Antti Katajala

LABVIEW OHJELMIEN TESTAUS JA DOKUMENTOINTI

Opinnäytetyö

KESKI-POHJANMAAN AMMATTIKORKEAKOULU

Tekniikan ylempi ammattikorkeakoulu

Teknologiaosaamisen johtamisen koulutusohjelma

Toukokuu 2011

KESKI-POHJANMAAN AMMATTIKORKEAKOULU
MELLERSTA ÖSTERBOTTENS YRKESHÖGSKOLA

TIIVISTELMÄ OPINNÄYTETYÖSTÄ

Yksikkö Tekniikka ja liiketalous	**Aika** Toukokuu 2011	**Tekijä** Antti Katajala
Koulutusohjelma Teknologiaosaamisen johtamisen koulutusohjelma		
Työn nimi LABVIEW OHJELMIEN TESTAUS JA DOKUMENTOINTI		
Työn ohjaaja KTT Pekka Nokso-Koivisto, Tkl Eero Pikkarainen		**Sivumäärä** [60 + 1]
Työelämäohjaaja TkT Mikko Keskilammi		

Tämä tutkimustyö tehtiin Sasken Finland Oy:lle LabVIEW tiimin suunnittelijoiden tarpeeseen. Tämän tutkimustyön tarkoitus oli LabVIEW:lla ohjelmoitujen ohjelmien testauksen ja dokumentoinnin yhtenäisen ohjeistuksen laadinta. Miten testausprosessin tulisi edetä? Miten lopputestaus tulisi suorittaa? Mitä tulisi dokumentoida LabVIEW:lla ohjelmoitaessa?

Työ toteutettiin teemahaastattelemalla yrityksen suunnittelijoita testauksesta ja dokumentoinnista. Näistä tehtiin yhteenveto ja verrattiin tuloksia teoriassa kerrottuihin suosituksiin. Teoriasta ja haastatteluista luotiin yleinen ohjeistus suunnittelijoille testaukseen ja dokumentointiin. Tutkimuksessa kerrotaan myös eri ohjelmien kehittämis- ja vaihejakomalleja.

Tutkimustuloksissa kerrotaan miten tulisi suorittaa yksikkötestaus, moduulitestaus, integrointitestaus ja järjestelmä- eli hyväksymistestaus. Lisäksi kerrotaan kuinka tulisi ohjelmia dokumentoida, sekä ohjeistetaan koodin kommentointi ja tuotedokumentaatio. Tuotedokumentaatio pitää sisällään ohjelmien käyttöohjeen, asennusohjeen, koulutusmateriaalin sekä teknisen dokumentaation.

Asiasanat
Dokumentointi, LabVIEW, ohjeistus, ohjelmointi, teemahaastattelu, testaus

CENTRAL OSTROBOTHNIA
UNIVERSITY OF APPLIED SCIENCES

ABSTRACT

CENTRAL OSTROBOTHNIA UNIVERSITY OF APPLIED SCIENCES	Date May. 2011	Author Antti Katajala
Degree programme Master`s Degree for Technology Competence		
Name of thesis LABVIEW SOFTWARE TESTING AND DOCUMENTATION		
Instructor Pekka Nokso-Koivisto, Eero Pikkarainen		**Pages** 60+1
Supervisor Mikko Keskilammi		

This research work was done for Sasken Finland Oy to the needs og the LabVIEW designers´ team. This research was done to design standard instructions for LabVIEW-programmed software testing and documentation. How should the testing process move forward? How should the final testing be performed? What should be documented during LabVIEW programming?

The work was carried out theme interviewing the designers of the company about testing and documentation. These were summarized and compared with the results of theoretically informed recommendations. The general guideline for testing and documentation for the designers was created of theory and interviews. The research also presents various program development and phase distribution models.

It is presented in the research results how to perform unit testing, module testing, integration testing and system i.e acceptance testing. In addition, it is told programs should be documented. The code annotation and product documentation are instructed as well. The product documentation includes a software manual, installation instructions, training materials and technical documentation.

Key words
Documentation, LabVIEW, guidance, programming, interviews, testing

SISÄLLYS

1 JOHDANTO

Kuinka tehdään LabVIEW:llä ohjelmoitujen ohjelmien dokumentointi ja testaus? Siinä on kysymys, johon tämän työn on antaa vastauksen. Tämän työn tarkoitus on kartoittaa LabVIEW:llä toteutetun ohjelman testauksen ja dokumentoinnin nykykäytäntö sekä määritellään, valitaan ja dokumentoidaan sopivat käytännöt ja prosessit tai ohjeet, millä mennään eteenpäin tulevaisuudessa Sasken Finland Oy:ssä, joka on työn tilaaja. Tätä ohjeistusta tai prosessia on tarkoitus käyttää LabVIEW tiimin suunnittelijoiden apuna/ työkaluna jokapäiväisessä työskentelyssä.

Teoriaosiossa käsitellään pintapuolisesti, mikä LabVIEW ohjelma on ja mitkä on LabVIEW:n yleisiä ohjelmointi sääntöjä. Tämän jälkeen käsitellään ohjelmien kehittämis-/ vaihejakomalleja. Vaihejakomalleista katsotaan, missä vaiheessa tulisi dokumentointi ja ennen kaikkea testaus ottaa mukaan suunnitteluun. Teorian loppuosassa tullaan kertomaan yleisesti ohjelmien dokumentoinnista ja erikseen LabVIEW ohjelmien dokumentoinnista ja viimeiseksi kerrotaan testauksen dokumentoinnista. Dokumentoinnissa on rajattu dokumentoinnin keskeisimpiin tuotedokumentteihin. Tämä työ ei käsittele projektien dokumentointia, eikä laatujärjestelmien määräämiä dokumentointeja, jos ne eivät kuulu tuote-/ koodindokumentointiin.

Työn tavoite ja menetelmät osiossa kerrotaan tarkemmin, miten tutkimus tullaan tekemään. Tarkoitus on käsitellä testausta ja dokumentointia, kuinka niitä tällä hetkellä tehdään ja kuinka tulisi suunnittelijoiden mielestä tehdä ja näitä verrata teoriaan. Näistä vedetään yhteenveto ts. synteesi ja kerrotaan lukijalle kuinka tulisi toimia dokumentoinnin ja testauksen suhteen.

Johtopäätöksissä tullaan kertomaan työn saavutuksista. Tässä osiossa kerrotaan kehittämistyön tekemisen ongelmista. Mitkä asiat vaikuttivat työn lopputulokseen? Kerrotaan mitä voisi vielä tutkia aiheesta. Vaikuttiko kehittämistehtävä Sasken Finland Oy:n organisaation toimintaan?

2 LABVIEW

2.1 LabVIEW:n taustaa

2.1.1 LabVIEW:n historia

Ymmärtääksemme paremmin LabVIEW ohjelmointia, on hyvä tarkistaa taustatietoja ensimmäisestä korkeamman tason ohjelmointikielestä. Nykyaikaisen tietokoneen ikä alkaa 1950-luvun puolivälistä. Pieni ryhmä IBM:ssä päätti luoda käytännön vaihtoehdon ohjelmointiin valtavan IBM 704 keskuskoneen (sen ajan supertietokone) matalan tason konekielelle, joka oli tuolloin nykyaikaisin käytettävissä oleva ohjelmointikieli. Tuloksena oli Fortran, enemmän ihmisen luettavissa oleva ohjelmointikieli, jonka tarkoituksena oli nopeuttaa ohjelmistojen kehitysprosessia. (National Instruments 2010.)

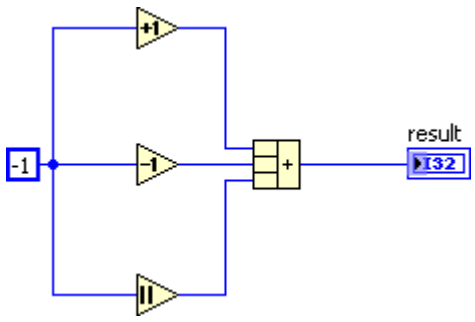
Suunnitteluyhteisö oli aluksi skeptinen, että tämä uusi menetelmä voisi jotenkin olla parempi ohjelma sovelluksissa, mutta pian kävi ilmi, että Fortranilla laaditut ohjelmat pyörivät lähes yhtä tehokkaasti kuin konekielisesti kirjoitetut ohjelmat. FORTRAN on nopeasti saavuttanut hyväksyntää tiedeyhteisöissä ja on edelleen vaikutusvaltainen ohjelmointimenetelmä. (National Instruments 2010.)

Viisikymmentä vuotta myöhemmin, on edelleen tärkeitä asioita mitä tässä tarinassa oli. Ensinnäkin, yli 50 vuotta, insinöörit ovat pyrkineet helpompaan ja nopeampaan tapaan ratkaista ongelmat tietokoneiden ohjelmoinnissa. Toisaalta suunnittelijoilla on tärkeämpiä tehtäviä monesti kuin pelkästään keskittyä ohjelmointiin. Kerrotut asiat toivottavasti auttavat ymmärtämään valtavan suosion graafiseen ohjelmointikieleen (G-kielinen). G-kielinen ohjelmointi on laajalti omaksuttu sen perustamisesta vuodesta 1986 lähtien, G-kielinen ohjelmointi edustaa erittäin korke-

an tason ohjelmointikieltä, jonka tarkoituksena on lisätä ohjelmoijien tuottavuutta, kun taas loppukäyttäjillä ohjelmien tulisi toimia lähes samalla nopeudella kuin alemman tason kielillä ohjelmoitujen ohjelmien, kuten Fortran-, C-ja C + +. (National Instruments 2010.)

2.1.2 LabVIEW ohjelmointikielenä

LabVIEW on itse asiassa sovelluskehitin, jolla tuotetaan graafista koodia, joka käännetään konekieliseksi ohjelmaksi kääntäjällä. Tällaista ohjelmakoodia kutsutaan usein G-kieliseksi ohjelmaksi tai G-kieliseksi koodiksi. LabVIEW poikkeaa perinteisistä ohjelmointikielistä myös suoritustapansa suhteen. Tavallista tekstipohjaista koodia suoritetaan rivi kerrallaan, kun taas LabVIEW'n graafisessa koodissa useita aliohjelmia voidaan suorittaa samanaikaisesti niin kuin kuviossa 1 on esitetty. Tässä tapauksessa kokonaisluku viisi jaetaan kolmeen eri signaaliin, jotka tuodaan kolmen funktion tuloportteihin. Funktiot tarkistavat, sisältääkö signaali nollaa suuremman kokonaisluvun. Funktiot suoritetaan siis rinnakkain ja niistä lähtee ulostulosignaalina *Tosi*-tyyppinen muuttuja. Tämän LabVIEW voi jakaa tarvittaessa automaattisesti useammalle prosessorille suoritettavaksi.



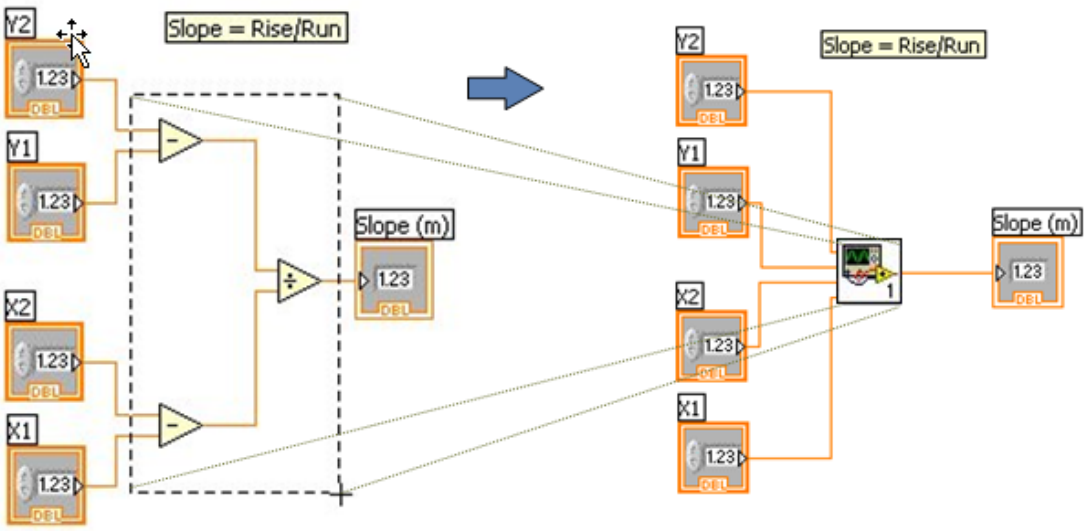
Kuvio 1. G-kielisen koodin moniajo

LabVIEW'n graafisen koodin luonteesta johtuen aloituskynnys ohjelmointiin on matalampi kuin perinteisillä ohjelmointikielillä. Tämä on toisaalta G-kielisen koodin vahvuus, mutta toisaalta kokemattomalle ohjelmoijalle haaste, sillä graafisesta

koodista saa aloittelija helposti tuotettua vaikeasti luettavaa ja sekavaa koodia jota on myös myöhemmin todella työlästä kehittää tai etsiä virheitä. Tästä syystä LabVIEW-ohjelmoinnissa on tärkeä muistaa hyviin ohjelmointitapoihin kuuluvat seikat, kuten funktioiden selkeä sijoittelu sekä selkeät, suorat johtimet. G-kielinen koodi on tämän lisäksi suunniteltu luettavaksi vasemmalta oikealle ja ylhäältä alas, joten ohjelmoijan on kaikin keinoin vältettävä johdottamasta oikealta vasemmalle. Objekteja ei myöskään ole syytä asetella johdotuksen päälle, koska tällöin sovelluksen luettavuus kärsii.

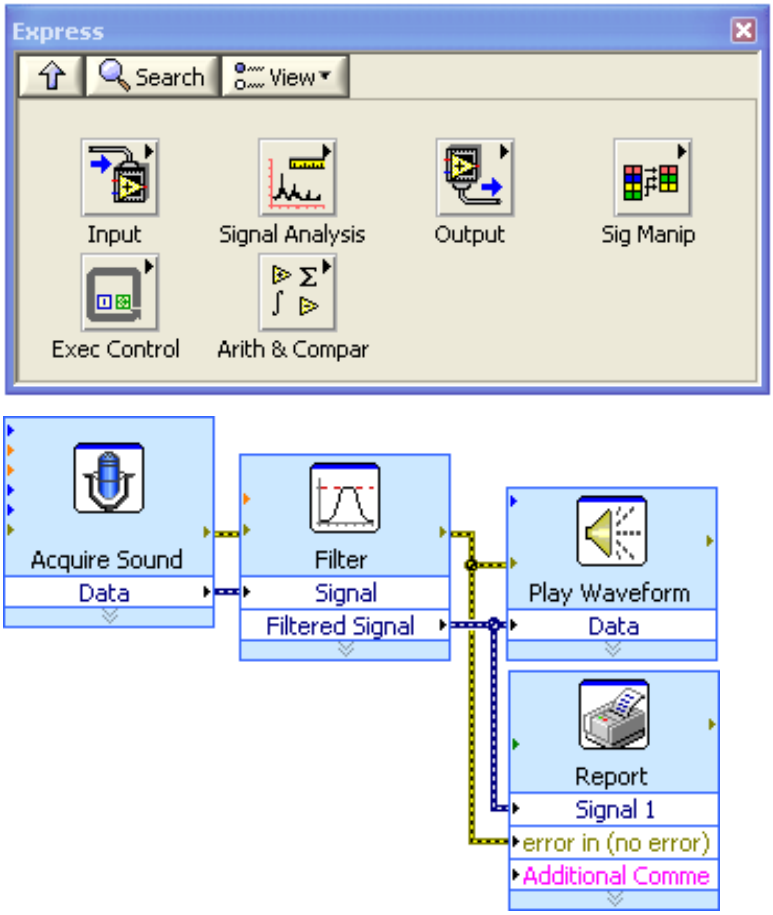
2.1.3 Virtuaali-instrumentit

LabVIEW-sovelluksia kutsutaan virtuaali-instrumenteiksi, koska ne muistuttavat reaalimaailman mittauslaitteita, kuten oskilloskooppeja tai yleismittareita. Virtuaali-instrumentti koostuu tietokoneesta ja siihen kytkettävistä lisälaitteista, joilla mitataan erilaisia reaalimaailman suureita. Ohjelmamoduulien nimen tiedostopäätteenä käytetään lyhennettä ”vi”, eli ohjelma voi olla nimeltään esimerkiksi *Mittaus.vi* (National Instruments, 2007b).



Kuvio 2. Ali vi:n luominen

Jokaisessa vi:ssä on oltava etupaneeli eli käyttöliittymä sekä G-kielisestä koodista koostuva osuus, jota kutsutaan diagrammiksi. LabVIEW:ssä on käytettävissä laaja valikoima valmiita funktioita. Datan keruuta varten LabVIEW:ssä on myös valmiita vi:tä, joilla pystytään helposti ohjaamaan lisälaitteita. Kuviossa 2 näytetään yksinkertaisimmillaan kuinka LabVIEW:ssa luodaan uusia ali vi:tä/ funktioita ja kuinka alifunktiot näkyvät ylemmässä ohjelmassa. Suurin syy, minkä takia aliohjelmia/ funktioita tehdään, on koodin selkeyttäminen ja tiettyjen funktioiden käyttö useammassa paikassa. Näin saadaan ohjelmasta modulaarisempi ja helpommin luettava. (National Instruments, 2007b.)



Kuvio 3. Express funktiot jaottelu ja käyttäminen

Uusimmissa LabVIEW ohjelmissa on myös express funktioita, jotka mahdollistavat vieläkin nopeamman ohjelmoinnin LabVIEW:lla. Kuviossa 3 kerrotaan kuinka helppoa ohjelmointi voi yksinkertaisuudessaan olla ja kuvion 3 yläosassa näkee, kuinka selkeästi LabVIEW:n Express funktion on jaoteltu funktioihin, joilla saadaan luettua tieto esimerkiksi mittalaitteelta tai tässä tapauksessa mikrofonilta (Input). Tämän jälkeen on otettu seuraavasta kohdasta signaalin analysointi (Signal Analysis) kohdasta suodatin Express funktio ja viimeiseksi otetaan funktio ulostulo (Output) kohdasta. Toistetaan muokattu ääni kaiuttimilla ja samanaikaisesti tallennetaan raportti exceliin, jonne tulostuu graafi ääninäytteestä suodatuksen jälkeen ja muut halutut tiedot.

