman valintaan sekä konkreettisempien ohjelmointiratkaisujen tekemiseen. Vaatimusmäärittelyt vaikuttivatkin liittyvän uuden tiedon esille tuomiseen ja yhdistelemiseen suunnittelutyön ulkoisessa tilassa (vrt. Tahsiri, Hale ja Niblock, 2017).

Aineiston tarkastelun perusteella IDE:eilla ei tässä suunnittelu- ja toteutusprosessissa vaikuttanut olevan merkittävää roolia työskentelyn tavoitteen hahmottamisessa, uusien suunnitteluongelmien havaitsemisessa tai oivallusten kokemisessa. Silloin kun muistiinpanot käsittelivät IDE:n käyttöön liittyvien suunnitteluratkaisujen hahmottelua, kyse oli valmiiden ohjelmistoteknisten ratkaisuiden käyttöönotosta. Arduino IDE:n yhteydessä Firmata-protokollan ja -kirjaston käyttö edellyttivät, että Arduino IDE:ssä sijaitsevasta valmiskoodien valikoimasta käyttöön valittaisiin Arduino-mikrokontrolleriin ladattava, projektin tarkoitukseen sopiva arduino-kielinen valmiskoodi, eli sketch. Myös pyFirmata-kirjaston käyttöönotto edellytti tämän toteuttavan *import*-lauseen lisäämistä kirjoitetun python-koodin alkuun. Kun jonkin valmiin ohjelmointiteknisen ratkaisun käytöstä oli tehty päätös, oli varsinainen käyttöönotto käytännössä pelkkä (tarvittaessa Arduino- tai python-tutoriaalien ohjeistuksen perusteella suoritettu) rutiinitoimenpide.

Ohjemointikielten kanssa työskennellessa suunnittelutyön tavoitetta sanallistettiin moneen kertaan uudestaan. Ohjelmointityötä käsittelevän muistiinpanoaineiston yhteydessä esiintyy myös runsaasti uusien suunnitteluongelmien havaitsemiseen viittaavia otteita. Myös valtaosa prosessin mittaan koetuista oivalluksista esiintyi ohjelmointikielten yhteydessä. Tästä huolimatta myös ohjelmointityön äärellä työskentelyssä korostuivat ratkaisujen *etsiminen* ja *toteuttaminen*. Valmiiksi olemassaolevaksi mielletyt ratkaisut eivät kuitenkaan näyttäytyneet yhtä konkreettisina tai ehdottomina kuin laitteiston yhteydessä hahmotetut ratkaisut. Esimerkiksi olio-ohjelmoinnin ohjelmointiparadigman soveltamiseen liittyvä luokkien suunnittelu ja Arduino-mikrokontrollerin kautta kokonaisuuteen kytkettyjen laitteiden käskyttäminen asettavat molemmat koodin kirjoittamista varten tiettyjä, ennalta määrättyjä tehtäviä, joiden yhtäaikainen toteuttaminen näyttäytyi ikään kuin päällekkäin asetettavien kehikkojen sovitteluna.

Ohjelmointiratkaisujen muodostamisessa tyypillistä oli *kokeilemalla työskentely*. Tämä tuli analyysissä selkeästi esiin, olipa kyseessä sitten arduinokielisen esimerkkikoodin perusteella pythonkielisten versioiden ohjelmointi tai moottoreiden liikkeessä esiintyneen odottamattoman käytöksen syiden selvittely. Usein ratkaisujen työstäminen johti uusien suunnitteluongelmien havaitsemiseen, mikä taas toisinaan johti myös työskentelyn tavoitteen täsmentymiseen tai sanallistamiseen uudesta näkökulmasta. Ohjelmointikielen ja -työn yhteydessä aineistossa kuvaillut oivallukset eivät niinkään liittyneet itse ohjelmointityöhön, vaan tilanteisiin, joissa jonkin laitteistoon liittyneen ongelman (esimerkiksi kytkentöihin liittyvän virheen tai ajurin ylikuumenemisen) ratkettua myös oman tai esimerkkikoodin toimintaan liittynyt epävarmuus hälventyi.

Työskentelyn alkuvaiheessa tehty vaatimusmäärrittelyjen kirjoittaminen eteni rinnakkaisena prosessina Arduino-laitteistoon (ja askelmoottoreihin) tutustumisen kanssa. Vaatimusten alustava aukikirjoittamisen ja laitteistoon liittyvien faktojen ja toimintaperiaatteiden selvitystyö näyttäytyvät muistiinpanoissa toisiinsa sulautuvina työskentelyn juonteina. Sen sijaan, kun vaatimusmäärittelyjen ääreen palattiin suunnitteluprosessin myöhemmissä vaiheissa, esiintyy ’vaatimukset’-muuttujan yhteydessä oivalluksia ja suunnitteluongelman havaitseminen, mikä osoittautuu tärkeäksi huomioksi. Vaatimusten äärellä koetut oivallukset liittyivät tavoitejärjestelmän määrittelyn kannalta keskeisten käsitteiden, ’käyttäjän’ ja ’käyttötavan’, merkityksen täsmentymiseen. Näillä täsmennyksillä oli konkreettisia vaikutuksia myös kehitettävän järjestelmän toteutukselle.

Vaatimusten pohjalta havaittu, uusi suunnitteluongelma liittyi ristiriitaan, jonka muodostivat vaatimuksissa esitetty tarve ylläpitää tietoa moottorin ottamista askelista ja tarjota samanaikaisesti käyttäjälle mahdollisuus määrätä, millaisella nopeudella moottoreita liikutetaan. Alkuselvitysten myötä muodostui käsitys, että liian nopea moottoreiden liikuttelu saattaisi johtaa todellisuudessa otettujen askelien ”menettämiseen”. Tästä ilmiöstä järjestelmä ei kuitenkaan projektissa toteutetussa muodossaan saisi tietoa Arduino-mikrokontrollerin avulla.

Kaiken kaikkiaan suunnitteluprosessissa Arduino-laitteiston rooli korostui työskentelyn alkuvaiheessa eräänlaisena porttina, edellytyksenä, jonka oikeanlaisen kokoamisen ja

ohjailun ymmärtämisen kautta työskentely saattoi vasta alkaa. Yhdessä askelmoottoreiden kanssa Arduino edusti konkreettisimmillaan kehitettävän järjestelmään sovellusalaa, josta monet ohjelmointitarpeet oli johdettavissa. Vaatimusmäärittely näyttäytyi työkaluna, joka oli läsnä prosessin alusta sen loppuun asti liimaten muita työkaluja prosessin jatkumoon ja pitäen prosessin liikkeessä. Ohjelmointikielet työkaluna olivat prosessissa aktiivisimmin käytössä, eikä kielen asema työkaluna aina ollut helposti erotettavissa sen asemasta osana työn lopputuotetta.

Sen sijaan IDE:ien rooli jäi havaintojen kannalta kokonaisuudessa varsin vähäiseksi, mikä saattoi johtua siitä, että järjestelmän luomisesta toteutettiin vasta prosessin ensimmäinen suunnittelu- ja toteutussykli, eikä prosessissa näin ollen koettu tarvetta IDE:ien tarjoamille erikoistuneemmille työkaluille. Koodeja testattiin läpi työskentelyn Arduino IDE:n ja -laitteiston avulla, sekä Spyder-ympäristössä IDE:n oman konsolin ja debuggerin sekä laitteiston avustuksella. IDE:jen osuus vaikutti kuitenkin tältäkin osin tämän projektin puitteissa läpinäkyvältä, sillä IDE-muuttujasta ei esiintynyt aineistossa tutkimuskysymysten kannalta merkittäviä havaintoja.

4 Tutkimuksen tulokset

Tässä luvussa käydään läpi tutkimuksen tulokset tutkimuskysymyksittäin. Luvussa 4.1 käsitellään ensimmäistä tutkimuskysymystä (TK1) koskevat tulokset. Luku 4.1.1 esittelee yksittäiset aineistosta tunnistetut tavoitekuvaukset ja niihin liittyvät ratkaisut. Luvussa 4.1.2 tavoitekuvaukset jaotellaan kategorioihin niitä yhdistävien näkökulmien perusteella. Näin päästään käsittelemään tavoitemuotoilujen osajoukkoja muodostavia piirteitä. Luvussa 4.1.3 tarkastellaan, millaisia tavoitekuvauksia eri työkalujen yhteydessä esiintyi. Luvussa 4.2 läpikäydään toista tutkimuskysymystä (TK2) koskevat tulokset, jotka kuvailevat sitä, miten eri näkökulmien välillä työskentely suunnittelu- ja toteutusprosessin mittaan eteni.

4.1 Eri työkalujen yhteydessä muodostetut tavoitteet ja ratkaisut

Ensimmäinen tutkimuskysymys (TK1) kuuluu: *Millaisia näkökulmia suunnittelu- ja toteutusprojektin mittaan käsiteltäviksi tulevat ohjelmointikielet, työkalut ja työskentelymetodit suunnittelijalle tarjoavat työskentelyn kohteena olevaan ongelmaan ja sen mahdollisiin ratkaisuihin?* Tutkimuskysymyksen vastaus muodostuu käsilläolevan luvun mittaan, sillä tutkimuskysymykseen vastatessa voidaan painottaa, joko kysymyksen ”*Millaisia näkökulmia*”- tai ”*ohjelmointikielet, työkalut ja työskentelymetodit suunnittelijalle tarjoavat*”-osaa.

4.1.1 Tavoitemuotoilut

Aineistosta tunnistettiin yhteensä neljätoista työskentelyn tavoitteesta esitettyä erilliseksi tunnistettavaa kuvausta. Suhteessa toisiinsa kuvaukset saattoivat olla luonteeltaan täsmentäviä tai tavoitetta kokonaan uudelleen eri näkökulmasta muotoilevia. Jatkossa kaikkiin tavoitteen sanallisiin kuvauksiin viitataan termeillä ’tavoitekuvaus’ tai ’tavoitemuotoilu’.

Taulukko 1. esittelee tavoitekuvaukset kronologisen esiintymisjärjestyksen perusteella numeroituina. Numerointi juoksee ensimmäisessä sarakkeessa taulukon vasemmassa

reunassa. Tavoitekuvaukset on pyritty nimeämään kuvauksen sisältöä luonnehtivasti taulukon keskimmäisessä sarakkeessa. Viimeisessä sarakkeessa on esitetty kunkin tavoitekuvauksen yhteydestä aineistosta poimittu ote, joka joissain tapauksissa sisältää koko tavoitemuotoilun sellaisenaan ja muulloin tarjoaa siitä jonkin kokonaisuutta edustavan näytteen. Tavoitekuvaus 9:n kohdalla esitetty lainaus sisältää tekstiotteen, joka on peräisin toimeksiantajan kanssa käydystä sähköpostikeskustelusta, mistä se suunnittelutyön aikana kopioitiin ja kommentoitiin (suunnittelijan kommentit kursiivissa) osaksi suunnittelumuistiinpanoja.

Nro	**Tavoitekuvauksen nimi**	**Ote aineistosta**
1	Kokonaisjärjestelmän ohjelmistotekniset osat ja kommunikaatio	”Rakennetaan tarvittavat piirit. Suunnitellaan protokolla, jonka mukaan pc:llä toimiva koodi ja arduino-sketch kommunikoivat. Ohjelmoidaan pc-ohjelmisto ja Arduino-sketch, jotka toimivat yhteen suunnitellun protokollan mukaisesti.”
2	Tehtävän kokonaiskuva ja järjestelmän reunaehdot	”[Suunnitelmaan tulee muodostaa] kokonaiskuva joka sisältää ohjelmistotarpeiden lisäksi myös laitteistokomponentit.”
3	Moottorinäkökulma	”Tulee käyttää kahta isoa ja yhtä pientä moottoria”
4	Valmiiden ratkaisujen käyttö toteutuksessa	”Etsin mahdollisimman toimivat valmiit ratkaisut vaatimusten täyttämiseksi”
5	Käytettäväksi tarkoitettu ohjelmisto	”Suunnittelen ohjaimen, jolle on mahdollista määrittää lineaarisiirtimien [*eli lineaariskannerin*] liikuttelulle matkan pituus (ja myöhemmin myös

Nro	Tavoitekuvauksen nimi	Ote aineistosta
		askelmoottorin liikkumiselle nopeus) ja joka on tietoinen siirtimien kulloinkin liikkumista askelista, joiden avulla se pitää lukua siirtimien sijainnista”
6	Käyttäjä ja käyttö uudelleentarkasteltuina	”input(prompt) ei muuten ole ollenkaan oikea funktio [suunniteltavan ohjelmiston tarpeisiin]... sillä eihän tämän ohjelmiston ’käyttäjä’ ole se ’perinteinen käyttäjä’”
7	Koodin muunnos	”kirjoitan python-koodin, joka on pythonlaisittain toteutettu versio arduinokoodista”
8	Käyttö yksinkertaisemmin	”[Ymmärsin ensin] että ohjaimen tulisi käyttäjän asettamien alkuasetusten jälkeen toteuttaa vaatimuksissa kuvattu viipaleittain etenevä liikerata. Uudestaan vaatimuksiin palatessa tulinkin käsitykseen, että käyttäjän on voitava itse toteuttaa se ohjelmiston tarjoamia funktioita käyttäen”
9	Kuvaustilanteen algoritmi	”Eli käyttäjä tekee jotain seuraavan kaltaista: 1. Skannerin intialisaatio [initialisointi] (aloituskohdan määritys*) liikutetaan [skannerin kelkkaa] kytkimeltä tiettyyn [aloitus]paikkaan, sijaintitieto (x, y, z) ylös* 2. Käynnistää kameran [...]”
10	Järjestelmä työkalusettinä	”Teen joukon funktioita, työkalusetin, jota

Nro	Tavoitekuvauksen nimi	Ote aineistosta
		käyttäen käyttäjä voi ohjailla kameraa kolmiulotteisesti.”
11	Kameran kolmiulotteisen siirtelyn työkalut	”Toteutetaan ohjainohjelmisto, joka käytännössä sisältää paletillisen työkaluja, joilla käyttäjä voi luoda haluamansa kaltaisia kuvaussessioita, ohjata kameraa hallitusti 3:n akselin suhteen ja saada mahdollisimman luotettavaa tietoa kameran sijainnista.”
12	Harha-askelia: 'skriptit'	”Toteutettavana tältä osin on ainoastaan algoritmin kuvauksen pohjalta toteutettava skriptitiedosto”
13	Python-ohjelmoinnin konventiot	”Tulee kirjoittaa ohjelmisto, jossa myös lähdekoodi on käyttäjälle helposti luettavaa, jotta sitä voidaan helposti hyödyntää käyttäjän omiin tarpeisiin.”
14	Kohti seuraavaa kehityssykliä	”[… suunnittelutyön] 2.kierroksella koko ohjelmistoa pitäisi täydentää output-tyyppisillä komponenteillä, joiden kautta saataisiin mm. varoitus, jos z-suunnassa tullaan liian lähelle kuvattavaa kohdetta (käyttötapaus 10) [...]”

