Juho Järvilehto

Testausjärjestelmän kehitys LabVIEW-ohjelmointiympäristössä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

7.5.2018

Metropolia

Tekijä Otsikko Sivumäärä Aika	Juho Järvilehto Testausjärjestelmän kehitys LabVIEW-ohjelmointiympäristössä 36 sivua 7.5.2018
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Sulautetut järjestelmät
Ohjaaja	Lehtori Keijo Länsikunnas

Insinöörityön tarkoituksena oli kehittää toimiva sovellus ja tehdä alustavaa tutkimustyötä osaksi testausjärjestelmän projektikokonaisuutta. Työn tavoitteena oli luoda ammattikorkeakoululle LabVIEW-ohjelmointiympäristöllä toteutettu testaussovellus osana erillistä asiakasprojektin sulautettua laitetta ja innovaatioprojektin automaatiolaitetta. Ideana oli, että testaussovelluksella pyritään suorittamaan uusien laitteiden toiminnallinen testaus, antureiden kalibrointi ja laadunvarmennus sekä järjestelmien välinen lopullinen laiteohjaus.

Työssä perehdyttiin testausjärjestelmän kehitysprosessiin ensisijaisesti käytännönläheisen projektityöskentelyn näkökulmasta. Järjestelmän suunnittelussa perehdyttiin laitteiden elektroniikkaan komponenttitasolla, teknisiin rajapintoihin ja itse ohjelmistokehitykseen.

Testattavana sulautettuna laitteena oli tasapainokeppi, joka sisälsi mikrokontrollerin, kiihtyvyysantureita ja Bluetooth-moduulin. Automaatiolaite koostui servo-ohjauksella toimivasta testausräkistä. Järjestelmien välinen kommunikaatio ja tiedonsiirto toteutettiin langattomalla Bluetooth-verkolla ja sarjaportilla. Testausprosessi suoritettiin asettamalla tasapainokeppi testausräkkiin, jolloin testaussekvenssi voitiin suorittaa järjestelmää ohjaavalta testaussovellukselta.

Laitteiden prototyyppimaisen luonteen takia LabVIEW-sovellusta pyrittiin kehittämään ketterästi vaiheittain pienissä moduuleissa, mikä johti kolmen täysin uuden version toteutumiseen. Testausohjelman kehitys aloitettiin keväällä 2017, ja se luovutettiin tilaajalle jatkokehitettäväksi syksyllä 2017.

Lopputuloksena saatiin yksinkertainen testaussovellus, joka alustaa kommunikointiyhteyden laitteistojen välillä, tarjoaa työkalut tiedonsiirtoa, vianetsintää ja testausprosessia varten sekä mahdollistaa mittaustulosten raportoinnin. Sovellus voidaan myös helposti muuntaa muille testausprosesseille yhteensopivaksi. Lopullinen sovellus vaatii kuitenkin vielä hieman hienosäätöä, sillä langattoman Bluetooth-yhteyden satunnainen katkeilu aiheutti ongelmia pääohjelmassa.

Avainsanat	testausjärjestelmä, Labview, sulautettu laite, kiihtyvyysanturi, bluetooth, prototyyppi

Author	Juho Järvilehto
Title	Development of testing system in LabVIEW programming environment
Number of Pages	36 pages
Date	7 May 2018
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Embedded Systems
Instructor	Keijo Länsikunnas, Senior Lecturer

The purpose of this thesis was to develop a functional application and preliminary research work as part of a test system project entity. The objective of the thesis was to create a testing application with LabVIEW programming environment for Metropolia University of Applied Sciences as part of a separate embedded device customer project and an automation device innovation project. The idea was to use the testing application to perform functional performance tests for new devices, sensor calibration and quality assurance test as well as final device management between embedded systems.

This thesis is orientated toward development process of the test system application, primarily from a pragmatic project work point of view. During the system development process, an introduction is done to electronics at the component level, technical interfaces and software development itself.

The embedded device to be tested was a balance stick that includes a microcontroller, accelerometers and a Bluetooth module. The automation device consists of a servo-controlled test rack. Communication between the systems and data transfer was implemented with a wireless Bluetooth network and a virtual serial port connection. The test process was carried out by inserting the balance stick in to a test rack and by running a test sequence controlled by the testing application.

Due to the prototype nature of the devices, the aim was to develop the LabVIEW application agilely in small modules, which led to the implementation of three completely new versions of the testing application. The development of the testing application started in spring 2017 and was handed over to customer for further development in the fall of 2017.

As a result, a simple testing application was created, capable of initiating communication between the hardware, providing tools for data transfer, troubleshooting and testing process, as well as enabling reporting of measurement results. The application can also be easily converted to be compatible with other testing processes. However, the final application requires a bit of fine-tuning as the occasional disconnects in wireless Bluetooth connection cause problems in the main program.

Keywords	test system, labview, embedded device, accelerometer, bluetooth, prototype

Sisällys

Lyhenteet

A/D	Analog to Digital
ATE	Automated Test Equipment
BLE	Bluetooth Low Energy
C/V	Coulomb to Voltage
COM	Communication port
DUT	Device Under Test
EVO	Evolutionary development
I^2C	Inter-Integrated Circuit
LabVIEW	Laboratory Virtual Instrument Engineering Workbench
MEMS	Micro Electro Mechanical Converter
UART	Universal Asynchronous Receiver Transmitter
UI	User interface
UML	Unified Modeling Language
USB	Universal Serial Bus
VRW	Velocity Random Walk
VSP	Virtual Serial Port

1 Johdanto

Insinöörityön tarkoituksena on mittaus- ja testaussovelluksen kehittäminen sulautettuun järjestelmään. Työ toteutetaan osana Metropolia Ammattikorkeakoulun testausjärjestelmän projektikokonaisuutta, joka koostuu asiakasprojektin testattavasta sulautetusta laitteesta ja innovaatioprojektin testausta suorittavasta automaatiolaitteesta. Tehtävänä on suunnitella ja toteuttaa helppokäyttöinen testausohjelma mainittujen sulautettujen laitteiden ohjaukseen. Testausjärjestelmän tavoite on uusien laitteiden toiminallinen testaus, laadunvarmistus ja mittaustulosten raportointi. Yksi testauksen tärkeimmistä tehtävistä on varmentaa ja todentaa, että laitteen liikeanturien mittaustulokset ovat määriteltyjen viitearvojen sisällä.

Suunnitteluvaiheessa otettiin huomioon projektin kokeellinen luonne, minkä takia päädyttiin National Instrumentsin LabVIEW (Laboratory Virtual Instrument Engineering Workbench) -ohjelmointiympäristöön, joka soveltuu erityisesti PC-pohjaisten sovellusten nopeaan kehittämiseen ja rajapintojen vaivattomaan yhdistämiseen. Lisäksi on tärkeä huomioida LabVIEW-ohjelmointiympäristön tarjoama laaja tuki eri rajapintojen ja laitteiston välillä isommissa projektikokonaisuuksissa. Testattavana laitteena toimii kuntoilu- ja venyttelyharjoitteluun suunniteltu tasapainokeppi. Laitteen toimivuus varmennetaan liikuttamalla testattavaa tasapainokeppiä testausräkissä ennalta määritellyillä liikeradoilla ja vertaamalla tasapainokepin liikeantureista saatuja tuloksia testausräkin kalibroituihin arvoihin.

Projektin laitteistokokonaisuus koostuu elektroniikaltaan mikrokontrollereista, kiihtyvyysantureista, Bluetooth-moduuleista ja servo-ohjaimista sekä moottoreista. Laitteiden kommunikointi ja tiedonsiirto toteutetaan Bluetooth-verkolla, joka on tarkemmalta tyypiltään BLE (Bluetooth Low Energy). Klassiseen Bluetoothiin verrattuna BLE säästää laitteen energian kulutusta pelkistämällä ja vähentämällä tiedonsiirtoa. Järjestelmän tiedonsiirto toteutetaan käyttäen virtuaalista UART-sarjaliikennerajapintaa (Universal Asynchronous Receiver Transmitter).

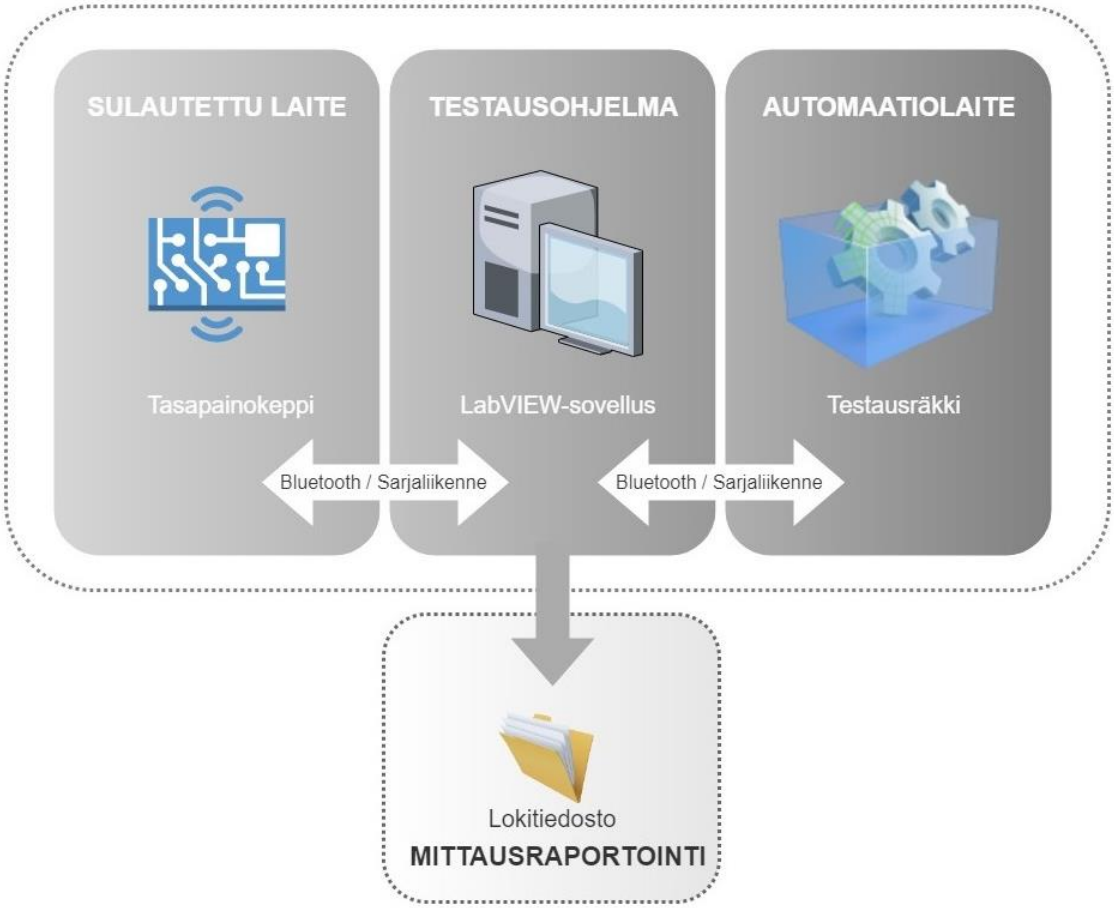
Insinöörityö aloitettiin testausohjelman osalta 27.3.2017, ja ohjelma luovutettiin jatkokehitystä varten 18.9.2017. Työssä tuotetun prototyyppisovelluksen on tarkoitus toimia alustavana tutkimustyönä ja jatkokehitysprojektina tuleville sulautetuille laitteille sekä järjestelmille. Lisäksi tulee huomioida, että dokumentoinnin teknisiä osioita on pyritty kirjoittamaan varjellen asiakkaan ja tilaajan liikesalaisuutta.

2 Testausjärjestelmä

Automaattisella ATE-testausjärjestelmällä (Automated Test Equipment) tarkoitetaan järjestelmää, joka suorittaa testattavalle laitteelle erilaisia mittauksia ja analysoi testien tulokset. Testausjärjestelmä voi olla laajuudeltaan hyvinkin yksinkertainen, tai se voi olla erittäin monimutkainen järjestelmä. Automaattisia testausjärjestelmiä käytetään laajalti elektroniikan tehdasteollisuudessa seuraamaan valmistettavien tuotteiden toimivuutta. [1.]

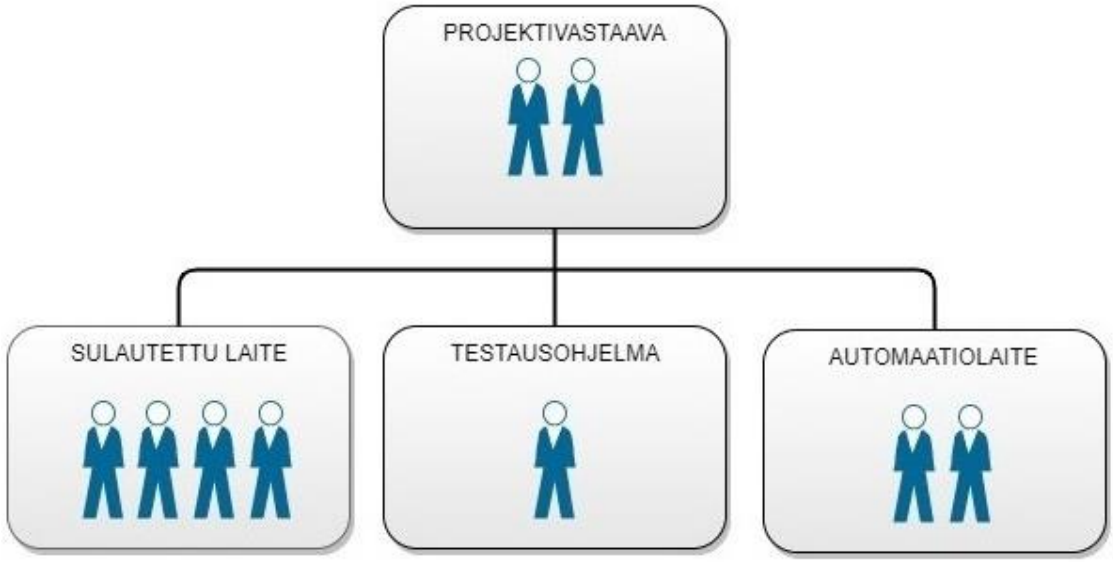
Yksinkertaisimmillaan automaattinen testausjärjestelmä koostuu kolmesta osasta, jotka ovat fyysinen testauslaitteisto, johon testattava DUT-laite (Device Under Test) asetetaan, tietokone tai mikrokontrolleripohjainen testausjärjestelmä sekä itse testausohjelmisto, joka todentaa ja vahvistaa testattavan laitteen toiminnallisuuden. Näiden lisäksi tarvitaan kommunikointiväylä testauslaitteiston ja tietokoneen tai mikrokontrollerin välille. [2.]

