OULUN AMMATTIKORKEAKOULU

Eero Räsänen

KÄYTTÖLIITTYMÄOHJELMISTO TUOTANNON TESTAUSLAITTEEN OHJAUKSEEN

KÄYTTÖLIITTYMÄOHJELMISTO TUOTANNON TESTAUSLAITTEEN OHJAUKSEEN

Eero Räsänen
Opinnäytetyö
Kevät 2024
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, langattomien laitteiden suuntautumisvaihtoehto

---

Tekijä(t): Eero Räsänen
Opinnäytetyön nimi: Käyttöliittymäohjelmisto tuotannon testauslaitteen ohjaukseen
Työn ohjaaja(t): Ville Kiiskinen ja Olli Himanka
Työn valmistumislukukausi ja -vuosi: Kevät 2024 Sivumäärä: 26

Opinnäytetyön aiheena oli suunnitella ja toteuttaa uusi käyttöliittymä olemassa olevaan tuotannon piirilevyjen testauslaitteen ohjaukseen. Testauslaitteella testataan ja ohjelmoidaan valmiiksi kokoonpantuja piirilevyjä. Työssä tavoitteena oli saada ohjelmisto, joka kykenee testaamaan piirilevyt kattavasti ja tehokkaasti sekä löytämään mahdolliset vikatilanteet. Lisäksi työn tuli noudattaa lääketieteellisen ohjelmiston standardin vaatimuksia.

Opinnäytetyö tehtiin suomalaiselle terveysteknologiayhtiö Optomed Oyj:lle. Optomed suunnittelee ja valmistaa kädessä pidettäviä silmänpohjakameroita sekä ohjelmistoratkaisuja. Opinnäytetyössä esitellään aluksi tuotannon testauslaite, jota ohjelmiston tulisi ohjata. Lisäksi käydään läpi ohjelmiston suunnitteluprosessi, jolla standardin mukaiset vaatimukset täyttyvät.

Ohjelmisto tuotannon testauslaitteen ohjaukseen saatiin valmiiksi ja otettua käyttöön tuotannossa. Työn aikana tekijälle syntyi kattava ymmärrys lääketieteellisen ohjelmiston standardin vaatimuksista ja niiden soveltamisesta ohjelmistoprossiin.

Asiasanat: Lääketieteellinen ohjelmisto, tuotannon testaus, Qt, QML

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information and Communication Technology, Option of Wireless Devices

---

Author(s): Eero Räsänen
Title of thesis: Graphical user interface software for controlling production tester
Supervisor(s): Ville Kiiskinen and Olli Himanka
Term and year when the thesis was submitted: Spring 2024
Number of pages: 26

The topic of the thesis was to design and implement a new graphical user interface for controlling an existing production circuit board tester. Assembled circuit boards are tested and programmed with the tester.

The goal was to implement a program for the tester that is capable of comprehensively and efficiently testing circuit boards and finding possible fault cases. In addition, the work had to comply with the requirements of the medical software standard.

This thesis was done for a Finnish healthcare technology company Optomed Oyj. Optomed designs and manufactures handheld fundus cameras and software solutions.

The thesis initially presents the production tester that should be controlled by the program. In addition, the software design process is reviewed to ensure it meets the requirements of the relevant standard.

During the work, the author gained a comprehensive understanding of the requirements of the medical software standard and its application to the software process. The user interface software for controlling the production tester was completed.

Keywords: Medical software, production testing, Qt, QML

SISÄLLYS

1 JOHDANTO

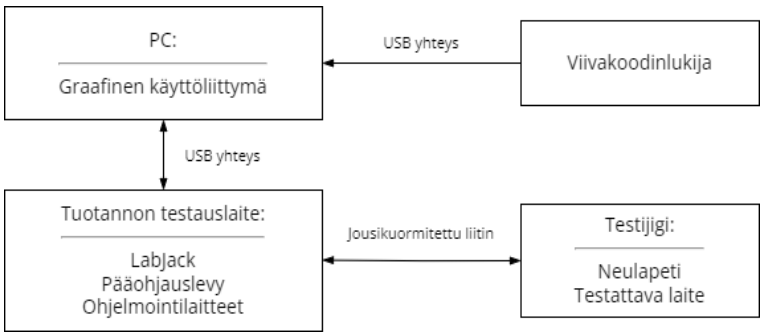
Opinnäytetyön aiheena on suunnitella ja toteuttaa uusi käyttöliittymä olemassa olevaan tuotannon piirilevyjen testauslaitteeseen. Testauslaitteella testataan ja ohjelmoidaan valmiiksi kokoonpantuja piirilevyjä. Työssä tavoitteena on saada testauslaitteen ohjaukseen käyttöliittymäohjelmisto, jonka avulla voidaan testata piirilevyt kattavasti ja tehokkaasti sekä voidaan löytää mahdolliset vikatilanteet. Ohjelmisto toteutetaan noudattaen lääketieteellisen ohjelmiston standardin vaatimuksia ja työssä käydään läpi sen suunnitteluprosessia.