Taulukko 1. Prosessin mittaan muodostetut 14 tavoitekuvausta

Suunnittelutyön alkaessa ensimmäinen tavoittemuotoilu (1) ilmensi pyrkimystä hahmottaa yleisellä tasolla, millaisen järjestelmän parissa olisi tarkoitus työskennellä. Ratkaisut koostuivat Arduino-mikrokontrollerin kytkentöihin, kommunikaatio-protokolliin ja ohjelmointiin liittyviin tutoriaaleihin ja esimerkkikoodeihin tutustumisesta. Näiden lähteiden kautta muodostettiin yleiskuva siitä, millä tavoin konkreettiset kytkennät näkyvät

arduinokielisessä koodissa. Tässä vaiheessa aloitettiin myös järjestelmän vaatimusmäärittelyjen kirjoittaminen, jonka toteuttamiseksi toimeksiantajan kanssa kommunikoitiin kirjallisesti ja videopalaverin välityksellä.

Toisessa tavoitemuotoilussa (2) huomio siirtyi laitteisto- ja ohjelmistoteknisistä kysymyksistä työskentelyn metatasoon. Tavoitemuotoilussa pyrittiin sen tunnistamiseen, mitkä käsilläolevan tehtävänannon elementit tulevat valmiiksi määrättyinä ja miten näiden elementtien pohjalta voisi tai täytyisi edetä, jotta kuva tavoitejärjestelmästä täsmentyisi entisestään ja jotta tähän mennessä hahmottuvia piirteitä (esim. protokolla ja sen mukainen kommunikointi) voitaisiin käytännössä toteuttaa. Ratkaisu pyrittiin löytämään tarpeeseen soveltuvan UML-kaavion laatimisesta. Vaatimusmäärittelyn tueksi päätettiin tehdä käyttöönottokaavio.

Seuraavaksi työskentelyssä seurasi askelmoottoreiden erityispiirteisiin, käyttöön ja kytkentöihin perehtymisen vaihe, jonka aikana keskeiseksi tavoiteeksi (3) muodostui askelmoottoreiden toiminnan ymmärtäminen ja käytettäväksi määrättyjen moottoreiden tyyppien (bipolaarinen vai unipolaarinen) ja mallien tunnistaminen. Tämän lisäksi tavoitteeseen sisältyi pyrkimys saada moottorit liikkumaan koodin avulla. Taulukko 1:ssä tavoitekuvaus 3:n yhteydessä esitetty ote muistiinpanoista on hieman irrallisen kuuloinen laitteistotasoinen huomio, joka kuitenkin kiteyttää tilanteen, josta tiedonhaussa lähdettiin liikkeelle: suunnittelija sai käteensä moottorit, joiden yksityiskohtaisempien tietojen selvittely täytyi aloittaa pelkän sarjanumeron perusteella. Ratkaisu syntyi askelmoottoreiden ominaisuuksia, Arduino-kytkentöjä ja käyttöä koskevien sekalaisten tiedonjyvien yhdistelemisen ja käytännönkokeilujen lopputuloksena.

Työskentelyn seuraavassa vaiheessa suunnittelijan huomio siirtyi laitteisto-kysymyksistä ohjelmointia ja ohjelmointikieltä koskeviin seikkoihin. Tavoitteessa (4) ja sen mahdollisissa ratkaisuissa korostui ensimmäistä kertaa ohjelmistokoodin näkökulmasta ajatus etsimisestä ja löytämisestä, eli ideasta, jonka mukaan ratkaisu tai joukko ratkaisuja on jo valmiiksi olemassa ja sopiva ratkaisu täytyisi vain osata tunnistaa ja valita. Alkukartoitusten myötä tietoa oli ehtinyt kertyä sen verran, että suunnittelija oli tullut tietoiseksi valmiiksi olemassa olevien ratkaisujen suuresta määrästä. Niinpä tavoitekuvaus

muotoutui nyt työskentelyperiaatteen muotoon, minkä mukaan toteutuksessa tulee hyödyntää mahdollisuuksien mukaan tavoitetta palvelevia valmiita ratkaisuja. Ratkaisujen, eli protokollan, valmiskoodin ja ohjelmointikirjastojen valinnan myötä kehitettävän järjestelmän kuva alkaa hahmottua toteutettavissa olevana suunnitteluongelmana. Tässä yhteydessä työkalujen (Firmata, FirmataStandardPlus, pyFIrmata) valinnat näyttäytyvät ensimmäisen kerran ennalta määrätyn ominaisuuden sijaan suunnitteluratkaisuina.

Työkaluvalintojen jälkeen työskentelyn tavoitteenmuotoilussa (5) palattiin järjestelmälle annettuihin vaatimuksiin, ja nyt tavoitejärjestelmää luonnehdittiin sen toiminnan ja käytön näkökulmista. Ratkaisun arveltiin löytyvän olio-ohjelmoinnin näkökulmaa soveltavasta ohjelmoinnista. Koodin ensiversion ohjelmoinnin edetessä suunnittelija tarkasteli tarjolla olevia käyttäjän syötteen lukemiseen liittyviä metodeita, ja tuli tehneeksi tärkeän huomion: vaikka vaatimuksien yhteydessä on keskusteltu komentorivikäyttöliittymän tekemisestä, ei tämä välttämättä tähän projektiin liittyvän loppukäyttäjän näkökulmasta tarkoita samaa kuin mitä se saattaisi tarkoittaa ”perinteisen” käyttäjän tapauksessa. Toimeksiantajan kanssa käyty keskustelu vahvisti huomion oikeaksi, minkä jälkeen myös ’ohjelmiston käyttö’ näyttäytyi suunnittelijalle uudesta näkökulmasta (tavoite 6). Tavoitteeseen liittyvät ratkaisut olivat yksinkertaisia: Komentorivisyötettä ei tarvitse lukea koodista. Vaatimuksiin liittyvien skriptien toteuttaminen jätetään myöhemmäksi.

Seitsemännen tavoitemuotoilun (7) yhteydessä tavoitetta lähestyttiin koodin näkökulmasta, ja mukaan tuli kokeiluluontoisesti idea arduino-kielisen sketchin pythonkielisen 'muunnoksen' tai 'käännöksen' tekemisestä. Ohjaimesta ei ollut olemassa arduinokielistä versiota. Sen sijaan ajatus liittyi tarjolla olevien arduino-kielisten esimerkkikoodien mahdolliseen hyödyntämiseen python-kielisen koodin kirjoittamistyössä. Tässä vaiheessa suunnittelijalle oli selvää, että kehitettävässä järjestelmässä Arduino odottaa Firmata-valmiskoodin ansiosta pc-ohjelmiston käskyjä, joten ohjelmoitavaksi tulisi ainoastaan pc-ohjelmisto. Ratkaisuna luonnosteltiin luokkasuunnitelman ensimmäinen versio, missä yhtenä tavoitteena oli hahmotella arduino-sketcheissä esiintyvää ”määrittelyt - void loop()” -rakennetta vastaava, luokka-perustainen toteutus.

Ratkaisu perustui kolmeen luokkaan – Moottori, Moottorit ja Kuvaus – missä Moottori-luokka vastaa yksittäisen askelmoottorin tiedoista ja sijaintilaskurista, Moottorit-luokka vastaa mikrokontrollerin kanssa kommunikoinnista ja kolmen moottorin liikuttamisesta ja Kuvaus-luokka vastaa käyttäjän käskyjen välittämisestä moottorit-oliolle ja yksittäiseen kuvaustilanteeseen liittyvien arvojen, esimerkiksi kuvaukselle määritetyn aloituspaikan, muistamisesta. Kuvaus-luokan metodeita suunniteltiin vaatimuksissa kuvattua käyttötilanteen esimerkkikuvausta hyödyntäen. Esimerkkikuvauksessa hahmoteltiin kameran liikuttelun liikesarja, jota varten ohjaimelle tulisi antaa liikesarjan määrittämisessä tarvittavat tiedot (kuvattavan jakson aloituskohta, lopetuskohta ja kuvausten välisten siirtymien leveys) ja jonka ohjain annettujen tietojen pohjalta toteuttaisi.

Myös suunniteltuja luokkia ohjelmoitaessa palattiin jälleen tarkastelemaan vaatimusmäärittelmässä esitettyä järjestelmän käytön kuvausta, minkä seurauksena suunnittelija kyseenalaisti nyt vuorostaan käsityksensä ohjaimen toivotusta toiminnasta. Keskustelu toimeksiantajan kanssa vahvisti tälläkin kertaa heränneet epäilyt ja tavoitekuvauksesta syntyi jälleen uusi versio (8): ohjaimen tulisi tarjota riittävän yleisluontoisia toimintoja, jotta käyttäjä voisi itse määrittää kuvaustilanteessa haluamansa liikesarjat. Ratkaisuna koodia täydennettiin eteenpäin proseduraalisempaa näkökulmaa soveltaen. Uudessa versiossa käyttäjä pystyi nyt määrittämään, mitä moottoria haluataan liikuttaa ja millaisen matkan moottorin halutaan liikkuvan.

Aikaisemmin aloitettua koodiaihiota ei kuitenkaan kokonaan poistettu, vaan käyttäjälle päätettiin tarjota myös mahdollisuus asettaa kuvausten liikeratoja määrittäviä arvoja muistiin ja suorittaa tämän pohjalta kerralla laajempia liikesarjoja. Jotta ideasta olisi helpompi keskustella, suunnitteludokumenttien yhteyteen päätettiin luoda algoritmin kuvaus (tavoitekuvaus 9), jonka määrittelyssä hyödynnettiin erityisesti yhtä vaatimusmäärittelyjen yhteyteen liitettyä toimeksiantajalta saatua käyttötilanteen kirjallista kuvausta. Ajatuksena oli myös, että algoritmin pohjalta voitaisiin suoraan ohjelmoida ja toisaalta myös tarkistaa, että kaikki tarvittavat toiminnot ovat tulleet toteutetuiksi. Ratkaisuna oli kirjoittaa algoritmin vaiheet auki ja kuvata vaiheiden suhteet vuokaaviona.

Jatkossa työskentelyn tavoite hahmottui yhä konkreettisempana näkymänä koodiin, sen rakenteeseen ja käyttöön (tavoitekuvaus 10). Ratkaisussa löysivät paikkansa sekä olioperustainen että proseduraalinen ajattelu. Aliohjelmiin perustuva ajattelu auttoi hahmottamaan käyttäjän ja käyttötilanteen tarpeita, kun taas olioihin perustuvat rakenteet turvasivat laitteiston käyttämien tietojen säilytyksen ja niiden hallitut muutokset.

Vaatimuksissa esiintyi kaksi piirrettä, joiden havaittiin yhdessä muodostavan käytännön toteutukseen liittyvän ristiriidan. Piirteet olivat käyttäjälähtöisesti määriteltävissä oleva moottoreiden kuvauksen aikainen liikuttelunopeus ja moottoreiden liikutteleman kelkan sijaintitiedon seuranta. Ristiriita perustuu tutoriaaleista löytyneeseen tietoon, jonka mukaan liian nopeassa tasaisessa liikkeessä askelmoottorit ”hukkaavat askelia” esimerkiksi liikkeelle lähtiessään, pysähtyessään tai moottorille liian raskasta kuormaa siirtäessään.

Ongelman ratkaisemiseksi päätettiin soveltaa oliomaista lähestymistapaa, jossa moottoreille suunniteltiin alustavasti toteutettavaksi kaksi ”liikuttelumodea”. Toinen mode mahdollistaisi lineaariskannerin kelkan joutuisammankin liikuttelun silloin, kun kamera ei kuvaa. Tällöin voitaisiin hyödyntää kiihdytys- ja jarrutus-ominaisuuksia otettujen askelten menettämisen välttämiseksi. Toisessa liikuttelumodessa ideana oli mahdollistaa kuvausten vaatima tasainen liikke siten, että käyttäjä voisi asettaa liikuttelulle haluamansa nopeuden. Tämä nopeus voitaisiin kuitenkin antaa vain sellaisten arvojen rajoissa, joiden puitteissa moottoreiden vielä voidaan odottaa pystyvän liikuttamaan kameraa ilman kiihdytys- ja jarrutusominaisuuksia siten ettei askeleita menetetä. Toiminnot ohjelmoitiin vain alustavasti, sillä lopullinen toteutus edellyttää varsinaisella laitteistolla toteutettua testausta.