Kuvassa 1 havainnollistetaan työssä toteutetun testausjärjestelmän osa-alueet, jotka koostuvat testattavasta tasapainokepistä, testausta suorittavasta testausräkistä ja järjestelmää ohjaavasta testausohjelmasta. Prosessin ideana on, että tasapainokeppi asetetaan testausräkkiin ja testausohjelmalta ajetaan laitteiden alustus, toiminallinen testaus ja mittaustulosten raportointi.



Kuva 1. Insinöörityössä toteutetun testausjärjestelmän osa-alueet.

Testausjärjestelmän toteutus oli laaja projektikokonaisuus, joka jaettiin osa-alueiden mukaisesti kolmeen eri projektiryhmään, jotka havainnollisestaan kuvassa 2. Työn tekijä vastasi koko testausohjelman kehitystyöstä ja laitteiden yhdistämisestä lopulliseen järjestelmään. Sulautetun laitteen ryhmään kuului kolme laiteohjelmistokehittäjää ja yksi elektroniikka- ja mekaniikkasuunnittelija. Automaatiolaitteen kehityksestä vastasi kaksi tietotekniikan sulautettujen järjestelmien opiskelijaa. Projektiryhmät raportoivat omasta alueestaan ja tekemisestään kahdelle vanhemmalle projektivastaavalle.



Kuva 2. Testausjärjestelmän projektikokonaisuuden projektiryhmät.

2.1 Testauksen tarkoitus

Automaattisen testausjärjestelmän tarkoituksena on yleensä mahdollistaa laitteiden nopea testaus tuotantoympäristössä. Automaattista testausta voidaan kuitenkin toteuttaa tuotteen koko elinkaaren ajan erilaisista lähtökohdista. Sitä voidaan käyttää esimerkiksi tuotteen

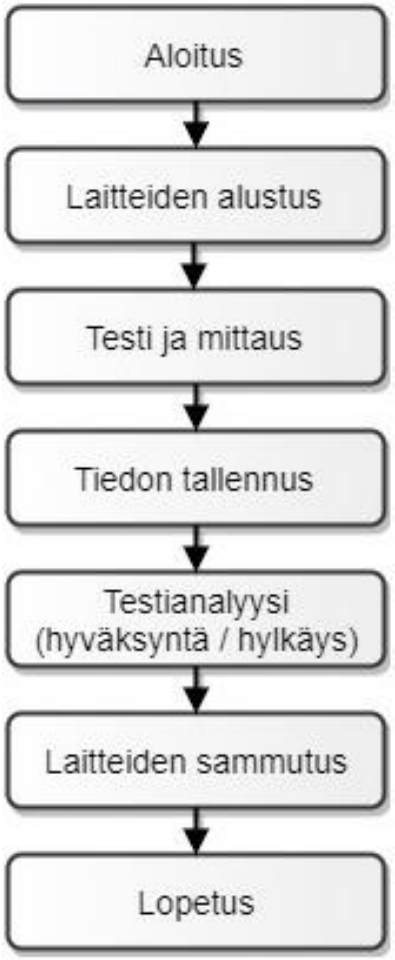
- suunnittelussa
- tuotekehityksessä
- tuotannossa
- luotettavuus- ja sertifiointitesteissä
- huollon yhteydessä.

Testausjärjestelmän avulla haetaan laitteiden testauksesta syntyneiden kustannusten ja ajan säästöjä sekä korkeampaa testausvarmuutta ja isompaa mittausotantaa testauksen aikana. Automaattisella testausjärjestelmällä myös poistetaan ihmisen toiminnasta johtuvan virheen mahdollisuus. [3.]

Työssä toteutetun testausjärjestelmän tarkoituksena oli varmentaa tasapainokeppien eri laitteistoversioiden ja anturien toimivuus ja mittaustulosten raportointi. Kyseessä oli siis lähtökohtaisesti tuotekehityksessä käytettävä testausjärjestelmä. Tasapainokepin toimivuus varmennetaan liikuttamalla testattavaa tasapainokeppiä testausräkissä ennalta määritellyillä liikeradoilla ja vertaamalla tasapainokepin liikeantureista saatuja tuloksia testausräkin kalibroituihin arvoihin.

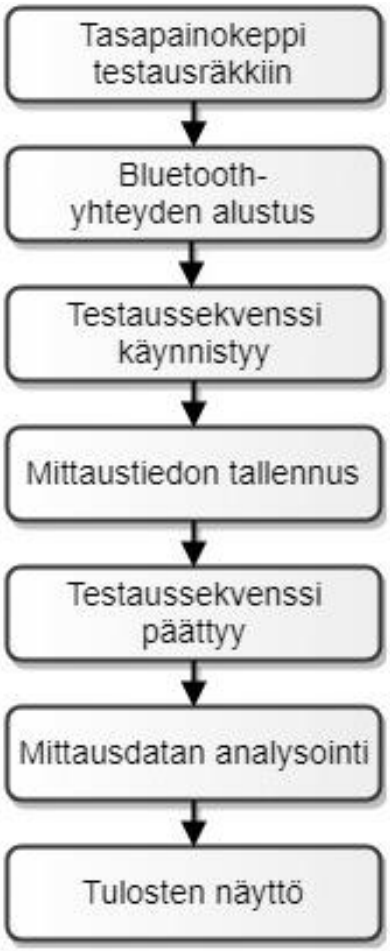
2.2 Testauksen vaiheet

Automaattisten testausjärjestelmien laajuus vaihtelee käyttötarkoituksen mukaan. Tyypillisen testauksen vaiheet voidaan yksinkertaistaa kuvan 3 mukaisesti.



Kuva 3. Tyypillisen ATE-testauksen vaiheet yksinkertaistettuna.

Projektissa tasapainokepin testaus alkaa, kun tasapainokeppi asetetaan testauslaitteistoon eli testausräkkiin. Testausohjelma alustaa Bluetooth-yhteyden järjestelmien välille. Kun yhteys on muodostettu, testausohjelma käynnistää testiräkin testaussekvenssin. Sekvenssin aikana testattavaa tasapainokeppiä liikutetaan testiräkissä ennalta määritetyn liikeradan mukaisesti. Ajon aikana testausohjelma lukee tasapainokepin liikeantureilta saadut mittaustiedot ja tallentaa ne mittausraportiksi. Testiräkin testaussekvenssin päätyttyä testausohjelma vertaa mittaustietoja oletusarvoihin ja laskee näiden erotuksen. Testausohjelma tarkistaa ovatko tasapainokepin liiketiedot sallittujen virhemarginaalien sisällä, ja tulostaa mittaustulokset käyttäjälle. Kuvassa 4 esitetään tasapainokepin testauksen eri vaiheet.



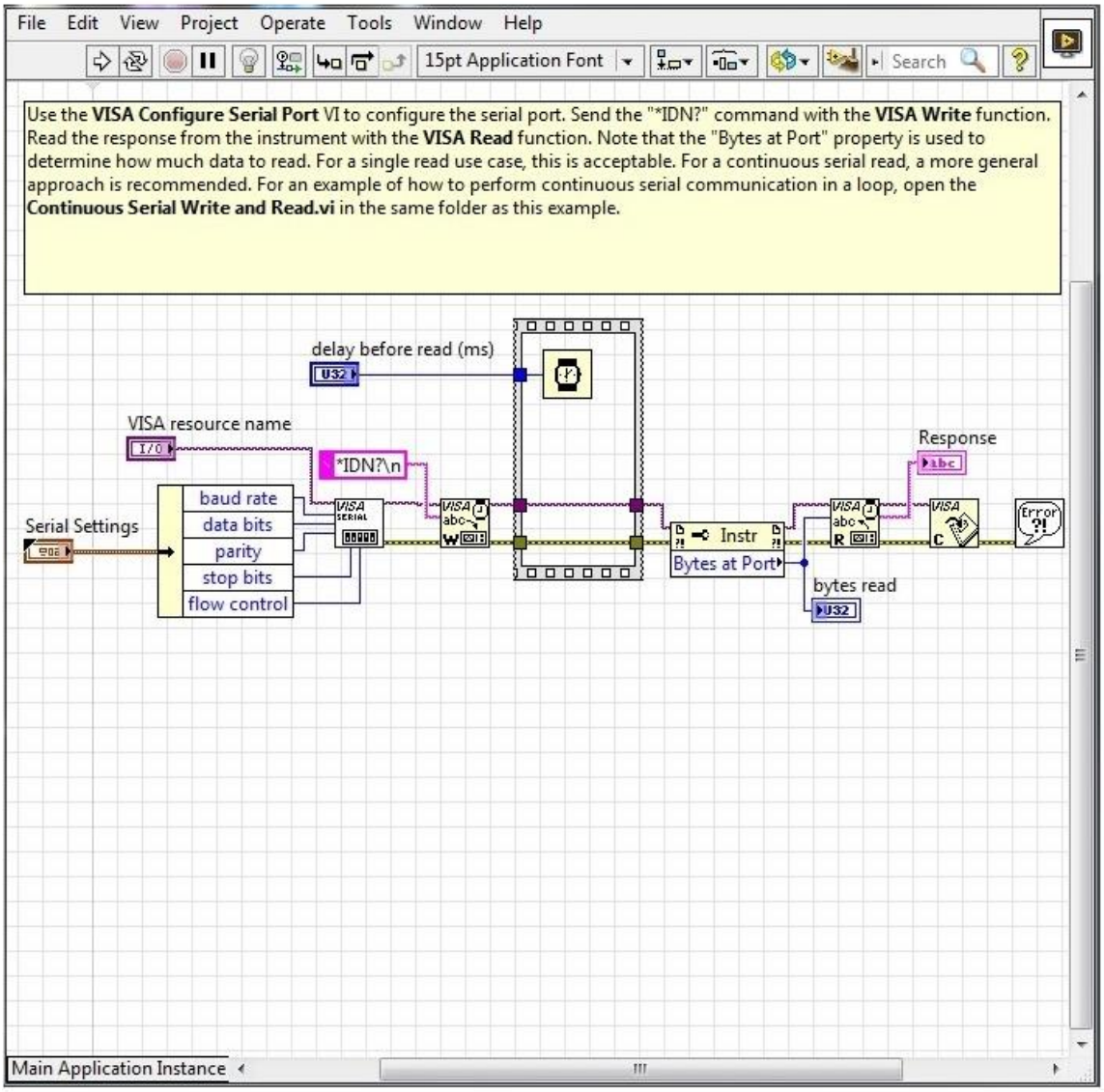
Kuva 4. Tasapainokepin testauksen vaiheet.

2.3 Testauksen ohjelmointi

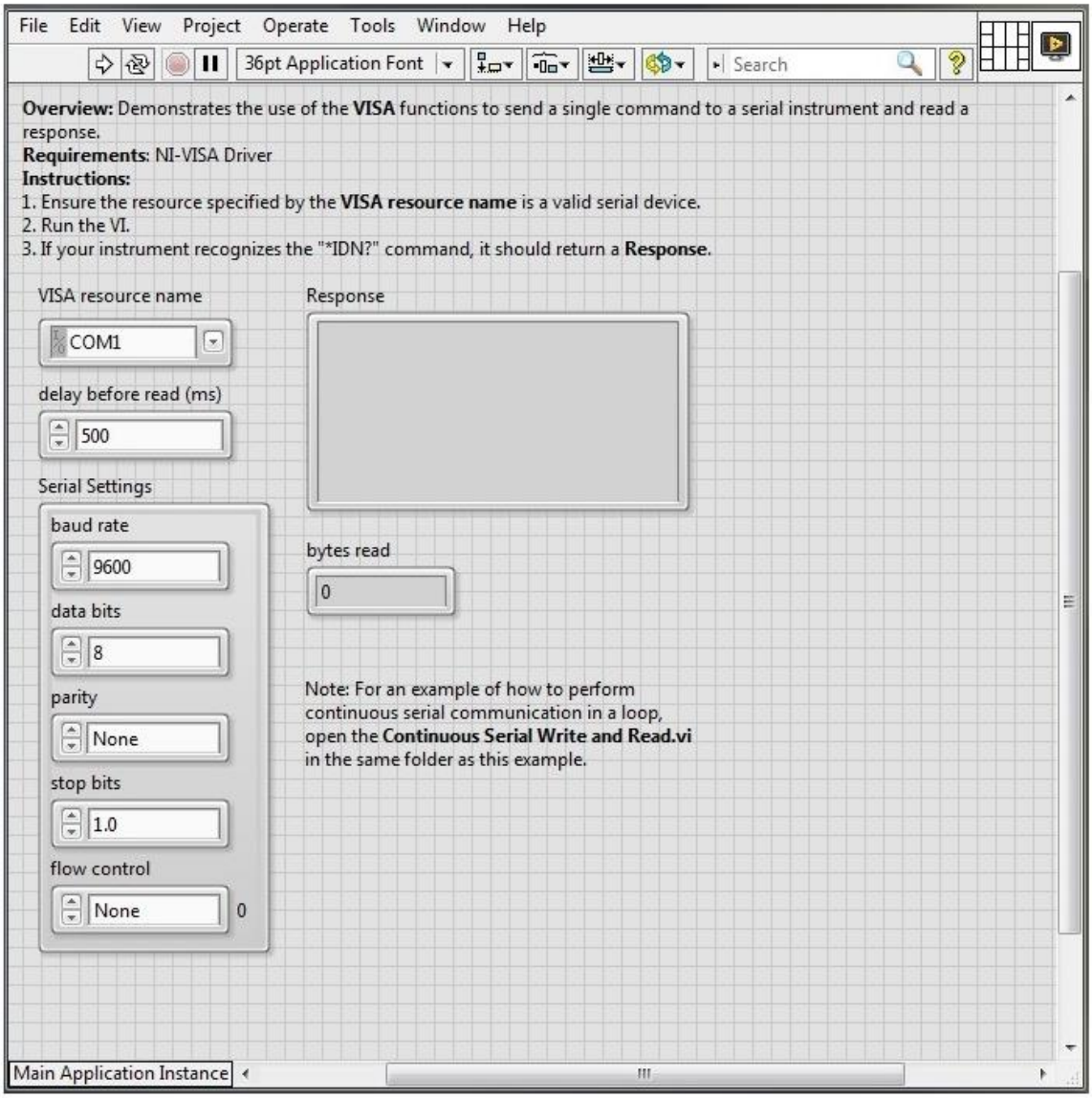
Testausohjelman ohjelmointi voidaan tyypillisesti toteuttaa käyttäen moderneja ohjelmointikieliä, kuten C, C++, Java tai Python. Vaihtoehtoisesti järjestelmän ohjelmointi voidaan toteuttaa ripeästi graafisella ohjelmointikielellä, kuten esimerkiksi National Instrumentsin G-kieleen perustuvalla LabVIEW-ohjelmointiympäristöllä, joka soveltuu erinomaisesti PC-pohjaisen mittaus- ja testaussovelluksen yleisohjelmointikieleksi.