Opinnäytetyö tehdään suomalaiselle terveysteknologiayhtiö Optomed Oyj:lle. Optomed on yksi johtavista kädessä pidettävien silmänpohjakameroiden suunnittelijoista ja valmistajista (1.).

2 TUOTANNON TESTAUSLAITE

Testaus on tärkeä osa koko tuotekehityksen elinkaarta ja on yksi laadunvarmistuksen prosessi. Tuotannon testauksen tehtävänä on suorittaa testattaville laitteille laadukas ja tehokas testaus, jotta voidaan minimoida vikojen esiintymisen tuotannon myöhemmässä vaiheessa. Testit tulisi olla toistavissa, jolloin samat mittaustulokset olisivat joka kerta samat, pienellä marginaalin erolla. Mahdolliset viat tulisi löytää ja korjata varhaisessa vaiheessa, jolloin tuotannon- ja laitteen valmistuskustannukset pysyisivät pienempänä. (2.)

Tuotannon testaus voidaan jakaa kolmeen erilliseen fyysiseen laitteeseen. PC tietokoneessa sijaitsee graafinen käyttöliittymä, jonka tarkoituksena on ohjata tuotannon testauslaitetta USB:n kautta. PC:hen voidaan myös yhdistää viivakoodinlukija helpottamaan testattavien laitteiden sarjanumeroiden syöttämistä. Tuotannon testilaitteen arkkitehtuuri on esitetty kuvassa 1.



KUVA 1. Tuotannon testauslaitteen arkkitehtuuri.

Tuotannon testauslaite sisältää pääohjauslevyn, LabJack-tiedonkeruulaitteen sekä ohjelmointilaitteita testattaville laitteille. Testijigit ovat yhteydessä tuotannon testauslaitteeseen jousikuormitetuilla liittimillä. Jigit sisältävät neulapedin, joka on yhteydessä testattavan laitteen testipisteisiin.

LabJack on tiedonkeruulaite, joka sisältää useita digitaalisia ja analogisia I/O-linjoja. Laitetta käytetään ohjaamaan pääohjauslevyä sekä PC:n ohjelmiston ja testauslaitteen kommunikointia USB, SPI ja I2C -väylien kautta. Lisäksi LabJack testaa sekä monitoroi pääohjauslevyn jännitteitä ja virrankulutusta mahdollisten oikosulkujen varalta.

Pääohjauslevy on tuotannon testauslaitteen ydin. Käyttöjännite saadaan muuntajan kautta pistorasiasta, jonka pääohjauslevy muuttaa itsensä ja testattavien laitteiden tarpeiden mukaiseksi. Pääohjauslaite sisältää yksittäisiä analogi-digitaali- ja digitaalianalogimuuntimia, I/O-laajentimia sekä multipleksereitä. Lisäksi laite sisältää akkusimulaattorin, lediohjauksen, virrantestaus- sekä äänentasomittauskytkennän. Näitä kaikkia käytetään eri testausvaiheissa.

Tuotannon testauslaite sisältää useita eri mikropiirivalmistajan ohjelmointilaitteita, joilla nimensä mukaisesti ohjelmoidaan eri laitteiden mikrokontrollereille laiteohjelmisto. Vanhemmat piirilevyt ohjelmoidaan pääsääntöisesti vielä näillä laitteilla, mutta uusien piirilevyjen ohjelmointi tapahtuu suoraan tässä opinnäytetyössä tehtävän ohjelman kautta, ilman erillisiä ohjelmointilaitteita.

Jokaista testattavaa piirilevytyyppiä kohden on yksilöllinen testijigi, joka yhdistää neulapedin kautta testattavan piirilevyn pääohjauslevyyn. Jigit sisältävät, kuten pääohjauslaite, yksittäisiä analogi-digitaali- ja digitaalianalogimuuntimia, I/O-laajentimia sekä multipleksereitä, joilla saadaan testattua piirilevy kattavasti.