Usein on tarpeellista tallentaa saatuja mittaustuloksia helposti luettavaan muotoon. Tämän vuoksi LabVIEW’ssä on monia taulukonkäsittelyfunktioita valmiiksi sisäänrakennettuna. Tallennuksessa käytetään yleisesti sarkaimin erotettua taulukkomuotoa, jota pystytään lukemaan kaikilla taulukkolaskentaohjelmilla käyttöjärjestelmästä riippumatta. Myös kaikki muut mahdolliset ulkomuodot ja raportointiformaatit ovat mahdollisia.

2.1.4 Yleisiä sääntöjä LabVIEW:lla ohjelmoitaessa

LabVIEW:lla ohjelmoitaessa ei mitään virallisia sääntöjä ole. Jokaiselle on tullut oma tapa, voisi sanoa käsiala ohjelmointiin. Joitakin yleisiä ohjeita toki on, joita olisi hyvä noudattaa. Tähän on otettu tärkeimmät ”säännöt” joiden mukaan Saskenillakin tulisi jokaisen toimia.

Taulukko 1. LabVIEW ohjelmoinnin tärkeimmät säännöt/ ohjeet (National Instruments 2006a).

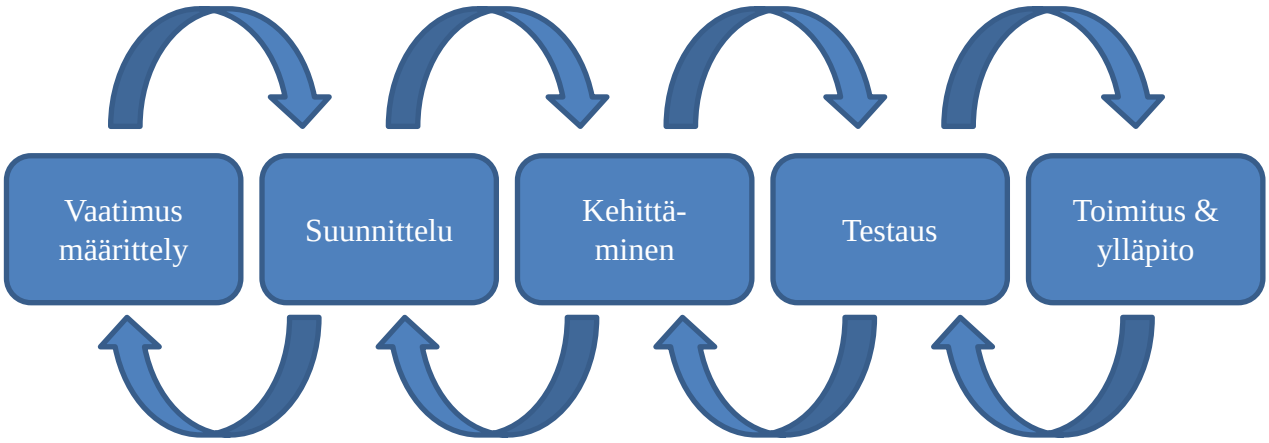
1.	Suunnittele ensin koodi ja ohjelmoi vasta sitten
2.	Kokeile koodinpätkiä ennen käyttöä
3.	Suunnittele uudelleen
4.	Pidä koodi selkeänä
5.	Kommentoi
6.	Käytä kuvaavia nimiä
7.	Huomio käyttöjärjestelmän rajoitukset nimeämisissä
8.	Käytä standardi LabVIEW controlleja VI:ssä
9.	Käytä pieniä kirjaimia controllien nimeämisissä
10.	Ryhmittele controllit loogisesti
11.	Huomioi käyttöliittymän koko
12.	Älä sijoita controlleja liian tiiviisti (erityisesti käytettäessä kosketusnättöä)
13.	Varmistu ettei kaksoispainallus aiheuta vääriä toimintoja
14.	Käytä kursorin ohjausta ja oletuksia controlleihin
15.	Käytä painonappien (controllien) nimissä yksilöllisiä sanoja, joilla voidaan erotella eri painonappien toiminnat
16.	Käytä aina peruta tai takaisin valintaa myös
17.	Boolean tyyppisissä controlleissa nimen pitää kertoa todellisen tilan
18.	Laita kytkennöissä controllit vasemmalle ja indikaattorit oikealle
19.	Laita controllien nimiin oletusarvot sulkuihin
20.	Käytä oletuskytkentää ali vi:ssä (12 terminaalia)
21.	Valitse terminaaleille ovatko ne vaatimuksia, suositeltavia vai valinnaisia
22.	Kirjoita vähintään ikoneihin
23.	Tee myös mustavalkoinen ikoni
24.	Käytä LabVIEW:n oletusfontteja ja -värejä
25.	Tee selkeät johdotukset vasemmalta oikealle
26.	Käytä virheklustereita ja käsittele ne
27.	Käytä negatiivista virhekoodia vakaville virheille ja positiivista muille
28.	Tee kuvaavia virheilmoituksia
29.	Muista sulkea referenssit

Taulukkossa 1 on LabVIEW ohjelmoinnin tärkeimpiä sääntöjä/ ohjeita, joita voidaan käyttää ohjeena kaikkeen LabVIEW ohjelmointiin. Hyvin monet näistä käyvät yleisestikin ohjeeksi ohjelmoitaessa, mitä tulisi ottaa huomioon. Tätä taulukkoa voi suunnittelija käyttää myös yksittäisten aliohjelmien testauksen tarkistuslistana.

2.2 Ohjelmien kehittämis-/ vaihejakomalleja

2.2.1 Vesiputousmalli

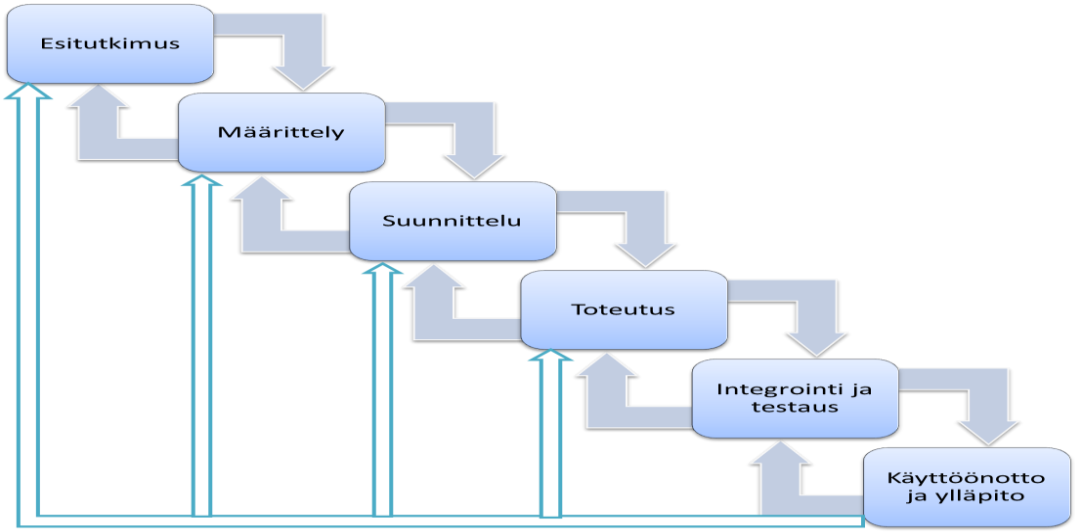
Kuviossa 4 on kuvattu National Instrumentin mielestä ideaalisin ohjelmointiprosessin malli. Tämä malli on perinteinen vesiputousmalli, vaikkakin vähän erilailla kuvattuna, kuin perinteisesti vesiputous mallina on totuttu.



Kuvio 4. Ideaali ohjelmointiprosessi (National Instruments 2007a.)

Käytännössä tämä ei koskaan toimi näin ideaalisesti, että mentäisiin vain enintään yksi pykälä taaksepäin. Tässä prosessissa vaatimusmäärittelyn jälkeen tulee suunnittelu, josta voidaan tarvittaessa palata vaatimusmäärittelyyn. Suunnittelun jälkeen on kehittäminen/ ohjelmointi, josta voidaan palata suunnitteluun. Ohjelmoinnin jälkeen testaus ja testauksessa esiin tulleet virheet korjataan, jolloin palataan ohjelman kehittämiskohtaan, ja tämän jälkeen testaukseen. Testauksen jälkeen siirrytään toimitukseen ja ylläpitoon. Joskus voi olla tilanteita, että joudutaan palamaan testauksesta vaatimusmäärittelyyn, jos huomataan vasta testauksessa virhe, joka on tullut vaatimusmäärityksiä tehtäessä. Tähän

onkin ratkaisu kuviossa 5, jossa on esimerkki iteroivasta vesiputousmallista ja alempana on kerrottu tarkemmin mitä missäkin vaiheessa vesiputousmallissa tehdään.

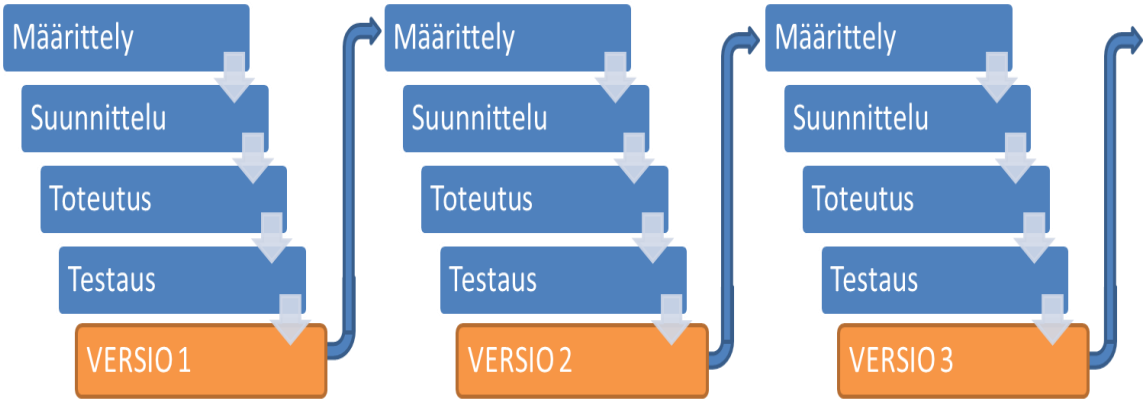


Kuvio 5. esimerkki iteroivasta vesiputousmallista (Märijärvi & Haikala 2001, 24.)

Iteroiva vesiputousmalli, joka on esitetty kuviossa 5, joka on tavallisin vaihejakomalli. Esitutkimuksessa selvitetään mikä on ratkaistava ongelma, onko sitä mahdollista ratkaista, mitä se maksaa ja mitä rajoituksia on. Määrittelyssä selvitetään millainen järjestelmä täyttää vaatimukset. Suunnittelussa selvitetään miten järjestelmä toteutetaan ja pienitään pienempiin osiin. Toteutusvaiheessa ohjelman osat ohjelmoidaan. Integrointi ja testausvaiheessa osat yhdistetään yhdeksi kokonaisuudeksi ja testataan osien ja kokonaisuuden toiminta, jonka jälkeen ohjelma menee käyttöön otto vaiheeseen ja ylläpitotilaan. (Märijärvi & Haikala 2001, 25- 29.)

2.2.2 Evo-malli

Vesiputousmallin lisäksi yleisesti käytettyjä elinkaari tai ohjelman kehitysprosessimalleja ovat mm. Evo-malli (evolutionary delivery) ja eri protoilumallit. (Märijärvi & Haikala 2001.)



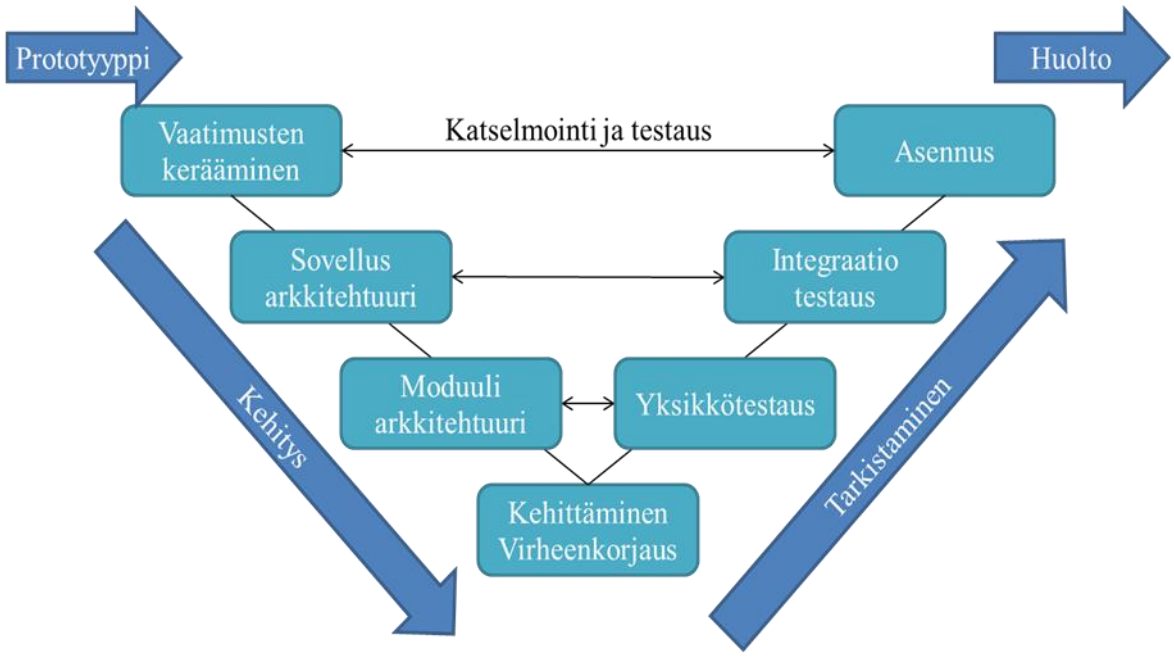
Kuvio 6. Evo-malli (Märijärvi & Haikala 2001, 30.)

Yleensä ohjelmasta tehdään ensimmäinen versio, jossa on vain tietyt ominaisuudet käytössä. Tämän jälkeen tulee seuraava versio, jossa on enempi ominaisuuksia ja samalla on korjattu esiin tulleita version 1 bugeja. Kuvion 6 mukainen Evo-malli havainnollistaa tätä hyvin. Tämä kuvio voisi jatkua vaikka kuinka pitkään samalla tavalla. Aina tehdään ensin määrittely, jonka jälkeen suunnittelu, toteutus, testaus ja versio asiakkaalle.

2.2.3 Ohjelmien yleinen kehittämisprosessimalli eli V-malli

Kuvion 7 mukaan ohjelman kehittämisen V-malli havainnollistaa hyvin ohjelman kehittämisen eri kehitysvaiheet ja niitä seuraavat testausvaiheet. Ohjelmien kehitys eli prosessivaiheisiin kuuluu vaatimusten kerääminen, sovellusarkkitehtuuri, moduuli arkkitehtuuri ja kehittäminen eli ohjelmointivaihe. Virheen korjausta

tehdään samanaikaisesti kehittämisen kanssa. Tarkistaminen eli testaus aloitetaan yksikkötestauksella eli moduulitestauksella jatkuen integraatio testauksella ja asennuksella eli hyväksymistestauksella, järjestelmätestauksella tai validioinnilla. Samalla asialla on monta eri nimeä. Tarkistamis- eli testausvaiheissa on tarkoitus vaiheittain tarkistaa ja testata ohjelmaa, vastaako ohjelma V-mallin vasemmalla puolella olevia suunnitteluvaiheita.

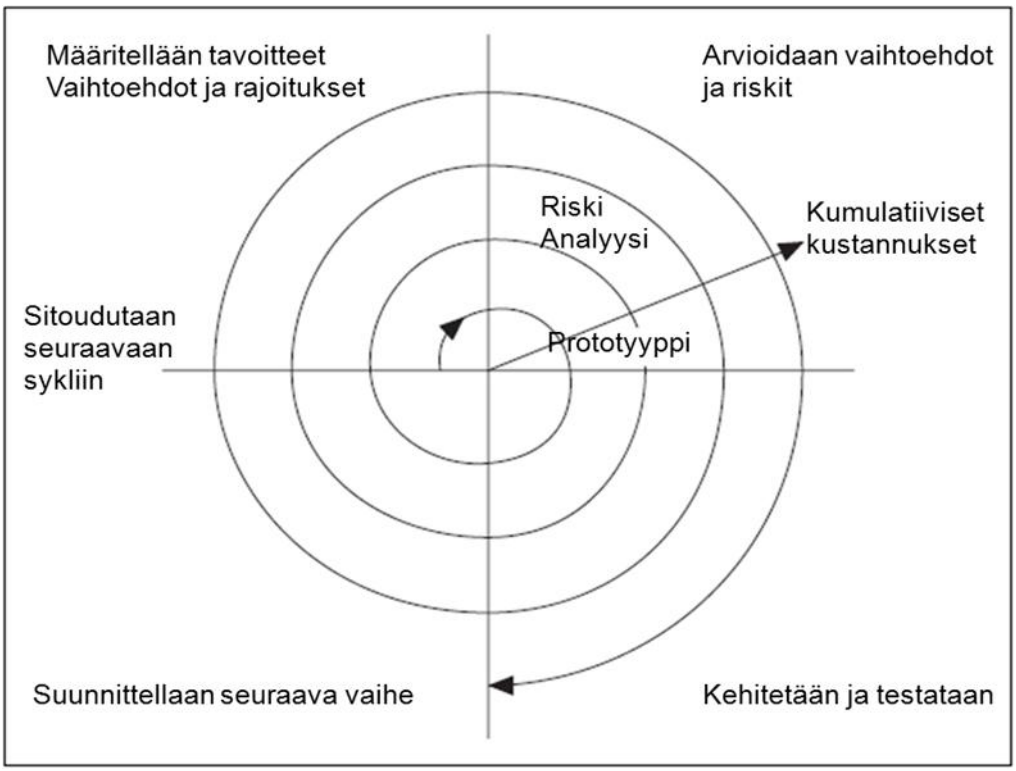


Kuvio 7. Ohjelman kehittämisen V-malli (Märijärvi & Haikala 2001, 33.)