Graafinen ohjelmointiympäristö mahdollistaa ohjelmien rakentamisen vaivattomasti perusohjelmointitaidoilla, ja ohjelmien ymmärtäminen on helppoa loogisesti luettavan graafisen ohjelmoinnin takia. On kuitenkin hyvä tiedostaa, että ohjelmointiympäristön valmiit työkalut ja funktiot vaativat huomattavasti syventymistä ohjelmointiympäristöön ja itse pääohjelman pitäminen helposti luettavana saattaa osoittautua haasteelliseksi isoissa ohjelmakokonaisuuksissa.

Työn testausohjelman kehityksessä päätettiin käyttää LabVIEW-ohjelmointiympäristöä, koska se soveltuu erinomaisesti prototyyppiohjelmien ketterään kehittämiseen ja haastavien rajapintojen helppoon yhdistämiseen. Ohjelmointiympäristöllä toteutettu ohjelmamoduuli on nimeltään virtuaalinen instrumentointi, ja se koostuu kahdesta osasta, jotka ovat lohkokaavio ja etupaneeli. Esimerkki käyttöliittymän lohkokaaviosta esitetään kuvassa 5, ja esimerkki käyttöliittymän etupaneelista esitetään kuvassa 6. Lohkokaavio sisältää ohjelman loogisen toiminnan tai ohjelmakoodin graafisessa lohkokaavion omaisessa muodossa. Etupaneelissa taas toteutetaan ohjelman konkreettinen käyttöliittymä.



Kuva 5. LabVIEW-käyttöliittymän lohkokaavio.



Kuva 6. LabVIEW-käyttöliittymän etupaneeli.

2.4 Testauksen kehitysprosessi

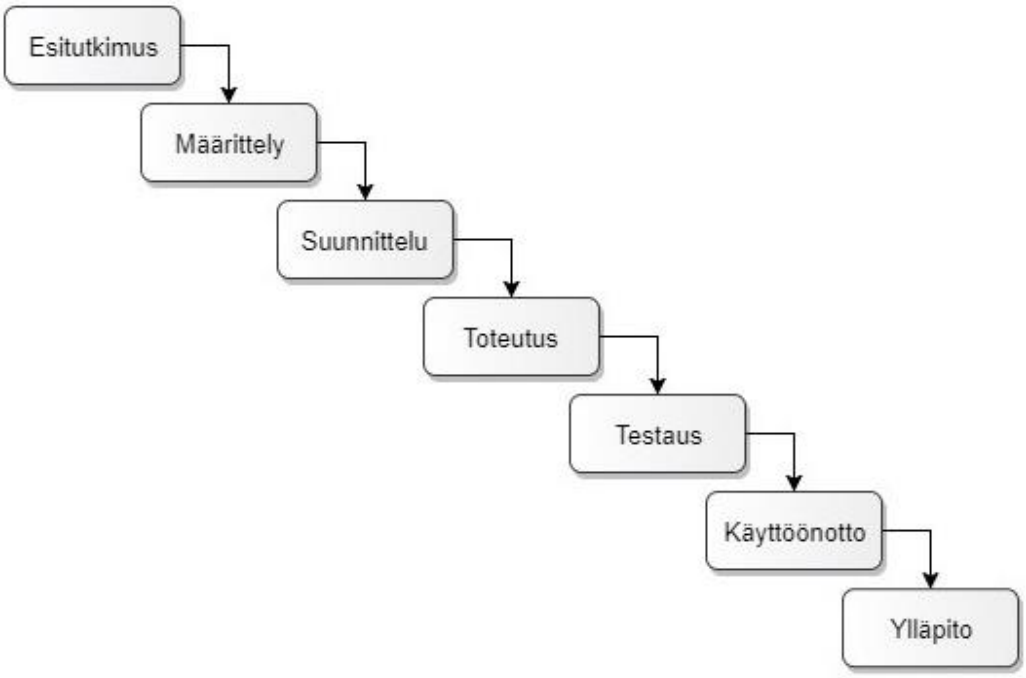
Työn testausohjelman kehitysprosessi voidaan toteuttaa soveltaen useaa erityyppistä ohjelmistotuotantomallia. Kehitysprosessin eri mallit voidaan karkeasti jakaa suunnitelmakeskeiseen (Plan-driven), iteratiiviseen (Iterative and incremental) ja ketterään (Agile) ohjelmistokehitykseen. Työn prototyyppimäisen luonteen ja ohjelmoijan lähtötason perusteella sovelluskehityksessä päädyttiin soveltamaan kehitysmallia, joka eniten muistuttaa mukautettua iteratiivista mallia ja soveltui parhaiten ohjelman nopeaan kehittämiseen.

Kehitysprosessien alle lukeutuu vielä useita eri tavalla toisistaan poikkeavia kehitysmalleja. Oli kyseessä mikä tahansa kehitysmalli, kehitysprosessi sisältää lähes aina seuraavat vaiheet:

1. vaatimusten määrittely
2. suunnittelu
3. toteutus
4. testaus
5. käyttö ja ylläpito.

Suunnitelmakeskeisessä ohjelmistokehityksessä eri vaiheet suunnitellaan ja dokumentoidaan mahdollisimman tarkasti ennen varsinaisen toteutuksen alkua. Ohjelmistokehitys nähdään elinkaarena, joka alkaa vaatimusten määrittelystä ja päättyy ylläpitoon. Ketterässä ohjelmistokehityksessä huolellisen dokumentoinnin sijaan ohjelmistokehitys perustuu suoraan viestintään eri kehitystyön avainhenkilöiden välillä. Itse ohjelmistokehitys tapahtuu lyhyissä iteraatioissa, jotka kukin ovat kuin oma pieni ohjelmistoprojekti. [11.]

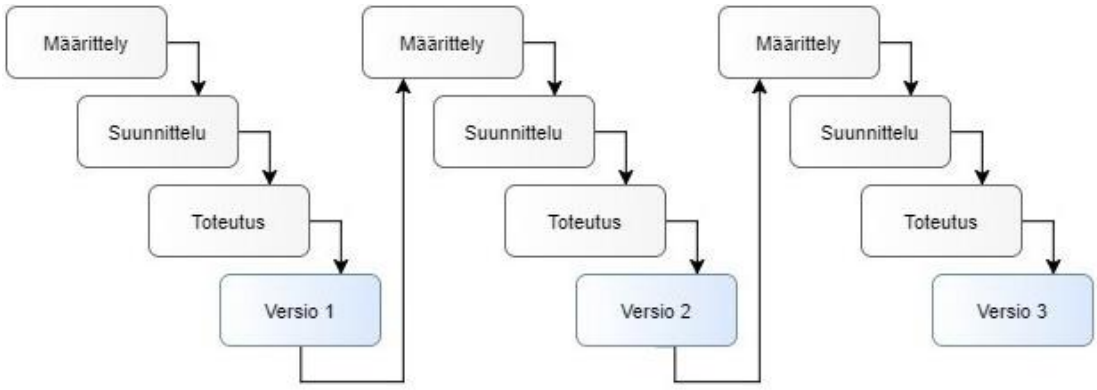
Vesiputousmalli (Waterfall model) on laajasti käytetty suunnitelmakeskeinen kehitysmalli, johon usea muu kehitysmalli pohjautuu. Vesiputousmallissa tavoitteena on toteuttaa ohjelmisto vaiheissa porrastetusti siten, ettei aikaisempiin jo suoritettuihin vaiheisiin enää palata ohjelmistokehityksen muissa vaiheissa. Ohjelmistokehitys on siis jaettu osiin, jotka suoritetaan loppuun ja dokumentoidaan ennen seuraavaan vaiheeseen siirtymistä. Käytännössä kuitenkin ohjelmistokehityksen aikana aikaisempiin vaiheisiin on joissain tapauksissa palattava, eikä vesiputousmallia voi monessakaan tapauksessa hyödyntää aivan sellaisenaan. Parhaiten se soveltuu ohjelmistokehitykseen, jossa jo suunnitteluvaiheessa tiedetään hyvin tarkasti lopputuotteen vaatimukset. Vesiputousmalli sisältää yleensä ainakin määrittely-, suunnittelu- ja toteutusvaiheet. Kuhunkin vaiheeseen kuuluu toimenpiteitä, joilla testataan ja varmistetaan toteutetun vaiheen laatu. [4; 11.] Kuvassa 7 on esimerkki vesiputousmallista.



Kuva 7. Esimerkki ohjelmistokehityksen vesiputousmallista [4].

Vesiputousmallin esitutkimusvaiheessa selvitetään ohjelmistolle asetetut vaatimukset ja se, miten ohjelmisto on mahdollista toteuttaa. Määrittelyvaiheessa täsmennetään ohjelmiston toiminnalliset ja tekniset vaatimukset. Suunnitteluvaiheessa tehdään tekninen suunnitelma ohjelmiston toteutuksesta. Suunnitelma voi koostua useasta dokumentoidusta suunnitelmasta, kattaen muun muassa ohjelmiston teknisen toteutuksen, käyttöliittymän, tallennusratkaisut, testauksen ja käyttöönoton suunnitelmat. Toteutusvaiheessa toteutetaan ohjelmiston ohjelmointi tai muu toteutus ja laaditaan ohjelmiston dokumentointi. Testausvaiheessa testataan toteutettu ohjelmisto huolellisesti mahdollisten ohjelmointivirheiden varalta. Testausvaiheessa voidaan myös testata järjestelmän suorituskykyä tai toimintaa normaaleista poikkeavissa tilanteissa. Käyttöönottovaiheessa valmis ohjelmisto luovutetaan asiakkaalle. Käyttöönottovaiheeseen saattaa kuulua esimerkiksi ohjelmiston asennusta, vanhojen tietojen siirtoa tai käyttäjien koulutusta. Ylläpitovaiheeseen sisältyy käytön aikana havaittujen virheiden korjaus sekä valmiin ohjelmiston muutos- tai laajennustyöt. [4.]

Käytännössä sovelluskehitys hyvin harvoin etenee vesiputousmallin tapaisesti vaiheesta toiseen, sillä lopputuotteen täydellinen määrittely jo sovelluskehityksen määrittelyvaiheessa on usein mahdotonta tai aikaa vievää. Sen sijaan kehitystyö usein tehdään iteratiivisesti, siten että suunnittelu ja toteutus tehdään pienissä osissa pikkuhiljaa muodostaen lopullisen ohjelmiston. Iteratiivisia kehitysmalleja on useita. Vesiputousmalliin pohjautuva EVO-malli (Evolutionary development) käyttää vesiputousmallin porrastettua rakennetta, mutta jakaa ohjelmistokehityksen pienempiin kehitysvaiheisiin, joista jokaisesta muodostuu oma versio ohjelmasta. Uuden version kehitys pohjautuu aina edellisen version kehitystarpeeseen. [4.] Kuvassa 8 on esimerkki tyypillisestä EVO-mallin rakenteesta.



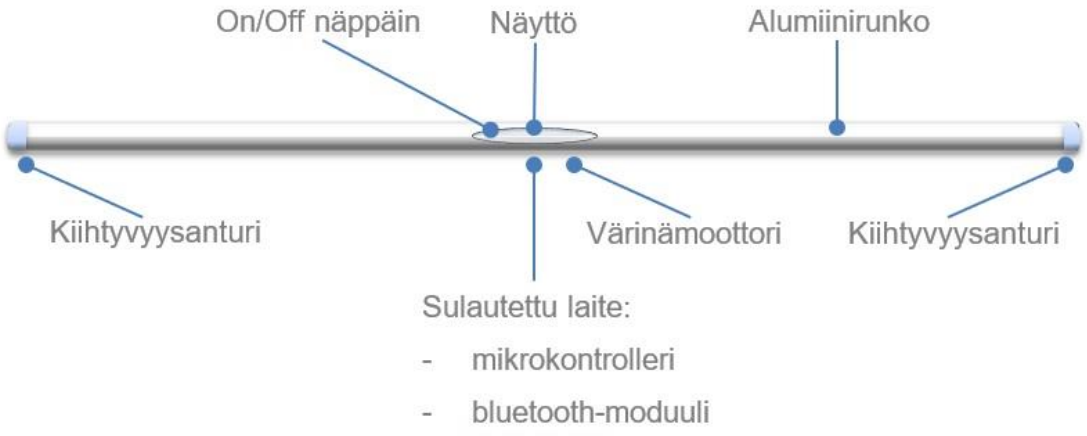
Kuva 8. Esimerkki ohjelmistokehityksen EVO-mallista [4].

Perinteisten kehitysmallien rinnalle on noussut useita ketterän ohjelmistokehityksen kehitysmalleja. Ketterällä ohjelmakehityksellä tarkoitetaan nopeatempoista ja asiakaskeskeistä sovelluskehitystä, joka pystyy paremmin sopeutumaan muutoksiin. Ketterän ohjelmakehityksen ominaispiirteitä ovat suora viestintä asiakkaan kanssa läpi koko ohjelmistokehityksen, kevyt dokumentointi ja ohjelmistokehityksen lyhyet iteraatiot, joiden tavoitteena on aina toimiva ohjelmisto sekä muutokseen sopeutuminen suunnitelman noudattamisen sijaan. Ketterän kehityksen kulmakivenä pidetään ketterän ohjelmistokehityksen julistusta (Manifesto for agile software development), joka määrittelee ketterälle ohjelmistokehitykselle neljä tyypillistä arvoa:

- yksilöitä ja kanssakäymistä enemmän kuin menetelmiä ja työkaluja
- toimivaa ohjelmistoa enemmän kuin kattavaa dokumentaatiota
- asiakasyhteistyötä enemmän kuin sopimusneuvotteluja
- vastaamista muutokseen enemmän kuin pitäytymistä suunnitelmassa [13].