3 OHJELMISTON SUUNNITTELUPROSESSI

Lääketieteellistä ohjelmistoa toteuttaessa ohjelmiston suunnitteluprosessin tulee noudattaa ja täyttää IEC 62304 Medical device software – Software life cycle processes standardin vaatimukset (3.).

Optomedillä uuden tuotteen suunnitteluprosessi ohjelmiston osalta jaetaan neljään eri vaiheeseen, joista jokainen jaetaan pienempiin vaiheisiin. Nämä neljä vaihetta ovat konsepti- ja toteutettavuustutkimus, vaatimus- ja järjestelmäsuunnittelu, toteutus ja yksikkötestaus sekä testaus ja validointi. Näiden lisäksi toimintaa ohjaavia dokumentteja ja prosesseja on useita, joista tärkeimpinä koodaustyyliopas ja koodin katselmointi.

Tässä opinnäytetyössä tehtävä ohjelmisto ei ole lääketieteellinen ohjelmisto, jolloin ei edellytetä noudattamaan kaikkia edellä mainitun prosessin vaiheita. Noudattamalla samoja sääntöjä ja prosesseja kaikkien ohjelmistojen laatu ja jäljitettävyys pysyy hyvällä ja halutulla tasolla.

3.1 Konsepti- ja toteutettavuustutkimus

Konsepti- ja toteutettavuustutkimusvaiheessa määritellään järjestelmätason vaatimusluettelo sekä tehdään toteutettavuustutkimus. Järjestelmätason vaatimusluettelo on yksinkertainen listaus tuotteen määrittelystä, esimerkiksi materiaaliluettelo.

Toteutettavuustutkimus aloitetaan, kun on valittu konsepti, jota lähdetään toteuttamaan. Tutkimus voi sisältää muun muassa teknologiatutkimuksen, ohjelmistoteknologian analyysin ja immateriaalioikeuksien tutkimuksen. Ohjelmistokehitysnäkökulmasta teknologiatutkimus on erittäin tärkeä: Se koostuu ohjelmisto-, laitteisto- ja mekaniikkavalinnoista ja siitä, mikä on paras ratkaisu alkuvaatimusten ja konseptin täyttämiseen. Ohjelmisto ei ole koskaan itsenäinen kokonaisuus, vaan sitä suoritetaan aina jossain laiteympäristössä, johon myös ympäröivä mekaniikka vaikuttaa.

Ohjelmistoteknologian analyysin tarkoituksena on näyttää kriittisimmät ongelmat ja niiden lievennykset, jotka liittyvät käytettyihin ohjelmistoteknologioihin. Immateriaalioikeuksien tutkimuksessa

esitetään yhteenveto käytetyistä ohjelmistolisenssityypeistä ja luetellaan käytetyt kirjastot ja niitä vastaavat lisenssit.

3.2 Vaatimus- ja järjestelmäsuunnittelu

Ohjelmiston suunnittelu alkaa vaatimusten määrittelyllä. Vaatimusmäärittelyn tavoitteena on listata ohjelmistovaatimukset järjestelmätasolla osaksi vaatimusmääritystä. On tärkeää, että määritellyt vaatimukset ovat kattavia, koska vaatimusmäärittely on tärkein asiakirja, joka määrittelee tuotteen. Jos jotain asiaa ei vaadita tässä asiakirjassa, on hyvin todennäköistä, että sitä ei myöskään toteuteta.

Ohjelmistovaatimuksilla tulee olla yksilöllinen tunnus, otsikko ja kuvaus vaatimuksesta. Lisäksi jokaisesta ohjelmistovaatimuksesta tulisi määrittää linkki yhteen tai useampaan järjestelmävaatimukseen, sekä on myös oltava linkki vaatimuksen varmennukseen, tämä on tyypillisesti testitapauksen tunnus, jota käytetään myös testisuunnitelmassa ja -raportissa. Taulukossa 1. on esimerkki ohjelmistovaatimuksesta.

TAULUKKO 1. Esimerkki ohjelmistovaatimuksesta.

Tunnus	Nimi	Kuvaus	Systeemitason vaatimukset	Testitapaukset
REQ-3	Cold test support	SW must support cold tests for the following PCBAs: - PCBA 1 - PCBA 2 - PCBA 3	SYS-REQ-1	TESTCASE-3 TESTCASE-4 TESTCASE-5

Järjestelmäsuunnittelu on seuraava vaihe ohjelmistoprosessissa. Sen tavoitteena on määritellä ohjelmisto järjestelmätasolla. Järjestelmä jaetaan pienempiin kokonaisuuksiin, joiden välille on määritelty rajapinnat.