Ohjelmointityön etenemisen tuloksena syntyi seuraava tavoitekuvauksen versio (11), jossa ohjelmistoa kuvaillaan sovellusalan ja käyttäjän tarpeiden näkökulmasta. Ratkaisu koostui edellisten vaiheiden myötä muokatusta luokkasuunnitelmasta ja sen pohjalta toteutetusta koodista.

Työskentelyn loppusuoralla tarkastelun kohteeksi nousivat jälleen komentoriviskriptit, joiden suhteen suunnittelijalle oli jäänyt epämääräinen mielikuva osana ohjaimen suunnittelutyötä toteutettavaksi tulevista valmiista komentoriviskripteistä. Niinpä skriptit sisältyivät hetkellisesti myös työskentelyn tavoitteisiin (tavoittekuvaus 12). Kun asiasta

keskusteltiin toimeksiantajan kanssa, kävi selväksi, ettei ohjaimen toteutukseen tältä osin tulisi toteutettavaa.

Kun sekä ’käyttäjää’ että ’käyttöä’ koskevat, tähän projektiin liittyvät erityispiirteet olivat prosessin mittaan täsmentyneet, ja ohjaimen koodista oli saatu kokoon sen ensiversio, muodostui lopulta tavoitekuvaus (13), jossa järjestelmän koodin luettavuuden vaatimus nousi yhdeksi järjestelmän käytettävyyden edellytykseksi. Ratkaisuna tälle vaatimukselle koodin kirjoitusasun osalta pyrittiin varmistamaan, että se vastaa python-koodille muodostuneita yleisiä kirjoituskonventioita.

Lopulta prosessissa tuotettu suunnitteludokumentaatio ja koodin ensimmäinen versio tarjosivat ainekset seuraavan kehityssyklin työtehtävien hahmottelulle (tavoitekuvaus 14). Ratkaisuna oli tarkastella toteutettua suunnitelmaa ja ohjelmaa kriittisesti niiden keskeisimmät puutteet ja keskeneräisyydet tunnistaen. Tämän perusteella ehdotettiin erilaisia jatkotoimenpiteitä ja näihin liittyvää priorisointia.

4.1.2 Tavoitemuotoilujen kategorisointi

Edellä kuvaillut tavoitekuvaukset voidaan jakaa kuvauksessa ilmennetyn näkökulman suhteen kolmeen kategoriaan. Alla esitetyssä kategoria-luettelossa esiintyviä suuraakkosia käytetään jatkossa kyseisen kategorian tunnuksena. Kategorioiden jäljessä esiintyvät numerot viittaavat edellisessä luvussa (luku 4.1.1) esiteltyihin tavoitekuvauksiin (ks. Taulukko 1.). Näkökulman ovat:

- T: suunnittelutyön konkreettisten tehtävien tai toimintatapojen, sekä laitekysymysten näkökulma (1, 2, 3, 4, 14),
- J: työskentelyn kohteena olevan järjestelmän, käyttäjän ja käytön näkökulma (5, 6, 8, 9, 10, 11, 12, 13) ja
- K: kirjoitettavan koodin ja ohjelmointityön näkökulma (7, 9, 13).

Työskentelyn ja laitekysymysten näkökulmista muodostetuissa tavoitekuvauksissa (T) hahmotellaan työskentelyn reunaehtoja, edellytyksiä ja etenemisperiaatteita sekä laitteistosta nousevia järjestelmän suunnitteluun vaikuttavia tekijöitä. Työskentelyn

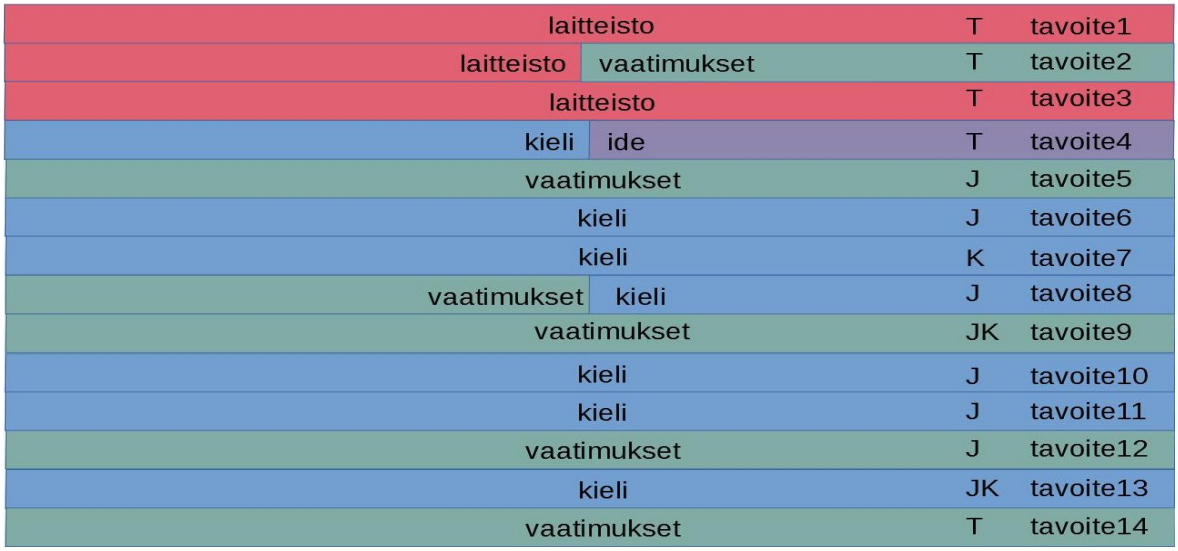
ensimmäisten vaiheiden mittaan muodostui neljä peräkkäistä työskentelyä ja laitetason kysymyksiä käsittelevää tavoitemuotoilua. Lisäksi tähän näkökulmaan palattiin prosessin viimeisessä tavoitemuotoilussa, jossa aikaansaatua tuotosta tarkasteltiin kriittisesti kehitystyön seuraavan syklin tehtäviä hahmotellen.

Tunnistetuista tavoitemuotoiluista suurin osa, kahdeksan kappaletta, liittyi suoraan työskentelyn päämääränä olevaan järjestelmään, käyttäjään tai järjestelmän käyttöön. Tähän kategoriaan (J) kuuluvien tavoitemuotoilujen sisältöjen välillä oli myös selkeä keskinäinen jatkumo. Tavoitteen uudelleenmuotoiluissa tavoitetta pyrittiin joko laajentamaan esimerkiksi siten että ensin pelkkää teknistä järjestelmää luonnehtinut tavoitemuotoilu huomioisi jatkossa myös järjestelmän kohderyhmän tai käyttötilanteen erityispiirteitä. Lisäksi uudelleenmuotoiluissa saatettiin myös palata uudelleen täsmentämään jotain edellä mainituista elementeistä. Esimerkin täsmennyksestä tarjoaa tavoitemuotoilu 6:n yhteydessä muistiinpanoissa esiintynyt huomio: ”*sillä eihän tämän ohjelmiston ’käyttäjä’ ole se ’perinteinen käyttäjä’*”. Toinen esimerkki löytyy tavoitemuotoilusta 8, jonka yhteydessä aikaisempi käsitys ohjelman toivotusta toiminnasta ja käyttötavasta korjattiin geneerisempään muotoon.

Kirjoitettavaan koodiin ja ohjelmointityöhön keskittyvä näkökulma esiintyi ainoastaan kolmessa tavoittekuvauksessa, mikä on hieman yllättävää, sillä välituloksissa ohjelmointikielten asema työkaluna vaikutti hyvin keskeiseltä. Tähän näkökulmaan kuuluvista tavoitekuvauksista kaksi (7 ja 9) muodostettiin työskentelyjakson puolivälin tienoilla. Kolmas koodiin ja ohjelmiontityöhön liittyvä tavoitekuvaus muodostettiin kokonaisprosessin toiseksi viimeisenä tavoitekuvauksena (13). Tämä liittyi koodin ulkoasun ja järjestelmän käytettävyyden välille prosessin loppuvaiheessa tunnistetuihin erityisvaatimuksiin.

Kuvio 4:ssa havainnollistetaan muuttujina tarkasteltujen työkalujen (laitteisto, IDE:t, ohjelmointikielet ja vaatimusmäärittely) käyttöä suhteessa aineistosta tunnistettuihin tavoitekuvauksiin (oikeassa reunassa tavoite 1–14). Kuvion yläreuna merkitsee työskentelyn alkuvaihetta, joten kronologisesti prosessin etenemistä voi hahmottaa kuviota

ylhäältä alaspäin lukien. Kategorioihin, joihin tavoitekuvaukset edellä luokiteltiin, viitataan kuviossa tunnuksin T (työskentely), J (järjestelmä) ja K (koodi).



Kuvio 4. Työkalujen käytön ja tavoitemuotoilujen esiintymisen suhde

Vaikka näkökulma tavoitemuotoiluissa vaihtelee huomattavastikin, ei yksikään tavoitteen uudelleenmuotoiluista edusta itsenäistä, toisista muotoiluista irralliseksi jäävää näkökulmaa. Sen sijaan algoritmin muotoiluun liittyvä tavoitemuotoilu (9) sekä ohjelmointikielen luettavuuteen liittynyt tavoitekuvaus (13) sisälsivät sekä lopputuotteeseen ja käyttäjään (kategoria J) että ohjelmointityöhön (kategoria K) liittyviä huomioita. Toinen Kuvio 2:sta havaittavissa oleva seikka on se, ettei tietyn työkalun käyttö rinnastu näissä työvaiheissa muodostetuissa tavoitekuvauksissa käytettyyn tavoitteen tarkastelunäkökulmaan. Myöskään siirtymä tietyn työkalun käytöstä toiseen ei välttämättä tarkoita sitä, että tavoitekuvauksen näkökulma muuttuisi yhtälailla. Näin olisi ollut, vaikka tavoitekuvausten ensimmäisestä aihepiiriltään laajasta kategoriasta (T) laitekysymykset olisi eristetty suunnittelutehtävistä omaksi kategoriakseen.

4.1.3 Eri muuttujien yhteydessä esiintyvät tavoitemuotoilut ja ratkaisuihin liittyvä tieto

Koko työskentelyprosessi käynnistyi alustavalla tutustumisella laitteistoon. Arduino-mikrokontrollerin ja sen oheislaitteistojen kanssa toimiessa työskentelyn tavoite jäsentyi konkreettisina rakentelu-, suunnittelu- ja ohjelmointitehtävinä. Tavoitekuvausten myötä

pyrittiin saavuttamaan yleiskuva siitä, millaisten vaiheiden kautta työskentelyn täytyisi edetä ja millaisista laitekomponenteista ja ohjelmistoista valmis järjestelmä tulisi lopulta koostumaan. Laitteiston äärellä työskennellessä huomio kohdistui (varsinkin prosessin alussa) työskentelyn kohteena olevan järjestelmän edellyttämän laitteiston kokonaiskuvaan. Sittemmin täytyi keskittyä myös yksityiskohtiin, joiden ratkominen muodosti muuta työskentelyä hidastavia pullonkauloja. Laitteistonäkökulmasta tavoite näyttäytyi myös toimeksiantajan (tavoitekuvaus 3) tai laitteistosuunnittelijan (tavoitekuvaus 1) valmiiksi määräämänä kokonaisuutena. Koska erityisesti laitteiden kytkentöihin ja sähkötekniikkaan liittyvät tiedontarpeet liittyivät suunnittelijan asiantuntemuksen ulkopuolisiin aihepiireihin, eikä näihin aiheisiin ollut tämän projektin yhteydessä tarvetta perehtyä enempää kuin tavoitejärjestelmän kehitystyö edellyttäisi, ratkaisuja muodostettaessa etsittiin tietoa (’totuutta’), jonka voisi löytää *inhimillisen tekijän*, siis aiheasiantuntijan, *antamana*.

IDE-muuttujan noustessa aineistossa esiin pian projektin ensimmäisellä kolmanneksella työskentelyn tavoitteen hahmottuminen liittyi valmiisiin ohjelmistoteknisiin ratkaisuihin ja siihen, missä määrin näitä päätettiin työn toteutuksessa käyttää. Kyse oli siis ensisijassa työskentelyä ohjaavan toimintaperiaatteen muodostamisesta, vaikkakin tällä periaatteella toimiessa (valmisratkaisuja mukaan valitessa) ratkaisut olivat järjestelmää itseään määrittäviä suunnitteluratkaisuja. IDE:n kanssa työskentely ei tapahtunut muihin työkaluihin nähden itsenäisesti vaan aina yhdessä ohjelmointikielen kanssa – Arduino IDE:n tapauksessa arduino-kielen ja Spyderin tapauksessa pythonin yhteydessä. Myös IDE:ien yhteydessä korostui valmiiden ratkaisujen etsiminen. Useiden mahdollisten ratkaisujen tilanteessa ratkaisuvaihtoehtoja arvioitiin käsitellen ’totuutena’ tietoa, joka oli luonteeltaan *inhimillisen tekijän antamaa tietoa*, ts. tarkastellun ohjelmistokirjaston toteuttaneiden ohjelmistosuunnittelijoiden kirjoittamaa dokumentaatiota.

Ohjelmointikielet olivat työskentelyn mittaan työkaluista suurimman osan aikaa käytössä. Prosessin alkuvaiheen jälkeen niitä käytettiin lähes koko ajan ja miltei prosessin loppuun asti. On kiinnostavaa havaita, miten monet ohjelmointityön ja koodin äärellä muodostetut tavoitekuvaukset (6, 8, 10, 11 ja 13) tarkastelevat tavoitejärjestelmää, eivätkä niinkään vain koodin tasoisia tavoitteita. Ohjelmointikielten yhteydessä esiintyvissä tavoitekuvauksissa

järjestelmää pyrittiin pääosin tarkastelemaan kokonaisuutena, ei yksittäisiin ominaisuuksiin keskittyen. Varsinkin tavoitekuvaus 11 nostaa esiin loppukäyttäjän ja käyttötilanteen asettamat tarpeet.