Tämä malli havainnollistaa hyvin, missä vaiheessa tulisi miettiä minkäkin testauksen suunnittelu. Esimerkiksi, kun ohjelma on siinä vaiheessa, että se voidaan asentaa lopulliseen käyttöön, sen testauksen ja katselmoinnin vaatimukset tulevat vaatimusten keräämisestä ts. vaatimusmäärittelystä. Jos ohjelma täyttää vaatimusmäärittelyssä annetut vaatimukset, ohjelma siirtyy ylläpitotilaan ja asiakkaalle käyttöön.

2.2.4 Spiraali kehitysmalli

Spiraalimalli on suosittu vaihtoehto vesiputousmallille. Spiraalimalli korostaa riskienhallintaa, jotta löydetään suuremmat ongelmat aiemmin kehityksen aikana. Vesiputousmallissa tehdään täydellinen suunnittelu ennen kuin aloitetaan ohjelmointia. Spiraalimallilla voidaan projekti palastella joukoksi riskejä, joita arvioidaan ja ratkotaan suunnitellussa järjestyksessä. Tämän jälkeen aloitetaan toistamaan prosessia, jossa analysoidaan tärkein riski, arvioidaan vaihtoehtoisia ratkaisuja riskille, käsitellään riskiä, arvioidaan tuloksia ja suunnitellaan seuraavan sykli. Tässä mallissa ensin suunnitellaan ja tehdään kriittisimmät osiot ja saadaan palaute asiakkaalta ja asiakkaalta saatua palautetta hyödynnetään seuraavalla kierroksella. Kuviossa 8 on havainnollistettu spiraali kehitys-/ elinkaarimallia. (National Instruments, 2003.)



Kuvio 8. Spiraali-kehitysmalli (National Instruments 2003, 18.)

Spiraali-kehitysmallissa voitaisiin ajatella jokaista pyörähdystä spiraalissa omaksi ohjelman versioksi. Versiot voivat olla ensimmäisillä kierroksilla niin alku tekijöissä, että ne toimivat vain suunnittelijoiden käytössä ja version hallinnassa. Pitemmälle mentäessä voi yksi pyörähdys olla aina uusi versio, joka menee testaukseen ja sitä myöten loppukäyttäjälle.

2.2.5 Ketterä menetelmä

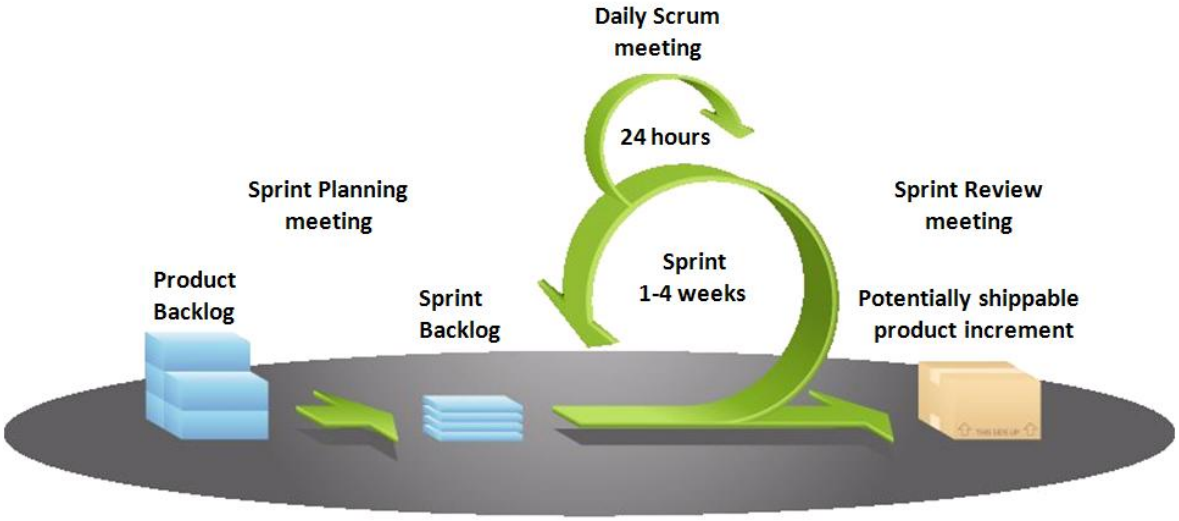
Ketteriä menetelmiä on useita, mm. Extreme Programming (XP), Scrum, DSDM, Crystal Methods, Agile modeling, Adaptive software development, Pragmatic Programming, Feature driven development ja Gilb-EVO (Wikipedia, 2011). Tässä tullaan käsittelemään vain Scrum menetelmää, koska tämä menetelmä on Saskenin LabVIEW tiimin käytössä. Pelkästään tästä aiheesta voisi kirjoittaa yhden opinnäytetyön, joten tässä kerrotaan miten Scrum prosessi toimii. Ketterien menetelmien ensisijainen tavoite on tuottaa toimiva sovellus nopeasti, mikä on siten arvokas asiakkaalle, kun taas suunnitelmalähtöisissä menetelmissä ensisijainen tavoite on tuottaa vakaa ja luotettava sovellus (Kettunen 2009). Tätä menetelmää käytetään Saskenin LabVIEW tiimissä, koska tekeminen painottuu jatkuvaan yhden ohjelmakokonaisuuden jatkokehittämiseen, paljon pieniä bugikorjauksia, pieni tiimi tekemässä ja LabVIEW ohjelmoinnissa tämä toimii todella hyvin.

Scrum perustuu Agile manifestoon, jonka sisältö on seuraava:

Me etsimme parempia keinoja ohjelmistojen kehittämiseen tekemällä sitä itse ja auttamalla siinä muita. Tässä työssämme olemme päätyneet arvostamaan:

1. Yksilöitä ja vuorovaikutusta enemmän kuin prosesseja ja työkaluja
2. Toimivaa sovellusta enemmän kuin kokonaisvaltaista dokumentaatiota
3. Asiakasyhteistyötä enemmän kuin sopimusneuvotteluita
4. Muutokseen reagoimista enemmän, kuin suunnitelman noudattamista.

(Agile 2011.)



Kuvio 9. Scrum prosessi (Saksa 2008.)

Kuviossa 9 esitetty Scrum prosessi on käytössä Sasken Finland Oy:ssä laajalti ja myös LabVIEW tiimissä. Product Backlog (tuote tilauskanta) on paikka, jonne kerätään ohjelmaan haluttuja uusia ominaisuuksia ja esimerkiksi bugeja, joita on testauksessa löydetty. Product Backlogissa uudet ominaisuudet ja bugit laitetaan kiireellisyysjärjestykseen ja laitetaan haluttuun toteutuskierrokseen. Toteutuskierrosta kutsutaan pyrähdykseksi, eli ns. sprintiksi (engl. Sprint). (Saksa 2008.)

Sprintin kesto on n. 2-4 viikkoa. Jokaisen sprintin sisältö sovitaan ennen periodin aloitusta, ja tehtäväksi valitaan vain niitä asioita, joilla on sillä hetkellä suurin merkitys projektin onnistumisen kannalta ja jotka tiimi uskoo pystyvänsä sovitussa ajassa tekemään. Päivittäisissä Scrum-palavereissa, jotka ovat kestoltaan noin 15 minuutin, tärkeintä on tiedon jakaminen tiimin jäsenille. Siellä kerrotaan mitä tein eilen, mitä teen huomenna ja onko mitään esteenä, etten saisi jotain tehtyä? (Saksa 2008.)

Sprint Review meeting (sprintin katselmuspalaveri) on tarkoituksellisesti pidetty hyvin epävirallisena. Tyypillisesti säännöt kieltävät PowerPoint-diat ja tämä mahdollistaa enintään kahdentunnin valmisteluajan ennen kokousta. Ihanneta-

pauksessa tiimi on suorittanut Product Baclog:sta sprintiin tuodut tehtävät, mutta on tärkeää, että ne saavuttavat yleistavoitteen sprintissä. Tässä palaverissa voidaan demota tehtyä ohjelmaa tai muulla tavalla kertoa miten lopputulokseen on päästy. Tärkein tehtävä tällä palaverilla on kuitenkin sprintin hyväksyminen, mitkä kohdat voidaan hyväksyä tehdyksi ja mitkä menevät seuraavaan asiakkaalle lähtevään versioon. (Saksa 2008.)

Ketterissä menetelmissä testauksen automatisointi on lähes välttämätöntä kattavuuden varmistamiseksi (Kettunen 2009). Tämäkään ei aina ole mahdollista. Kun töitä on paljon ja väkeä vähän tekemässä niitä, jää testaus monesti ohjelmoijan vastuulle. Kuinka hän sen tekee ja missä laajuudessa.

Elinkaari-, kehittämis- tai vaihejakomallit, millä nimellä kukin haluaa kutsua näitä ohjelman kehittämiseen liittyviä malleja kuvaavat eri vaihtoehtoja, joista olisi hyvä valita vähintään yksi malli. Käytännössä projektissa voi olla useampiakin malleja käytössä ja jopa yhtä aikaa. Esimerkiksi voidaan aloittaa projekti spiraalimallilla, joka auttaa tarkentamaan määrityksiä ja vaatimuksia useiden toistojen avulla prototyyppiin. Vaatimusmäärityksien ollessa vajavaisia, voidaan hakea apua vesiputousmallista ja jossa tehdä aluksi suunnittelu, seuraavaksi koodaus, testaus ja ylläpitovaihe.

2.3 Ohjelmien testaus

Testauksen tarkoitus on löytää virheitä. Testaukseen liittyvät työvaiheet ovat testauksen suunnittelu, testiympäristön luonti, testin suorittaminen ja tulosten tarkastelu. Näihin ja ohjelmoinnin aikaiseen koodin virheiden korjaamiseen kuluu keskimäärin yli puolet ohjelmisto projektin resursseista. LabVIEW ohjelmoinnissa tästä ajasta menee suurin osa ohjelmoinnin aikaiseen testaukseen ja virheiden korjaamiseen.

Moduulitestauksessa eli yksikkötestauksessa etsitään vikoja yksittäisistä moduuleista. Integrointitestauksessa etsitään vikoja moduulien yhteistoiminnasta ja järjestelmätestauksessa koko järjestelmän toiminnoista ja suorituskyvystä (Märijärvi & Haikala 2001, 28). Useimmat LabVIEW sovellukset ohjaavat tai lukevat ulkopuolisia laitteita ja myös kommunikoivat muiden sovellusten kanssa. Tällöin tulisi varmistaa testaamalla, miten sovellus toimii muiden sovellusten kanssa. Järjestelmätestauksessa, tulisi miettiä seuraavia kysymyksiä:

- Ovatko suorituskykyvaatimukset täyttyneet?
- Jos sovellus kommunikoi toisen sovelluksen kanssa, se osaa käsitellä toisen sovelluksen ilmoittamat odottamattomat virhetilanteet hyvin?

(National Instruments 2003.)

Useasti puhutaan ohjelmien testauksen yhteydessä validioinnista. Validiointi eli kelpoistaminen on sarja toimenpiteitä, joilla osoitetaan, että ohjelma toimii asiakkaan haluamalla tavalla. Validiointi varmistetaan testaamalla tuotetta oikeassa käyttöympäristössä. Puhutaan myös lopputestauksesta. (Märijärvi & Haikala 2001, 86.)

Luvussa 2.2.3, jossa käsitellään ohjelman kehittämisen V-mallia, kerrottiin tämän mallin perusidea, jossa testaussuunnitelmia tehdään ohjelman suunnittelun alusta lähtien. Ohjelman määrittelyvaiheessa mietitään jo mitä järjestelmätestauksessa tulisi testata, arkkitehtuurisuunnittelussa suunnitellaan integrointitestauksen testaussuunnitelma, moduulisuunnittelun yhteydessä suunnitellaan moduulitestauksen vaatimukset ja ohjelmoinnin yhteydessä suunnitellaan yksikkötestausta ja myös testataan yksikkötestaus. Järjestelmätestauksen jälkeen voi monesti myös seurata erillinen kenttätestaus ja hyväksymistestaus. Näitä kenttätestauksia ja hyväksymistestauksia kutsutaan monesti myös Alfa- ja Beetatestaukseksi, regressio- ja käytettävyystestaukseksi.

Yksikkötestauksella LabVIEW ohjelmoinnissa ajatellaan pääasiassa yksittäistä vi-tiedostoa tai vi:tä, jossa on muutamia ali vi, mutta joka suorittaa yksittäisen toiminnan. Yksikkötestaus jää ohjelmoijan vastuulle ja tehdään ohjelmoinnin yh-

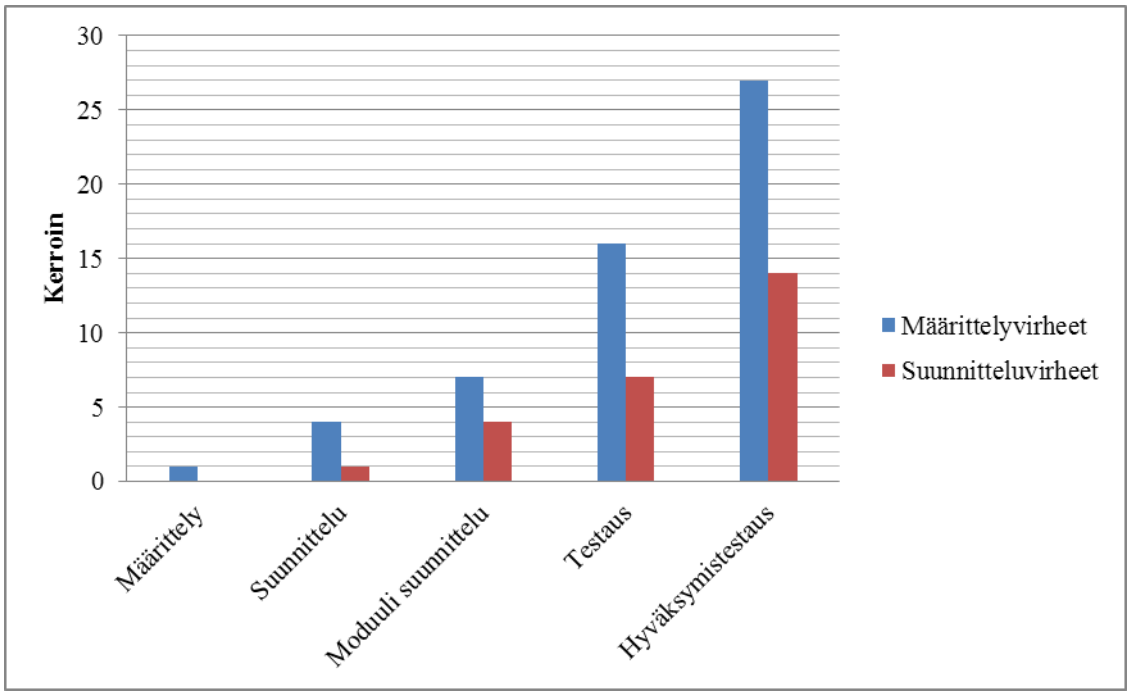
teydessä. Yksikkötestaus olisi hyvä automatisoida varsinkin suurempia ohjelmistoja tehtäessä. Yksikkötestauksen automatisoinnista kerrotaan kohdassa LabVIEW ohjelmien testaus teoriassa, jossa kerrotaan yksikkötestaustyökaluista.

Moduulitestaus LabVIEW:lla on esimerkiksi yksittäisen mittalaitteen ohjaukseen tehdyn ajurin testaamista, johon on tehty kuitenkin useampia ominaisuuksia. Moduulina voidaan ajatella jonkin ohjelman irrallista osaa, joka ei välttämättä yksittäin toimi, vaan voi tarvita lisäosia toimiakseen. Moduuleja voidaan testata myös hyvin pitkälle samalla lailla kuin yksikkötestaustakin tehdään. Moduulitestauksen tekee yleensä ohjelmoija itse tai se voidaan automatisoida. Automatisoitaessa moduulitestaus, tarvittavat parametrit ja syötteet tulevat ohjelmoijalta, tai jos on tehty erillistä moduulisuunnittelua, sieltä tulee vaatimukset moduulin toiminnasta. Moduulitestauksen automatisointi voidaan suorittaa yksikkötestaustyökaluilla.

Integrointitestauksessa yhdistetään moduuleja yhdeksi kokonaisuudeksi, tai lisätään uusi moduuli valmiiseen ohjelmistoon. Tässä testauksessa painopiste on rajapintojen toimivuuden tutkiminen. Monesti tämä testaus tehdään myös moduulitestauksen yhteydessä eikä erillisenä testauksena. Testaustuloksia verrataan monesti teknisiin määrityksiin, kuinka ohjelman tulisi toimia ja onko rajapinnat määrityksien mukaisia. Tämän vaiheen voi tehdä myös joku muu kuin ohjelmoija itse. Esimerkkinä voisi ottaa uuden mittalaitemoduulin lisääminen ohjelmistoon. Kun moduuli on testattu irrallisena, voidaan se liittää koko ohjelmistoon, jonka jälkeen testaaja käy läpi kaikki mahdolliset tilanteen mitä voi tulla eteen ja testaa, että ko. ohjelmistomoduuli toimii oikein kaikissa tilanteissa. (Märijärvi & Haikala 2001.)