Tämän jälkeen määritellään ohjelmiston arkkitehtuuri. Tämä vaihe määrittelee aiemmin määriteltyjen komponenttien ohjelmistoyksiköt. Ohjelmistoyksikkö on tyypillisesti ohjelmistokomponentin osa,

joka suorittaa tiettyjä toimintoja, jotka ovat loogisesti yhteisiä toisilleen. Ohjelmiston arkkitehtuurimäärittely sisältää vuokaavion, joka näyttää kaikki ohjelmistoyksiköt sekä niiden väliset rajapinnat.

Yksi valinnainen dokumentti on käyttöliittymän määrittely, joka kuvaa ohjelmiston toimintaa sekä esittelee kaikki ohjelmiston käyttöliittymät. Näitä voivat olla esimerkiksi graafinen ja konsolikäyttöliittymä.

Yksi järjestelmäsuunnitteluvaiheen tärkeimmistä dokumenteista on ohjelmistointegraation suunnitelma. Suunnitelmassa kuvataan aikataulu kaaviona, jossa ohjelmistoyksiköt integroidaan kehitettävään ohjelmistoon. Se näyttää myös julkaisukandidaattien aikataulun.

Vaatimusten määrittelyn jälkeen ja ennen ohjelmiston toteutuksen alkamista, testaustiimi luo yhteistyössä ohjelmiston kehittäjän kanssa julkaisutestaussuunnitelman. Suunnitelma sisältää listan testitapauksista, mitkä koostuvat yksilöllisestä tunnuksesta, testitapauksen nimestä, käytetystä ympäristöstä ja laitteista sekä linkistä ohjelmiston vaatimukseen. Taulukosta 2. löytyy esimerkki testaussuunnitelman yksittäisestä testitapauksesta.

TAULUKKO 2. Esimerkki testaussuunnitelman testitapauksesta.

Tunnus	Nimi	Käytetyt ympäristöt ja laitteet	Ohjelmiston vaatimus
TESTCASE-3	Cold test support for the PCBA 1.	PC Production tester Testjig 1 PCBA 1	REQ-3

3.3 Koodaustyyliopas

”Yhtenäisen ohjelmointityylin noudattaminen lisää lähdekoodin ymmärrettävyyttä, ylläpidettävyyttä, siirrettävyyttä, luettavuutta ja uudelleenkäytettävyyttä”. (4, s. 377.)

Optomedillä on luotuna sisäinen dokumentti koodauskäytännöistä, joka kuvaa ohjelmistosuunnittelu ja -toteutuksen tyyli- ja käytännönohjeita käytössä oleville tuotteille ja eri ohjelmistokielille. Opas määrittelee yhteisesti muun muassa:

- tiedostojen, funktioiden ja muuttujien nimeäminen pitää olla selkeitä sekä kuvata niiden toimintaa
- sisennyksissä tulee käyttää Allman-tyyliä (5, s. 8.)
- ei saa käyttää kovakoodattuja numeroja tai merkkijonoja

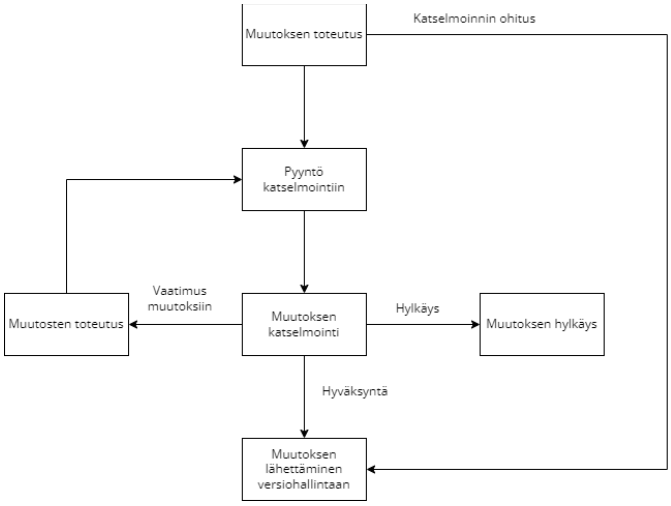
Kuvassa 2 on esimerkki hyvästä tavasta tehdä koodia.

```
const uint MaxConditionCount = 10;
bool condition = true;
uint count = 0;

while (condition == true)
{
    count++;
    condition = checkForConditionChange();

    if (count > MaxConditionCount)
    {
        break;
    }
    else
    {
        doSomethingElse(count);
    }
}
```

KUVA 2. Esimerkki hyvästä koodaustyylistä.