Ohjelmointikieli-muuttujan yhteydessä ohjelmointikielen ja ohjelmoinnin näkökulmaan (K) osittain tai kokonaan sitoutuvia tavoitekuvauksia esiintyi vain kaksi kappaletta (7 ja 13). Näistäkin jälkimmäinen (13) keskittyy oleellisesti myös kehitettävän järjestelmän näkökulmaan, sillä se käsittelee järjestelmän käytettävyyteen liittyviä tarpeita. Pelkästään kielelliseen näkökulmaan sitoutuva tavoitekuvauksen (7) muodostaminen käynnistää muistiinpanoissa sopivan ohjelmointiparadigman valintaan liittyvät pohdinnat. Tässä vaiheessa suunnittelutyössä pohdittiin, kannattaisiko arduinosta pythoniin siirtymisen yhteydessä siirtyä myös arduino-sketcheille ominaisesta proseduraalisesta lähestymistavasta esimerkiksi olionäkökulman soveltamiseen, ja millaisia muutoksia oliototeutus arduino-esimerkkeihin verraten tarkoittaisi (jos minkäänlaisia). Myöhemmin näiden pohdintojen ilmentämä ohjelmistoparadigmojen vastakkainasettelu loiventui toteamukseen, jonka mukaan tavoitejärjestelmän koodi voisi hyvin sisältää sekä olio- että proseduraalis-tyyppisiä piirteitä.

Koska ohjelmointikielen yhteydessä ratkaisuja muodostettiin ennen kaikkea esimerkkikoodien hyödyntämiseen ja omiin ohjelmointikokeiluihin perustuen, luotettiin ratkaisuissa tietoon, jonka totuudellisuus perustui *kokemukseen*. Esimerkkikoodien hyödyntämisen lähtökohtana toimi anekdootinomaiseen kokemukseen luottaminen, idea siitä, että mikäli kyseinen koodi oli toiminut jonkun muun käytössä, se voisi toimia käsilläolevassakin tapauksessa. Ohjelmointikokeilut tuottivat suunnittelijan omaan kokemukseen perustuvaa tietoa, jonka totuudellisuuden lähtökohtana toimi kokeilujen toistettavuus: Jos vain kokeilu oli suunniteltu oikein (esimerkiksi toimivana pidetty koodi olisi ajettavaa kaikissa kyseiselle ohjelmointikielelle soveltuvissa ympäristöissä), ratkaisua voidaan pitää validina.

Vaatimusmäärittely-muuttujan yhteydessä tarkasteltiin sekä vaatimusmäärittelyjen kirjoittamista työskentelyn alkuvaiheessa että prosessin mittaan tapahtuneita toistuvia palaamisia vaatimusdokumentin äärelle. Tavoitekuvaukset 2 ja 5 (ks. Taulukko 1.) ovat

peräisin vaatimusmäärittelyjen kirjoitustyön ajalta. Loput vaatimusmäärittely-muuttujaan liittyvistä tavoitekuvauksista on muodostettu tilanteissa, joissa vaatimusmääritelmien ääreen on palattu jonkin tietyn yksityiskohdan tarkistamiseksi.

Vaatimusmäärittelyt-muuttujan yhteydessä muodostettiin useita kehitystyön kohteena olevaa järjestelmää ja sen käyttöä hahmottelevia tavoitekuvauksia (5, 8, 9 ja 12). Kuten ohjelmointikielet-muuttujan yhteydessä, myös vaatimusmäärittelyjen yhteydessä esiintyvissä tavoitekuvauksissa järjestelmää lähestyttiin useammin sen kokonaiskuvan, ennemmin kuin yksityiskohtien, kautta. Viivaskanneri-tyyppisen kuvaustapahtuman algoritmin määrittämiseen liittyvässä tavoitemuotoilussa (9) näkökulma liittyi sekä järjestelmän kehittämiseen että ohjelmointiin, ja myöhemmin vaatimusdokumenttiin liitetty algoritmin kuvaus toimikin liiman tavoin järjestelmän suunnittelun ja ohjelmointityön välisenä siltana. Toisaalta vaatimusmäärittelyjen äärellä muodostettiin myös tavoitekuvauksia (2 ja 14), joissa huomio kohdistui suunnittelutyön itsensä edellyttämiin toimenpiteisiin ja joilla pyrittiin tehostamaan työskentelyn etenemistä.

Vaatimusmäärittelyjen yhteydessä muodostetut suunnitteluratkaisut eivät muiden edellä käsiteltyjen muuttujien yhteydessä esiintyneiden ratkaisujen lailla perustuneet selkeästi tiettyyn tai tiettyihin epistemologian totuus-statuksiin. Ratkaisut saattoivat muodostua osittain tutoriaalien tarjoamiin tietoihin (eli inhimillisen tekijän antamaan ’totuuteen’), toimeksiantajan kanssa käytyihin keskusteluihin (eli jälleen inhimillisen tekijän antamaan ’totuuteen’) ja näiden perusteella tuotettuihin *tulkintoihin*. Lisäksi ratkaisujen taustalla vaikuttivat myös suunnittelijan itsenäisemmin jostain vaatimusten osasta (kuten kohdekäyttäjästä tai käyttötilanteesta) luomat tulkinnat. Algoritmin suunnittelussa toiminnan perustana toimi työskentelyprotokolla, joka oli opittu ohjelmistotekniikan perusopetuksen yhteydessä, ja siten ratkaisussa (jonka mukaan algoritmin suunnittelu tässä vaiheessa ylipäätään oli tarpeen) luotettiin ’totuuteen’, joka oli opettaja-statusta kantavalta asiantuntijalta annettuna saatu.

Olleellista on, että vaatimusmäärittelyjen yhteydessä ratkaisujen ei välttämättä lainkaan oletettu (kaikiltaosin) perustuvan ’totuuksiin’ vaan nimenomaan mahdollisimman täsmällisesti ilmaistuihin ja huolellisesti perusteltuihin *tulkintoihin*. Ratkaisujen

soveltuvuutta tarkistettiin sekä testaamalla koodin osia, keskustelemalla toimeksiantajan kanssa että havainnollistamalla toteutetun järjestelmän toimintaa toimeksiantajalle näyttämällä siitä esimerkkejä videopalaverien välityksellä. Lisäksi myöhemmin toteutettavaksi suunniteltiin myös erillistä kokonaisjärjestelmän testausta.

4.2 Näkökulmat ja työskentelyn eteneminen

Toinen tutkimuskysymys (TK2) kuuluu: *Kuinka näkökulmien kanssa työskentely suunnittelu- ja toteutusprosessin myötä etenee?* Suunnittelu- ja toteutusrojektin alussa ei eksplisiittisesti sitouduttu mihinkään tiettyyn ohjelmistonkehitysmalliin. Muistiinpanoista on kuitenkin havaittavissa peräkkäisiä tai hieman lommitaisia perinteiseen vesiputousmalliin kuuluvia työvaiheita (esitutkimus, määrittely, suunnittelu, toteutus). Muistiinpanoissa esiintyvä ajatus *ensimmäisestä* ja *toisesta kehityssyklistä* voisivat viitata myös inkrementaaliseen ohjelmistonkehitysmalliin liittyviin taustaoletuksiin. Toisaalta ajatus kehityssykleistä saattoi olla ensisijaisesti aikarajoitteista seuraava vaikutelma: graduprojektiin liittyvän suunnittelu- ja toteutusprojektin työvaiheelle oli opintojen aikataulutuksessa varattu aikaa reilut kaksi kuukautta, ja ne ominaisuudet, jotka lopuksi jäivät toteuttamatta, ovat luonnollisesti oletetun ”seuraavan vaiheen” aihe.

Työskentelyssä käytettyjen työkalujen käyttö ei vaihdellut ennalta suunniteltujen työvaiheiden seurauksena, vaan vähän kerrassaan hahmottuvien tarpeiden mukaisesti. Tarpeet puolestaan hahmoteltiin tunnistamalla yleisen tason kysymyksissä pienempiä, konkreettisia kysymyksiä ja vastaamalla näihin kysymyksiin – tai käyttämällä jotain työkalua kysymykseen vastaamiseksi. Näin ollen eteneminen tapahtui rajaamalla määrätietoisesti tarkasteltavia seikkoja, konkretisoimalla työskentelyn kohdetta ja kohdistamalla huomio aina seuraavaan etenemisen kannalta ensisijaiseen haasteeseen. Kysymysketju saattoi edetä esimerkiksi seuraavasti (suluissa muuttuja-työkalu, jonka ääreen kysymys suunnittelijan ohjasi): Miten askelmoottorit kytketään Arduinoon (Arduino-laitteisto)? Miten askelmoottoreita voidaan ohjailla arduinokielisesti (ohjelmointikielet)? Miten Firmata-sketch otetaan käyttöön (IDE:t)? Miten pythonilla voidaan ohjelmoida askelmoottoreiden liikuttelu (ohjelmointikielet)? Tarvitaanko valmiita kirjastoja (ohjelmointikielet)?

Kuvio 4.:stä hahmottuvista työskentelyvaiheista (ts. työskentelyn jaksoista, joiden päätyttyä työskentelyn tavoitteesta muodostettiin uusi tavoitekuvaus) yli kaksi kolmasosaa muodostui ohjelmointikielten käytön ja vaatimusdokumentin tarkastelun vuorottelusta. Näiden, tai muidenkaan, työkalusta toiseen tapahtuvien siirtymien yhteydessä ei siirtymään itseensä liittyen koettu erityisiä haasteita. Myös muistiinpanoaineistossa siirtymät näyttäytyvät läpinäkyvinä, eli niihin ei liity ilmaisullisia ongelmia, ristiriitaisuuksia tai seuraamista häiritseviä kerronnallisia aukkoja. Sen sijaan ohjelmointityön ja vaatimusten tarkastelun vuorottelu tuntui pitävän prosessia liikkeessä aivan kuin vaihtelu itsessään olisi synnyttänyt jonkinlaista dynamiikkaa.

Muistiinpanojen puolivälissä mainitaan ohjelmistotekniikan opinnoista suunnittelijan mieleen jäänyt toimintaperiaate, jonka mukaan *järjestelmän suunnittelua tulee ajatella isosta kuvasta liikkeelle lähtien ja toteutusta pienestä kuvasta lähtien*. Tätä periaatetta noudattaen ohjelmoitaessa pyrittiin toteuttamaan vain pieniä kokonaisuuksia kerrallaan, kun taas vaatimusdokumentin äärellä tarkasteltiin yleisemmän tason kysymyksiä ja kehitettävän järjestelmän kokonaiskuvaa. Edellä mainitun periaatteen avulla voidaan ymmärtää, miksi ohjelmointi- ja vaatimus-työkalujen vaihtelulle tuntui olevan tarvetta, joskaan vaihtelun ei suoraan voida olettaa seuraavan mainitusta periaatteesta. Työskentelyä aineiston kautta tarkasteltaessa voidaan kuitenkin havaita, että ”ison” ja ”pienen kuvan” oli vaihdeltava säännöllisesti, jotta työskentely saattoi edetä ja samalla säilytettäisiin oikea suunta.

Toisaalta se, että mainittua periaatetta ylipäätään sovellettiin, selittänee osaltaan sitä, miksi myös koodin ja ohjelmointityön äärellä tuotetut tavoitekuvaukset hahmottuivat kokonaisjärjestelmän – eivätkä kielen tai ohjelmoinnin – näkökulmasta: Suuri kuva ei missään vaiheessa päässyt karkaamaan liian kauas, joten se pysyi mielessä silloinkin, kun toteutustyössä keskityttiin etenemään pienin askelin. Erityisesti algoritmin kuvaus toimi vaatimusmäärittelyjen pohjalta tehtävän järjestelmän yksityiskohtaisemman suunnittelun ja varsinaisen ohjelmointityön välisenä liittymäkohtana.

Lopulta työkalujen käytön vaihteluun liittyen voidaan vielä todeta, että analyysin myötä käy ilmi, miten työskentelyssä suunnittelu- ja toteutusvaiheet eivät tapahtuneet perinteisen

vesiputousmallin kuvaamina peräkkäisinä vaiheina, vaan toisteisina, enemmän tai vähemmän limittäisinä vaiheina.

Tutkimusaineisto ei anna syytä olettaa, että eri tarkasteltujen työkalujen käytön välillä siirtymiseen olisi liittynyt erilaisten näkökulmien, tai jopa erillisisten paradigmojen sovitteluun liittyviä haasteita. Sen sijaan havainnot tukevat tulkintaa, jonka mukaan työskentely tapahtui koko ajan saman paradigman sisällä. Myöskään esimerkiksi ohjelmointikieli-muuttujan yhteydessä tarkastellut ohjelmointiparadigmat eivät näyttäytyneet yhteismitattomina, erillisinä ratkaisuina, vaan vaihtoehtoisina toimintatapoina, joita saattoi myös soveltaa saman järjestelmän sisällä.

5 Pohdinta

Ensimmäistä tutkimuskysymystä koskevassa työhypoteesissa ehdotettiin, että jokainen neljästä tarkasteltavasta muuttujasta täsmentäisi ja konkretisoisi suunnittelijan näkemystä tavoitteesta, ja kaventaisi tämän myötä hänen käsitystään mahdollisista ratkaisuista. Työhypoteesin asettama olettamus perustui Wernickin ja Hallin (2004, 239) esittämään ajatukseen siitä, kuinka ohjelmistotekniikan työkalujen kautta aktualisoituvat paradigmaelementit filtteröivät suunnittelijan maailmankuvaa, hänen näkemystään käsilläolevasta tilanteesta ja suunnitteluongelmasta sekä sen mahdollisista ratkaisuista. Wernickin ja Hallin (2004) esittämät tulokset ohjelmistotekniikan kypsyysasteesta esiparadigmaattisena alana motivoivat myös työn tutkimuskysymysten muotoilua.