2.5 Testattava laite

Insinöörityössä testattavana laitteena oli asiakasprojektin kuntoilu- ja venyttelyharjoitteluun suunniteltu tasapainokeppi, jossa on sisäänrakennettuna liikeantureita. Anturit havaitsevat tasapainokepin asennon ja liikekulmat, jolloin tasapainokeppi ilmaisee värähtelyllä laitteen ollessa epätasapainossa tai halutussa kulmassa. Tasapainokepin Bluetooth-moduuli välittää liikeantureiden datan testiohjelmalle langattomasti virtuaalista UART-sarjaliikennerajapintaa käyttäen, jolloin tasapainokeppi pystytään havaitsemaan normaalisti kytkettynä testausohjelman sarjaporttiin. Kuvassa 9 havainnollistetaan tasapainokepin tärkeimmät komponentit.



Kuva 9. Tasapainokepin komponentit.

Kuntoilun tai venyttelyharjoittelun aikana tasapainokeppiä voidaan kallistaa, nostaa ja laskea sekä liikuttaa eteen- ja taaksepäin. Tämän lisäksi tasapainokeppiä voidaan kiertää, mutta kepin kiertoa ei kuitenkaan ole otettu toteutuksessa huomioon, sillä kiertoradan mittaus on teknisesti haastava toteuttaa laitteessa käytettävillä kiihtyvyysantureilla.

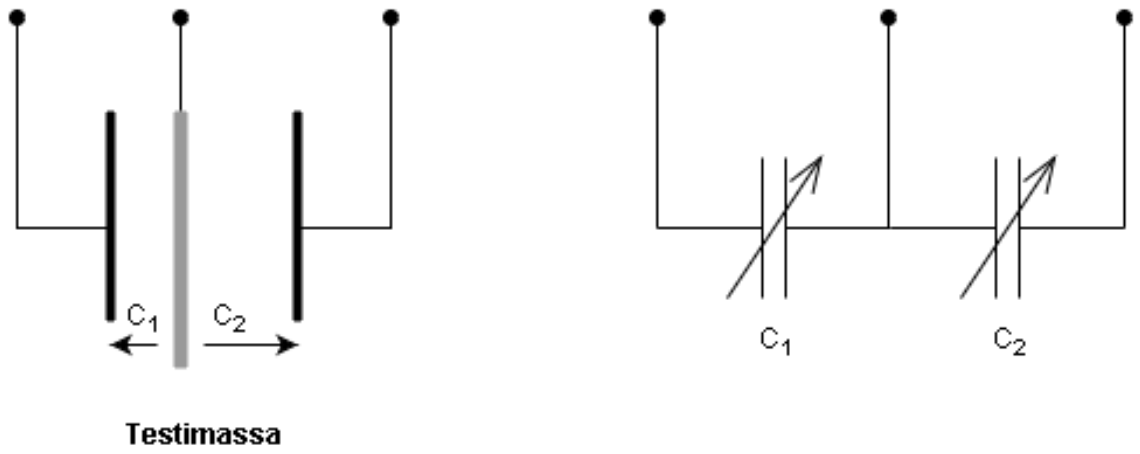
Testausjärjestelmän tarkoituksena on todentaa tasapainokepin toiminnallinen testaus ja se, että liikeanturien mittaustulokset ovat ennalta määriteltyjen viitearvojen sisällä. Todennäköisimmät virheet protovaiheessa johtuvat tasapainokepin sulautetun järjestelmän ohjelmistovirheistä tai kiihtyvyysantureiden kokoonpanon asennusvirheistä. Moduulikohtaiset virheet käydään läpi työn loppuvaiheessa.

2.5.1 Kiihtyvyysanturi

Kiihtyvyysanturi mittaa kiihtyvyyttä eli kappaleen nopeuden muutosta ajassa. Työssä testattu kiihtyvyysanturi oli tyypiltään kolmiakselinen kapasitiivinen MEMS-anturi (Micro Electro Mechanical Converter) eli niin sanottu mikroanturi, joka on vain alle yhden mikrometrin kokoinen. Perinteisin tavoin valmistettuun kiihtyvyysanturiin verrattuna MEMS-antureiden suurin heikkous on kuitenkin lievä epätarkkuus. MEMS-antureiden eduiksi taas luetaan

- edullinen valmistuskustannus
- pieni koko
- kevyt rakenne
- vähäinen energiankulutus
- nopea käynnistysaika. [5.]

Kapasitiivisen kiihtyvyysanturin toiminta perustuu kapasitanssin muutokseen testimassan liikkuessa. Yksinkertaistettu malli kapasitiivisen kiihtyvyysanturin toiminnasta havainnollistetaan kuvassa 10.



Kuva 10. Kapasitiivisen kiihtyvyysanturin yksinkertaistettu malli ja kytkentä [9].

Testimassa, joka toimii samalla kondensaattorin levynä, on joustavasti kiinni rungossa ja pääsee täten liikkumaan kiihtyvyyden aiheuttaman voiman vaikutuksesta. Yhdessä kiinteiden paikallaan olevien levyjen kanssa ne muodostavat kondensaattorin, jonka kapasitanssi muuttuu levyjen välisen etäisyyden muuttuessa. Kahden levyn välinen kapasitanssi voidaan laskea kaavan 1 mukaisesti.

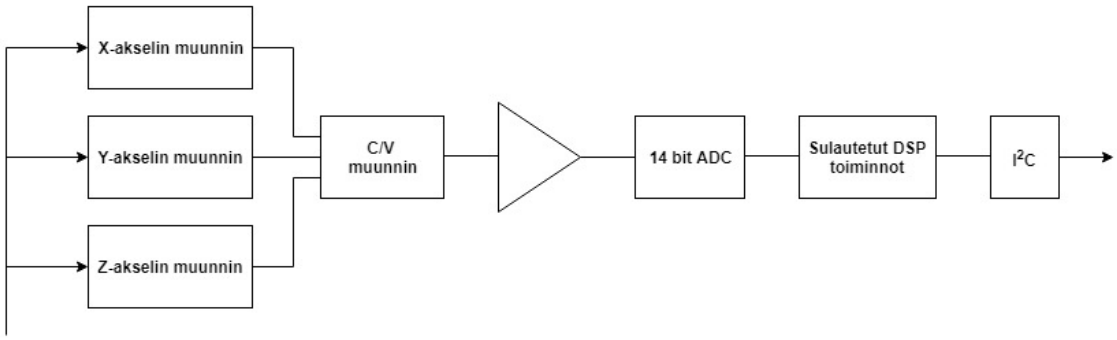
$$C = \epsilon_r \epsilon_0 \frac{A}{d}, \tag{1}$$

jossa C on kapasitanssi, ϵ_r on eristysmateriaalin suhteellinen permittiivisyys, ϵ_0 on tyhjiön permittiivisyys, A on levyjen pinta-ala ja d levyjen välinen etäisyys. Kiihtyvyysanturista saatu mittasignaali on puolestaan suoraan verrannollinen testimassaan vaikuttavaan voimaan F. Kappaleen kiihtyvyys saadaan laskettua käyttämällä hyväkseen Newtonin toista lakia kaavan 2 mukaisesti.

$$F = ma \tag{2}$$

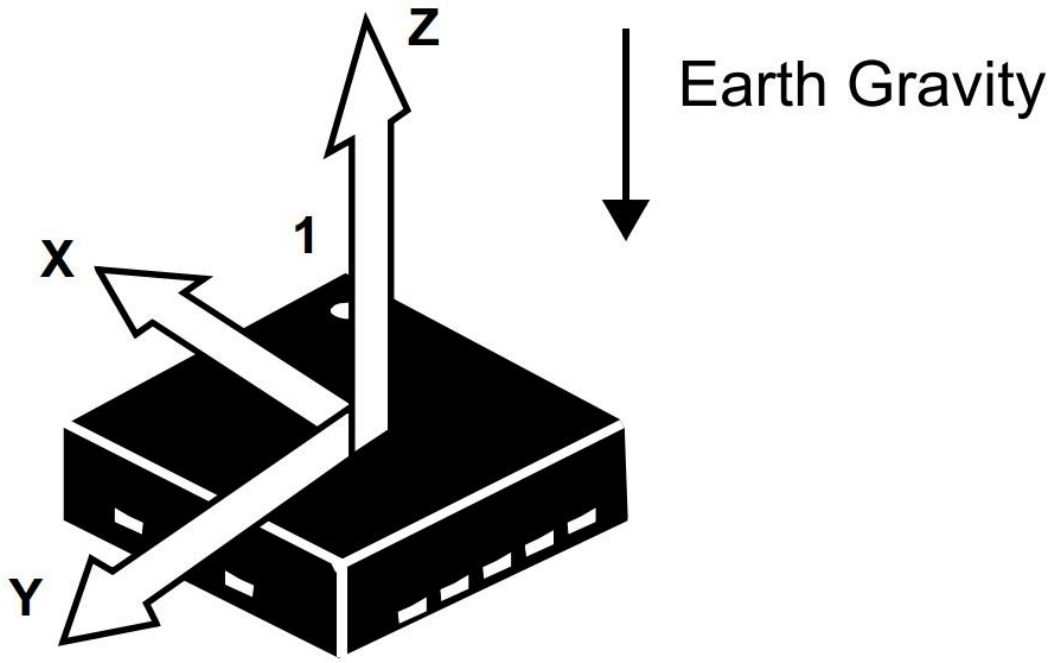
[5.]

Kuvassa 11 havainnollistetaan työssä käytetyn kiihtyvyysanturin yksinkertaistettu lohkokaavio. Kiihtyvyyttä mitataan kolmessa eri akselissa, X, Y ja Z. C/V-muuntimessa (Coulomb to Voltage Converter) kapasitanssimittaus muunnetaan jännitteeksi. Jännitesignaali vahvistetaan jännitevahvistimella ja muunnetaan 14-bittiseksi digitaaliseksi signaaliksi A/D-muuntimessa (Analog to Digital Converter). Digitaalinen signaalikäsittely suoritetaan ja signaali siirretään I^2C (Inter-Integrated Circuit) tiedonsiirtoväylää pitkin tasapainokepin mikrokontrollerille.



Kuva 11. Kiihtyvyysanturin yksinkertaistettu lohkokaavio [8].

Työssä käytetty kiihtyvyysanturi mittaa kiihtyvyyttä kolmelta eri kiihtyvyysakselilta. Mitattavat kiihtyvyysakselit on havainnollistettu kuvassa 12.



Kuva 12. Kiihtyvyysanturin mitattavat akselit [8].

Testauksessa kiihtyvyysanturia käytettiin vain tasapainokepin kallistuskulman mittaamiseksi pituusakselin suhteen eli kepin kallistukseen (roll). Kepin kiertoa (pitch) ei otettu tasapainokepissä ollenkaan huomioon. Kallistuskulma Y-akselin suhteen laskettiin ohjelmallisesti kiihtyvyysanturin eri akselien X, Y ja Z kiihtyvyyksistä kaavan 3 mukaisesti.

$$\alpha = \tan^{-1}\left(\frac{Y}{\sqrt{X^2 \times Z^2}}\right) \times 2\pi \qquad (3)$$

[7.]

Työn testausjärjestelmän kannalta kiihtyvyysanturin mahdollisten mittausvirheiden ymmärtäminen oli oleellista. MEMS-kiihtyvyysantureiden mittausvirheiden lähteitä ovat tyypillisesti

- vakio bias-virhe
- bias-epästabiilius
- valkoinen kohina ja nopeuden satunnaiskävely
- lämpötilan vaikutus
- kalibrointivirheet (skaalaus- ja kohdistusvirheet).

Bias-virhe (bias error) on kiihtyvyysanturin vakio poikkeama todellisesta arvosta. Bias-virheen arviointia hankaloittaa gravitaatiovoima, joka näkyy osana anturin bias-virhettä. Virhettä määrittäessä onkin tärkeä tietää laitteen kulma gravitaatiokenttää nähden. Helpoiten tämä onnistuu kiinteässä testausjärjestelmässä, jossa mitattavaa laitetta pystyy kääntämään hallitusti.

Bias-stabiiliudella (bias-stability) tarkoitetaan bias-jännitteen vakautta ajan suhteen. Bias-epästabiilius aiheutuu 1/f-häiriöstä (flicker noise), joka näkyy bias-virheen ryömimisenä. 1/f-häiriö on elektronisista komponenteista johtuvaa matalataajuista häiriötä, joka kasvaa taajuuden pienentyessä. Valkoista kohinaa (White noise) syntyy, kun laitteen sisäinen elektroniikka aiheuttaa häiriötä kiihtyvyysanturin kapasitanssin mittaukseen. Valkoinen kohina koostuu täysin satunnaisista normaalijakautuneista muuttujista, joiden odotusarvo on nolla ja varianssi on äärellinen. Kun valkoista kohinaa integroidaan, syntyy ajan suhteessa kasvavaa nopeuden satunnaiskävelyä VRW (Velocity Random Walk).

Kohinan suuruus määrittää sensorin pienimmän erotettavissa olevan mittauksen. Ympäristön ja itse laitteen lämpenemisestä johtuva lämpötilan muutos vaikuttaa anturin bias-virheen tasoon usein epälineaarisesti. Kalibrointivirheitä ovat skaalaus- ja kohdistusvirheet (Scale factor / alignment error). Niistä johtuva virhe näkyy osana bias-virhettä vain laitteen kiihdyttäessä. Skaalausvirhe kasvaa mitattavan kiihtyvyyden kasvaessa, sen vaikutus on sitä isompi, mitä laajempi mitattava kiihtyvyysalue on. [5; 6.]

Kiihtyvyysanturin kohdistustarkkuuteen vaikuttaa anturin kohdistusvirheiden lisäksi anturia paikallaan pitävän mekaanisen rakenteen vakaus käytön aikana. Työssä käytetty kiihtyvyysanturi tulee valmiiksi asennettuna kiihtyvyysanturimoduuliin. Itse kiihtyvyysanturin asennuksesta johtuvaan kohdistusvirheeseen ei siis pystytty vaikuttamaan tasapainokepin kokoamisvaiheessa. Kiihtyvyysanturimoduuli puolestaan kiinnitetään ruuveilla

asennusalustaan. Asennusalustan epätasaisuudesta ja itse asennustyöstä johtuvat virheet voivat näkyä kohdistusvirheenä. Tämän lisäksi havaittiin, että asennustyön aikana kiihtyvyysanturimoduuli on altis rikkoutumiselle, jos kiinnitysruuvit ruuvataan epätasaisesti kiinni. Tämä voi aiheuttaa jännitystä kiihtyvyysanturimoduulin piirilevyllä, mikä puolestaan saattaa rikkoa komponenttien juotoksia samalla tehden koko kiihtyvyysanturimoduulista käyttökelvottoman.