3.4 Koodin katselmointi

Koodin katselmointiprosessi alkaa siitä, että tekijä toteuttaa muutoksen. Muutos voi olla yksittäinen virheen korjaus tai osa isompaa kokonaisuutta. On kuitenkin suositeltavaa tehdä ja katselmoida useita pieniä muutoksia yhden suuren muutoksen sijaan, jotta voidaan varmistaa katselmoinnin laatu. Tämä mahdollistaa myös ongelmien tunnistamisen ja korjaamisen varhaisessa vaiheessa. Koodin katselmointiprosessin vuokaavio on esitetty kuvassa 3.



KUVA 3. Koodin katselmointiprosessin vuokaavio.

Pyydettäessä katselmointia muutoksen tekijän tulee kirjoittaa kuvaus muutoksesta ja valita sille katselmoijat. Muutoksen kuvauksen tarkoituksena on auttaa katselmoijia ymmärtämään katselmoitavan muutoksen. Kuvauksessa tulee olla syy muutokseen ja miten muutoksen on tarkoitus toimia. Sen tulee sisältää myös viittaukset asiaankuuluviin tehtäviin ja niihin liittyvään dokumentaatioon. Normaalisti katselmoijina toimivat yksi vanhempi kehittäjä ja yksi tai useampi nuorempi kehittäjä. Monimutkaisissa muutoksissa katselmoijina voi toimia useampi vanhempi kehittäjä. Katselmoijia valittaessa tulee ottaa huomioon, että ainakin yhden kehittäjän tulisi tuntea muokattu lähdekoodin, jos mahdollista. Lisäksi koulutustarkoituksiin on myös suositeltavaa ottaa mukaan kehittäjä, jolla ei ole ollenkaan tai on vähemmän tietoa katselmoitavasta lähdekoodista.

Katselmoijien tulee ensin keskittyä katselmoinnin ydinkysymyksiin. Katselmointivaiheessa katselmoijan tulee kysyä muutoksen tekijältä tarvittaessa selventäviä kysymyksiä ja ehdottaa mahdollisista parannuksista tekijän kanssa. Yleensä katselmoinnit suoritetaan kommentoimalla muutosta koodin katselmointijärjestelmään. Koodin katselmointijärjestelmästä pitäisi olla mahdollista löytää tietoa pyydetyistä muutoksista ja siitä, onko niitä korjattu tai hylätty.

Jos katselmoijat ovat pyytäneet muutoksia, ne on toteutettava. Kun muutoksia toteutetaan, katselmoijille on ilmoitettava ja muutokset tarkistettava.

Kaikkien katselmoijien on hyväksyttävä viimeisin versio muutoksesta, ennen kun se voidaan lähettää versiohallintaan. Katselmointi voidaan ohittaa vain erittäin hyvällä syyllä, joko muutos on hyvin pieni tai katselmoijia ei ole paikalla, jolloin koodi tulisi katselmoitava myöhemmin.

Jos muutoksen hyväksymisestä ei päästä sopimukseen, ohjelmistotiimin kokous tulee järjestää. Ohjelmistoryhmän kokouksessa voidaan tehdä päätös muutoksen hylkäämisestä.

3.5 Toteutus ja yksikkötestaus

Toteutusvaiheessa aloitetaan varsinainen koodausprosessi. Sen aikana on tärkeää noudattaa koodauskäytäntöjä helposti ymmärrettävän ja yksinkertaisen koodin saavuttamiseksi. Se hyödyttää myös myöhemmin ohjelmistosuunnittelun dokumentointiprosessia. Tässä vaiheessa ohjelmiston on mentävä myös katselmointiprosessin läpi, jotta se saadaan versiohallintaan.

Ohjelmistoyksiköiden kehittäjän on testattava ohjelmistoyksiköt asianmukaisesti, jotta voi taata niiden oikea toimivuus onnistuneissa ja virheellisissä tilanteissa. Yksiköiden tulee olla vakaita ja tarjota informatiivisia palautusarvoja yleisen vakauden ja oikean toiminnan tukemiseksi.