Järjestelmätestauksessa tuloksia verrataan ohjelmiston määrittely dokumentaatioon ja testaus kohdistuu koko järjestelmään. Järjestelmätestaukseen voi liittyä myös kenttä- ja hyväksymistestaus tai ne voivat myös tulla tämän jälkeen niin kuin aikaisemmin kerrottiinkin. Järjestelmätestausta ei tulisi suorittaa ohjelmoijien itse vaan mielellään ulkopuolisen henkilön, joko talon sisältä tai asiak-

kaan puolelta. Tässä testauksessa testataan myös ohjelman käytettävyyttä. Testataan käyttöohjeiden selkeyttä ja tilanteita, jos syötetään vääriä syötteitä ohjelmaan. Mitä ylempänä V-mallin mukaisia testauksia ollaan, sitä kalliimmaksi virheiden korjaus tulee, josta kuvio 10 kertookin. (Tauriainen 2005, 23- 24.)



Kuvio 10. Virhekustannusten kertoimia (Märijärvi & Haikala 2001, 257.)

Pitää päättää etukäteen, minkä tason testausta ohjelma edellyttää. Aikataulut ovat monesti tiukkoja, mistä syystä testaus saa vähemmän huomiota ja käytetään enemmän aikaa muuhun kehitykseen. Tietyn tason testaus säästää loppupelissä aikaa. Ohjelmoijien on ymmärrettävä, kuinka paljon on tarpeellista testata. Testaussuunnitelmat ja testausraportit auttavat näkemään kuinka paljon on testattu ja kuinka paljon olisi hyvä vielä testata. Testaus tulisi suorittaa mahdollisimman aikaisessa vaiheessa, koska mitä myöhemmin virhe löydetään, sitä kalliimmaksi sen korjaaminen tulee. Kuviossa 10 voidaan hyvin havaita edellä mainitut tilanteet. (National Instruments 2003.)

Staattiset menetelmät

Staaattisella koodin analysoinnilla tarkoitetaan ohjelmiston arkkitehtuuri- ja moduulisuunnitelmien tai itse ohjelmakoodin tarkastelua ja analysointia ilman ohjelmakoodin suorittamista. Analysointi voidaan suorittaa joko käsin tai automaattisesti koodin analysointiohjelmalla esimerkiksi National Instrumentsin VI Analyzer Toolkit. Staattinen analyysi voi paljastaa muuttujiin tai parametreihin liittyvät virhekäytöt, indeksien ja osoittimien virheellisen käytön tai funktiokutsuihin liittyvät ongelmat, kuten kutsumattomat funktiot. Staattista analyysiä voidaan suorittaa eri testaustasoilla, mutta näiden painottuminen testauksen alkupäähän pienentää kustannuksia. (Rauhala 2010.)

Dynaaminen koodin analysointi

Dynaaminen analyysi testausmenetelmänä on ohjelmiston varsinaista testaamista ohjelmakoodia suorittamalla niin, että pyritään löytämään virheitä. Tällöin keskitytään ohjelman ja koodin käyttäytymiseen ohjelman suorituksen aikana. Dynaamisella koodin analysoinnilla tutkitaan taulukossa 2 kerrottuja asioita.

Taulukko 2. Dynaaminen koodin analysointi

1.	Mikä kuluttaa muistia?
2.	Onko käsitelty kaikki virheet sovelluksessa?
3.	Mikä oli viimeinen tapahtuma, mikä esiintyy ennen ...?
4.	Mikä oli kutsu-ketju, joka johti meidät ...?
5.	Mikä on suorittava säie? (Säie on peräkkäisesti toimiva käskyjono, joka toimii muista säikeistä riippumatta; säikeellä on oma ohjelmalaskuri ja pino. Saman prosessin säikeet käyttävät kuitenkin tämän prosessin muistialuetta ja käyttöjärjestelmän resursseja)
6.	Olenko todella tullut haluttuun event-caseen?
7.	Mitä tapahtui rakenteen sisällä?
8.	Missä järjestyksessä nämä tapahtumat tapahtuvat?
9.	Onko daemon(palvelu) prosessia käynnissä taustalla?
10.	Toimiiko koodi eritavalla käännettynä?

Black box – testaus

Ohjelmisto kuvitellaan "mustana laatikkona" — implementoinnista ei ole tarkkaa tietoa. Black box -testaamistapoihin kuuluu: samanarvoisiin jaotteluihin, raja-arvoanalyysi, parien testaus, satunnaisella datalla testaaminen, mallipohjainen testaaminen, jäljitettävyysmatriisi, tutkiva testaaminen ja määrittelyyn perustuva testaus.

Hyödyt ja haitat: Black box -testaaminen ei ole sidottu koodiin, jolloin testaajan näkökulma on hyvin yksinkertainen: Koodin "täytyy" sisältää virheitä. Periaatteen "Kysy ja saat vastauksen" käyttäminen löytää virheitä sieltä, missä ohjelmoija ei niitä löydä. Toisaalta Black box -testaaminen on kuin "kävelisi pimeässä ilman taskulamppua", koska testaaja ei tiedä, kuinka ohjelmisto on rakennet-

tu. Tämän seurauksena tulee tilanteita, joissa käyttäjä kirjoittaa monta testitapausta tarkistaakseen jotain, jonka voisi testata yhdellä tapauksella ja/tai osa jää testaamatta kokonaan.

White box – testaus

White box – testaus on suomeksi käännettynä ”lasilaatikkotestaus”. Tässä testaaja tietää ohjelman koodin ja pystyy käyttämään hyödyksi testitapausten suunnittelussa. Testaaja pyrkii löytämään virheellisiä kohtia ja toiminnallisuuksia ohjelman koodin avulla.

Gray box – testaus

Gray box testaus eli harmaa laatikko testaus, joka on sekoitus vähän kumpaakin edellä mainittua testausmenetelmää, muttei selvästi kumpikaan. Harmaa laatikko testauksessa on otettu parhaat puolet Black box ja White box testauksista.

2.4 LabVIEW ohjelmien testaus teoriassa

LabVIEW:lla tehtyjen ohjelmien testaus ei merkittävästi poikkea normaalista ohjelmien testauksesta. National Instruments tarjoaa erilaisia lisäosia LabVIEW ohjelmien testaukseen. Näitä ovat:

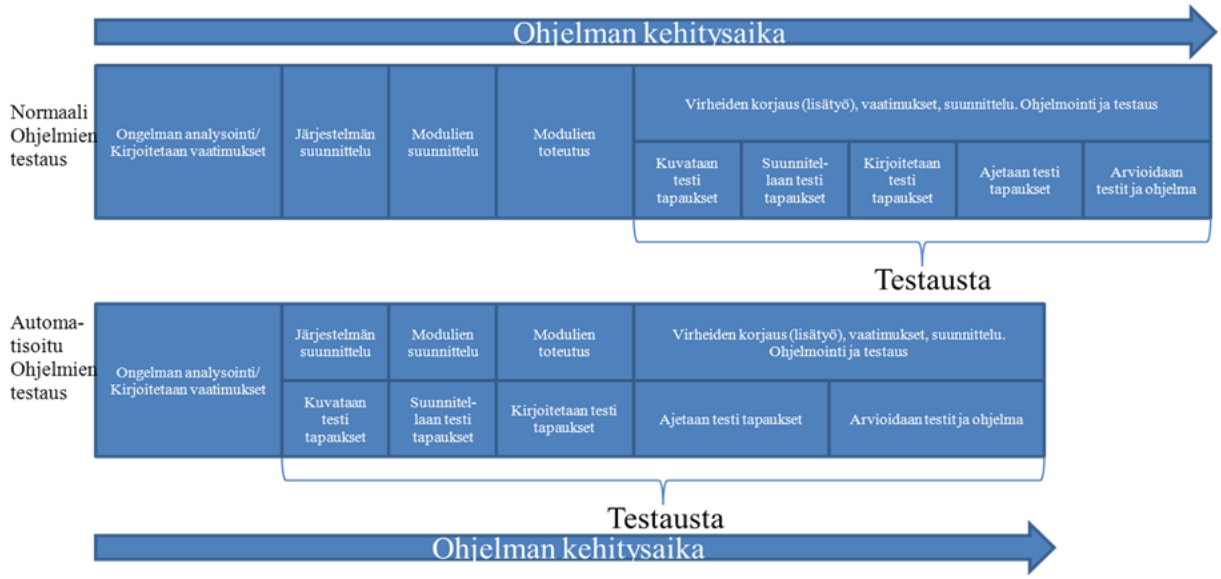
1. VI Analyzer Toolkit, staattiseen koodin analysointiin
2. Desktop Execution Trace Toolkit, dynaamiseen koodin analysointiin
3. Unit Test Framework Advanced, toiminnalliseen koodin validiointiin/ testaukseen

Lisäosia olisi otettava käyttöön järjestyksessä, mitä kompleksisempi ja kriittisempi ohjelma on, sitä useampi lisäosa olisi hyvä olla käytössä. (Kerry 2010.)

VI Analyzer Toolkitin avulla voidaan määrittää yli 80 testiä, jotka se automaattisesti tekee kaikille sovelluksessa oleville vi:lle tarkastellen koodia sen ulkoasun kannalta ja staattista koodin analysointia. Käyttämällä LabVIEW VI Analyzer Toolkit:ä vältetään vääriä koodaustekniikoita, jotka voivat vaikuttaa sovelluksen suorituskykyyn, toiminnallisuuteen tai ylläpitoon. Lisäksi voidaan valvoa tehokkaasti käytäntöjä ja koodaustyylejä tiimin ohjelmoijien välillä ja varmistaa koodin luettavuutta ja toiminnallisuutta. (National Instruments 2011d.)

Desktop Execution Trace Toolkit auttaa Windowsissa jäljittämään ohjelmasta LabVIEW VI:t ajon aikana, jotka voivat aiheuttaa odottamatonta toimintaa tai ongelmia koodissa. Se antaa kronologisen näkymän VI tapahtumista, jonotoiminnoista, referenssivuodoista, muistin varauksista, käsittelemättömistä virheistä ja aliohjelmien toiminnasta. Tällä voidaan ohjelmallisesti luoda käyttäjän määrittämistä tapahtumista lohkokaavio LabVIEW sovelluksesta. Tätä ohjelmaa voidaan käyttää paikallisesti siinä koneessa, jossa testattava ohjelma pyörii tai toiselta koneelta verkon yli, jolloin nähdään viimeiset tapahtumat esimerkiksi käyttöjärjestelmän kaatumistilanteessa. (National Instruments 2011a.)

NI LabVIEW Unit Test Framework Toolkit on yksikkötestaustyökalu. National Instrumentin virallinen kanta on, että kaikki vi:t testattaisiin tällä työkalulla tai ainakin moduuli tasolla. NI LabVIEW Unit Test Framework Toolkit auttaa automatisoimaan vi:n yksikkötestauksen, suorittamaan toiminnallisia validiointeja ja osoittamaan, että ohjelma käyttäytyy halutulla tavalla. Napsauttamalla hiiren kakkospainikkeella mitä tahansa vi:tä LabVIEW Project ikkunassa, luodaan yksikkö testaus, tai voidaan tuoda testiparametreja tekstitiedostosta muokkaamalla esimerkiksi Microsoft Excel:llä. Testit voivat sisältää useita testitapauksia, joissa määritellään lähtöarvot ja odotettu tulos, joka voi olla mikä tahansa tietotyyppi, kuten taulukko tai klusteri. (National Instruments 2011c.)



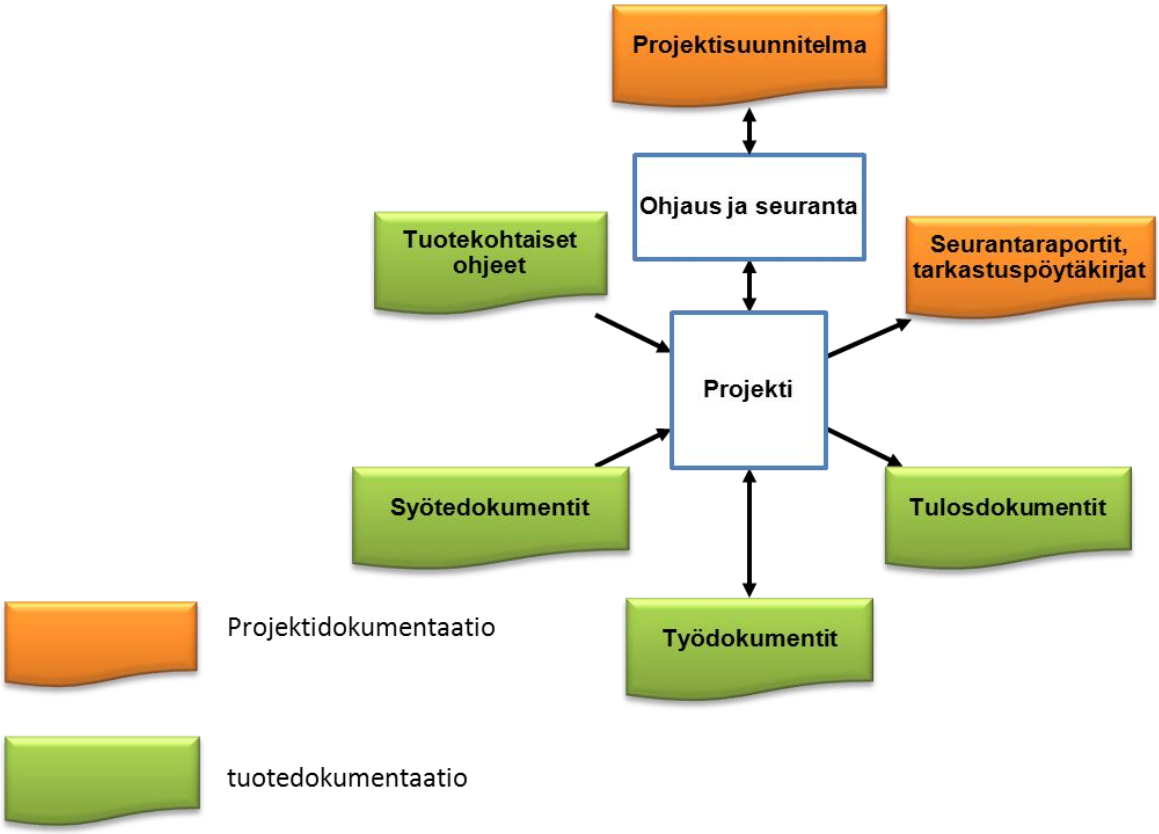
Kuvio 11. Testaus työkalujen käytön vaikutus kehitys aikaan (Poston 1996, 25).

Kuviossa 11 näytetään testaus työkalujen käytön vaikutus kehitysaikana, kun otetaan käyttöön automatisoitu ohjelmien testaus. Kun testitapaukset kuvataan järjestelmäsuunnittelun yhteydessä, tiedetään ilman erillistä pohdintaa, mitä tulisi testata. Moduulisuunnittelun yhteydessä mietitään testitapaukset ja toteutetaan testi toteutuksen yhteydessä. Tämä on samalla V-mallin mukainen ohjelman toteutusprosessi. Normaali ohjelmien testausprosessi kuvaa vesiputousmallia, jolloin seuraavaan kohtaan mennään, kun edellinen on saatu valmiiksi. Näiden kahden mallin ero on ohjelman kehitysaika, joka saavutetaan, kun asiat tehdään silloin kun ne ovat parhaiten mielessä.

2.5 Ohjelmien dokumentointi

Ohjelmistoprojektin dokumentoinnin määrä riippuu hyvin pitkälle projektin/ ohjelmiston laajuudesta. Ohjelmistoprojekteissa voi dokumentteja syntyä todella paljonkin. Keskeisimpiä tuotedokumentteja ovat toiminnallinen määrittely (määrittelydokumentti), tekninen määrittely (suunnitteludokumentti) sekä testausdokumentit (testaussuunnitelma) (Märijärvi & Haikala 2001, 60). Lisäksi projektis-

sa syntyy projektidokumentteja esimerkiksi projektisuunnitelma, seurantaraportit ja tarkastuspöytäkirjat. Edellä mainituista dokumenteista on havainnollistava kuvio 12, jossa ohjelmistoprojektista syntyviä dokumentteja on väreillä kerrottu mitkä dokumentit ovat projektidokumentaatiota ja mitkä tuotedokumentaatiota. Lisäksi yrityksen laatujärjestelmä voi vaatia dokumentointia. Tässä kehittämistyössä ei käsitellä laatujärjestelmien vaatimia dokumentointeja vaan keskitytään ennen kaikkea tuotedokumentointeihin ja projektin dokumentointiin siltä osin, kun ne koskevat ohjelmointia ja sen määrityksiä.



Kuvio 12. Ohjelmistoprojektista syntyviä dokumentteja

Jokaisessa ohjelmistoprojektissa ei välttämättä tarvitse tehdä kaikkia dokumentteja alusta lähtien esimerkiksi dokumentoitaessa kappaleessa 0 kerrotun Evo-mallin mukaan. Tämän mallin mukaisessa projektissa tehdään lisäyksiä, korjauksia ja muutoksia aikaisempaan ohjelmistoversioon, jolloin dokumentointia-

kaan ei tarvitse joka kerta uudestaan, vaan päivitetään olemassa olevia dokumentteja. Tällaisessa tapauksessa ei kuitenkaan saa jättää päivittämättä tuotetason dokumentteja, koska voidaan joutua ongelmiin ylläpidossa ja varsinkin kun aletaan taas tehdä seuraavaa versiota. Hyvin monesti kuitenkin nämä dokumentit jäävät kokonaan päivittämättä tai päivitetään aikaisintaan projektin lopussa. Tästä syystä olisikin hyvä tehdä esimerkiksi tuotteen tekninen määrittely hyvin yleisellä tasolla, jottei sitä tarvitse pienten muutosten takia edes päivittää. (Märijärvi & Haikala 2001.)

Projektin päätyttyä tuotedokumentaatiosta tehdään lisää esimerkiksi: käyttöohje, asennusohje, koulutusmateriaali sekä tekninen dokumentaatio. Suunnitteludokumentointi kannattaa sisällyttää itse ohjelmaan kommentteina, mitä missäkin tehdään, jolloin ohjelman muuttuessa ohjelmointivirheiden vuoksi, tarvitse päivittää muuta dokumentointia. (Märijärvi & Haikala 2001, 63.)