Projektissa toteutetun testausjärjestelmän ensisijainen tehtävä on todentaa tuotekehityksen aikaiset virhelähteet. Koska virhelähteiden syntymekanismeja on niin monenlaisia, tulee itse testausohjelma toteuttaa siten, että kaikki mahdolliset virhelähteet pyritään ottamaan huomioon. Todennäköisimmät ja oleellisimmat virhelähteet ovat kohdistusvirhe ja bias-virhe.

2.5.2 Bluetooth-moduuli

Tasapainokepissä on BL600-SA-Bluetooth-moduuli, joka on Laird Technologiesin valmistama pienikokoinen ja tehoa säästävä ohjelmoitava moduuli. Testausohjelma-PC:n rajapinta on puolestaan varustettu Laird Technologiesin valmistamalla BL620-USB-dongella. Tasapainokepin Bluetooth-moduuli ja testausohjelma PC:n USB-dongle ovat tyypiltään BLE (Bluetooth Low Energy), joka klassiseen Bluetoothiin verrattuna säästää laitteen energiankulutusta pelkistämällä ja vähentämällä tiedonsiirtoa. Tiedon kokoaikaisen lähetyksen sijaan BLE välittää vain laitteen tilatietoja määrätyin väliajoin, jolloin yhteys on muuna aikana lepotilassa (sleep mode).

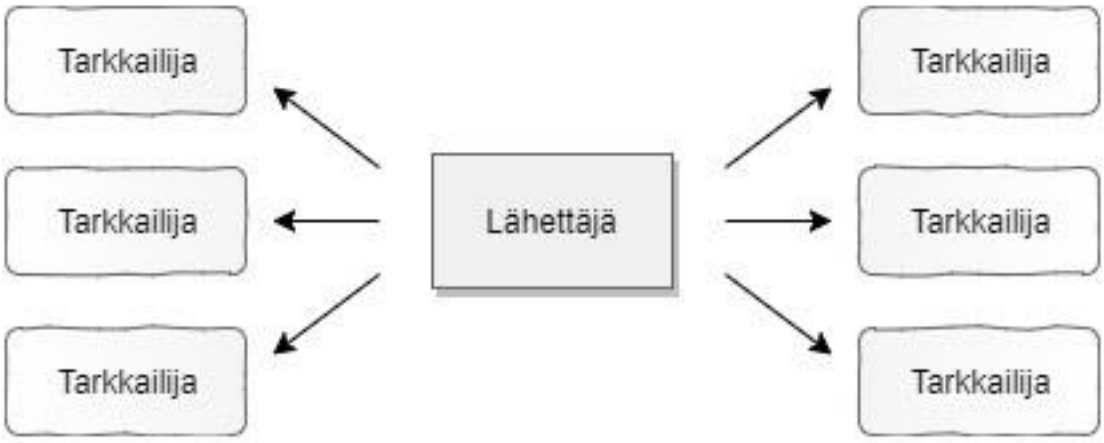
Järjestelmän tiedonsiirto toteutettiin langattoman Bluetooth-yhteyden välityksellä UART-sarjaliikennerajapintaa käyttäen. Järjestelmän tiedonsiirron periaatekaavio havainnollistetaan kuvassa 13.

Kuva 13. Järjestelmän Bluetooth-tiedonsiirron periaatekaavio.

Tiedonsiirto testausohjelman ja USB-donglen välillä tapahtuu virtuaalista UART-sarjaliikennerajapintaa käyttäen. USB-donglen ja tasapainokepin Bluetooth-moduulin välinen tiedonsiirto tapahtuu Bluetoothin välityksellä, ja se käyttää hyväkseen yleistä tiedonsiirtometodia nimeltä VSP (Virtual Serial Port). Laird Technologies on kehittänyt erityisesti oman VSP-protokollan BLE-laitteiden väliseen tiedonsiirtoon. Klassisesta Bluetooth-standardista poiketen, BLE-standardi ei määrittele valmista rajapintaa sarjaliikenteelle, vaan antaa laitevalmistajien määritellä omat toteutuksensa. USB-dongleen on lisäksi ohjelmoitu serial to vSP bridge, joka siirtää sarjaliikenteen vSP:hen.

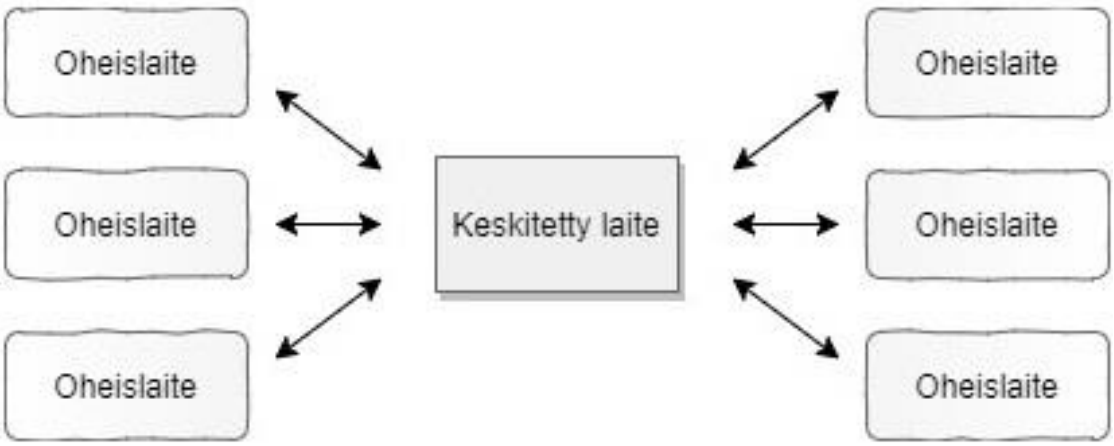
2.5.3 Bluetooth BLE-teknologia

BLE pohjautuu Nokian kehittämään langattomaan tiedonsiirtotekniikkaan nimeltä Wibree. BLE:n kehityksen lähtökohtana oli luoda mahdollisimman energiatehokas lyhyen matkan langaton likiverkkotekniikka. BLE-laite voi siis kommunikoida joko Broadcast-tyyppisellä tai Connections-tyyppisellä tiedonsiirrolla. Kun BLE-laite kommunikoi Broadcast-tyyppisellä tiedonsiirrolla, mikä tahansa lähellä oleva laite voi vastaanottaa laitteen lähettämän tiedon. Tiedonsiirto kuitenkin tapahtuu vain yhteen suuntaan lähettäjän (broadcaster) ja tarkkailijan (observer) välillä. Lähettäjä lähettää tietyn aikavälein mainospaketteja kaikille datan haluavalle laitteille. Tarkkailija puolestaan skannaa tiettyä taajuutta mainospakettien varalta. [10.] Kuvassa 14 havainnollistetaan Broadcast-tyyppinen tiedonsiirto.



Kuva 14. Broadcast-tyyppinen tiedonsiirto.

Broadcast-tyyppinen tiedonsiirto on käytännöllinen, kun halutaan samanaikaisesti lähettää dataa usealle laitteelle kerrallaan tai kun halutaan lähettää vain pieni määrä dataa tietyin aikavälein. Sen rajoitteena on kuitenkin tietoturva, sillä mikä tahansa laite voi vastaanottaa lähetettävän datan. Lisäksi lähetettävä mainospaketin tietosisältö voi olla kooltaan vain 31 tavua, joskin BLE mahdollistaa toisen 31 tavun kokoisen mainospaketin lähetyksen tarkkailijan pyynnöstä. Connections-tyyppistä tiedonsiirtoa käytetään, kun tiedonsiirron tulee tapahtua kahteen suuntaan tai kun lähetettävän datan koko ylittää 62 tavua. Connections-tyyppinen tiedonsiirto on kahden laitteen välistä pysyvää ja jaksoittaista datan lähetystä. Kuvassa 15 havainnollistetaan Connections-tyyppinen tiedonsiirto. [10.]



Kuva 15. Connections-tyyppinen tiedonsiirto.

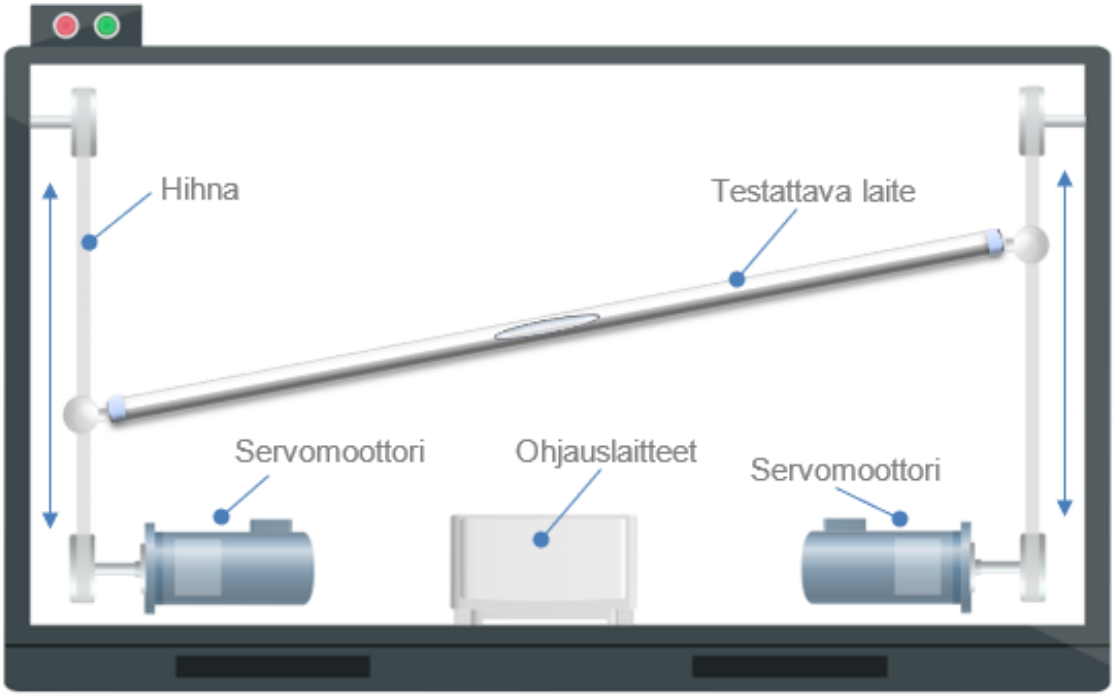
Laite voi toimia joko keskitetyssä roolissa (master) tai oheisessa roolissa (slave). Keskitetyssä roolissa laite skannaa mainospaketteja ennalta-asetetuilta taajuuksilta ja muodostaa Connections-tyyppisen yhteyden mainospaketin lähettäjän kanssa. Se myös käynnistää jaksoittaisen tiedonsiirron ja hallinnoi tiedonsiirron ajoitusta. Oheisessa roolissa oleva laite lähettää tietyin aikavälein mainospaketteja kaikille datan haluavalle laitteille ja hyväksyy Connections-tyyppisen tiedonsiirron. Yhteyden muodostumisen jälkeen se lopettaa mainospakettien lähettämisen ja kommunikoi keskeisessä roolissa olevan laitteen kanssa. [10.]

Projektissa käytettiin Connections-tyypistä tiedonsiirtoa, sillä se mahdollistaa tiedon striimauksen, ja tiedonsiirron molempiin suuntiin. Tasapainokepin Bluetooth-moduuli toimii oheisessa roolissa ja testausohjelman PC keskitetyssä roolissa.

2.6 Testauslaitteisto

Insinöörityössä testauslaitteistona käytettiin automatisoitua testausräkkiä, jonka tehtävänä oli suorittaa testaussekvenssejä testattavalle laitteelle testausohjelman ohjauskäskyjen mukaisesti. Testaussekvenssi suorittaa ennalta määriteltyjä fyysisiä liikeratoja, joiden tarkoitus oli simuloida käyttäjän suorittamia harjoitusliikkeitä testattavalle tasapainokepille.

Testausräkki sisälsi mikrokontrollerin, servo-ohjaimia ja servomoottoreita. Servo-ohjaimella ohjataan servomoottoreita kuvan 16 mukaisesti haluttuun asentoon testaussekvenssin antamien parametrien mukaisesti. Lisäksi testausräkissä oli käytössä aikaisemmin työssä mainittu Bluetooth-moduuli, jonka avulla kommunikointi tapahtuu testausohjelmalle UART-sarjaliikennerajapinnan välityksellä.

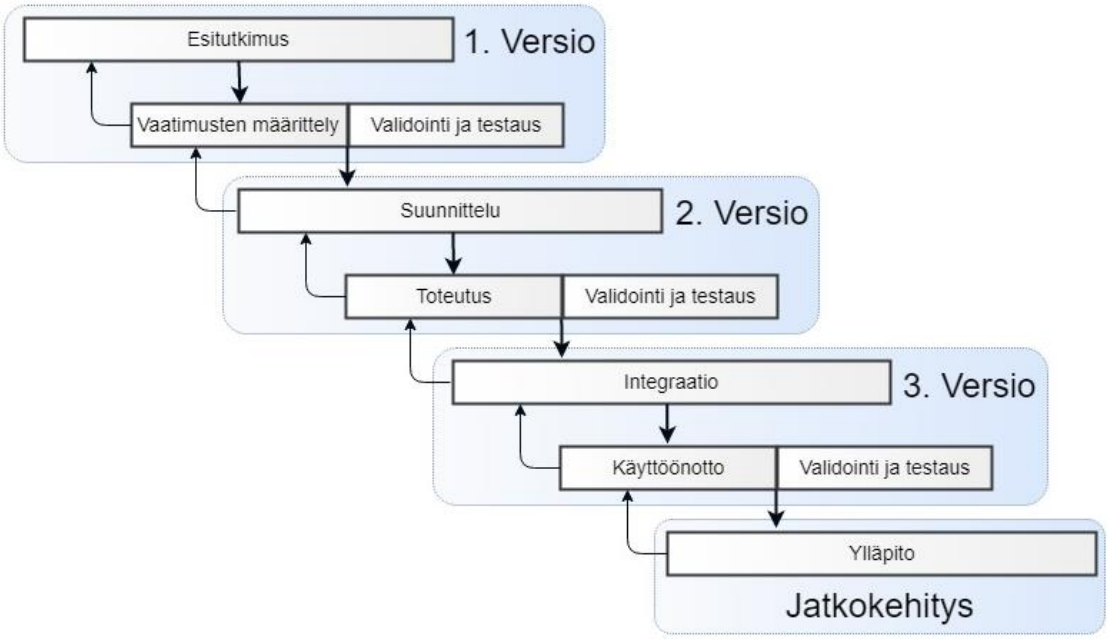


Kuva 16. Testauslaitteiston komponentit.