Tulosten perusteella ensimmäisen tutkimuskysymyksen yhteydessä esitetty työhypoteesi ei yksiselitteisesti pitänyt paikkaansa jokaisen muuttujan suhteen. IDE:jen vaikutus suunnittelijan ajatteluun ei toteutuneella työskentelyjaksolla ollut tavoitetta täsmentävä tai konkretisoiva, tai mahdollisten ratkaisujen määrää karsiva. Tosin kokemus olisi saattanut olla toisenlainen, mikäli tarkasteltava jakso olisi sisältänyt myös laajempaa ohjelmiston testausta. Nyt sekä arduino- että python-ohjelmoinnin IDE:t toimivat lähinnä ”vain” toteutustyön *mahdollistajina*.

Sen sijaan Arduino-laitteisto ja sen oheislaitteistot tarpeineen ja rajoituksineen asettivat tässä projektissa *konkreettisia täsmennyksiä ja vaatimuksia* sekä työskentelyn tavoitteelle että sen mahdollisille ratkaisuille. Käytännössä laitteistosta nousevat vaatimukset asettivat usein uusia työvaiheita tai ohjelmointityössä huomioitavia elementtejä. Suhde laitteistoon oli kuitenkin tässä projektissa melko yksioikoinen, sillä keskeiset laitevalinnat oli tehty ennen varsinaisen työskentelyn alkua. Toisenlaisessa ohjelmistotekniikan projektissa myös laitteistoa koskevat valinnat olisivat saattaneet olla osa suunnitteluratkaisuja.

Työkaluna ohjelmointikieli otti projektissa *muokattavan materiaalin* roolin. Kukin kieli toi projektiin mukanaan ominaisuuksiensa muodossa implisiittisesti sisältämänsä skaalan mahdollisia toteutustapoja. Koska ohjelmointikieli itsessään tarjoaa kielioppinsa muodossa valmiita tulkintoja, voidaan sen nähdä asettavan suunnittelijalle myös kognitiivisen linssin.

Tässä mielessä etsityt ratkaisut ovat myös valmiiksi olemassa olevia. Riippuukin merkittävästi suunnittelijan taustatiedoista ja ohjelmointitaidoista, missä määrin hän saattaa tehdä valintoja mahdollisten toteutustapojen välillä. Se, kuinka rajoittavana tietyn kielen asettama kognitiivinen linssi näyttäytyy, riippuu luonnollisesti myös kulloisenkin suunnitteluongelman tarpeista. Projektissa ohjelmointityö näyttäytyi yksittäisten koodia koskevien suunnitteluratkaisujen aktualisointina, toisin sanoen, muiden mahdollisten toteutustapojen poissulkemisena. Lopulta toteutetun koodin muodossa nimenomaan konkretisoitiin järjestelmäsuunnittelun yhteydessä tehdyt valinnat. Näin ollen erityisesti ohjelmointityön voidaan ajatella toimivan työhypoteesissa tarkoitetun täsmentävän ja konkretisoivan työskentelymenetelmän tavoin. Ohjelmointi oli myös se nimenomainen väylä, jonka kautta prosessissa hankittua uutta tietoa tuotiin esille ja sovellettiin suunnittelutyön ulkoisessa tilassa, eli työn tavoitteena olevan artefaktin tuottamisessa. Tämä havainto on samansuuntainen Tahsirin, Halen ja Niblockin (2017) suunnittelutyökalujen vaikutuksia koskevien tuloksien kanssa.

Tarkastelluista muuttujista vaatimusmäärittelyille hahmottuva rooli vastasi täsmällisimmin työhypoteesissa ehdotettua työkalujen mahdollista vaikutusta. Vaatimusmäärittelyt nimenomaisesti *täsmensivät* työskentelyn myötä tavoiteltua lopputulosta ja toisinaan myös sen mahdollisia ratkaisuja. Lisäksi vaatimusmäärittelyjen yhteydessä epämääräinen idea täytyi *konkretisoida luonnollisen kielen muotoon*. Vaatimusmäärittely-dokumentti toimikin eräänlaisena kehitysprosessin ajatustyön kiintotähtenä. Sen yhteydessä (kuten myös ohjelmointikielten yhteydessä) muodostettiin monipuolisesti eri näkökulmista työskentelyä hahmottavia tavoitekuvauksia. Huomion arvoista on, että nimenomaan vaatimusmäärittelyjen yhteydessä tehty *täsmennys*- ja *konkretisointityö* ei lähtökohtaisesti perustunut erityiselle epistemologisesti tunnistettavalle totuustyypille vaan tulkinnalle, joka perustui toimeksiantajan ja suunnittelijan väliseen kommunikaatioon.

Tulosten perusteella ohjelmistosuunnittelun työkaluista vaatimusmäärittelyjen ja ohjelmointikielen yhteydessä merkittävä rooli on tulkintatyöllä sekä tehtävän edellyttämällä olemassa olevan tiedon kokoamisella. Tämä tieto tulee löytää (esimerkiksi erilaisten valmiiden ratkaisujen olemassaolo) tai saada esiin (esimerkiksi käyttötilanteen tarpeet) ja liittää oikealla tavalla yhteen. Ogundare (2017) täsmentää, että

ohjelmistotekniikassa tuotetaan tietoa usein mallintamalla, mikä sekin sopii yhteen tässä esitetyn näkemyksen kanssa. Esimerkiksi ohjelmointi itsessään voidaan nähdä ongelmanratkaisupyrkimyksen ohjelmointikielisenä tulkintana.

Toiminnan perustana olevan tiedon totuudellisuutta perätessä havaitaan inhimillisen tekijän antaman tiedon ja kokemukseen perustuvan (oman kokemuksen tai anekdootin muotoon saatetun tiedon) keskeisyys (Holloway 1994;1995). Tämä on luonnollista, kun otetaan huomioon, että kyseinen tieto koskee nimenomaan ihmisen tekemää artefaktia – jonka perusteet palautuvat, kuten edellä esitettiin, eri tyyppisiin tulkintoihin. Tällaisesta artefaktista voidaan hankkia tietoa joko itse kokeilemalla (omaan kokemukseen perustuva tieto), tutustumalla artefaktille laadittuun dokumentaatioon (inhimillisen tekijän antama tieto, joka voi sisältää myös anekdootinomaista tietoa) tai kysymällä henkilöltä, jolla on kokemusta artefaktista (inhimillisen tekijän antama tieto).

Vaikka Hollowayn (1994) kritiikkiin vastaaminen ei sisälly tämän työn tavoitteisiin, tutkimuksessa tehdyt havainnot auttavat osaltaan ymmärtämään, miksi ohjelmistotekniikan teknisen juonteen – jos omaksutaan Lázaron ja Marcosin (2005) ehdottama ohjelmistotekniikan tutkimusaiheiden kahtiajako tieteellisiin ja teknisiin – tutkimusaiheiden yhteydessä esitetään vähäisesti empiirisiä todisteita tai loogisia perusteita.

Toisessa työhypotesissa ehdotettiin, että työkalun asettaman kognitiivisen linssin mahdollisen täsmentämisen tavan taustalta olisi tunnistettavissa jokin tietty ohjelmistotekniikan alalla vaikuttava paradigma. Myös tämä työhypoteesi perustui Wernickin ja Hallin (2004) esittämiin tuloksiin, joiden mukaan ohjelmistotekniikka ei vielä omaisi yhtä eheää paradigmaa, vaan useita kilpailevia paradigmoja.

Kun tulosten yhteydessä tarkasteltiin sitä, millaisia tavoitekuvauksia ja mahdollisia ratkaisuja suunnittelija oli muodostanut eri työkalujen käytön yhteydessä, ja miten eri näkökulmien välillä siirtyminen eteni, tulokset eivät tukeneet sellaista havaintoa, että jokin yksittäinen työkalu olisi ilmentänyt muista poikkeavaa paradigmaa. Tämä ei luonnollisestikaan tarkoita, etteikö työkalujen taustalla vaikuttaisi *minkäänlainen* ohjelmistotekniikan alan paradigma. Ennemminkin havaintoa on tulkittava siten, etteivät

käytetyt työkalut sattuneet edustamaan erillisiä ohjelmistotekniikan kokonaiskenttää jäsentäviä paradigmoja – mikäli sellaisia ohjelmistotekniikan kentällä tällä hetkellä on olemassa. Tulosten perusteella kilpailevien paradigmojen asetelmaa ei voida havaita, sillä työkalujen yhteydessä muodostetut tavoitekuvaukset ja niihin liittyvät mahdollisest ratkaisut eivät ilmentäneet sellaista *yhteismitattomuutta*, jolla Kuhn (1994, 18) kuvasi epäkypsän tieteenalan erillisten koulukuntien välisen kommunikaation mahdottomuutta. Sen sijaan tulokset tukevat Wernickin ja Hallin (2004) tuloksia siltä osin, että alan käytännönhajoituksessa voitaisiin jo tunnistaa yhtenäisen paradigman, ja siten myös *normaalitieteen* piirteitä.

Tämän luvun alussa muuttujina tarkastelluille työkaluille hahmotettiin erilaiset roolit. Oli konkreettisia täsmennyksiä ja rajoituksia asettava laitteisto, suunnittelijan muokattavaksi tarjoutuvaa materiaalia (ohjelmointikielet), materiaalin työstämisen mahdollistavat IDE:t ja ongelman ratkaisua sanalliseen muotoon ennen toteuttamista pakottava vaatimusmäärittely. *Yhdessä* tämä joukko työkaluja muodostaa kokonaisuuden, joka vaikuttaisi yhteismitalliselta ja yhden ja saman eheän maailmankuvan mukaiselta. Siten ne asettuvat muodostamaan perustaa, jonka pohjalta ohjelmistoteknisiä suunnitteluongelmia voidaan tunnistaa ja ratkaistaa. Tulosten perusteella ei kuitenkaan tule olettaa, etteikö ohjelmistotekniikan teknisen paradigman keskeisiin alkioihin kuuluisi paljon muutakin, kuin mitä tässä on käsitelty. Samaten, kuten Wernick ja Hall (2004, 240) huomauttavat, ohjelmistotekniikan kentällä ei vallitse yksimielisyyttä kehitysprosessin elementtien keskinäistä tärkeysjärjestystä koskien.

Tulosten kautta kuvatut paradigmapiirteet vaikuttavat yhteensopivilta Ogundaren (2017, 169) teknisen paradigman kuvauksen kanssa. Ogundaren (2017) ehdottaman ohjelmistotekniikan alan kahtiajakautuneen luonteen perusteella voitaisiinkin esittää kysymyksiä ohjelmistotekniikkaa epistemologian näkökulmasta tarkastelevaa jatkotutkimusta ajatellen. Mitä esimerkiksi voisivat olla sellaiset ohjelmistotekniset työkalut, joiden voitaisiin sanoa erityisesti ilmentävän ohjelmistotekniikan tieteellistä paradigmaa? Millaisia arvoja tai olettamuksia nämä työkalut ilmentävät?

Tekninen paradigma, joka tässä käsiteltyjen tulosten myötä hahmottuu, näyttäisi pitävän sisällään esimerkiksi ohjelmistokehityksen vesiputousmallin sisältämät vaiheet: esitutkimusksen, määrittelyn, suunnittelun ja toteutuksen. Myös malliin sisältyvät testauksen ja käyttöönoton vaiheet olivat projektissa implisiittisesti mukana (vaikka niiden toteutus jäi tulevaisuuteen, niitä varten tehtiin kuitenkin suunnitelmia). Sen sijaan vesiputousmallin esittämä työskentelyjärjestys ja tapa edetä yksittäisessä vaiheessa alusta loppuun ennen seuraavaan vaiheeseen siirtymistä eivät saaneet vahvistusta käytettyjen työkalujen kautta – työkalut itsessään eivät ohjaa juuri tällaiseen toimintaan vaan jättävät nämä valinnat kehittäjän itsensä päätettäviksi. Lisäksi työskentelyä ohjanneeseen paradigmaan voidaan katsoa kuuluvan esimerkiksi alalle valmistavan koulutuksen kautta jaetut opit algoritmien tarpeellisuudesta ja suunnittelusta sekä ”ison” ja ”pienen kuvan” soveltamisesta järjestelmäsuunnittelun ja teknisen toteutuksen yhteydessä.

Lopuksi tutkimuksen toteutuksen suhteen todettakoon, että ensimmäiseen tutkimuskysymykseen vastaamiseksi suoritettu teoriasidonnaisen sisällönanalyysin analyysimenetelmä soveltui suhteellisen hyvin erilaisten tavoitekuvausten ja ratkaisujen tarkasteluun. Toiseen tutkimuskysymykseen vastaamiseksi fenomenologinen lähestymistapa osoittautui haastavammaksi. Tämä johtui erityisesti siitä, että tarkastelun kohteena oleet *siirtymät* ja *työskentelyn eteneminen* olivat hyvin abstrakteja kohteita. Mikäli työkalujen yhteiskäytössä olisi esiintynyt ongelmia, tarkastelun kohde olisi luultavasti muuttunut helpommin lähestyttäväksi, sillä työskentelyn etenemisen *edistämiseen* olisi luultavasti liittynyt jonkinlaisia toimenpiteitä.

Jos ohjelmistotekniikan työkalujen vaikutusta käyttäjänsä ajatuksiin haluttaisiin jatkossa tutkia enemmän, kannattaisi tulevissa tutkimuksissa hyödyntää, tapaustutkimuksen lisäksi, laajempien kehitystiimien työskentelyn seuraamista esimerkiksi teemahaastattelujen avulla. Tällöin voitaisiin saada esiin yksittäisen työkalun käyttöön liittyviä, erilaisesta tieto-taidosta juontuvia yksilöiden välisiä eroja.