Dokumentteja varten kannattaa tehdä dokumenttimallit. ISO 9001 – standardi määrittelee tarkasti mitä dokumentista tulee löytyä. Dokumenttien olisi hyvä olla yhtenäisiä ja helposti luettavia (Märijärvi & Haikala 2001, 65). Myös itse lähdekoodi on osa dokumentaatiota. Tämän vuoksi olisi myös hyvä olla moduulien ulkoasu määritelty ja yhtenäinen, josta esimerkkinä kohdassa 2.1.4 esitetty taulukkossa 1 LabVIEW ohjelmoinnin tärkeimpiä säännötöjä/ ohjeita (National Instruments 2006a).

2.6 LabVIEW ohjelmien dokumentointi

LabVIEW ohjelmoijat, kuten kaikki muutkin ovat hyvin kiireisiä. Mutta se ei oikeuta olla kirjoittamasta hyviä kommentteja kaikkialle koodiin. Kommentoida pitäisi kaikki hankalammat koodin kohdat esimerkiksi matemaattiset algoritmit, mukautetut apuohjelmat. Hyvät kommentit eivät toista koodia tai selitä sitä. Ne selkeyttävät sen tarkoitusta. Kommentit kertovat ylemmällä tasolla mitä yritetään tehdä. Pitää

muistaa, että henkilö, joka katsoo tätä koodia pitemmänkin ajan jälkeen voi olla koodaaja itse. Myöhemmin on paljon nopeampi päästä perille koodista, jos se on hyvin kommentoitu. (National Instruments 2006a.)

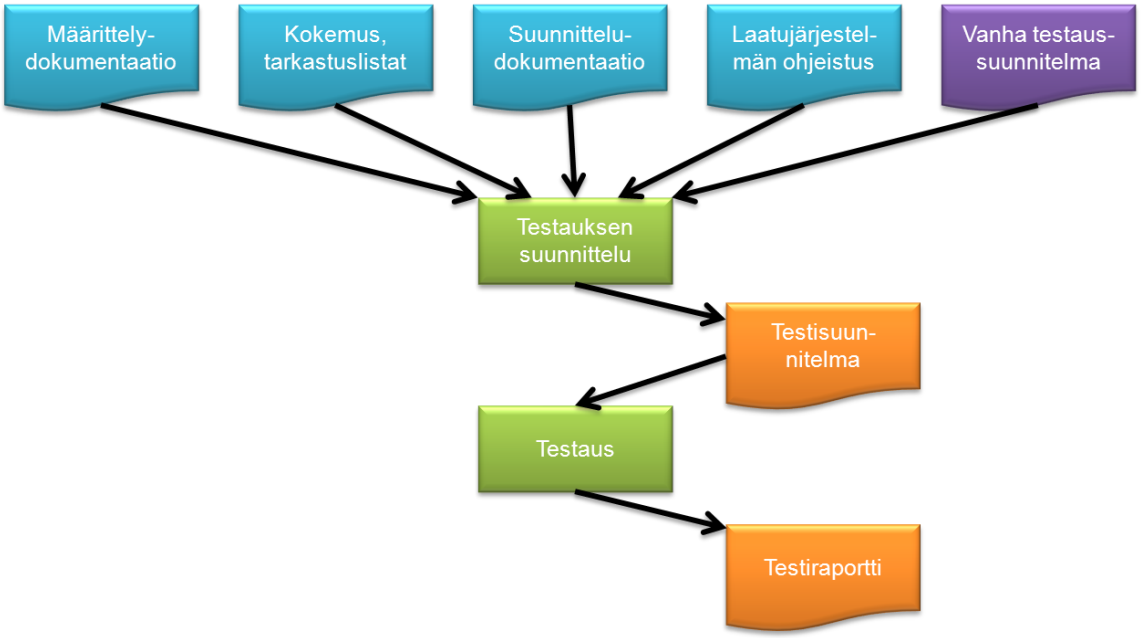
Suositeltavaa on kirjoittaa lyhyt (yleensä korkeintaan yksi tai kaksi lausetta) kuvaus jokaisen VI:n Info-näyttöön. Sitten lisätään kuvaukset kaikista controlleista ja indikaattoreista, jotka poikkeavat LabVIEW omista. Jos controllit ja indikaattorit on kuvattu siihen varattuun kenttään, käyttäjä voi nähdä ohjeikkunan, kun hän vie hiiren niiden päälle. Olisi hyvä, jos dokumentoitaisiin ne kaikki, mutta välttämättä ei ole aikaa siihen. Jos nämä kaikki edellä mainitut dokumentoitaisiin, ohjelman dokumentointi voitaisiin luoda automaattisesti käyttäen LabVIEW dokumentointi työkalua. Lisäksi olisi hyvä vähintään yrittää kommentoida kaikki rakenteet (While loop, For loop, Case rakenteet, sekvenssit, referenssit ja rekisterit). Tällä saadaan keskeiset osat koodia dokumentoitua. (National Instruments 2006a.)

2.7 Testauksen dokumentointi

Testauksesta voi tulla dokumentaatiota hyvinkin paljon, jos kaikista tehdyistä testeistä tehtäisiin testaussuunnitelma ja testausraportti. Esimerkiksi ISO 9000-3 ohjeessa testaussuunnitelmat ja -raportit tulisi tehdä integrointitestauksesta ja järjestelmä- eli hyväksymistestauksesta. Yksikkö- ja moduulitestauksessa testaussuunnitelman korvaa erillinen ohjeistus, jossa voi olla suuntaviivat miten tulisi testata ja kuinka kattavasti. Testaussuunnitelma, joka voi pienemmissä projekteissa olla yksi dokumentti, pitäen sisällään sekä integrointi- järjestelmätestauksen, voi olla sisällytettynä jo projektisuunnitelmaan, ohjelman määrittelydokumenttiin. (Märijärvi & Haikala 2001, 281.)

Testaussuunnitelmasta tulisi selvitä, mitä testejä tehdään, miten tehdään ja miten ohjelman tulisi toimia halutussa testissä. Testaussuunnitelmassa tulisi mää-

ritellä ennen kaikkea testauksen laajuus ja kuinka kauan tehdään testausta. Halutaanko, että testausta tehdään niin kauan, että kaikki virheet on korjattu ja testattu vai mikä on haluttu raja ts. kuinka paljon sallitaan virheitä ja minkälaisia.



Kuvio 13. Testauksen suunnittelu ja dokumentit (Märijärvi & Haikala 2001, 280)

Kuviossa 13 havainnollistetaan, mitkä kaikki asiat vaikuttavat testaussuunnitelman laadintaan. Testisuunnitelman pohjana kannattaa hyvin monesti käyttää vanhaa testaussuunnitelmaa. Testisuunnitelman tekemiseen vaikuttavat ohjelman määrittelydokumentti, suunnittelijan kokemus ja mahdolliset tarkastuslistat. Ohjelman suunnitteludokumentaatiosta ja laatujärjestelmästä tulevat ohjeistukset esimerkiksi dokumentin ulkoasuun ja sisältöön. Testisuunnitelman mukaisten testauksien jälkeen saadaan tehtyä raportti, jossa kerrotaan saadut tulokset ja verrataan niitä testisuunnitelmassa oleviin raja-arvoihin.

Testiraportissa tulisi virheet kirjata huolella ylös ja kuvata löytynyt virhe. Lisäksi olisi hyvä kertoa kuinka vakavasta virheestä on kysymys ja miten se löydettiin.

Loppukäyttäjiltä tulevia virheitä varten on monesti olemassa erityinen lomake, joka voi olla myös ohjelman sisälle rakennettu ja jonka voi lähettää verkon yli haluttuun paikkaan.

3 TYÖNTAVOITE JA MENETELMÄT

3.1 Toimeksiantaja

Tämän työn toimeksiantaja on Sasken Finland Oy. Sasken on langattoman teknologian alalla toimiva yritys, jonka toimialueeseen kuuluu: tuotekehitystä, tutkimusta, suunnittelupalveluja, elektroniikkasuunnittelua, tietoliikennepalveluja ja tietoliikennelaitteita. Sasken Finland on entinen Botnia Hightech Oy, jonka Intialainen pörssiyhtiö osti vuonna 2006. Botnia Hightech Oy on perustettu 1989. Sasken Finland Oy toimii Saskenin tytäryrityksenä. Saskenilla on työntekijöitä yhteensä 3200+ ja Sasken Finland Oy:ssä niistä on noin 190 kehittämistehtävän alussa ja lopussa työntekijöitä oli noin 100. Liikevaihto oli vuonna 2010 noin 18 miljoonaa euroa. Liiketoiminnan toimipaikat ovat Suomessa Kaustisella, Tampereella ja Oulussa. (Sasken 2011.)

Sasken Finland Oy:ssä on kaksi linjaa:

- HW liiketoimintalinja on erikoistunut langattoman teknologian tuotteiden tarjontaan, laitteisto- ja mekaanikka suunnitteluun ja ratkaisuihin. Erityinen vahvuus on RF-suunnittelu osaaminen. Se sisältää myös testaus-palvelut ja näin ollen langattoman teknologian tuotteiden T&K:ssa tarvittavat osa-alueet löytyvät yhdestä paikasta.

- SW liiketoimintalinja on keskittynyt ohjelmistojen suunnitteluun, integrointiin ja konsultointiin. Se edustaa huippuosaamista mobiiliohjelmistojen kehittämisessä. SW liiketoimintalinja on jatkuvasti mukana useissa vaativissa ohjelmistoprojekteissa.

HW tulee englannin kielensanasta hardware, joka on suomeksi laitteisto. SW tulee myös englannin kielen sanasta software, joka on ohjelmisto.

3.2 Tutkimuksen tavoitteet

Tämän tutkimuksen tavoitteena on tuottaa selkeä kuvaus siitä, miten tulisi suorittaa LabVIEW ohjelmalla toteutettujen ohjelmien testaus, validiointi ja dokumentointi. Tällä hetkellä jokainen suunnittelija testaa, validioi ja dokumentoi tekemänsä ohjelmat itse kokonaan tai toisen henkilön avustuksella. Validioinnilla tarkoitetaan valmiiden ohjelmien lopputestausta, jossa käydään läpi kaikki tarvittavat ohjelman osa-alueet ennen ohjelman siirtymistä asiakkaan käyttöön. Yrityksessä ei ole minkäänlaista prosessikuvausta tai ohjeistusta ohjelmien testaamiseen, validiointiin ja dokumentointiin. Kehittämistehtävän aikana on tarkoitus tehdä prosessi tai ohjeistus, jota jokaisen suunnittelijan tulisi tulevaisuudessa käyttää. Näistä tarpeista muodostuvat seuraavat tutkimusongelmat:

Päätutkimusongelma:

LabVIEW ohjelmien testaus ja dokumentointi

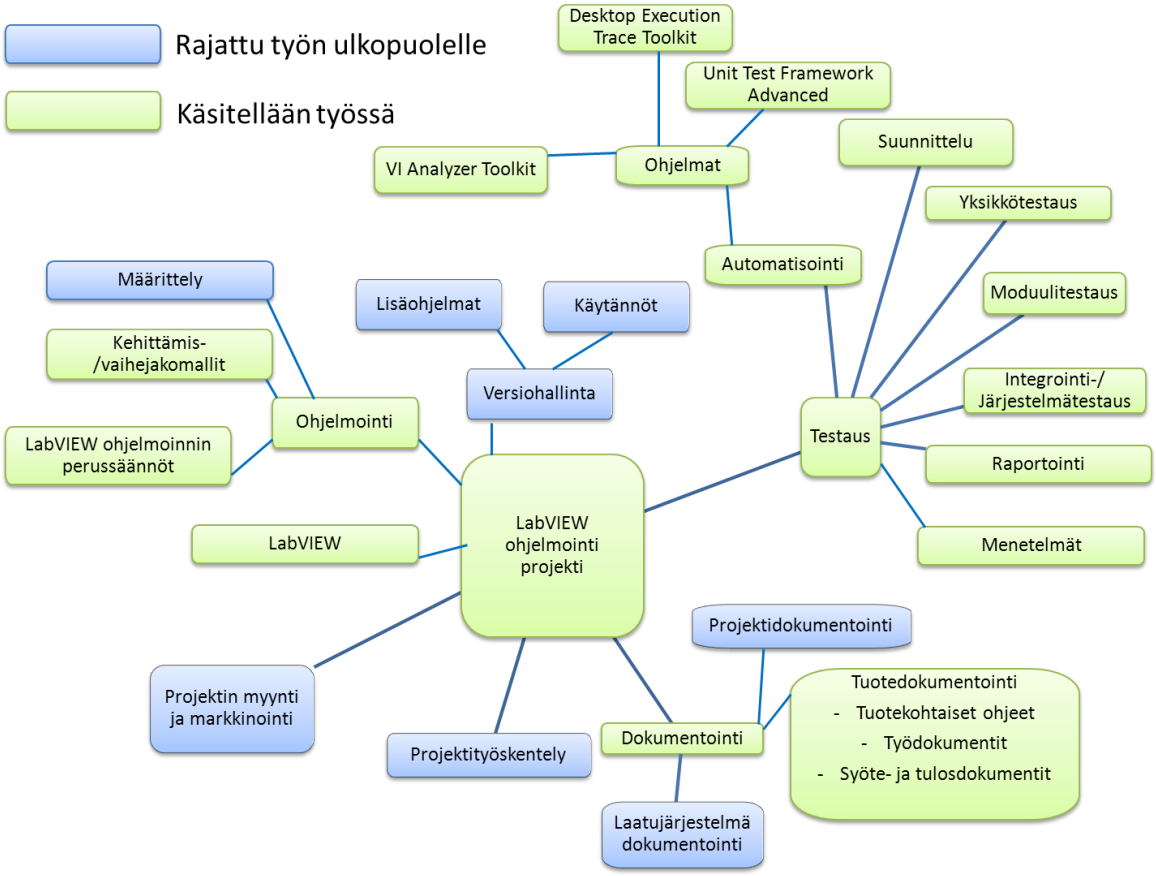
Ja alakysymykset:

1. ***Miten testausprosessin tulisi edetä?***
2. ***Miten lopputestaus tulisi suorittaa?***
3. ***Mitä tulisi dokumentoida LabVIEW:lla ohjelmoitaessa?***

Testauksen automatisointiin yrityksellä ei tällä hetkellä ole taloudellisesti mahdollisuuksia. Testauksen suunnitelmallinen kehittäminen on kuitenkin yritykselle tärkeää ja suurin hyöty saadaan, kun kartoitetaan LabVIEW testauksen nykykäytäntö sekä määritellään, valitaan ja dokumentoidaan sopivat käytännöt ja prosessit tai ohjeistukset millä mennään eteenpäin.

3.3 Työn rajaus

Tässä työssä on rajattu käsiteltävät alueet kuvion 14 mukaan. Tässä työssä keskitytään LabVIEW:lla tehtyjen ohjelmien testaukseen ja dokumentointiin. Tässä työssä ei käsitellä ohjelmien versionhallintaa, projektityöskentelyä, projektien ja ohjelmien myyntiä eikä markkinointia. Dokumentoinnissa ei oteta kantaa laatujärjestelmien vaatimiin dokumentointeihin, eikä projektin vaatimiin dokumentointeihin. Ainoastaan tuotedokumentointiin kiinnitetään huomiota.



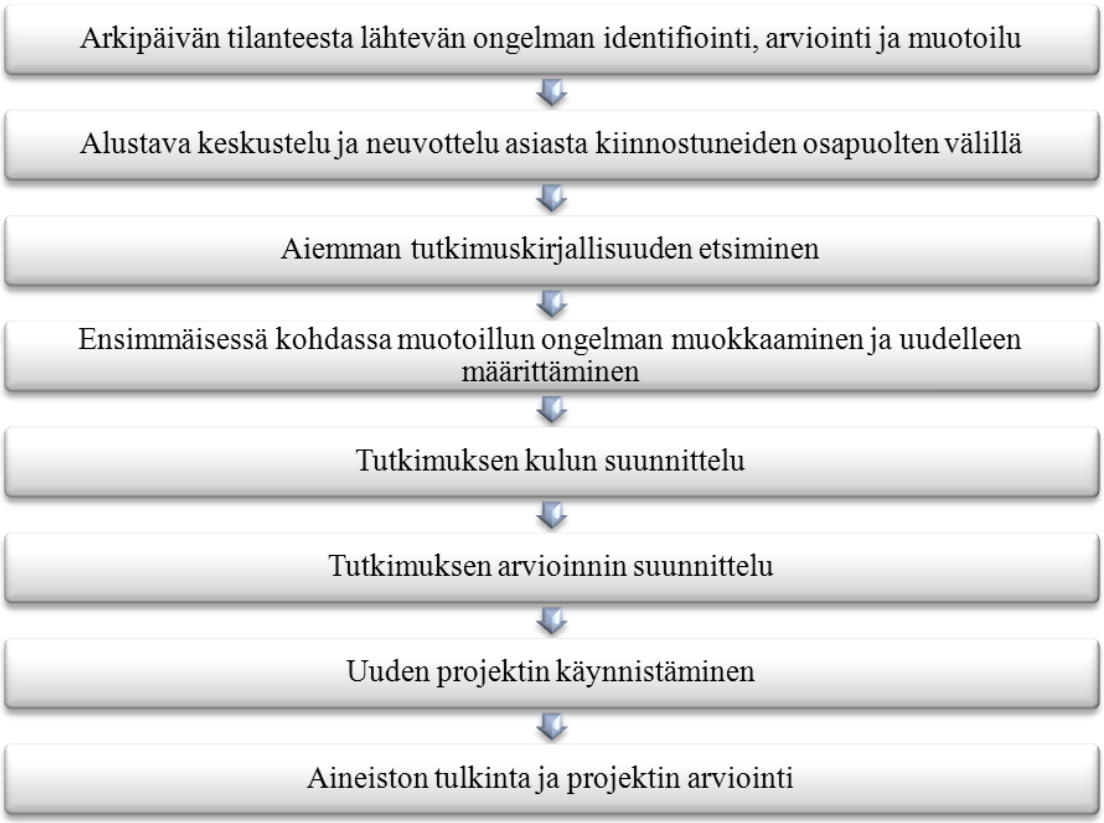
Kuvio 14. Kehitys tehtävän rajaus

Ohjelmointia itsessään käydään hyvin pintapuolisesti läpi. Ohjelmointia käydään läpi, jotta ymmärretään, minkälaista on LabVIEW ohjelmointi. Teoriassa käsitel-

lään eri vaihejako malleja ja niitäkin ymmärtääksemme, missä vaiheessa testaukseen pitäisi kiinnittää huomiota. Suurin käsiteltävä alue on testaus ja siihen vaikuttavat osa-alueet.