Yksikkötestit auttavat varmistamaan koodin oikeellisuuden, lyhentämään ohjelmistokehityksen aikaa, parantamaan kehittäjien tuottavuutta ja tuottamaan tehokkaampia ohjelmistoja (6.). Yksikkötestauksen suunnitelma tehdään ohjelmistovaatimusten ja -suunnittelun pohjalta ennen varsinaisen toteutuksen aloittamista. Ohjelmiston toteutuksen jälkeen kehittäjä suorittaa varsinaisen testauksen ja luo raportin, joka tulee sisältää testiympäristön ja käytetyt laitteet, annetut parametrit, testiaskeleet, hyväksymisperusteet, testin ulostulon sekä tuloksen.

3.6 Julkaisutestaus ja validointi

Kun ohjelmiston toteutus on valmis ja yksikkötestaus on suoritettu sekä se on käynyt katselmointiprosessin läpi, on vuorossa julkaisutestaus. Ohjelmiston kehittäjä luo julkaisukandidaatin ja toimittaa sen testaustiimille. Testaustiimi alkaa suorittamaan testausta aiemmin luodun suunnitelman mukaisesti. Testaustiimi luo testauksesta varmennusraportin, mikä sisältää testaussuunnitelman testitapauksien lisäksi niille saadut tulokset.

Jos testauksen aikana löytyy toiminnallisia virheitä, tulee ohjelmiston kehittäjä aloittaa heti niiden korjaaminen. Korjausten jälkeen julkaisutestaus aloitetaan alusta.

Kun kaikki testitapaukset on testattu hyväksytysti läpi, katselmoidaan tulokset laatu-, ohjelmisto-, testaus-, tuotanto- ja tuotehallintatiimin kanssa ja päätetään ohjelmiston validoinnista. Validoinnin jälkeen ohjelmisto voidaan julkaista ja ottaa käyttöön tuotannossa.

Lopuksi ohjelmiston kehittäjän on luotava ohjelmiston suunnittelumäärittely dokumentti, joka voidaan luoda automaattisesti Doxygen-työkalun avulla. (7.)

4 OHJELMISTON SUUNNITTELU

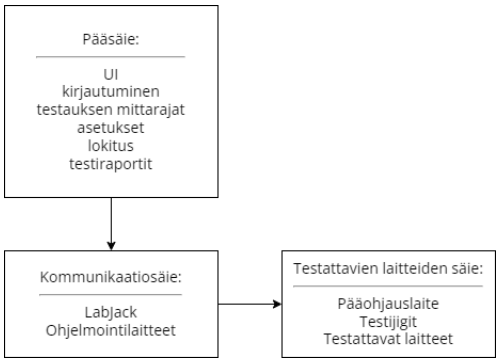
Ensimmäiseksi ohjelmistosuunnitteluvaiheessa luotiin ohjelmistointegraation suunnitelma. Suunnitelmassa hahmoteltiin ohjelmiston rakenne karkeasti. Ohjelmisto paloiteltiin pienempiin yksiköihin ja ohjelmiston toteutus aikataulutettiin. Aikataulussa määriteltiin taulukkona ohjelmistoyksiköiden integroinnin kesto ohjelmistoon ja julkaisukandidaatin ajankohdat, jolloin voitiin varata muun muassa testaustiimistä henkilöstöresursseja julkaisutestaukseen.

4.1 Ohjelmiston arkkitehtuuri

Ennen ohjelmointityön alkamista käyttöjärjestelmästä luotiin yrityksen sisäinen arkkitehtuurimäärittelydokumentti, jossa kuvataan tarkemmin ohjelmiston eri komponenttien toimintaa. Ohjelmisto koostuu useista yksiköistä, joille on määritelty sisäiset ja julkiset rajapinnat. Nämä yksiköt on jaettu edelleen kolmeen säikeeseen, jotka suorittavat toimintojaan rinnakkain ja itsenäisesti. Säikeet kommunikoivat toistensa kanssa signaalien ja niihin yhdistettyihin funktioiden avulla. Ohjelmiston arkkitehtuurin vuokaavio on esitetty kuvassa 4.

Pääsäie on vastuussa ohjelman loogisesta toiminnasta. Se luo useimmat yksiköt ja säikeet sekä vastaa kommunikoinnista suorittavan koodin ja graafisen käyttöliittymän välillä. Pääsäikeen kautta luodaan myös lokitiedostot ohjelman toiminnasta ja raportit tehdyistä testeistä.