6 Yhteenveto

Tämä pro gradu -tutkielma tarkastelee ohjelmistotekniikan alalla vaikuttavia paradigmapiirteitä sellaisina, kuin ne tulevat esiin ohjelmistokehitystyössä käytettyjen suunnittelu- ja toteutustyökalujen kautta. Tutkielmassa ollaan kiinnostuneita ohjelmistotekniikan luonteesta tieteenfilosofian, erityisesti tieteen tieto-opin eli epistemologian, näkökulmasta. Tutkimuksen motiivi on peräisin Paul Wernickin ja Tracy Hallin (2004) artikkelista ”*Can Thomas Kuhn's paradigms help us understand software engineering?*”, jossa tutkijat esittävät, että ohjelmistotekniikan teoriankehityksen kentällä vallitseva tilanne vastaa kuhnilaisin käsittein *esiparadigmaattisessa* vaiheessa olevan tieteenalan tilannetta, kun taas alan käytännönhajoituksessa voidaan jo nähdä kypsemmän alan piirteitä, nk. *normaalitieteen* piirteitä.

Paradigmalla tarkoitetaan tietyn tieteellisen yhteisön jakamien olettamusten, uskomusten, mallien ja lähestymistapojen muodostamaa kokonaisuutta (Wernickin ja Hall, 2004). Kullakin alalla juuri paradigma luo ajattelulle perustan, jonka avulla tietyn alan toimijat voivat olla yhtä mieltä siitä, mitä alalla koetaan merkitykselliseksi tarkastella ja millä tavalla alaa voidaan harjoittaa (Kuhn 1994). Toisin sanoen tietyllä alalla vallitseva paradigma määrittää suuresti sitä, miten tavoitteita ja mahdollisia ratkaisuja hahmotetaan. Toisaalta Kuhnin (2012) mukaan eri metodien tai työkalujen *käyttöönotto* on juuri sitä, miten tietty vaikuttava paradigma tulee esiin käytännön tilanteessa.

Opinnäytetyö toteutettiin empiirisenä laadullisena tutkimuksena, jossa selvitettiin tapaustutkimuksen avulla, miten ohjelmistokehittäjä kehitysprosessin mittaan ymmärtää työskentelyn tavoitteen ja mahdolliset suunnittelu- ja toteutusratkaisut eri työkaluja käyttäessään. Tutkimukselle asetettiin kaksi tutkimuskysymystä: Millaisia näkökulmia suunnittelu- ja toteutusprojektin mittaan käsiteltäviksi tulevat ohjelmointikielet, työkalut ja työskentelymetodit suunnittelijalle tarjoavat työskentelyn kohteena olevaan ongelmaan ja sen mahdollisiin ratkaisuihin? Kuinka näkökulmien kanssa työskentely suunnittelu- ja toteutusprosessin myötä etenee? Työn empiirisessä vaiheessa kerättyä muistiinpanoaineistoa analysoitiin teoriasidonnaisen sisällönanalyysin analyysimenetelmällä.

Analyysin tuloksia tarkasteltaessa eri työkalujen havaittiin asettuvan erityyppisiin, laajempaa kokonaiskuvaa täydentäviin rooleihin. Yhdessä niiden tulkittiin osaltaan ilmentävän eheän paradigmaattisen perustan vaikutusta, mikä mahdollistaa ohjelmistoteknisten suunnitteluongelmien tunnistamisen ja ratkaisemisen. Tutkimuksen tulokset tukevat Wernickin ja Hallin (2004) tuloksia ja tulkintaa siltä osin, että alan käytännönhajoituksessa voitaisiin jo tunnistaa yhtenäisen paradigman, ja siten myös *normaalitieteen* piirteitä. Wernick ja Hall (2004) kuitenkin painottavat, että heidän tuloksensa osoittavat myös erillisten koulukuntien olemassaolon ohjelmistotekniikan alalla. Yhdeksi tulevaisuuden tutkimuksen aiheeksi jääkin sen tarkempi määrittäminen, perustuvatko ohjelmistotekniikassa nykyisin toimivat koulukunnat myös paradigman laajuisiin erillisiin maailmankuviin, vaiko vain erilaisiin mutta kuitenkin yhteismitallisiin näkökulmiin, jolloin kyse ei olisi paradigman suhteen sirpaleisesta tilanteesta.

Kunkin alan tiedeyhteisön on tärkeää olla eksplisiittisesti tietoinen yhteisesti jaetuista olettamuksista, uskomuksista, malleista ja lähestymistavoista. Tältä perustalta käsin tarkasteltuna erilaiset lähestymistavat hahmottuvat niin tutkimuksen kuin alan peruskoulutuksen kannalta merkityksellisesti. Mikäli ohjelmistotekniikka hahmotetaan hybriditoimintakentäksi, joka koostuu sekä tieteellisen että teknisen paradigman rinnakkaiselosta, myös tieteellisen ja käytännön teknisen kehitystyön paradigmoihin liittyvien tendenssien ja mahdollisten keskinäisten ristiriitojen erittely ja analysointi epistemologisesta näkökulmasta on arvokasta jo alan sisäisen kommunikaation tukemiseksi. Tämä edesauttaa esimerkiksi alaa kohtaan esitetyn tiedon totuusstatukseen kohdistuvan kritiikin käsittelyä (esim. Holloway 1994; 1995). Tämän tutkimuksen tulokset osallistuvat osaltaan kirkastamaan ohjelmistotekniikan alan itsetuntemusta tältä osin.

Lähteet

Episkopou, D.M., & Wood-Harper, A.T. 1986. "Towards a framework to choose appropriate IS approaches." *Computer Journal* 29, 222–228. doi: 10.1093/comjnl/29.3.222.

Frakes, W. 1994. "Systematic software reuse: a paradigm shift." *Proceedings of 1994 3rd International Conference on Software Reuse*. 2-3. IEEE. doi: 10.1109/ICSR.1994.365817.

Garlan, D. 2010. "Software engineering in an uncertain world." *Proceedings of the FSE/SDP workshop on Future of software engineering research.* 125-128. doi: 10.1145/1882362.1882389.

Grogono, P., & Shearing, B. 2008. "Concurrent software engineering: Preparing for paradigm shift." *Proceedings of the 2008 C3S2E Conference.* 99-108. doi.org/10.1145/1370256.1370270

Gürel, A. 2018. *Cognitive comparison of using hand sketching and parametric tools in the conceptual design phase* (Doctoral dissertation, Bilkent University). http://hdl.handle.net/11693/49002

Hirschheim, R., & Klein, H. K. 1989. "Four paradigms of information systems development." *Communications of the ACM*, *32*(10), 1199-1216. doi: 10.1145/67933.67937

Holloway, C. M. 1994. "Epistemology, software engineering and formal methods." abstrakti. https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19950010065.pdf.

Holloway, C. M. 1995. "Software engineering and epistemology." *ACM SIGSOFT Software Engineering Notes*, 20(2), 20-21. https://dl.acm.org/doi/pdf/10.1145/224155.565638?casa_token=Baqiu4MXsZAAAAAA:BbElGGrefjtVINhwt07fwb3ikzTzeRfHY37g4pf-OlF_p2lSbJ3mNh0BdWv3pNUtPj_yBN6_32U

Kuhn, T. S. 1994. *Tieteellisten vallankumousten rakenne*. Alkuperäisteoksesta *The structure of scientific revolutions* (1962.) suom. K. Pietiläinen. Helsinki: Art House.

Kuhn, T. S. 2012. *The structure of scientific revolutions*. Fourth Edition. Chicago & London: University of Chicago press.

Lázaro, M., & Marcos, E. 2005. "Research in Software Engineering: Paradigms and Methods." *CAiSE Workshops (2)*, 517-522. http://kybele.escet.urjc.es/phise05/papers/sesionIII/LazaroMarcos.pdf

Northover, M., Kourie, D. G., Boake, A., Gruner, S., & Northover, A. 2008. "Towards a philosophy of software development: 40 years after the birth of software engineering." *Journal for General Philosophy of Science*, 39(1), 85-113. doi: 10.1007/s10838-008-9068-7

Ogundare, O. 2017. "How Do You Know What You Know: Epistemology in Software Engineering." *Journal of Software Engineering and Applications*, 10(02), 168. https://www.scirp.org/pdf/JSEA_2017022415054805.pdf

Saaranen-Kauppinen, A., & Puusniekka, A. 2006. *KvaliMOTV - Menetelmäopetuksen tietovaranto*, verkkojulkaisu. Tampere: Yhteiskuntatieteellinen tietoarkisto. https://www.fsd.tuni.fi/menetelmaopetus/.

Smith, L., Wernick, P., & Veneziano, V. 2011. "Is finding a Black Swan ‘Popper,(1936) possible in software development?" Teoksessa *The computational turn: Past, presents, futures?* toimittajat Charles Ess, & Ruth Hagengruber, 67-70. Aarhus University: Verlagshaus Monsenstein und Vannerdat OHG.

Tahsiri, M., Hale, J., & Niblock, C. 2017. "Knowledge distribution and the effect of design tools on the design process." *Design Computing and Cognition'16,* 437-455. Springer, Cham.

Tuomi, J. & Sarajärvi, A. 2018. *Laadullinen tutkimus ja sisällönanalyysi* (Uudistettu laitos.). Helsinki: Kustannusosakeyhtiö Tammi.

Wernick, P., & Hall, T. 2004. "Can Thomas Kuhn's paradigms help us understand software engineering?" *European Journal of Information Systems*, 13(3), 235-243. doi: 10.1057/palgrave.ejis.3000501

Liitteet

A Hyperspektrikameran lineaariskannerin ohjainohjelmiston suunnitteludokumentti

Johdanto

Tämä dokumentti sisältää hyperspektrikameran lineaarisiirtimien liikutteluun tarkoitetun ohjainohjelmiston suunniteludokumentaation. Suunnitellun järjestelmän avulla käyttäjä on tarkoitus kuvata pinta-aloja skannaustyyppisesti hyperspektrikameraa kuvatessa ja kuvausten välillä liikuttaen. Näin ollen järjestelmän käyttö voi liittyä esimerkiksi taideteosten viipaleittaiseen kuvantamiseen. Järjestelmässä ohjataan moniakselista lineaariskanneria, jonka avulla liikutellaan spektrikameraa. Skannerin ohjaus perustuu askelmoottorien käskyttämiseen, ja järjestelmä toimii viivaskannerin tavoin kolmessa suunnassa. Suunnista kaksi (X,Y) muodostavat skannattavan alueen pinta-alan ja kolmatta (Z) hyödynnetään kameran tarkennuksessa. Akseleita vastaaviin kameroihin viitataan dokumentaatiossa vastaavaa kirjainta käyttämällä.

Järjestelmä liittyy askelmoottorijärjestelmien sovellusalaan. Nämä järjestelmät koostuvat kontrollerista, ajureista ja askelmoottoreista, joita ohjataan jonkinlaisten käyttöliittymän kautta (https://en.wikipedia.org/wiki/Stepper_motor#Stepper_motor_system). Tässä tapauksessa kontrollerina toimii mikrokontrolleri Arduino Uno, ajureina Polulo A4988 -ajurit ja askelmoottoreina yksi bipolaarinen Nema 17 -moottori ja kaksi unipolaarista Nema 23 -moottoria.

Ohjainjärjestelmän loppukäyttäjät ovat joko tietotekniikan opiskelijoita tai tutkijoita, eli ns. asiantuntijakäyttäjiä. Näin ollen heidän odotetaan hallitsevan tietotekniikan perustaidot vähintään alan maisteriopiskelijan tasoisesti. Yksi käyttäjien tarpeista nouseva lähtökohta on, että ohjainta voidaan käyttää graafisen käyttöliittymän sijaan komentoriviltä käsin python-skripteillä ohjaten. Niinpä esimerkiksi järjestelmän helppokäyttöisyys ei tässä tapauksessa tarkoita samaa, kuin käyttöliittymien suunnittelun yhteydessä yleensä. Tavoitteena onkin tarjota ohjausjärjestelmän kautta riittävän geneerisiä, monenlaisiin

kuvaustilanteisiin soveltuvia toimintoja, joilla käyttäjä voi toteuttaa kuvaustilanteensa tarpeitaa vastaavat ohjausliikkeet ilman että moottoreiden ohjaukseen täytyy kuitenkaan kiinnittää tarpeettoman paljon huomiota.

Järjestelmän määrittelyyn liittyviä käsitteitä:

- **Hyperspektrikamera** - kamera, joka pystyy kuvaamaan useita valon eri aallonpituuksia erillisille kanaville. Kamerat toimivat yleensä 400-1000 nm ja 1000-2500 nm aallonpituusalueilla.
- **Viivaskanneri** - kuvantava fotodiodi, joka kuvantaa yhtä spatiaalista pikseliriviä kerrallaan
- **Rajakytkin** - kytkin, joka pysäyttää moottorin, kun kelkka/pidike/alusta koskettaa sitä
- **Lineaariskanneri** - automaatio järjestelmä, jossa kelkka/pidike/alusta kulkee kiskoa pitkin edestakaisin

1. Käyttöskenaariot

Tässä luvussa kuvataan joukko järjestelmän ohjausjärjestelmän käyttöskenaarioita. Skenaariot kuvaavat avaininteraktiot käyttäjän ja systeemin, systeemin ja ympäristön sekä systeemin osien välillä.