3.4 Tutkimusmenetelmät

Koska kysymyksessä on kehittämishanke, on pohdittava, mitä tulisi kehittää. Kehittämisessä muuttamalla toimintatapaa luodaan parannusta aiempaan tilanteeseen. Tässä kehittämistehtävässä tullaan käyttämään laadullista tutkimusmenetelmää ja prosessiin kohdistuvaa tutkimusta. Tätä voisi kutsua myös toimintatutkimukseksi. Tällä yritetään parantaa käytännön ongelmia, muuttamaan käytäntöjä ja kehittämään olemassa olevaa käytäntöä paremmaksi (Metsämuuronen 2008, 29). Kuviossa 15 on havainnollistettu toimintatutkimuksen kulku.



Kuvio 15. Toimintatutkimuksen kulku (Metsämuuronen 2008, 32)

Toimintatutkimusta kritisoidaan, koska tutkimuskohde on tilanteeseen sidottu ja spesifi. Toimintatutkimuksen otos on hyvin rajoitettu, eikä siis ole edustava, eikä näin ollen voida tuloksia yleistää. Tästä saadut tulokset ovat päteviä tutkitussa tapauksessa. Tässä tutkimusmenetelmässä myös teorian ja käytännön kytkeminen toisiinsa on usein hankalaa. (Metsämuuronen 2008, 32.)

Tämän kehittämistehtävän/ tutkimuksen tarkoitus on selkeyttää testauksen, validioinnin ja dokumentoinnin nykytila. Tähän päästään teemahaastattelemalla tätä työtä tekeviä henkilöitä heidän näkemyksistä, miten pitäisi testaus suorittaa ja myös miten he todellisuudessa tekevät ohjelmien testauksen, validioinnin ja dokumentoinnin. Haastattelujen jälkeen kaikki vastaajat osallistuvat yhteispalaveriin, jossa keskustellaan parhaista käytännöistä, mahdollisten lisäosien hankinnasta ja tähän kehittämistehtävään liittyvistä asioista.

Teemahaastattelu on puoli strukturoitu haastattelu, jossa on haastattelulle tehty runko, jotta kaikilta haastateltavilta kysyttäisiin samoista asioista välittämättä kysymysten muotoilusta tai järjestyksestä (Ruusuvuori & Tiittula 2005). Haastattelut nauhoitetaan, jotta ne voidaan raportoida tarkemmin ja myös haastattelussa voidaan keskittyä olennaiseen. Laadullisen tutkimusmenetelmän luotettavuudesta ei löydy yhtenäistä käsitystä, vastuu tulosten yleistettävyydestä tai siirrettävyydestä jää myös lukijalle. Niin kuin aikaisemminkin on tässä luvussa kerrottu, varsinkaan laadullisessa toimintatutkimuksessa on hyvin vaikea yleistää saatuja tutkimus tuloksia laajalti (Metsämuuronen 2008).

Teemahaastattelu on keskustelua, jolla on etukäteen päätetty tarkoitus. Teemahaastattelussa haastattelun rakenne pitää pysyä haastattelijan hallinnassa. Teemahaastattelun etu on siinä, että kerättävä aineisto rakentuu aidosti vastaajan henkilön kokemuksista käsin. Teemahaastattelussa piilee suuri vaara, että vastaaja/ henkilö ja hänen kertomuksensa alkaa johdatella haastattelun kulkua liikaa. Silloin syntyvän aineiston eri haastattelut eivät ole riittävässä määrin samanlaisia ja vertailukelpoisia teemarakenteensa puolesta. (Tilastokeskus 2011.)

Tämän hetkisistä tavoista ja mielipiteistä kerätään parhaat ominaisuudet ja tavat, tutkitaan yleisiä ohjelmistokehityksen testaukselle ja dokumentoinnille kehiteltyjä prosesseja. Näistä tämän hetkisistä käytännöistä ja teorioista tehdään uusi paranneltu testaus ja dokumentointi prosessi tai ohjeistus LabVIEW:llä toteutetun ohjelman testaukseen ja dokumentointiin Sasken Finland Oy:ssä. Tämä uusi prosessi tai ohjeistus dokumentoidaan ja otetaan käyttöön kouluttamalla suunnittelijat.

4 TUTKIMUSTULOKSET

4.1 Taustaa

Haastateltavia oli yhteensä viisi henkilöä, joista vastaaja E oli National Instrumentin myyntipäällikkö ja teknisen tuen henkilö. Muut neljä vastaajaa oli Saskenilla LabVIEW ohjelman parissa työskenteleviä henkilöitä. Vastaajat olivat käyttäneet LabVIEW:a kolmesta kahdeksaantoista vuotta, keskimäärin kahdeksan ja puoli vuotta. Ohjelmointikokemusta henkilöillä oli 10–18 vuotta, keskimäärin 12 vuotta. Vastaajat olivat ohjelmoineet keskimäärin kolmella muullakin ohjelmointikielellä esimerkkinä: vastaaja B kertoi ohjelmointikokemusta olevan C, BHP, Java ja C++.

4.2 Testauksien tämän hetkinen tilanne

4.2.1 Yksittäisien VI:n testaus

Vastaajat testasivat tällä hetkellä yksittäiset vi:t ajamalla lähdekoodia LabVIEW:lla satunnaisilla syötteillä ja katsomalla mitä saadaan tulokseksi laittamalla syötteet halutulle alueelle ja tarvittaessa raja-arvoilla tai niiden yli. Ei ole mitään tiettyä logikkaa vaan tapauskohtaisesti testataan. Vastaaja A kertoi käyttäneensä myös ongelmien ratkomiseen Profile työkalua muistivuotojen tarkistamiseen sekä ajamalla ja debuggaamalla lampun kanssa, joka näyttää johdotuksissa sillä hetkellä olevat arvot ja ohjelman ajojärjestyksen. Debuggaus on ohjelmistotuotannon osa, jossa testauksessa löytyneen virheellisen toiminnan aiheuttanut virhe paikallistetaan ja korjataan. Myös virheiden hallinta ja niistä saatu informaatio oli tärkeässä osassa ohjelmoinnin aikaista testausta (vastaaja A).

Tulevaisuudessa yksittäisen vi:n testaus tulisi haastattelujen pohjalta tehdä hyvin kattavasti. Vastaaja C sanoi esimerkiksi, että kaikki vi:t pitäisi kuitenkin ajaa läpi ja tarkistaa toiminta. Tarkistaminen tulisi suorittaa kokeilemalla ajamalla erilaisilla syötteillä raja-arvojen sisällä, raja-arvoilla ja raja-arvojen ulkopuolisilla arvoilla. Myös muistivuodot tulisi tarkistaa ja mahdolliset aikaa vievät kohdat, varsinkin jos ko. vi:tä ajetaan useita kertoja monessa paikassa. Tähän muistivuotojen ja pyörimisajan tarkistamiseen lähes kaikki suosittelivat käytettäväksi LabVIEW ohjelmasta löytyvää ”Profile Performance and Memory” työkalua.

NI LabVIEW Unit Test Framework Toolkit on yksikkötestaustyökalu. Tämä työkalu oli sellainen, jonka haastatteluissa vastaajat A, D ja E halusivat käyttöön ja pitivät järkevänä testauksen automatisointityökaluna. Näiden haastattelujen jälkeen pidetyssä yhteispalaverissa kuitenkin todettiin yhteisymmärryksessä, ettei tällä hetkellä ole järkevää ottaa tätä lisäosaa käyttöön vähäisten resurssien vuoksi.

4.2.2 Mittalaiteajurien testaus (moduuli testaus)

Mittalaiteajurien testaaminen suoritettiin tällä hetkellä hyvin monella tavalla. Vastaaja A kertoi kokeilevansa mittaukset ensin käsin, näppäilemällä mittalaitetta, tämän jälkeen LabVIEW:llä remotena (GPIB väylän kautta). Kun itse mittaus on saatu tehtyä, aloitetaan koodaus. Ohjelma tulee tarkistaa vielä, jotta käskyt menevät oikeanlaisina perille asti. Vastaaja A ei tee kuitenkaan tulosten vertailuja, koska yleensä uupuu esimerkiksi kaapeleiden kalibroinnit ym., jolloin tulosten vertailu aikaisemmin saatuihin tuloksiin on hyvin vaikeaa.

Jotkut suunnittelijat taas testasivat pelkästään käännetty versio (vastaaja C). Pääasiassa yksittäiset vi:t testattiin jollakin datalla tarkistaen samalla, että komennot toimivat ja menevät mittalaitteelle saakka. Tämän jälkeen mittalaiteajureita testattiin kokonaisena, tarkistaen komentojen meneminen mittalaitteelle ja

niiden oikea järjestys. Mittalaiteajurien testaus suoritettiin lähdekoodina kirjoittamalla parametrit mittalaiteajurien käyttöliittymään käsin, tai ajamalla käännettyä versiota valmiilla testijonolla. (vastaajat B ja E.)

Mittalaite ajurien testaamiseen haluttiin suurinta muutosta. Suunnittelijat halusivat erillisen testausohjelman, joka katsoisi mitä parametreja ja käskyjä mittalaiteajurille voisi syöttää. Ohjelmalla voisi testata mittalaiteajuria valituilla parametreilla tarkistaen meneekö komennot mittalaitteelle ja saadaanko halutunlaisia tuloksia. Suunnittelija kertoi seuraavasti: ”Olen huomannut ettei se riitä vielä testaukseksi vaikka alimmalla tasolla käskyt toimisikin. Pitää koko järjestelmän kautta ajaa oikeita testi caseja ja sieltä ne vasta todelliset virheet tulevat, joita ei koskaan osaa ennakoida, eikä muista kuin se kutsuu drivereita ja mitä parametreja (vastaaja B).”

Mittalaiteajurien testaus tulisi suorittaa vastaajan E mielestä ihan samalla menetelmällä oikeastaan kuin yksittäiset vi:t. Yksittäisen funktion testaaminen voidaan automatisoida hyvin pitkälle, mutta mittalaitetta ei välttämättä voida automatisoida, jos se on ulkoinen mittalaite, kun siellä on väyläliikennettä. Meidän pitää jollakin verifioida miten se mittalaite todellisuudessa käyttäytyy. Tällainen testaaminen ei mene välttämättä automaattisesti, vaan puoliautomaattisesti. Jonkun täytyy lukea mittalaitteen ruudusta, missä tilassa mittalaite on. Periaatteessa yksikkötestaustyökalua voidaan käyttää mittalaiteajurienkin testaukseen.

Haastateltavien A ja B mielestä mittalaiteajurien testauksessa tulisi käyttää testaukseen fyysistä mittalaitetta, kun ei ole emulaattoria, jolle voisi lähettää suoraan GPIB komennot ja tarkistaa oikea muoto. Mittalaite ja NI spy päällä yhtä aikaa ja tarkistetaan, ettei mittalaite herjaa muodosta tai lähetä vääriä käskyjä. Pitää tietää minkälaiseen tilaan mittalaite halutaan ja tarkistetaan, että mittalaite on halutussa tilassa ja onko käskyt mennyt mittalaitteelle perille. Vastaajan C mielestä ajureiden testaaminen on tällä hetkellä hankalaa, koska meillä pitäisi olla jokin ohjelma, jonka sisään ajuri laitetaan. Tämä helpottaisi testausta, ettei tarvitsisi kirjoittaa parametrointeja joka käskyyn erikseen. Testaaminen on hidasta ja tuskallista ja on

suuri vaara, ettei välttämättä kaikkia tule testattua. Olisi hyvä olla erillinen ohjelman, jolla testata. Tämä ohjelma näyttäisi, mitä funktioita ja parametreja ajurissa olisi. Tämä sama asia tuli esille myös muiden haastateltavien kohdalla.

4.2.3 Ohjelmien testaaminen (integrointi-/ järjestelmätestaus)

Kokonaisten ohjelmien testaaminen oli hyvin vaihtelevaa. Jotkut olivat testanneet ohjelmia hyvinkin paljon kääntämisen jälkeen. Jotkut olivat testanneet vain kriittiset kohdat ja katsoneet, että pääpiirteissään toimii, luottaen alimoduulien testaukseen. Voisi sanoa, että hyvin vaihteleva määrä testausta riippuen henkilöstä ja toki myös minkä laajuisesta ohjelmasta oli kysymys. Esimerkiksi: ” Aika vähälle jää kokonaisuuden testaaminen. Koitan perusominaisuudet pyöräyttämällä koko prosessin läpi, jotta kaikki toimii oikein. Ei voi testata kaikkia ominaisuuksia, eikä kaikilla parametreilla. Menisi monta viikkoa, jos kaikki asetusvaihtoehdot kokeiltaisiin (vastaaja B).”

Kokonaisia ohjelmia pitäisi tulevaisuudessa testata monessa eri ympäristössä, mikä ei välttämättä ole mahdollista, kun ei ole aikaa eikä resursseja. Pitäisi testata ympäristön vaikutus esimerkiksi eri mittalaitekokoonpanoilla. Tulisi verrata tuloksia ympäristön vaihtumisen jälkeen edellisiin tuloksiin, kertoi vastaaja C. Hänen mielestä, esimerkiksi aina ajettaisiin jokin perusmittaus läpi ja verrattaisiin tuloksia edellisiin. Jos tehdään jotain uutta, myös ne tulisi erityisesti testata.

Vastaajan C mielestä olisi hyvä, jos olisi yksi työntekijä pelkästään testaukseen. Hän testaisi pitkiä testejä edestakaisin ja yrittäisi etsiä heikkoja kohtia. Mittapaikkojen pitäisi olla valmiina ja tulosten vertaamisen helppoa. Vastaajan D mielestä jokainen palikka pitäisi testata erikseen, kokonaisuutta tulisi testata erikseen ja kuinka kaikki toimii yhteen. Yksikkötestauksen, ohjelmoija tekee itse, jonka jälkeen ohjelma menee testaajille testattavaksi. Nämä testaa pienemmät kokonaisuudet erikseen ja koko paketin kanssa.

4.2.4 Ohjeistus vai prosessi testaukseen?

Vastaajien A, B ja E:n mielestä ohjeistus olisi meille parempi vaihtoehto. Prosessista tulisi helposti raskas ylläpidettävä. Ennemminkin haluttiin selkeät ohjeet, kuinka testaus tulisi suorittaa. Esimerkiksi vastaaja A epäili, että jos tarkka prosessi annetaan testaukseen eivätkä kuitenkaan kaikki ohjelmat vaadi tarkkaa testausta. Miten sitten käsitellään prosessia?

Vastaajan C mielestä prosessi olisi tärkeämpi lopputuotteen testauksessa. Pienemmissä vi:ssä jokainen tekee eri tavalla testauksen, niin ei yhtenäistä prosessia näiden testaamiseen, vaan ohjeistus olisi parempi vaihtoehto. Vastaajan E mielestä pienemmissä organisaatioissa on yleensä ohjeistus, kuinka tulisi tehdä. Isommissa organisaatioissa, joissa ei voi kommunikoida henkilökohtaisella tasolla vaan kaikki menee vaatimusmäärittelyjen tasolla, pitää olla jonkinlainen prosessi testaukseen. Saskenin kokoisessa organisaatiossa ei kannata prosessia lähteä kehittämään, koska se vie äkkiä aikaa muusta työstä liikaa. Tämän kokoisessa organisaatiossa pystytään vielä ohjeistuksella ja dokumentoinnilla hallitsemaan. Prosessi kääntyy hyvin äkkiä itseään vastaan, kun on vähän resursseja. Pitäisi olla eri henkilö ylläpitämään prosessia.

Kysyttäessä toimisivatko he prosessin mukaan, jos sellainen olisi, suurin osa heistä toimisi. Vastaajat muistuttivat kuitenkin, ettei se saa olla liian raskas, jos halutaan, että suunnittelijat ottavat sen omakseen.

4.3 ohjelmien dokumentointi tällä hetkellä ja mielipide dokumentoinnista

4.3.1 NI:n kanta LabVIEW:lla tehtyjen ohjelmien dokumentointiin

Vastaaja E kertoi, että National Instruments tarjoaa myös työkaluja dokumentointiin, jotta dokumentointi tulisi tehtyä. National Instruments:lla on työkalu, joka pystyy skannaamaan kaikki yksittäiset funktiot ja katsomaan, että dokumentaatio on kunnossa ja kasaamaan dokumentit. Funktioista voidaan tulostaa dokumentteja LabVIEW:lla, mutta ei välttämättä ole järkevää tulostaa yksittäisten vi:n käyttöliittymiä, vaan kokonaisuuksia. Esimerkiksi lohkotasolla dokumentoidaan kuin yksittäisiä vi:tä. NI Requirements Gateway työkalu on myös olemassa, joka linkittää vaatimusmäärittelydokumentit, koodit ja testitulokset keskenään.

Vastaaja E mielestä kevyimmällä tasolla testauksen dokumentoinnissa pitäisi olla jokin vaatimusmäärittelydokumentti, joka määrittelee kuinka tuote tulisi testata, mitä osa-alueita tulisi testata, mitä tuloksia pitäisi tulla. Se voisi olla Word, pdf tai Excel dokumentti. Requirements Gateway linkittää tämän vaatimusdokumentin koodimassaan, joka tekee varsinaista testausta. Jokainen vaatimus pitää pystyä osoittamaan, että se on jossakin testattu. Työkalu antaa myös raportteja kuinka paljon me olemme tehneet siitä testauksesta. Lopussa kattavuus pitäisi olla 100 %. Jos jompikumpi pää muuttuu, joko vaatimusmäärittelydokumentti tai testikoodit, järjestelmä skannaa kaikki testikoodit läpi ja ilmoittaa muutokset. Yksittäiset vaatimukset voidaan myös linkittää testituloksiin.