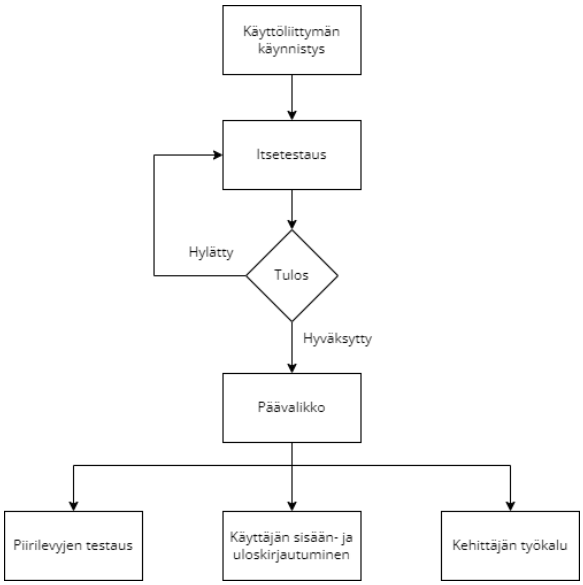
Kommunikaatiosäie vastaa fyysisten LabJack tiedonkeruu- sekä ohjelmointilaitteiden ohjauksesta. Testattavien laitteiden säie luodaan jokaista testiä varten uudestaan. Se huolehtii testien käyttötapahtumista ja ohjaa pääohjauslaitteen sekä testijigien piirejä kommunikaatiosäikeen kautta. Testattavien laitteiden säie lähettää signaaleja myös testien edistymisestä ja -tuloksista pääsäikeen kautta graafiselle käyttöliittymälle.



KUVA 4. Ohjelmiston arkkitehtuurin vuokaavio.

4.2 Käyttöliittymän määrittely

Ohjelman käynnistyessä ohjelman tulee ajaa sisäinen testaus, jossa käydään läpi testauslaitteen sisäisten komponenttien toiminta ja yhteys käyttöjärjestelmälle. Käyttöliittymän päätoiminta koostuu kolmesta osiosta: piirilevyjen testauksesta, käyttäjän sisään- ja uloskirjautumisesta sekä kehitystyökalusta. Käyttöliittymän päätoiminnan vuokaavio on esitetty kuvassa 5.



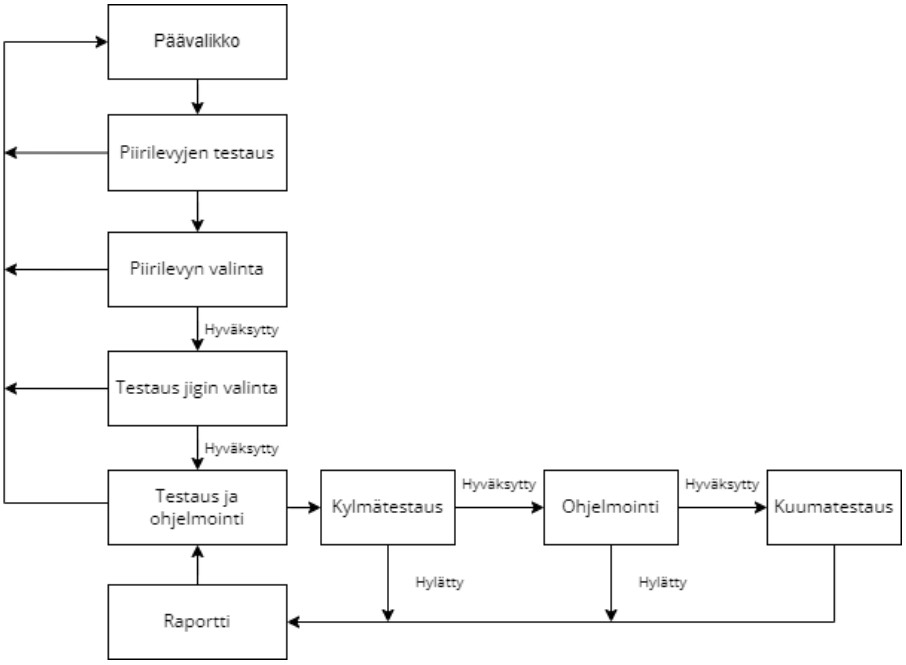
KUVA 5. Käyttöliittymän päätoiminnan vuokaavio.