I. Peruskäyttö: käyttäjä kuvaa pinta-alan

Käyttäjä haluaa kuvata taulun pinta-alan skannaamalla. Ensin käyttäjä tarkentaa kameran z-akselin suuntaista liikettä käyttäen. Alue skannataan lähtien liikkeelle koordinaateista (X0,Y0). Kuvatessa kameraa siirretään kohtaan (X0,Y1) mahdollisimman tasaisella liikkeellä. Tämän jälkeen palataan takaisin kohtaan (X0,Y0) ja siirrytään kohtaan (X1,Y0), jonka jälkeen skannaataan tasaisella liikkeellä alue kohtaan (X1,Y1), palataan takaisin kohtaan (X1,Y0), siirrytään kohtaan (X2,Y0) jne.

II. Peruskäyttö: käyttäjä selvittää kuvattavan alan korkeuden

Käyttäjä haluaa tietää, kuinka monta askelta moottorin täytyy ottaa skannatakseen koko taulun korkeuden. Niinpä hän käynnistää moottorin Y ja antaa sen liikkua, kunnes kelkka on edennyt taulun yläreunaan. Tällöin hän pysäyttää moottorin ja pyytää järjestelmää palauttamaan kameran sijaintitiedon.

III. Peruskäyttö: viimeisimmän sijaintitiedon palautus

Käyttäjä haluaa kuvata taulun pinta-alan skannaamalla. Alue skannataan lähtien liikkeelle koordinaateista (X0,Y0). Kameraa siirretään kuvatessa kohtaan (X0,Y1), missä Y1 vastaa edellä selvitettyä korkeutta. Kuvauksen päätteeksi käyttäjä pyytää järjestelmää palauttamaan kameran viimeisimmän sijainnin sijaintitiedon ja saa kameran koordinaatit Numpy Array -muotoisena taulukkona, missä sijainti on ilmaistu kameran ottamien askelien perusteella.

IV. Peruskäyttö: kuvaus aloitetaan määrätystä kohdasta

Käyttäjä määrittelee kuvauksen aloituskohdan, joka perustuu hänen aikaisempaan kuvauskokemukseensa. Järjestelmä siirtää kameran haluttuun kohtaan. Käyttäjä käynnistää kameran. Hän skannaa matkan Y, sammuttaa kameran ja palaa aloituskohtaan. Käyttäjä siirtää kelkan aloituskohdasta pykälän verran suuntaan X ja toistaa skannaamisen, kameran sammuttamisen, palaamisen ja kelkan siirtämisen X:n suhteen niin monta kertaa kuin sillä erää on tarve.

V. Peruskäyttö: snapshot-kuvaus

Käyttäjä haluaa kuvata snapshot-tyyppisesti, jolloin kamera on kuvanottohetkellä pysähdyksissä ja siirtyy sitten seuraavaan kohtaan. Kameran ja moottoreiden synkronaatiosta ei tarvitse välittää vaan käyttäjä huolehtii asiasta.

VI. Peruskäyttö: komentorivikäyttöliittymän käyttöesimerkki 1

Käyttäjä ohjaa järjestelmää komentoriviltä syöttämällä seuraavanlaisen käskyn: kuvaus.skannaa(matka, nopeus), jolloin kelkkaa y-suuntaisesti liikuttava moottori liikkuu skannaus-modessa (=mahdollisimman tasainen liike) tiettyyn suuntaan halutun määrän askelia.

VII. komentorivikäyttöliittymän käyttöesimerkki 2

Käyttäjä ohjaa järjestelmää komentoriviltä syöttämällä seuraavanlaisen käskysarjan:

kuvaus.siirry_askelta(4000, 500, 200)

kuvaus.vaihda_suunta(”z”)

kuvaus.siirry_askelta(0, 0, 50)

jolloin ensin liikutetaan kelkkaa sijaintiin (4000, 500, 200), sitten tullaan z-suuntaisesti hieman takaisin päin.

VIII. Poikkeustilanne: kun ohjataan rajakytkimen taakse 1 (minimivaatimus)

Käyttäjä ohjaa kameraa Y-akselin suhteen sijaintiin, joka olisi rajakytkimen takana. Hän koettaa käskyttää kameraa liikkumaan rajakytkimen ulkopuolelle, mutta saavuttuaan rajakytkimen kohdalle kamera pysyy paikallaan. Komentoriville tulostetaan teksti, joka kertoo, ettei kamera voi siirtyä Y-akselilla edemmäs. Pyytäessään käyttäjä saa kameran sijaintitiedon, joka kertoo kameran todellisen viimeisen sijainnin Y-akselilla (ei kuvitteelista sijaintia rajakytkimen takana).

IX. Poikkeustilanne: kun ohjataan rajakytkimen taakse 2 (edistyneempi)

Käyttäjä aikoo ohjata kameraa Y-akselin suhteen sijaintiin, mutta järjestelmä ei anna syöttää näin suurta askelmäärää, vaan hyväksyy askelmäärät, jotka pysyttelevät aina rajakytkimien rajaaman alueen sisäpuolella siten, ettei rajakytkimille käytännössä koskaan edes tulla (paitsi ohjelman alussa tarvittava nollakytkimellä käynti, jonka avulla toteutetaan sijainnin laskenta).

X. Poikkeustilanne: Varotoimi kohteen puhkaisemisen estämiseksi

Käyttäjä määrittää kuvauksen aloituskohdan, ja siirtyy z-suuntaisesti vielä lähemmäs kuvauskohdetta. Käyttäjä arvioi liikuttamiseen sopivan askelmäärän väärin, jolloin ollaan vaarassa puhkaista kohde. Järjestelmä havaitsee kohteen läheisyyden, eikä tämän vuoksi siirrä kelkkaa skriptissä käskytettyyn sijaintiin saakka vaan pysähtyy kesken matkan. Tämänkin jälkeen pyydettäessä palautettu sijaintitieto vastaa todellista sijaintia.

2. Vaatimusmäärittely

Tämän suunnitteluprojektin luonteeseen kuului, että valtaosa järjestelmän laitteistovalinnoista oli tehty ennen suunnitteluprojektin alkamista, joten varsinaisena suunnittelutehtävänä oli lähteä toteuttamaan järjestelmää laitevalintojen ehdoilla. Järjestelmän tilaaja tiesi millaiset askelmoottorit järjestelmään tulisi ja juuri ennen projektin alkua tehtiin päätös, että kontrollerina käytettäisiin Arduino mikrokontrolleria.

Toiminnalliset vaatimukset

- Käyttäjän tulee voida käskyttää kustakin suunnasta (x/y/z) vastaavaa moottoria erikseen.
- Kunkin moottorin akselilla tulee voida liikkua määrätysti edestakaisin, rajakytkimien rajaamalla välillä (myöhemmin: rajakytkimien rajaaman alueen sisäpuolelle määritetyllä turva-alueella).
- Yksittäisen moottorin viimeisimmästä sijaintitiedosta tulee pitää yllä tietoa, ja kelkan viimeisin sijainti tulee voida palauttaa käyttäjälle Numpy Array -muotoisena taulukkona.
- **Skannaus-modessa** kelkan liike on mahdollisimman tasaista. Koska sijaintitieto halutaan määrittää, tulee tasaisen liikkeen olla suhteellisen hidasta, sillä kiihdytyksin ja hidastuksin varmistettava todellisten askelien laskenta (ts. sen varmistaminen ettei askelia hukata) ei ole käytössä.
- Skannauksen aloituskohdan voi halutessaan määrittää itse (oletuksena (x0, y0, z0)).
- Aloituskohdan määrittämisen lisäksi käyttäjä voi määrittää kuvattavien viipaleiden paksuuden ja lukumäärän. Tämän jälkeen skannaamisen tulisi olla yksinkertaista. Ellei toisin määrätä, käytetään skannatessa sopivia oletusarvoja.
- **Perus-modessa** kelkan liike on skannaus-modea ripeämpää (koska oletus on, että skannaus-moden liikkeen tasaisuus ja sijaintitiedon tarkkailu edellyttävät suht hidasta liikettä). Liikkeeseen kiihdytetään vähitellen ja ennen pysähdystä liikettä hidastetaan. Tällä pyritään varmistamaan, ettei moottorin askelia hukattaisi liian voimakkaan väännön vuoksi. Väännön voimakkuuteen vaikuttaa liikuteltavan kameran paino, joka on tässä tapauksessa suhteellisen korkea (8-15 kg).
- Kuvaaminen ja kelkan liikuttelu ovat erillisiä asioita. Snapshot-kuvaaminen onnistuu kameraa perus-modella haluttuihin sijainteihin liikutellen.
- Käyttöliittymältä odotetaan yksinkertaisuutta, ja käytön joustavuutta käyttäjän ideoiden suhteen.
- Käyttöliittymän tulee olla komentorivipohjainen.
- Käyttöliittymän tulee mahdollistaa ohjaaminen python-skripteja käyttäen.

- **Turvallisuustekijöitä:**
- Rajakytkimet rajoittavat kelkan liikettä: kun kelkka saapuu rajakytkimelle, se pysähtyy eikä yritä jatkaa tästä eteenpäin. Rajakytkimelle tulon jälkeen valmistaudutaan palaamaan, eli liikkumaan (niin pyydettäessä) päinvastaiseen suuntaan. Suunnanvaihdos voidaan siis tehdä oletuksena, sillä tämän suhteen käyttäjällä ei ole vaihtoehtoja. Oletuksena ei kuitenkaan liikuta, vaan rajakytkimelle tultaessa pysähdytään ja käskystä mahdollisesti jäljelle jääneet askeleet jätetään ottamatta.
- Z-akselin suuntaisesti liikuttaessa tulee pyrkiä varmistamaan ohjaimen turvallinen käyttö niin, ettei kuvattavaa kohdetta erehdyksissä päädytä puhkaisemaan. Tässä voidaan käyttää esimerkiksi jonkinlaista etäisyysmittaria.
- Rajakytkimien rajaama pinta-ala moottoreiden liikutteluun käytettävien askelien suhteen tulee määrittää, ja tämän (ja moottorien sijaintitietojen) perusteella rajoitetaan askelmäärää, jonka käyttäjä voi syöttää skripteille. Muodostetaan ns. turva-alue, jolla kelkka pysyy, eikä sen näin ollen pitäisi edes tulla rajakytkimille.

Laadulliset vaatimukset

- Ohjaimen tarjoamien toimintojen soveltuminen erilaisiin käyttötilanteisiin osana mahdollista muuta koodia.
- Käytön mutkattomuus, yksinkertaisuus, työkalumaisuus. Tätä vaatimusta tulee tulkita käyttäjäryhmän oletetut tietotekniset perustaidot huomioiden!
- Turvallisuus: järjestelmä ei saa rikkoa liikuttelemaansa laitteistoa tai kuvattavaa kohdetta.

Käyttäjät tullaan aina kouluttamaan järjestelmän käyttöön. Erillisiä CE-merkintöjä laitteistolle ei haeta.

3. Järjestelmän kokoonpano

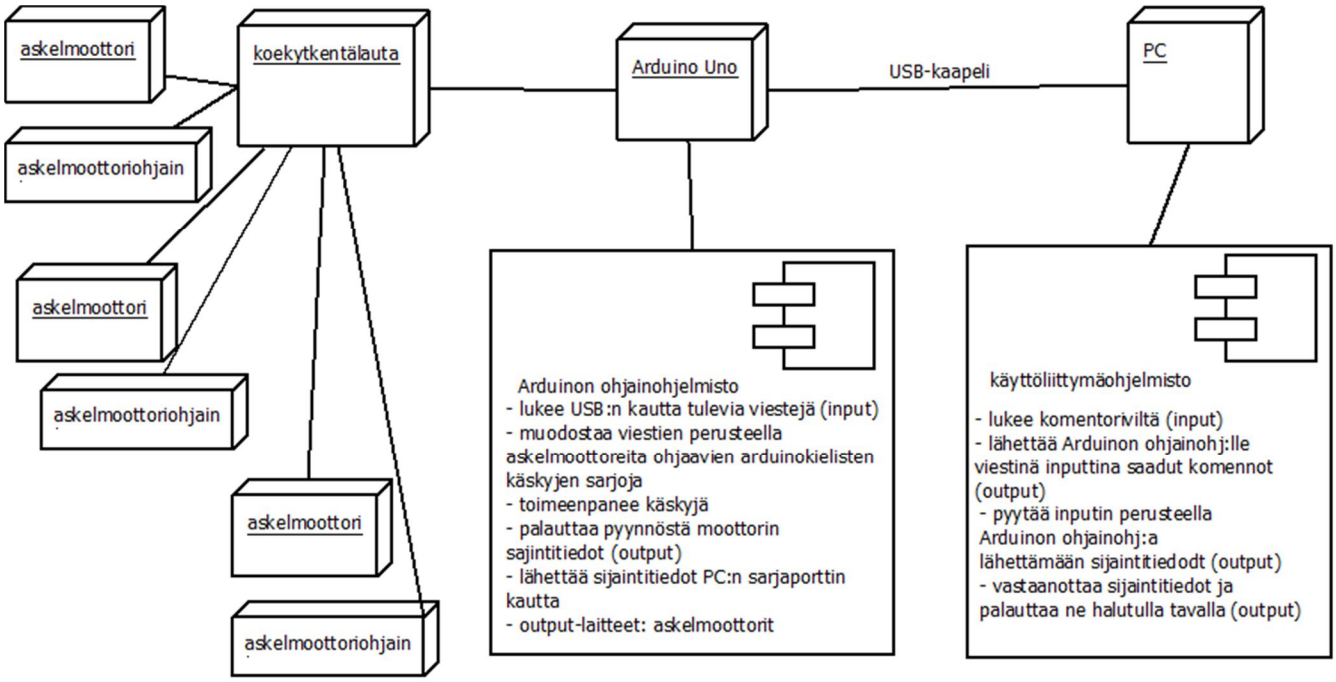
Järjestelmä sisältää seuraavat komponentit:

- iso askelmoottori Nema23, 2 kpl (x- ja y-suunnista vastaavat moottorit)
- askelmoottori Nema17, 1 kpl (z-suunnasta vastaava moottori)
- Pololu A4988 -ohjaimet, 3 kpl
- Arduino UNO -mikroprosessori, 1 kpl
- Arduino IDE -sovellus ja sen StandardFirmataPlus-sketsi
- pyFirmata-kirjasto
- (myöhemmin lisättävä: etäisyysmittari)

Lisäksi kehitysympäristöön kuuluivat:

- laboratoriokäyttöön tarkoitettu virtalähde GW GPS-3030D
- Spyder 4 -kehitysympäristö,
- Python 3.7.6
- ThinkPad -kannettava tietokone
- Windows 10 -käyttöjärjestelmä

4. Käyttöönottokaavio

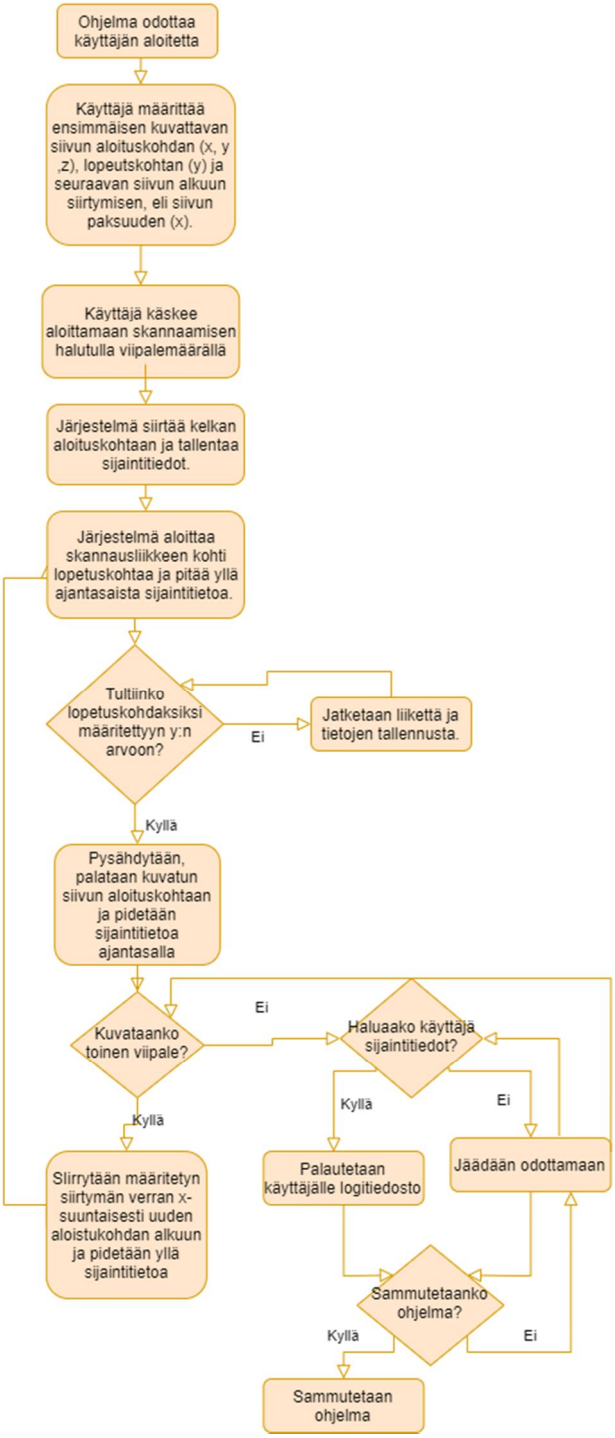


5. Algoritmi

Lähtökohtaisesti ohjainohjelmistoa voidaan käyttää kahdella eri tavalla. Alla olevassa kuvassa (ks. seuraava sivu) esitetään ”määritetty skannaus” -tyyppisen käytön taustalla oleva algoritmi. ”Määritetty skannaus” tarjotaan käyttötapaus I:ssä kuvatun kaltaisen säännöllisen liikuttelukuvion toteuttamisen helpottamiseksi.

Tämän lisäksi ohjainohjelmistoa voidaan käyttää myös ilman allaolevassa algoritmissa mainittujen aloitus- ja lopetuskohtien, ja viipaleiden paksuuksien ja -määrien määrittämistä. Tällöin käyttäjä rakentaa haluamansa kaltaisen liikutteluskriptin määrittämällä kutakin akselia vastaavan moottorin liikkeet erikseen. Tässä tapauksessa käyttäjän tulee määrittää käskytetyn moottorin liikumat matkat, suunnanvaihdokset ja nopeudet/liikuttelumodet (perus- vai skannaus-tyyppinen liikuttelu).

Ohjaimen käyttö viivaskanneri-tyyppisessä kuvaustilanteessa



6. Luokkasuunnitelma

Ohjainohjelmiston ensimmäinen versio koostuu kolmesta luokasta, jotka ovat Moottori, Moottorit, ja Kuvaus. Luokkienvälinen vastuunjako on seuraavanlainen:

Moottori-luokan olio säilyttää tietoa yhden moottorin nimestä, sijaintilukemasta, moottorin kytkentään liittyvistä pinninumeroista (step ja dir) ja moottorin rajakytkimien pinninumeroista ja tiloista. Lisäksi moottorilla on atribuutit, joiden avulla tallennetaan tieto, saako moottori liikkua ja kumpaanko suuntaan moottori on parhaillaan kulkemassa. Moottorin metodit ovat pääasiassa settereitä ja gettereitä sekä erilaisia ylemmällä tasolla hyödynnettäviä tietojen tulostus-metodeita.

Moottorit-luokan olioon tarvitaan kolme moottori-oliota (x-, y- ja z-moottorit). Luokan tehtävä on ohjata käskyt oikealle moottorille ja vastata kommunikaatiosta Arduino-mikroprosessorin suuntaan. Moottorit-oliolla on merkkijono-muotoinen tieto käytetystä USB-portista ja porttia käyttävä kortti-attribuutti, joka vastaa mikroprosessoria. Arduino-kommunikaatio toteutetaan pyfirmata-kirjaston avulla, joten suunnitelma edellyttää, että Arduinoon on ajettu sopiva pyFirmata-sketsi, minkä seurauksena se odottaa kommunikaatiota isäntäohjelmalta. Moottorit-oliota luotaessa sille luodaan ja käynnistetään iteraattori, joka läpikäy Arduino-korttia. Moottorit-olion luomisen yhteydessä kolmen moottorin rajakytkimien käyttämien Arduino-pinnien käyttö tulee myös määritellä input-tyyppiseksi.

Moottorit-olion metodit sisältävät moottorien liikutteluun, suunnanvaihtoon, rajakytkinten tarkkailuun ja sijainnin palauttamiseen liittyvät toiminnot. Lisäksi on tulostusmetodeita, joiden avulla käyttäjä voi tarkistaa kolmen moottorin kokoonpanon kytkentöihin liittyvät tiedot (tämä ei liity ohjainohjelmiston peruskäyttöön, mutta saattaa ohjelmiston kehittyessä myöhemmin tulla hyödylliseksi).

Kuvaus-luokan olio käyttää yhtä moottorit-oliota. Luokan tehtävänä on tarjota käyttäjälle metodeita, joiden avulla moottorien käskyttäminen on yksinkertaista ja selkeää ja toteuttaa liikuttelutehtäviä nimenomaan *hyperspektrikameran lineaarisiirtimen käytön* tarpeita

huomioiden. Oliota luotaessa asetetaan moottoreille lineaarisiirtimen rakennetta vastaavat aloitussuunnat (niin että aivan aluksi käydään aina ensimmäiseksi nollakytkimellä). Lisäksi oliolla on dictionary-tyyppinen määritykset-kokoelma tietoja, johon käyttäjä voi luokan metodeita käyttäen tallettaa muistiin skannaustapahtumaan liittyviä arvoja, joiden avulla voidaan luoda säännöllinen skannauskuvio, jossa skannataan (eli liikutetaan kelkkaa tasavauhtisesti) haluttu määrä saman mittaisia ja paksuisia y-suuntaisia viipaleita. Luokka sisältää myös metodit, joilla käyttäjä voi yksinkertaisemmin liikutella kelkkaa eri suuntiin.

7. Huomioita nykyisestä järjestelmästä

Ohjelmistosuunnittelun ensimmäinen sykli toteutettiin etätyössä kesä- heinä- ja elokuun aikana 2020. Tämän syklin tuloksena syntyi hyperspektrikameran lineaarisiirtimien ohjainohjelmiston ensimmäinen prototyyppi. Syklin työskentelyssä käytössä olivat lineaariskannerin kokoonpanoon kuuluvia moottoreita vastaavat irralliset askelmoottorit, joiden liikuttelua varten rakennettiin Arduino UNO -mikrokontrollerin käyttöön perustuva laitteisto. Moottoreiden ja Arduinon lisäksi laitteisto sisälsi askelmoottoreiden ajurit, ulkoisen virtalähteen sekä rajakytkimien toimintaa demonstroivan painonappisarjan, joka on tarkoitettu pelkästään väliaikaiseksi, kehitystyötä tukevaksi komponentiksi. Ensimmäisen toteutussyklin puitteissa ohjelmoitiin Arduino-kytkentöihin ja ohjaimen perustoimintoihin liittyviä ominaisuuksia, kuten esimerkiksi rajakytkimien seuranta, moottorien liikuttelu ja suunnanvaihdot, sijaintitiedon laskeminen ja tämän tiedon palauttaminen käyttäjälle vaatimuksissa esitetyssä muodossa.

Ohjelmiston kehitystyön seuraavassa vaiheessa työskentely edellyttää tarkempaa varsinaiseen laitteistoon tutustumista ja kehitetyn ohjelmiston kokeilemista sen äärellä. Erityisesti useat alustuksiin liittyvät alkuarvot jouduttiin vielä ensimmäisessä vaiheessa asettamaan arvaukseen perustuen (esimerkiksi kumpaanko suuntaan moottoreiden on pyörittävä, jotta tullaan 0-sijainnissa oleville rajakytkimille). Lisäksi hyperspektrikameran painon vaikutukset kelkan sijainnin määrittämiseen saadaan esiin vain testaamalla ohjelmistoa kameraa (tai sen painoa vastaavaa massaa) lineaarisiirtimellä liikuttelemalla. Tässä yhteydessä tulee erityisesti selvittää, hukataanko liikkeellelähtöjen tai pysähtymisten yhteydessä askelia ja vastaavatko samojen liikutteluskriptien jälkeen havaittavat kamerojen sijainnit toisiaan – ohjelmiston palauttamista sijaintitiedoista tulee panna merkille, että ne on laskettu Arduinoon toteutettavaksi määrättyjä liikutteluimpulsseja laskien. Varsinaisista *toteutuneista* askelista järjestelmä ei tällä hetkellä saa tietoa.

Kun edellämainittuja kameran painon vaikutuksia on selvitetty, pystytään tämän jälkeen määrittelemään skannauksessa käytetylle liikuttelutoiminnolle sopivat nopeusrajat, joiden puitteissa liike voidaan toteuttaa tasaisena, mutta siten, että myös skannauksen jälkeinen

sijaintitieto on paikkansapitävä. Tässä yhteydessä tulee ratkaista, millaista tarkkuutta sijaintitiedolta edellytetään. Selvitettäväksi tulee myös, millä tavoin perusliikuttelussa mahdollisesti tarvittavat kiihdytykset (liikkeellelähdössä) ja jarrutukset (ennen pysähtymistä) tulevat toteutettavaksi.

Kehitystyön ensimmäisen syklin aikana moottoreiden liikuttelun yhteydessä havaittain ajoittaista x- ja y-moottoreiden liikkeiden epätasaisuutta. Liikkelleelähdön alku vaikutti etenevän hitaasti, minkä jälkeen liike eteni nopeammin nykäisten haluttuun asentoon. Tämä ei vaikuttanut ”alkuvaiheen kiihdytykseltä” ja ilmiö tuli ja meni niin ettei tullut selväksi mistä se johtui. Yhtenä mahdollisena syynä epäiltiin ajureiden kuumenemista, mutta liikuttelun tukkoisuutta esiintyi myös jähdyttämisestä huolehtivien osien lisäämisen jälkeen.

Tulevaisuudessa laitteistokokoonpanon z-suuntaisesta liikkeestä vastaavan moottorin yhteyteen tulee kytkeä etäisyysmittari, jonka avulla pyritään turvaamaan se, ettei kamera missään tilanteessa voi vahingoittaa kuvattavaa kohdetta. Tähän liittyen ohjainohjelmistoon tulee lisätä z-kameran liikuttelun yhteydessä toteutettava etäisyysmittarin tarkkailu.

On myös tärkeää, että ohjainohjelmiston kehitystyön toisessa syklissä tullaan tarjoamaan ohjainohjelmiston loppukäyttäjille mahdollisuus kokeilla ohjainohjelmiston alustavaa demoversiota. Näin voidaan saada parempi kuva siitä, mitä hyperspektrikameran käyttöön liittyviä tarpeita tulisi huomioida ohjaimen toteutuksessa paremmin. Nykyisessä muodossaan ohjelma ei esimerkiksi jätä aikaa käyttäjälle kameran kanssa mahdollisesti suoritettaville toimenpiteille kesken skriptin suorituksen (esimerkiksi määritetyn skannauksen aikana kameraa ei ehdi sammuttaa ja käynnistää uudestaan). Skriptin suoritusta ei myöskään tällä hetkellä voi (hallitusti) keskeyttää tai pistää *pauselle*. Tällaisten toimintojen mahdollinen tarve tulee kuitenkin kartoittaa loppukäyttäjien kanssa.