4.3.2 Suunnittelijoiden mielipide dokumentoinnista

Haastatteluissa kysyttiin dokumentoinnista, kuinka se tulisi tehdä, vai tulisiko ohjelmia dokumentoida ja minkä verran? Haastateltavien mielestä ohjelmia tulisi

jollakin tasolla dokumentoida aina, vaikka onkin kyse graafisesta ohjelmointikielestä, joka dokumentoi periaatteessa jonkin verran itse ja josta on helppo lukea mitä ohjelmassa tehdään missäkin kohdassa.

Vastaajan mielestä varsinaista koodin toiminnallisuutta ei tarvitse dokumentoida (vastaaja B). Jollakin ylemmällä tasolla olisi kuitenkin hyvä dokumentoida, esimerkiksi vuokaavio tasolla, ei kuitenkaan moduuli/ vi tasolla. Hänen mielestä koodista voi helposti katsoa miten se toimii. Vastaaja D:n mielestä taas pitäisi dokumentoida kaikki vi:t, joka kertoisi mitä vi ottaa sisään ja mitä antaa ulos, mitä virheitä palauttaa ja minkälainen rajapinta vi:ssä on. Vastaaja E otti tärkeän asian esille, joka joskus jää hyvin vajavaiseksi, ohjelmien käyttöohjeet tulisi aina myös tehdä ja hyvin.

Testausten dokumentoinnista vastaajan A mukaan, jos tehdään yleinen ohjeistus, se on samalla testauksen määrittelyä. Olisi hyvä esimerkiksi luetteloida mitkä testit tulisi ajaa ja millä parametreilla. Tämä testi ajetaan jollakin mittalaitekokoonpanolla läpi, jolloin tiedetään toimiiko ohjelma halutulla tavalla. Samaa mieltä oli myös vastaaja B, jonka mukaan voisi myös kaikille ohjelmille tehdä omat tarkistuslistat, jotka toimisivat samalla myös dokumenttina testauksesta. Vastaaja C otti myös esille asian, jos eri henkilöt testaavat ohjelmia, niin tiedettäisiin mitä toinen on testannut ja mitä siinä pitäisi edes testata. Virhetilanteessa kirjoitettaisiin listaan mitä tapahtui. Virhetilanteet tulisi korjata ja aloittaa tarkastuslista alusta. Mittauksien testaukseen tarkistuslistana kävisivät testijonot, joissa olisi kaikki testit teknologioittain. Testijonot ajetaan läpi ja katsotaan, ettei virheitä tule mihinkään ts. että mittalaitteet ohjautuu oikein ja pitäisi lisäksi olla työkalu, millä verrataan tuloksia, josta tulisi raportti testaamisesta. Raporttiin pitää määritellä rajat, kuinka paljon tuloksissa saa olla eroavaisuutta ja mikä on normaalia vaihtelua tuloksissa, kertoi vastaaja B. Samoilla linjoilla, olivat myös muut haastatteluun osallistuneet henkilöt.

4.4 Yhteenveto

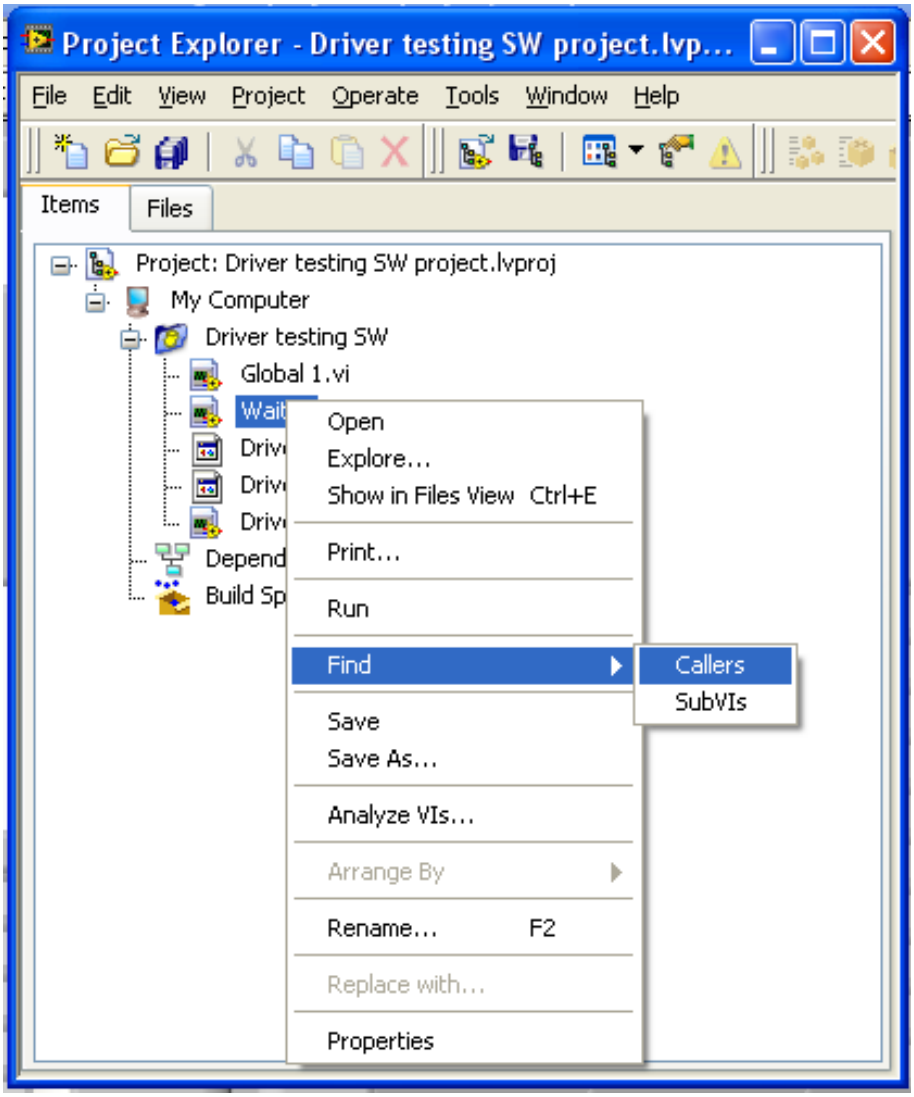
Tässä osiossa käsitellään tutkimustuloksia vertaamalla niitä teoriaan ts. tekemällä synteesin. Synteesi on teoriasta saatujen tietojen ja haastattelutulosten yhdistelemistä ja kokoamista yhdeksi tiedoksi (Haaparanta & Niiniluoto, 1986). Näiden pohdintojen pohjalta tullaan tekemään ohjeistus Sasken Finland Oy:lle LabVIEW:lla toteutettujen ohjelmien testaukseen ja dokumentointiin, joka on erillinen ohjeistus.

Yksi tavoite tutkimukselle oli selvittää, kumpi olisi parempi vaihtoehto Sasken Finland Oy:ssä prosessi vai ohjeistus dokumentointiin ja testaukseen. Sekä kirjallisuus ja haastattelut antoivat vahvistusta käsitykselle, että ohjeistus on järkevin vaihtoehto tämän kokoiselle organisaatiolle (Märijärvi & Haikala 2001). Prosessi vaatisi toimiakseen paljon enempi resursseja ja monesti siitä tulee liian raskas tämän tyyliseen ohjelmointiin. Ohjeistus ei ole niin sitovaa, jos tulee kiireellisiä korjauksia, tai jokin ominaisuus tarvitaan hyvin nopeasti loppukäyttäjille. Ohjeistuksessa on toki vaaransa, jotka pitää tiedostaa.

4.4.1 Yksikkötestauksen suorittaminen

Yksikkötestaus eli yksittäisten vi: n testaamisen vastuu kuuluu ohjelmoijalle itselleen niin kuin kirjallisuus ja ohjelmoijat sanoivat. Tähän olisi hyvä käyttää automatisoituja testausohjelmia esimerkiksi National Instrumentin yksikkötestaustyökalua, jolloin saataisiin testauksen kattavuus paremmaksi. Yksikkötestauksessa olisi kuitenkin tärkeää testata, vaikka ohjelmaa tehtäessä, ajamalla lähdekoodia satunnaisilla syötteillä halutulla alueelle ja tarvittaessa raja-arvoilla ja niiden yli. Virheiden hallinta tulisi tarkistaa jokaisesta yksittäisestä aliohjelmasta ja mielellään lisätä kuvaavia virheilmoituksia virhetilanteista. LabVIEW:ssa löytyvää ”Profile Performance and Memory” työkalua olisi suotavaa käyttää vähänkin monimutkaisempien aliohjelmien testaamisessa, jolla voi helposti tarkistaa onko jäänyt muistivuotoja tai aukinaisia referenssilinjoja.

Tällä hetkellä Saskenilla ei ole mahdollista pienien resurssien vuoksi ottaa käyttöön automatisoitua yksikkötestaustyökalua. Tästä syystä jää suurempi vastuu ohjelmoijille ja heidän ammattitaidolle. Jos tehdään muutoksia johonkin aliohjelmaan, tulisi varmistua, ettei ko. aliohjelmaa käytetä muualla. Jos aliohjelma on käytössä useammassa paikassa, varmistetaan, että se tulevaisuudessakin toimii kaikkialla halutulla tavalla. Kuviosta 16 näkyy, kuinka LabVIEW project ikkunassa voidaan tarkistaa missä kaikissa paikoissa ko. aliohjelma on käytössä ts. mitkä ohjelmat kutsuvat ko. vi:tä.



Kuvio 16. Kutsuvien ohjelmien löytäminen

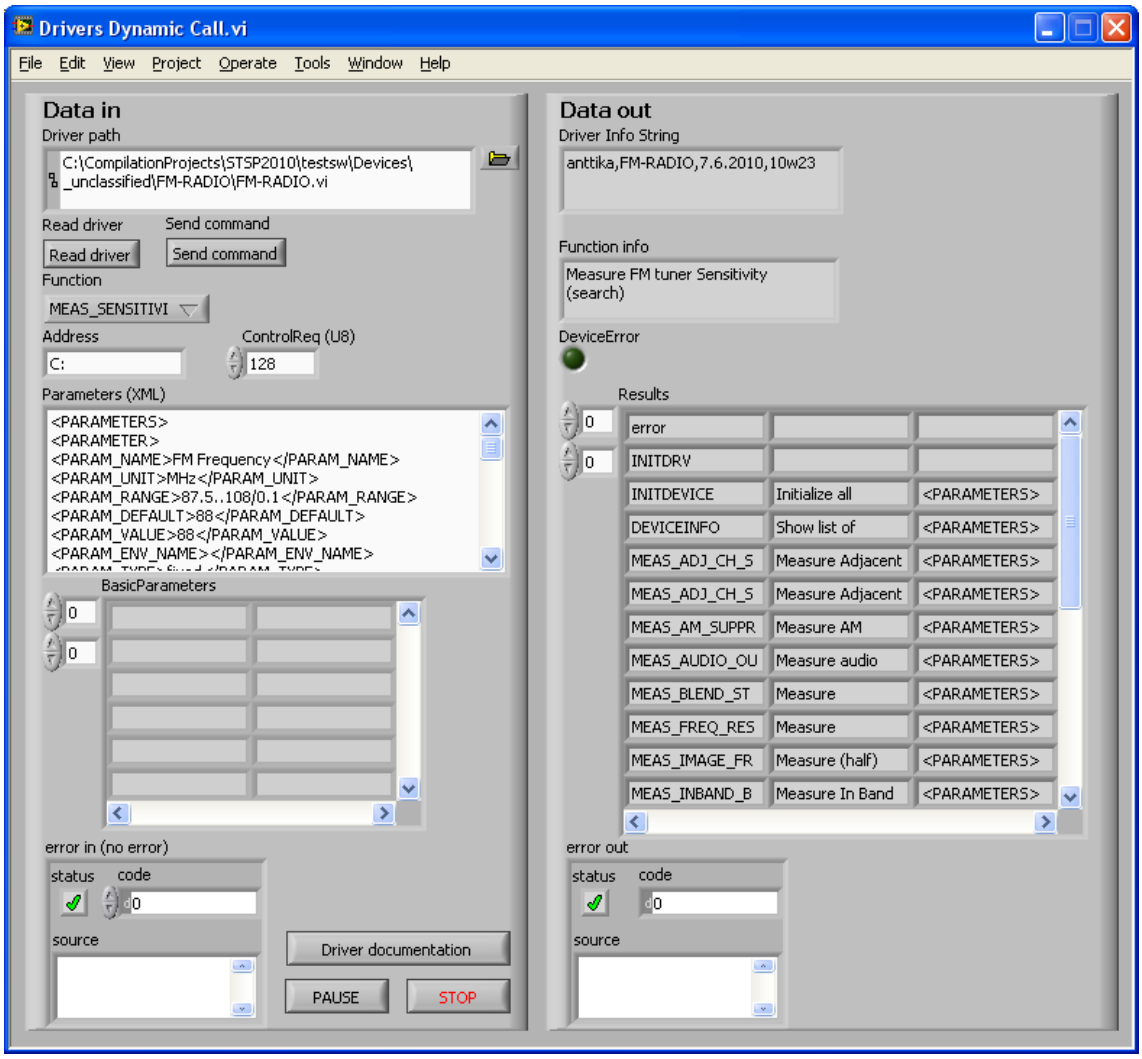
LabVIEW Project ikkunasta voidaan tehdä hyvin paljon muutakin esimerkiksi analysoida vi, katsoa missä kyseinen tiedosto sijaitsee levyllä jne. LabVIEW Project ikkuna on tullut käyttöön LabVIEW 8.6 versiosta lähtien ja on ohjelmoinnin kannalta välttämätöntä käyttää nykyään sitä.

4.4.2 Moduulitestauksen suorittaminen

Moduulitestauksessa esimerkkinä Saskenin tapauksessa on yksittäisen mittalaiteajurin testaaminen, vaikka moduuli voi olla muukin ohjelmanosa esimerkiksi käyttöliittymä tai loki-tiedoston tallennus. Moduulitestaus kuuluu ohjelmoijalle itselleen. Ohjelmoijan tulee testata moduuli yksittäin ennen sen liittämistä integrointi vaiheessa itse ohjelmaan.

Mittalaiteajurien testaamisen aluksi tulisi selvittää miten mittaus tulisi suorittaa vaatimusmääritysten mukaan. Tässä voidaan käyttää kappaleessa 2.3 kohdassa White Box testauksessa kerrottua menetelmää, jossa tarkistetaan kohta kohdalta onko mittauksen järjestys oikein. Mittalaiteajurien testaaminen on ollut hyvin hankalaa, koska ajureille meneviä syötteitä on useita ja parametrointi tapahtuu XML-tiedosto muodossa. Tämä ongelma nousi haastatteluissakin esille hyvin vahvana. Tähän ongelmaan tehtiin tämän kehittämistyön aikana testausalustan, jolla mittalaiteajurien testaus helpottuu huomattavasti.

Tämä mittalaiteajurien testauksen apuohjelma lukee ajurissa olevat funktiot läpi, lukee niiden parametrien oletusarvot ja antaa ajureille rajapinnan mukaiset tiedot. Tällä ohjelmalla voidaan ajurit alustaa, valita testattava funktio ja parametroida halutuilla arvoilla. Tällä samaisella ohjelmalla voi luoda myös perusdokumentaation mittalaiteajurista. Ohjelman käyttöliittymä on esitetty kuviossa 17. Ohjelma on tarkoitettu ohjelmoijille apuvälineeksi, josta syystä ulkonäköön ei ole kiinnitetty huomiota.



Kuvio 17. Mittalaiteajurien testaamiseen laadittu apuohjelma

Ajurien testaamisessa olisi hyvä ensin manuaalisesti kokeilla mittalaitetta ja tämän jälkeen ohjelmallisesti, jolloin voidaan katsoa meneekö mittalaite samaan tilaan ohjelmallisesti kuin manuaalisesti. Ajurien testaaminen tulisi tehdä oikealla mittalaitteella, jolloin mittalaite ja NI Spy ovat päällä yhtä aikaa. Tarkistetaan ettei mittalaite herjaa muodosta tai, ettei lähetetä vääriä käskyjä. Tässä vaiheessa on kuitenkin vaikeaa alkaa testaamaan mittausten oikeellisuutta, koska useasti vaaditaan useampien laitteiden samanaikaista ohjaamista. Esimerkiksi vaaditaan signaaligeneraattorin, radiokommunikointitesterin ja signaalianaly-

saattorin samanaikaista käyttöä, jotta saadaan tehtyä koko mittaus. Edellä mainitusta syystä mittausten oikeellisuuksien ja tulosten vertailu tulisikin tapahtua integrointi-/ järjestelmätestauksen yhteydessä.