3 LabVIEW-testausohjelma

Työssä toteutetun testausjärjestelmän testausohjelman sovelluskehityksessä lähtökohtana oli toteuttaa testausohjelma, jonka lopullisia toiminnallisia ja teknillisiä vaatimuksia on esitutkimusvaiheessa vaikea selvittää. Teknistä toteutusta oli haastava arvioida vielä suunnitteluvaiheessa ohjelmoijan osaamistason ja mahdollisten testaustarpeiden mukaan. Tämän vuoksi käytettävässä sovelluskehityksessä päädyttiin soveltamaan kehitysmallia, joka eniten muistuttaa EVO-mallin ja ketterän kehityksen yhdistelmää. Projektin ohjelmistokehitys jaettiin useaan pienempään kehitysvaiheiseen, joista jokaisesta muodostui oma toimiva sovellus, jotka pohjautuivat edellisiin kehitysvaiheisiin. Kehityksen aikaista suunnittelua ja dokumentointia pyrittiin pitämään mahdollisimman minimaalisena. Tavoitteena oli lisätä sovelluksen sopeutumista jatkuvasti kehittyviin ja muuttuviin vaatimuksiin.

Projektin kokeellisen luonteen ja vianetsinnän laajuuden takia suunnittelutyö toteutettiin soveltaen kuvan 17 mukaista muunneltua vesiputousmallia. Ohjelmointiympäristönä LabVIEW oli ohjelmoijalle melko tuntematon, ja siksi iso osa suunnittelutyöhön käytetystä ajasta kului oppimisprosessiin ja etenkin prototyyppilaitteiston vianetsintään.



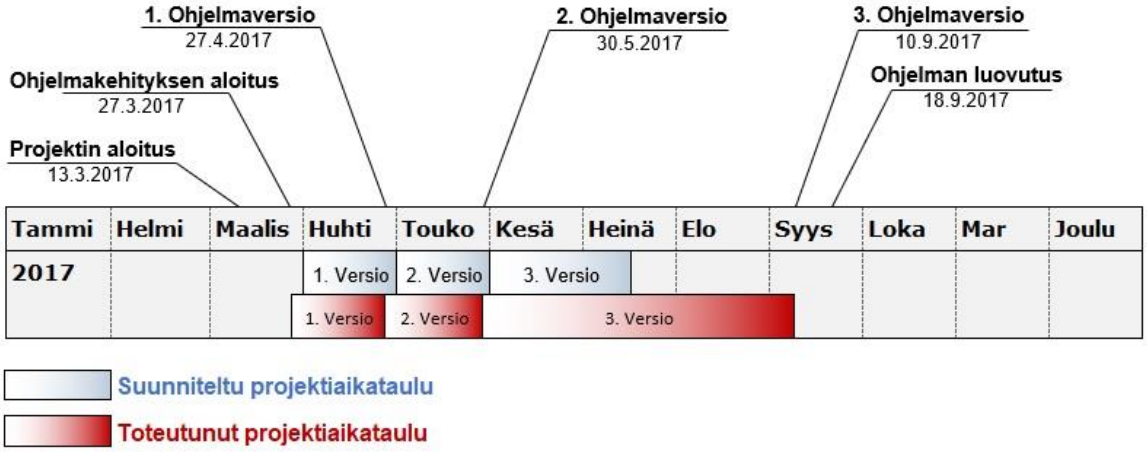
Kuva 17. Insinöörityöprojektin muokattu vesiputousmalli.

Sovelluksen tärkeimmät suunnitteluperusteet olivat prototyyppivaiheen nopea kehitystyö ja rajapintojen vaivaton yhteensovittaminen. Työn alkuvaiheessa sovittiin alustavasti, että kyseessä on testausohjelma, jolla pyritään suorittamaan yksinkertainen laiteohjaus ja mittausdatan käsittely. Ohjelmalla luodaan yhteinen sovellusrajapinta järjestelmien välillä, ja sen kautta suoritetaan laitteiden

- yhteyden alustaminen
- järjestelmien yhdistäminen
- ennalta määriteltyjen testaussekvenssien suorittaminen
- laitteiden mittaustulosten käsittely
- mittaustulosten raportointi.

3.1 Ohjelman suunnittelu ja toteutus

Testausohjelman suunnittelun lähtökohta oli toteuttaa ohjelma kolmessa eri vaiheessa. Ohjelmakehityksen aikajana on hahmotettu kuvassa 18.



Kuva 18. Insinöörityön ohjelmakehityksen aikajana.

Käytännössä ohjelmasta toteutettiin kolme täysin eri versiota, joista jokaisen toteutus pyrittiin aloittamaan puhtaalta pöydältä, aikaisempaa kehitystyötä ja toteutettuja moduuleja soveltaen. Kehitystyön kolme vaihetta ovat

1. yhteyden alustus
2. logiikan ohjelmointi
3. prototyyppiohjelma.

Ensimmäisen vaiheen aikana keskityttiin selvittämään, onko testausohjelma ylipäätään mahdollista toteuttaa annettujen resurssien, laitteiston ja ohjelmointiympäristön puitteissa. Vaiheessa siis perehdyttiin etenkin käytettävään laitteistoon rautatasolla ja siihen miten ohjelmistot saataisiin alustettua yhteiselle ohjelmistorajapinnalle. Lopullisen testausohjelman arkkitehtuurin ja vianetsinnän rakenteeseen ei vielä tässä vaiheessa haluttu ottaa kantaa. Aluksi luotiin hyvin yksinkertainen ohjelmistoversio, jonka avulla pyrittiin havainnollistamaan ja selvittämään järjestelmien toimivuus sekä yhteensopivuus. Tämä ohjelmistoversio toimi sovellusselvityksenä (Proof of concept) LabVIEW-ohjelmointiympäristölle ja käytettäville rajapinnoille.

Toisen vaiheen tavoite oli luoda testausohjelman karkea rakenne sekä kehittää työkalut ohjelmoinnin ja laitteiston vianetsintää varten. Ohjelman logiikka toteutettiin yksinkertaisella tilakoneella (State machine) ja tapahtumakohtaisella ohjelmoinnilla (Event-driven programming), jolloin jo valmiiksi toteutettuja ohjelmamoduuleja voitiin helposti lisätä jälkikäteen itse pääohjelmaan. Ohjelman toiminnallisuuden varmentaminen oli toisessa vaiheessa avainasemassa, joten toimivuutta pyrittiin testaamaan emuloimalla kaikkia järjestelmän osia, samalla poissulkien mahdolliset laitteistokohtaiset vikatilat.

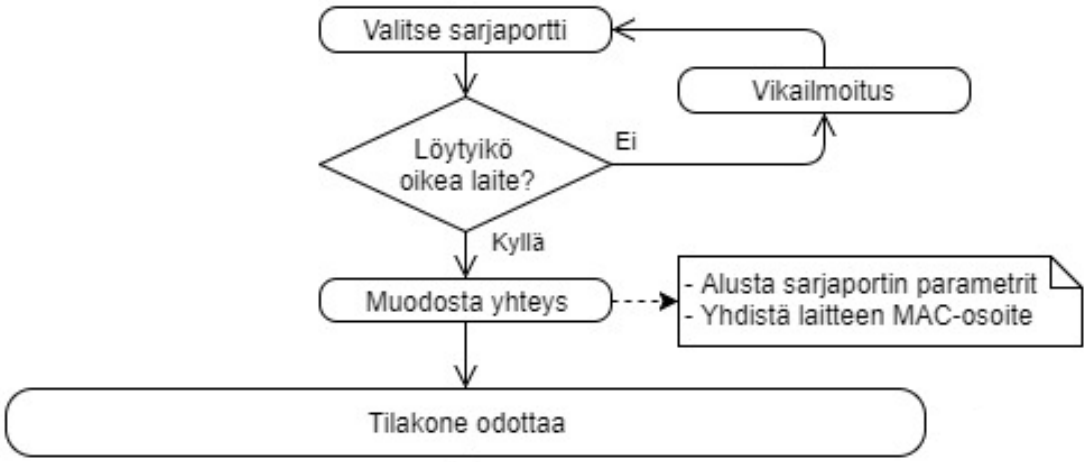
Kolmannessa vaiheessa yhdistettiin aikaisempien vaiheiden ohjelmamoduulit viimeisteltyyn demoversioon. Viimeisessä vaiheessa suoritettiin niin sanottu integrointi todelliseen käyttöympäristöön ilman järjestelmän osa-alueiden emulointia. Ohjelmalle luotiin pelkistetty käyttöliittymä eli UI (User interface), jonka kautta käyttäjä suorittaa järjestelmän ohjauksen. Käyttöliittymään kehitettiin lisäksi lisäominaisuutena virheenkorjausosio (Debugging) mahdollisia laitteisto- ja ohjelmistokohtaisia virheilmoituksia varten. Lisäksi tässä vaiheessa alustettiin mahdollisia kehitystarpeita jatkokehitysprojektia varten.

3.1.1 Yhteyden alustus

Ohjelman varsinainen kehittäminen aloitettiin perehtymällä laitteiston Bluetooth- ja virtuaaliseen UART-sarjaliikennerajapintaan ja siihen miten halutut testausjärjestelmän funktiot voitaisiin toteuttaa LabVIEW-ohjelmointiympäristössä. Yhteyden alustaminen LabVIEW-ohjelmalla koostui seuraavista vaiheista:

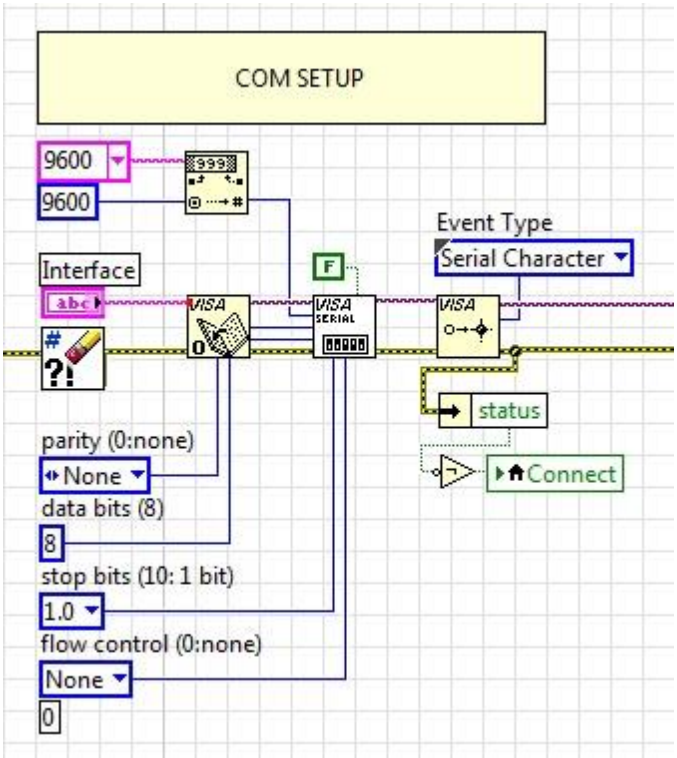
1. Alustetaan Bluetooth-dongle.
2. Asetetaan sarjaliikenneparametrit.
3. Etsitään oikea laite.
4. Yhdistetään MAC-osoite.
5. Rajapinta odottaa.

Järjestelmien rajapinnan alustusta kuvaava UML-kaavio (Unified Modeling Language) esitetään kuvassa 19. Alustuksen tuloksena tilakone jää odottamaan ohjelmasta suoritettavia komentoja.



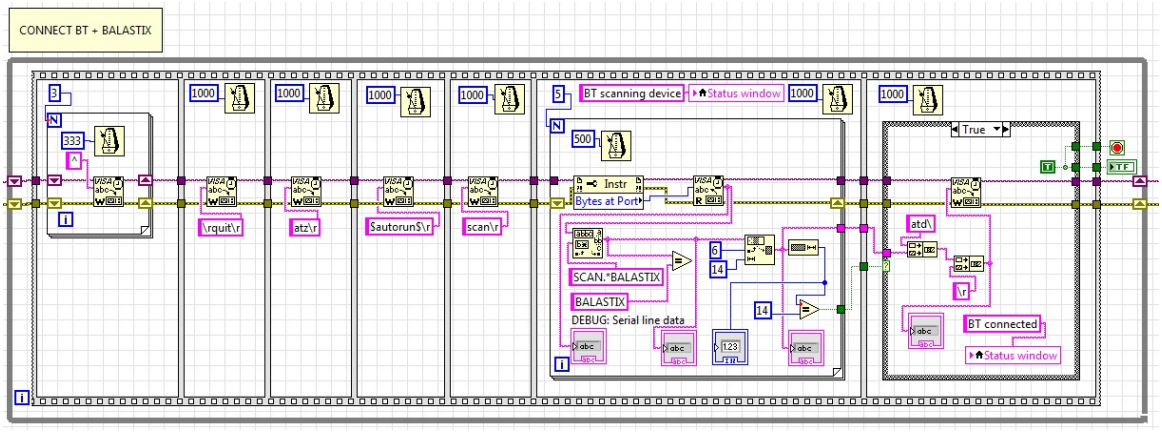
Kuva 19. Yhteyden alustus.

Sarjaliikennettä varten LabVIEW-ohjelmointiympäristössä on valmis funktio, joka hoitaa sarjaliikenteen alustuksen ja avaamisen. Ohjelmassa tulee kuitenkin määritellä sarjaliikenteen asetukset, kuten käytettävä COM-portti (Communications-port), siirtonopeus ja tiedonlähetysasetukset. Kuvassa 20 havainnollistetaan työssä käytettävä LabVIEW-lohkokaavion funktio, jolla työn sarjaliikenteen alustaminen ratkaistiin.



Kuva 20. LabVIEW-yhteyden alustaminen.