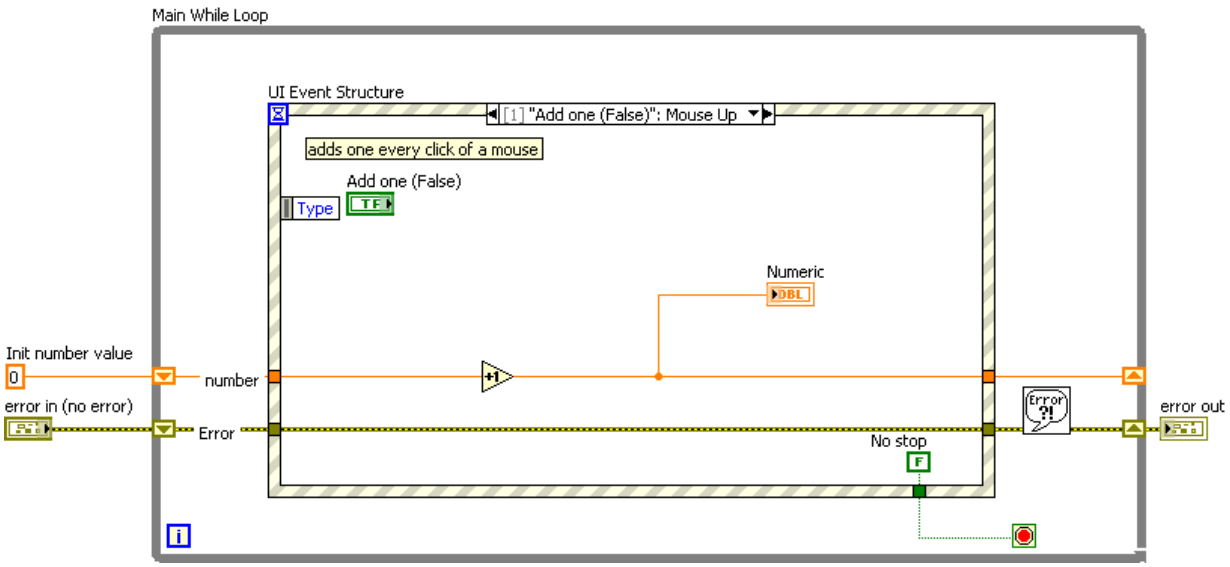
4.4.3 Integrointi- ja järjestelmätestauksen suorittaminen

Integrointi- ja varsinkin järjestelmätason testauksen tulisi tehdä jonkun muun kuin ohjelmoijan itse. Tämä tuli esiin niin haastatteluissa kuin kirjallisuudesta. Ohjelmoija ”sokeutuu” virheille ja voi tiedostamattaan välttää tilanteita, jotka eivät toimi täydellisesti. Ongelmana on ainainen kiire, jolloin varsinkin järjestelmä tason testaus jää vähälle. Jonkin tasoisen integrointitestauksen ohjelmoija joutuu tekemään väkisinkin, vaikka tämän olisi hyvä tehdä lisäksi jonkun toisen. Tällä hetkellä järjestelmätason testaus jää hyvin pitkälle talon sisäisen asiakkaan testauksen vastuulle. Sitä voisi sanoa Alfa testaukseksi. Järjestelmätestaus on jäänyt hyvin monesti lähes kokonaan tekemättä. Ei ole aina tarpeen testata kaikkia ominaisuuksia, eikä kaikilla parametreilla, koska tähän menisi aikaa useita viikkoja, jos kaikki asetusvaihtoehdot kokeilisi. Suunnittelijan tai testaajan olisikin hyvä määritellä jo ennen testauksen aloittamista mitkä osa-alueet tulisi testata ja millä laajuudella.

Kun puhutaan mittausohjelmista, tulisi mittaustuloksia pystyä vertaamaan toisessa ympäristössä ajettuihin tuloksiin tai jotenkin muuten varmistua tuloksien oikeellisuuksista. Järjestelmätestauksen yhteydessä olisi hyvä ajaa jokin perusmittaus läpi ja verrattaisiin tuloksia edellisiin tuloksiin. Tällä varmistettaisiin, ettei toimiviin mittauksiin ole tullut vikoja. Kun on vähäiset resurssit ja kova kiire uusien päivityksien tekemisessä, pitäisi testauksessa käytettävien mittapaikkojen olla valmiina ja tulosten vertaaminen helppoa jolloin aikaa säästyisi itse testaamiseen.

4.4.4 Ohjelmien dokumentointi

Teoriaosuudessa kerrottiin kuinka ohjelmia tulisi yleensä ottaen dokumentoida. Dokumentointi on yhtä tärkeää myös LabVIEW:lla ohjelmoitaessa, vaikka ohjelma dokumentoi myös itse itseään. Moduulitasolla olisi kuitenkin hyvä dokumentoida, esimerkiksi vuokaavioden tasolla. Graafisessa ohjelmoinnissa ohjelmakoodissa ei välttämättä jokaista kohtaa tarvitse kommentoida, mutta ohjelmien osa-alueet vi:ssä tulisi kuitenkin dokumentoida ja jokaiseen aliohjelmaan olisi hyvä laittaa jonkinlainen kuvaus mitä ko. aliohjelma tekee. Kuviossa 18 on näytetty yksinkertainen ohjelmaesimerkki kuinka koodia tulisi dokumentoida.



Kuvio 18. LabVIEW koodin dokumentointi

Kappaleessa 2.1.4 taulukkossa 1 on kerrottu, mitä tuli ottaa huomioon LabVIEW:lla ohjelmoitaessa ja mitä asioita tulisi vähintään dokumentoida. Näiden sääntöjen lisäksi ohjelmassa olevat ”luuppi-”, ”case”- jne. rakenteet tulisi kommentoida, mitä missäkin tapahtuu. Lisäksi pitkät johdotukset ja siirtorekisterit tulisi kommentoida. Kaikkiin kontrolleihin tulisi laittaa otsikkoon näkyviin oletusarvot ja kontrollit ja indikaattoreihin olisi hyvä kirjoittaa kuvaukset.

National Instruments tarjoaa myös työkaluja dokumentointiin, että dokumentointi tulisi tehtyä. Ongelmanahan on, ettei dokumentteja tule tehtyä kuin pakon edessä, vaikka teoriassa pitäisi ja kaikkien ohjelmoijien mielestä myös. National Instrumentsilla on työkalu, joka pystyy skannaamaan kaikki yksittäiset funktiot ja katsomaan, että dokumentaatio on kunnossa ja kasaamaan dokumentit. Funktioista voidaan tulostaa dokumentteja LabVIEW:lla, mutta välttämättä ei ole järkevää tulostaa yksittäisten vi:n käyttöliittymiä, vaan kokonaisuuksia. Myös tämän kehittämistehtävän yhteydessä tehty mittalaiteajurien testaamiseen tarkoitettu apuohjelma käyttää LabVIEW:ssa valmiina olevia valmiita funktioita dokumenttien tekemiseen.

Ohjelmakoodin lisäksi tulisi dokumentoida ohjelmien käyttöohjeet hyvin. Käyttöohjeet tulisi tarkastuttaa järjestelmätestauksen yhteydessä testaajalla, joka pitäisi olla henkilö, joka ei ole ohjelman tekemisen kanssa missään tekemisessä. Ohjelmien käyttö- ja asennusohjeiden tulisi olla niin hyviä, että jokainen ihminen pystyisi käyttämään ohjelmaa niiden avulla.

4.4.5 Testauksen dokumentointi

Kevyimmällä tasolla testauksen dokumentoinnissa pitäisi olla jokin vaatimusmäärittelydokumentti ja raportti testauksesta. Vaatimusmäärittelydokumentti määrittelee kuinka tuote tulisi testata, mitä osa-alueita tulisi testata ja mitä tuloksia pitäisi tulla. Tämä voi olla yksinkertaisimmillaan Excelissä lista kohdista, jotka tulisi testata. Lisäksi testauksen määrittelynä voisi olla yleinen ohjeistus. Ohjeistuksessa olisi hyvä olla luettelo mitkä asiat tulisi tarkistaa vähintään testauksen yhteydessä. Testauksen raporttipohjana voisi käyttää vaatimusmäärittelyssä olevaa listaa testeistä, mitä testataan ja tähän samaiseen raporttiin merkitään miten testi on mennyt. Jos eri henkilöt testaavat ohjelmia olisi tärkeää tietää mitä toinen on testannut ja mitä pitäisi vielä testata. Virhetilanteissa kirjoitettaisiin listaan, mitä tapahtui ja mikä aiheutti virhetilanteen. Virhetilanteet tulisi korjata ja aloittaa tarkastuslista alusta.

Mittausten testaukseen tarkistuslistana kävisivät myös testijonot, joissa olisi kaikki mittaukset teknologioittain. Testijonot ajetaan läpi ja katsotaan, ettei virheitä tule mihinkään ts. että mittalaitteet ohjautuu oikein ja tulokset ovat oikeanlaisia. Kehittämistyön alussa Saskenilla ei ollut mitään ohjelmaa tulosten vertailuun. Tätä varten on kuitenkin tehty ohjelma, millä verrataan tuloksia. Ohjelmasta saadaan raportti mitkä tulokset menevät määritettyjen raja-arvojen sisään ja mitä tuloksia tulisi vielä tarkastella lähemmin.

Tämän työn liitteenä ovat ohjeistukset testauksen eri tasoille. Nämä ohjeistukset on tehty hyvin yleispäteviksi esimerkeiksi ja näitä samoja ohjeistuksia voidaan käyttää myös testauksen raporttipohjina. Olisi tärkeää kuitenkin lisätä jokaisen ohjelman omat tärkeimmät kohdat jotka tulisi testata jo ohjelman vaatimusmäärittelyä tehtäessä niin kuin ohjelmoinnin V-mallissa asia esiteltiin.

5 JOHTOPÄÄTÖKSET

Tämän kehitystyön tuloksena saatiin kaksi erillistä apuohjelmaa ohjelmien testaamiseen, ohjeistus ohjelmien dokumentoinnista ennen kaikkea koodin osalta ja yleisohjeistus LabVIEW:lla tehtyjen ohjelmien testaukseen Sasken Finland Oy:ssä. Toisen näistä ohjelmista tein itse ja toisen teki toinen suunnittelija. Tutkimusongelma oli kuinka LabVIEW:lla tehdyt ohjelmat tulisi dokumentoida ja testata. Tähän ongelmaan tämä kehittämistehtävä antaa vastauksen. Tässä työssä ei ole liitteitä mukana, jossa on itse ohjeistukset ja esimerkit.

Kehittämistehtävä jäi haastattelujen osalta aika suppeaksi alkuperäisestä tavoitteesta, koska LabVIEW suunnittelijoiden määrä Sasken Finland Oy:ssä supistui Itseni lisäksi kahteen henkilöön, aikaisemmasta seitsemästä suunnittelijasta kehittämistyön aikana. Vähennykset johtuivat lomautuksista, hankalassa taloudellisessa tilanteessa, joka oli seurausta yleisestä talouden taantumasta ja tärkeimmän asiakkaan strategisista päätöksistä. Kehittämistyössä oli tarkoitus tehdä uusintahaastattelu kahdenkuukauden kuluttua käyttöönotosta. Tätä ei kuitenkaan tehty kun, ei voitu ottaa uutta käytäntöä täysimääräisenä käyttöön ja osa ensimmäisessä ja ainoasta haastattelukierrokseen osallistujista olivat lomautettuina. Kehittämistehtävän lopussa myös itseltäni loppuivat työt Sasken Finland Oy:ssä. Tämä myös osaltaan on ollut vaikuttamassa tämän työn vajavaisuuteen.

Työn luotettavuutta on vaikea arvioida. Työstä olisi saatu huomattavasti luotettavampi, jos olisin pystynyt haastattelemaan kaikkia yrityksen LabVIEW suunnittelijoita. Kaikkien mielipidettä ei pystytty kuulemaan tätä tehtäessä. Lisäksi olisi varmasti saatu parempi lopputulos, jos olisi pystytty etenemään alkuperäisen suunnitelman mukaisesti, jolloin olisi pidetty toinen haastattelu koulutusten ja käyttökokemuksien jälkeen. Kuitenkin viiden ihmisen haastattelu ja teoria aiheesta, antoivat yleiskäsityksen, minkälainen ohjeistuksen tulisi olla, kuinka tulisi dokumentoida ohjelmaa ja miten testaus tulisi suorittaa milläkin tasolla.

Tästä olisi hyvä tehdä jatkotutkimus, jossa kysyttäisiin suunnittelijoilta mielipidettä ohjeistuksista, paranneltaisiin niitä, tehtäisiin valmiita dokumentti pohjia, mietittäisiin testauksien automatisoimista ja mahdollisien LabVIEW:n lisäosien käyttöönottoa testaukseen mukaan. Tämä jää ainakin omalta osaltani tähän, työsuhteen päättymisen vuoksi.

Työ ei aiheuttanut muutoksia organisaatiossa ainakaan vielä. Monien syiden johdosta LabVIEW tiimi pienennettiin kahteen henkilöön. Tästä syystä Saskenilla ei mielestäni pystytä tekemään perusteellista testausta, ei ole aikaa uuden kehittämiseen, vaan henkilöiden aika menee ylläpitotehtävissä ja pienien lisäosien tekemisessä. Nämä ovat strategisia päätöksiä joita monet yritykset joutuvat pohtimaan tänä aikana.

LÄHTEET

Agile. (2011). Manifesto for Agile Software Development. Www-dokumentti. Saatavissa: http://agilemanifesto.org. Luettu: 30. 3 2011.

Alasuutari, P. (1999). Laadullinen tutkimus. Tampere: Vastapaino.

Haaparanta, L. & Niiniluoto, I. (1986). Johdatus tieteelliseen ajatteluun. Helsinki: Yliopistopaino.

Kaner,C., Bach, J. & Pettichord, B. (2002). Lessons learned in software testing. New York: John Wiley & Sons, Inc.

Kerry, E. (2010). Software Engineering for LabVIEW Applications. USA.

Kettunen, V. (2009). Ohjelmistotestaus ja ketterät menetelmät. Lappeenranta: Lappeenrannan teknillinen yliopisto.

Metsämuuronen, J. (2008). Laadullisen tutkimuksen perusteet . Jyväskylä: Gummerus kirjapaino Oy.

Märijärvi, J. & Haikala, I. (2001). Ohjelmisto tuotanto. Pieksämäki, Suomi: Talentum Media Oy.

Narikka, J. (1996). LabVIEW ohjelmointi/ Ohjelmistotekniikan seminaarityö. Www-dokumentti. Saatavissa: http://www.mit.jyu.fi/opiskelu/seminaarit/bak/labview. Luettu: 31. 10 2010

National Instruments. (2003). LabVIEW Development Guidelines. Austin, Texas.

National Instruments. (2006a). NI Developer Zone. Rules to Wire By -- Part I. Www-dokumentti. Saatavissa: http://zone.ni.com/devzone/cda/tut/p/id/5560. Luettu: 2. 2 2011

National Instruments. (2006b). NI Developer Zone. Rules to Wire By -- Part II. Www-dokumentti. Saatavissa: http://zone.ni.com/devzone/cda/tut/p/id/4822. Luettu: 2. 2 2011

National Instruments. (2007a). LabVIEW Advanced I: Architectures Course Manual. Austin.

National Instruments. (2007b). LabVIEW Fundamentals. LV 8.6.

National Instruments. (2010). Product Information: What is NI LabVIEW? Www-dokumentti. Saatavissa: http://www.ni.com/labview/whatis. Luettu: 2. 2 2011

National Instruments. (2011a). NI LabVIEW Desktop Execution Trace Toolkit. Www-dokumentti. Saatavissa: http://sine.ni.com/nips/cds/view/p/lang/en/nid/209044. Luettu: 11. 3 2011

National Instruments. (2011b). NI LabVIEW Unit Test Framework Toolkit. Www-dokumentti. Saatavissa: National Instruments Corporation: http://sine.ni.com/nips/cds/view/p/lang/en/nid/209043. Luettu: 1. 2 2011

National Instruments. (2011c). NI LabVIEW Unit Test Framework Toolkit. Www-dokumentti. Saatavissa: http://sine.ni.com/nips/cds/view/p/lang/en/nid/209043. Luettu: 1. 2 2011

National Instruments. (2011d). NI LabVIEW VI Analyzer Toolkit. Www-dokumentti. Saatavissa: http://sine.ni.com/nips/cds/view/p/lang/en/nid/209042. Luettu: 11. 3 2011

Poston, R. M. (1996). Automating Specification- Based Software Testing. Washington: Matt Loeb.

Rauhala, E. (2010). Ohjelmistotestauksen suunnittelu - Case: A-lehdet Oy:n laskujen tulostusohjelma. Www-dokumentti. Saatavissa: https://publications.theseus.fi/bitstream/handle/10024/23687/Rauhala_Eija.pdf?sequence=1. Luettu: 18. 3 2011

Ruusuvuori, J. & Tiittula, L. (2005). Haastattelu. Tampere: Vastapaino.

Sasken Finland Oy (2011). Www-dokumentti. Saatavissa: http://www.sasken.com. Luettu: 2.2.2011

Saksa, O-P. (2008). Scrum, in Theory and in Practice. Tampere.

Stenberg, A. (2004). Ohjelmiston testaus. Www-dokumentti. Saatavissa: http://www.pori.tut.fi/~stenberg. Luettu: 29. 3 2011

Tauriainen, S. (2005). Ohjelmistotestauksen kehittäminen. Kajaani: Kajaanin AMK.

Tilastokeskus. (2011). Virtual statistics. Www-dokumentti. Saatavissa: http://www.stat.fi/virsta/tkeruu/04/03. Luettu: 25. 4 2011

Wikipedia. (2011). Ketterä ohjelmistokehitys. Www-dokumentti. Saatavissa: http://fi.wikipedia.org/wiki/Ketter%C3%A4_ohjelmistokehitys. Luettu: 29. 3 2011

LIITTEET

HAASTATTELU POHJA

1. Kuinka kauan olet käyttänyt LabVIEW:a?
2. Kuinka kauan olet ohjelmoinut yhteensä muut kielet mukaan lukien?
3. Millä muilla kielillä olet ohjelmoinut?
4. Miten testaat tällä hetkellä yksittäiset vi:t?
5. Miten testaat driverit?
6. Miten testaat kokonaisuuden?
7. Miten tulisi testata yksittäiset vi:t?
8. Miten tulisi testata driverit/mittalaite ajurit?
9. Miten tulisi testata kokonaisuus?
10. Dokumentointi? Testaussuunnitelma, Testausraportti
11. Kuinka paljon ohjelmistokehityksestä pitäisi mielestäsi olla testausta?
12. Tarvitaanko prosessia testaukseen?
13. Toimisitko prosessin mukaan, jos sellainen olisi?
14. Mitä LabVIEW:n työkaluja käytät testaukseen ja mitä lisäosia haluaisit meillä olevan käytössä?
15. Millä testaus saataisiin nopeaksi?
16. Millä tavalla mielestäsi saataisiin kriittiset bugit esille testauksessa?
17. Kuinka laaja testauksen tulisi olla ennen kuin ohjelmisto annetaan asiakkaalle?
18. Mitä muuta asioita tulee mieleen LabVIEW ohjelmien testauksesta?