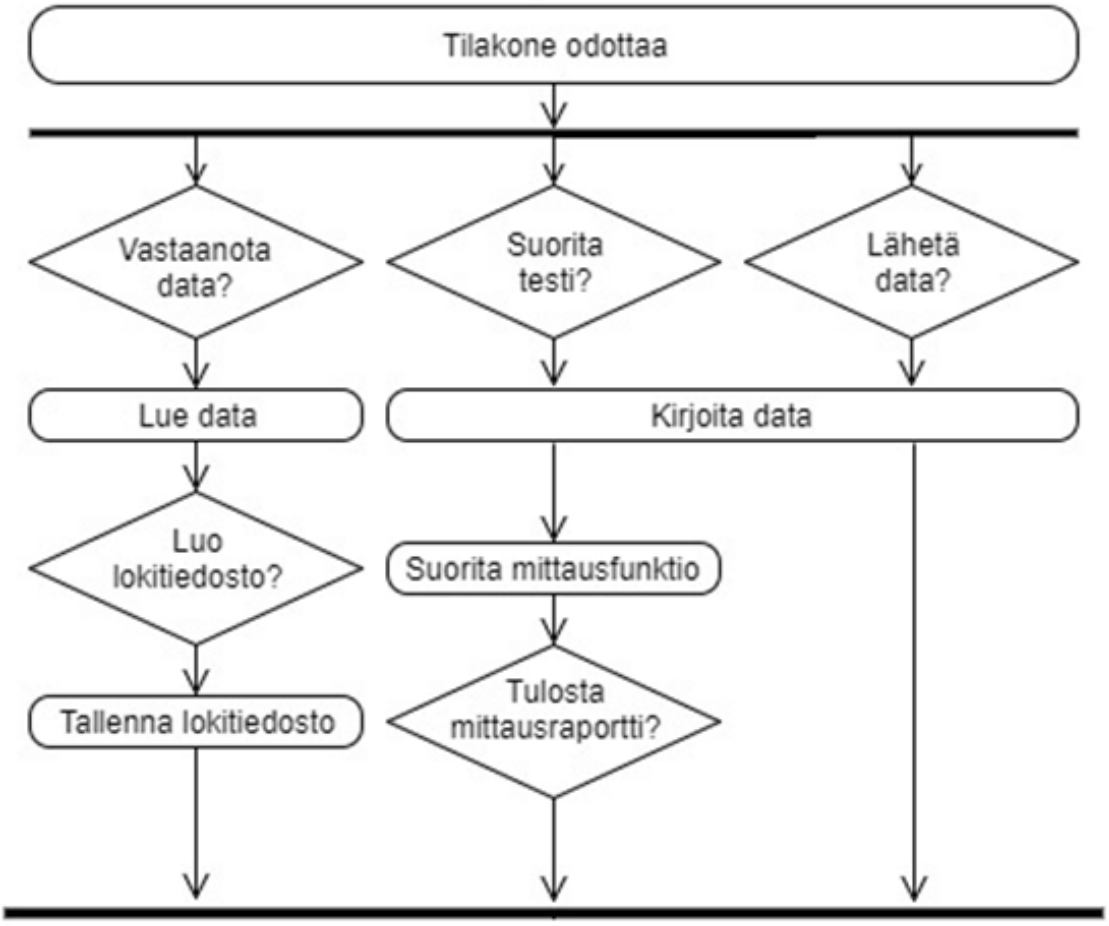
Tasapainokepin projektista oli saatavilla Laird BL620-Bluetooth-moduulin alkulatausohjelma (Bootloader), jonka lähdekoodista pystyttiin päättelemään, miten virtuaalisarjaliikenteen muodostaminen laiterajapintojen välillä tapahtui. Alkulatausohjelman lähdekoodia ei varsinaisesti sellaisenaan pystytty käyttämään, vaan virtuaalisarjaliikenteen alustaminen ratkaistiin käyttäen kuvan 21 LabVIEW-lohkokaavion funktioita.



Kuva 21. LabVIEW-virtuaalisarjaliikenteen alustaminen.

3.1.2 Logiikan ohjelmointi

Ensimmäinen karkea ohjelmaversio toteutettiin ajatellen pelkästään laitteiden yhdistämistä LabVIEW-ohjelmarajapintaan, eikä se varsinaisesti soveltunut sellaisenaan muiden vaadittavien ominaisuuksien toteutukseen. Tämän vuoksi seuraava ohjelmaversio toteutettiin ensisijaisesti arkkitehtuuri edellä ja moduuli kerrallaan. Kriittisiin moduuleihin kuului datan lukeminen ja lähettäminen sekä vianetsintätyökalut. Ohjelman logiikka perustuu kuvan 22 mukaiseen tapahtumapohjaiseen tilakoneeseen, johon on helppo liittää jo aikaisemmin kehitettyjä ja testattuja moduuleja.



Kuva 22. Tilakoneen logiikka.

Ohjelman kehittäminen tässä kehitysvaiheessa vaati huomattavasti enemmän laitteistoon syventymistä ja vianetsintää, joka kulutti arvioitua enemmän resursseja itse ohjelmistokehityksestä. Ohjelman testaus toteutettiin näin ollen emuloimalla koko järjestelmää virtuaalisessa ympäristössä ennalta asetetuilla komennoilla ja ilman fyysisiä laitteita. Lisäksi tässä vaiheessa tuli huomioida ohjelman testaukseen vaadittavien vianetsintätyökalujen suunnittelu ja toteutus, jotta lopullisessa versiossa voitiin tarkkaan paikantaa laitteistosta tai ohjelmistosta johtuvat ongelmat.

On tärkeä tiedostaa, että sarjaportti toimii ainoastaan Point-to-Point-yhteydellä, joten sarjaporttiyhteydessä voi olla enintään vain kaksi liittymärajapintaa. Tämä tekee yhteyden ylläpitämisestä ja seurannasta haastavan. Vianetsinnässä ja takaisinmallintamisessa (Reverse engineering) käytettiin seuraavia työkaluja:

- VSPE (Virtual Serial Ports Emulator)
- Serial Port Monitor
- Windows batch skriptaus
- PuTTY.

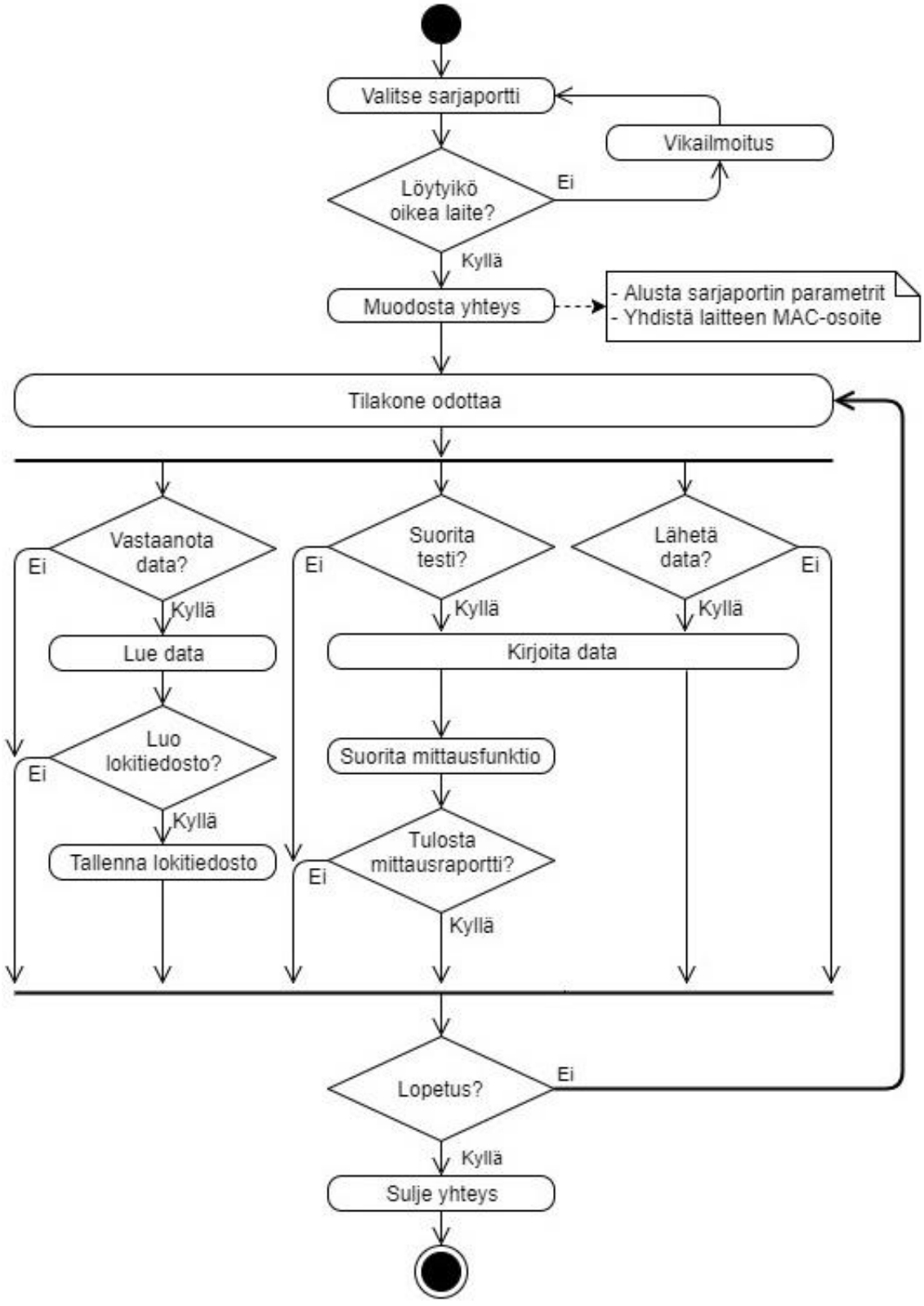
VSPE-ohjelman tarkoitus on auttaa ohjelmistosuunnittelijoita ja kehittäjiä sarjaporttien testauksessa. Ohjelmalla voidaan luoda uusia virtuaalisia sarjaportteja sekä haaroittaa ja kloonata käytössä olevia sarjaportteja. Ohjelmaa käytettiin tasapainokepin ja testausräkin kommunikaatiorajapintoja emuloimisessa. Vianetsinnässä sarjaporttiyhteys haaroitettiin, jotta pakettien kulkua pystyttiin monitoroimaan eri työkaluilla. [12.]

Serial Port Monitor-ohjelmalla mahdollistetaan tietoliikennepakettien kaappaus, tallentaminen ja analysointi. Langattoman tiedonsiirron ja BLE-lepotilan takia osa paketeista saattoi kadota matkan varrella tai olla vajaita vastaanottavan laitteen rajapinnassa, minkä vuoksi ohjelmaa käytettiin laitteiden välisten bittien vertailuun ja seurantaan.

Windows batch-skriptauksella voidaan ohjelmoida yksinkertaisia käskyjä esimerkiksi Microsoft Windows- ja DOS-käyttöjärjestelmiin. Käskyt koostuvat komentojonoista, jotka komentorivin tulkki suorittaa. Käskyjä voidaan kirjoittaa tavalliseen tekstitiedostoon, joka muunnetaan tiedostonimeltä .bat-päätteiseksi. Työssä batch-skriptausta käytettiin pääasiassa sarjaportin tilan selvittämiseen ja testikäskyjen nopeaan suorittamiseen. PuTTY-pääte-emulaattoriohjelmalla (Terminal emulator) voidaan muodostaa yhteys monitoroitavaan sarjaporttiin. Tekstipohjaisen pääteohjelman avulla on siis helppo seurata tai ohjata yksittäisen sarjaportin tietoliikennettä.

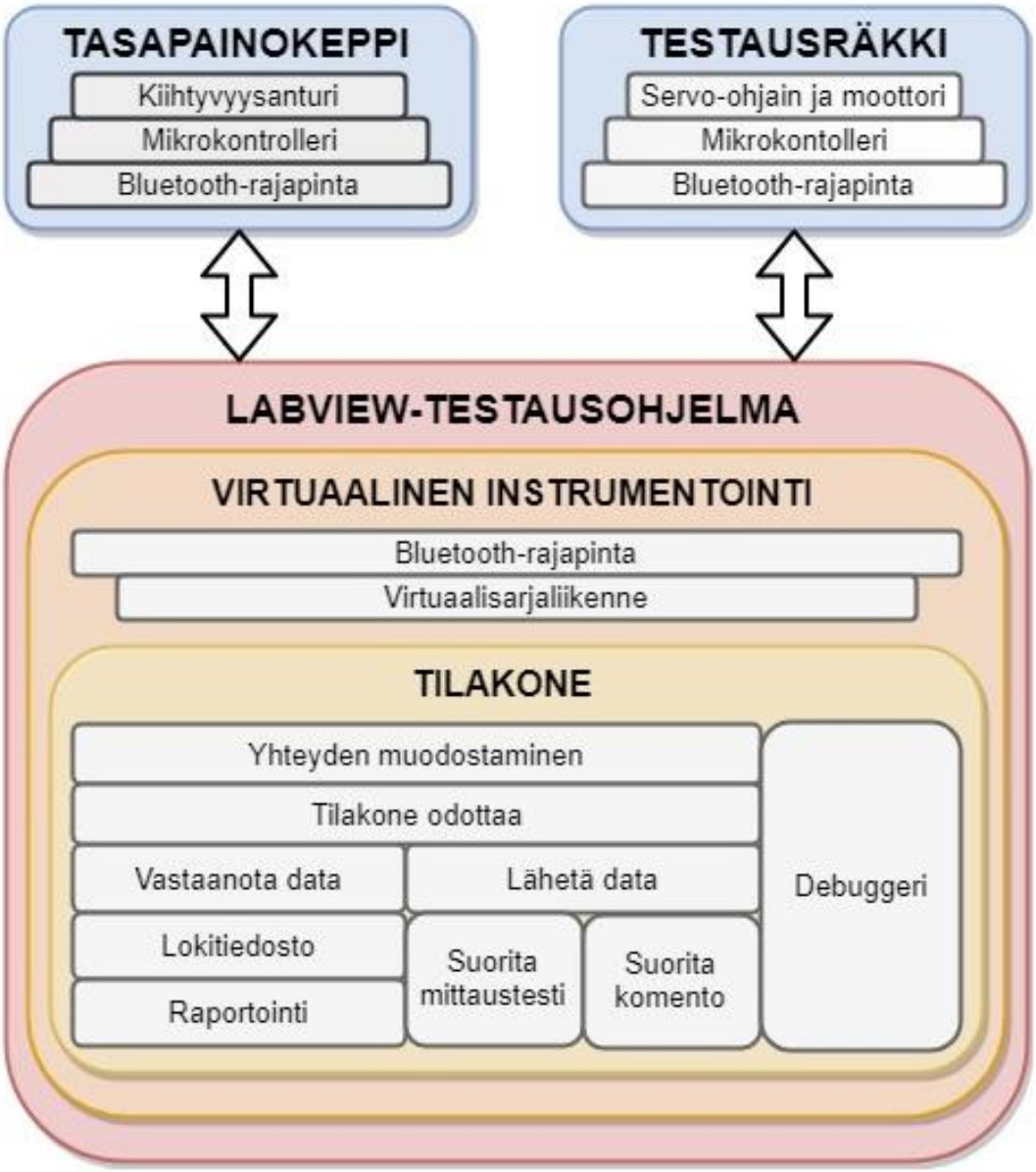
3.1.3 Prototyyppiohjelma

Viimeisessä vaiheessa integroitiin kehitetyt moduulit viimeisteltyyn demoversioon. Ohjelman rakenne jouduttiin kuitenkin vielä kertaalleen miettimään uusiksi, koska itse ohjelmoijan ohjelmointitaidot ja uudet menetelmät olivat kehittyneet parempaan suuntaan. Osaa moduuleista ei siis voitu siirtää sellaisenaan pääohjelmaan aikaisempien ohjelmointivirheiden ja epäkäytännöllisen tilakoneen rakenteen takia. Lopullinen testausohjelman toimintaperiaate pyrittiin silti pitämään alkuperäisen suunnitelman mukaisena, mikä esitetään kuvan 23 uml-kaaviolla.



Kuva 23. Prototyyppiohjelman toimintaperiaate.

Uuden suunnittelumallin vuoksi järjestelmän rakenne toteutettiin perusteellisesti uusiksi kuvan 24 arkkitehtuurin ja moduulien mukaisesti. Syy tähän oli, että vikatilanteissa osa aikaisemmin toteutetuista moduuleista saattoi jumiutua tai kaataa itse testausohjelman ilman selkeää indikaatiota ohjelmointiviasta. LabVIEW-testausohjelman uudella mallilla pyrittiin luomaan selkeämpi modulaarinen arkkitehtuuri, jotta jokainen tilakoneen tapahtumapohjainen funktio toimisi omana itsenäisenä aliohjelmana ja raportoisi virhetilat suoraan debuggerille, jonka avulla pystyttäisiin estämään tai uudelleen käynnistämään tilakoneen kaatuneet moduulit.

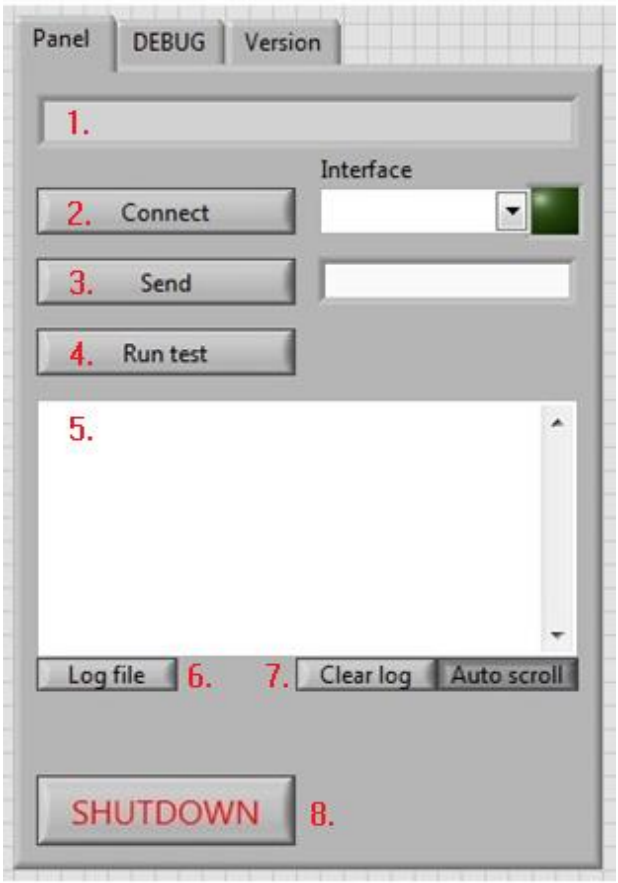


Kuva 24. Testausjärjestelmän arkkitehtuuri ja moduulit.

Testausohjelmaan toteutettiin karkea käyttäjäystävällisempi käyttöliittymä, joka esitetään kuvan 25 mukaisesti:

1. ohjelman tila
2. sarjaportin valinta ja yhdistäminen
3. komennon lähettäminen
4. testisekvenssin suorittaminen
5. datan vastaanottaminen
6. lokitiedoston tulostaminen
7. lokitiedon siivous
8. ohjelman sammutus.

Lisäksi järjestelmän ylläpitoa ja laitteistosta johtuvien vikatilojen varalle toteutettiin ohjelman välilehteen yksinkertainen vianetsintäosio.



Kuva 25. Testausohjelman käyttöliittymä.

3.2 Ohjelman testaussekvenssi

Testausjärjestelmän ensisijainen tehtävä on todentaa tuotekehityksen aikaiset virhelähteet. Ohjelman testaussekvenssin aikana testataan kiihtyvyysanturin mahdolliset virhelähteet, joista todennäköisimmät ja oleellisimmat ovat kohdistusvirhe ja bias-virhe. Kiihtyvyysanturin virheiden lisäksi ohjelman testaussekvenssissä testataan itse tasapainokepin ohjelman toimivuus.

Kohdistusvirheen testaaminen onnistuu parhaiten liikuttamalla testattavaa laitetta gravitaatiokentän suuntaisesti, jolloin kiihtyvyyden muiden akselien suhteen tulisi olla nolla. Jos anturi on asennettu virheellisesti, näkyy muiden akselien kiihtyvyydessä muutos. Bias-virheen testaus onnistuu parhaiten testattavan laitteen ollessa paikallaan vaakatasossa. Y- ja X-akselin suuntaisten kiihtyvyyskomponenttien tulisi teoreettisesti olla nolla, ja Z-akselin kiihtyvyyskomponentissa tulisi näkyä vain gravitaatiokentän aiheuttama kiihtyvyyskomponentti. Laitteen ohjelman toimivuuden testaus onnistuu parhaiten tekemällä ennalta määritelty liikerata useaan kertaa peräkkäin. Toistamalla bias-virheen testaus koko testaussekvenssin lopussa voidaan myös mahdollisesti havaita bias-epästabiiliudesta tai laitelämpenemisestä johtuvat virheet.

Testausohjelman testaussekvenssi koostuu taulukon 1 mukaisesti neljästä eri vaiheesta, joita ovat bias-virhetesti, kohdistusvirhetesti, ohjelmatesti ja bias-epästabiiliustesti.

Taulukko 1. Testausohjelman testaussekvenssi.

Vaihe	Testi	Kesto	Toistoa	Sekvenssi
1	Bias-virhetesti	10 s	1	• Laite vaakatasossa paikallaan • Luetaan kiihtyvyyskomponentit
2	Kohdistusvirhetesti	20 s	1	• Laitetta liikutetaan Z-akselin (gravitaatiokentän) suuntaisesti • Luetaan Y-akselin kiihtyvyyskomponentti
3	Ohjelmatesti	30 s	4	• Laitetta kierretään Y-akselin suuntaisesti • Luetaan Y-akselin kiihtyvyyskomponentti • Verrataan saatuja arvoja viitearvoihin
4	Bias-epästabiiliustesti	10 s	1	• Laite vaakatasossa paikallaan • Luetaan kiihtyvyyskomponentit

3.3 Ongelmat

Lähtökohtaisesti projektissa ei varattu riittävästi aikaa ohjelmiston vaatimusten määrittelyyn ja suunnitteluun, jolloin suurin osa suunnittelusta tapahtui tutkimisen ja moduulien toteutuksen aikana. Projektin aikana haasteita tuottivat lisäksi

- prototyyppilaitteiden tyypilliset laiteviat
- riittävä kommunikointi projektiryhmien välillä
- työn laajuuden ajanhallinta
- suunnitelmallisuuteen liittyvät ongelmat.

Työn suurimpia haasteita oli se, että erillinen ohjelmistokehitys käynnistyi samaan aikaan, kuin itse fyysistä testauslaitteistoa aloitettiin kehittämään, eikä itse testattava laite ollut missään vaiheessa täydellisesti toimiva. Ohjelmiston toteutuksessa tämä näkyi etenkin laitteistojen välisen kommunikoinnin ongelmina. Ratkaisuna ongelmaan kehitettiin tapa emuloida koko järjestelmää virtuaalisessa ympäristössä, jotta pystyttiin rajaamaan fyysisen laitteiston tuomat ongelmat ja testaamaan kehitettyä ohjelmistoa tehokkaasti erillisessä ympäristössä. Testausohjelman suunnitelluista ominaisuuksista saatiin siis toteutettua karkeasti 75 %. Lopullisessa ohjelmaversiossa jäi kuitenkin vielä hieman auki, miten varsinainen testaussekvenssi, liikeradat ja hyväksyntäkriteerit tulee olla määritelty testauslaitteiston päässä.

Isoin ratkaisematon ongelma oli BLE-yhteyden vakaus. Bluetooth-yhteys laitteiden välillä saattoi katketa satunnaisesti, eikä ongelman syntyperää onnistuttu paikallistamaan. Syy tähän ongelmaan lienee joko BLE-yhteyden tai prototyyppilaitteiden lepotilassa ja herätystilassa (wake-up mode), jotka sekoittivat LabVIEW-testausohjelman yhteyden. Tämä taas aiheutti sarjaliikenteen jumiutumisen, jolloin koko järjestelmä jouduttiin käynnistämään uudestaan.

4 Yhteenveto

Insinöörityössä luotiin Metropolia Ammattikorkeakoululle LabVIEW-ohjelmistoympäristöllä toteutettu testausjärjestelmä, jolla voitiin yhdistää ja ohjata erillisiä sulautettuja laitteita. Lisäksi projektissa tuotettiin alustavaa tutkimustyötä LabVIEW-ohjelmistoympäristössä kehitettäville uusille laiteprojekteille. Työ toimii hyvänä pohjustuksena jatkokehitysprojekteille ja osoittaa LabVIEW-ohjelmointiympäristön soveltuvuuden testausjärjestelmien ohjelmoimiseen.

Testausohjelman toteuttamisessa tehtiin kompromisseja, joten osa suunnitelluista ominaisuuksista jouduttiin jättämään toteuttamatta. Suunnittelua ja itse ohjelman toteutusta vaikeutti huomattavasti se, että fyysisen testauslaitteiston kehitys jatkui samaan aikaan testausohjelmiston kehityksen kanssa, eikä itse testattava laite ollut kehitysvaiheessa täydellisesti toimiva. Tämän vuoksi sovelluskehitys toteutettiin emuloimalla koko järjestelmää virtuaalisessa ympäristössä, poissulkien testattavan laitteen ja testauslaitteiston toimimattomuudesta johtuvat ongelmatilanteet.

Lopputuloksena saatu testausohjelma alustaa kommunikointiyhteyden laitteistojen välille, tarjoaa työkalut tiedonsiirtoa, vianetsintää ja testausta varten sekä mahdollistaa mittaustulosten tulostamisen. Toteuttamatta jäi testaussekvenssin ajo, mittausdatan analysointi ja raportointi.

Yksi tärkeimmistä havainnoista oli insinöörityön aikana oikeaoppinen ja analyyttinen insinöörityöskentely sekä ajanhallinnan merkitys nopeassa ohjelmistokehitysprosessissa. Ohjelmiston vaatimusten määrittely ja suunnittelu ovat ensisijaisen tärkeitä, eikä ennen niiden huolellista laatimista tulisi aloittaa sovelluksen toteutusta. Työn aikana havaittiin myös sujuvan yhteistyön merkitys projektityöskentelyn tilanteessa, jossa samanaikaisesti useampi projektiryhmä työskentelee yhteistyössä saavuttaakseen yhteisen tavoitteen. Henkilökohtaisesti tärkeä havainto oli tehokkaan ja riittävän kommunikaation merkitys isossa projektikokonaisuudessa, mihin selkeänä vaikuttimena on työskentely samoissa tiloissa eri projektiryhmien kanssa.

Lähteet

1 Automatic test equipment. 2017. Verkkodokumentti. Wikipedia. https://en.wikipedia.org/wiki/Automatic_test_equipment, Luettu 5.10.2017.

2 Bakshi, U.A & Bakshi, A.V. 2009. Electronic Measurement Systems. Technical Publications Pune.

3 Brindley, Keith. 1991. Automatic Test Equipment. Newnes.

4 Immonen, Jarkko. 2002. Verkkodokumentti. Johdatus ohjelmistotuotantoon. <http://cs.joensuu.fi/~jimmonen/jot_moniste/jot_moniste_121.html>. Luettu 11.3.2018.

5 Woodman, Oliver. 2007. Techical Report number 696 - An introduction to inertial navigation. University of Cambridge Computer Laboratory.

6 Chow, Raymond. 2017. Verkkodokumentti. Evaluating inertial measurement units. <https://www.edn.com/design/test-and-measurement/4389260/Evaluating-inertial-measurement-units>. Luettu 12.11.2017.

7 Kari, Janne. 2016. Langattoman peliohjaimen varusohjelmisto. Insinöörityö. Metropolia Ammattikorkeoulu. Theseus-tietokanta.

8 Tekninen datalehti MMA8451Q 3-axis 14-bit/8-bit digital accelerometer. 2017. NXP Semiconductors.

9 Pedley Mark. 2013. Application Note - Tilt Sensing Usign a Three-Axis Accelerometer. Freescale Semiconductor.

10 Davidson, Robert; Townsend, Kevin; Wang, Chris & Cufí, Carles. 2014. Getting Started with Bluetooth Low Energy: Tools and Techniques for Low-Power Networking. O'Reilly Media.

11 Huttunen, Janne. 2006. Ketterän ohjelmistokehitysmenetelmän määrittely, vertailu ja käyttäjäkysely. Diplomityö. Teknillinen Korkeakoulu.

12 Virtual Serial Ports Emulator. 2018. Verkkodokumentti. Eterlogic. <http://www.eterlogic.com/Products.VSPE.html>. Luettu 11.3.2018.

13 Ketterän ohjelmistokehityksen julistus. 2018. Verkkodokumentti. Ward Cunningham. <http://agilemanifesto.org/iso/fi/manifesto.html>. Luettu 11.3.2018.