4.2.1 Piirilevyjen testaus

Jokaiselle testattavalle laitteelle on luotava erilliset testitapaukset sekä mittaustuloksen ylä- ja alarajat. Niiden tarkoituksena on testata laitteen toiminta riittävän kattavasti. Testitapaukset jaetaan

kolmeen testivaiheeseen: kylmätestaus, ohjelmointi ja kuumatestaus. Testi tulee voida myös tarvittaessa keskeyttää, jos todetaan, että sitä ei voida viedä laitteelle turvallisesti loppuun, esimerkiksi oikosulkutapauksessa. Piirilevyn testauksen vuokaavio on esitetty kuvassa 6.

Kylmätestauksen tarkoituksena on testata ja havaita mahdolliset oikosulut eri jännite- ja maatason välillä sekä varmistaa testien turvallinen jatkuminen. Ohjelmointivaiheessa laitteen mahdollinen vanha ohjelmisto poistetaan ja laite ohjelmoidaan uusiksi. Lisäksi varmistetaan, että laite on onnistuneesti ohjelmoitu ja uusi ohjelma tunnistetaan. Kuumatestauksen tarkoituksena on testata laitteen funktionaalinen toiminta. Laitteen jokaisen piirin toiminta testataan.



KUVA 6. Piirilevyn testauksen vuokaavio.

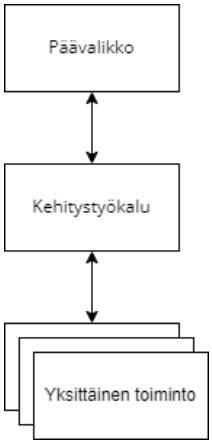
4.2.2 Käyttäjän sisään- ja uloskirjautuminen

Käyttäjän ensimmäisenä toimena on kirjautua sisään. Käyttäjä antaa oman yksilöllisen tunnuksensa, joka liitetään myöhemmin testiraportteihin mukaan. Käyttäjä voi myös antaa ohjelmistolle määritellyn salasanan, jolla saadaan kehittäjän työkalun käyttöön. Käyttäjän sisään- ja uloskirjautumisen vuokaavio on esitetty kuvassa 7.

KUVA 7. Käyttäjän sisään- ja uloskirjautumisen vuokaavio.

4.2.3 Kehitystyökalu

Kehitystyökalun avulla voidaan ohjata ja lukea pääohjauslaitteen sisältämiä yksittäisiä analogi-digitaali- ja digitaalianalogimuuntimia, I/O-laajentimia sekä multipleksereitä. Työkalun avulla voi myös lähettää I2C-linjoille luku- ja kirjoituskäskyjä. Näiden toimintojen avulla voidaan luoda ja testata yksittäisiä testitapauksia. Kehitystyökalun vuokaavio on esitetty kuvassa 8.



KUVA 8. Kehitystyökalun vuokaavio.

5 TOTEUTUS

Toteutusvaihetta voitiin lähteä toteuttamaan, kun kaikki työtä ohjaavat dokumentit oltiin saatu valmiiksi. Toteutuksen tuli noudattaa suunnitteluvaiheessa määriteltyjä dokumentteja.

Toteutusvaiheen aluksi määriteltiin käytetyt työkalut ja ohjelmistokehitysympäristö. Tämän jälkeen voitiin aloittaa varsinainen ohjelmointi. Ohjelmoinnin yhteydessä luotiin ja ajettiin yksikkötestit.

5.1 Käytetyt työkalut ja ohjelmistot

Ennen ohjelmointityön alkamista käytettävissä olevista kehitysympäristöistä ja ohjelmointikielistä tehtiin alustava kartoitus. Ohjelmointikielistä lopulta kaksi vaihtoehtoa erottautui muista, Python ja C++.

Pythonin hyviin puoliin kuului, että sillä oli ohjelmoitu aiempi ohjelmisto ja vanhaa koodipohjaa olisi voitu käyttää hyväksi uuden ohjelmiston ohjelmointiin. Lisäksi Python on alustariippumaton käyttöjärjestelmästä. Pythoniin saa myös runsaasti kirjastoluokkia.

Huonoina puolina Pythonin valinnassa pidettiin, että aiempi ohjelmisto oli ohjelmoitu Python 2:lla, ja sen siirtäminen Python 3:lle toisi aikataulullisia lisähaasteita. Lisäksi vanhassa koodipohjassa oli niin sanottua ”kuollutta koodia”, jonka läpi käymiseen ja korjaamiseen olisi mennyt paljon aikaa. Pythonin valintaa ohjelmointikieleksi ei myöskään puoltanut, että sen käytöstä ei löytynyt paljoa kokemusta.

Lopulta päädyttiin käyttämään C++-ohjelmointikieltä, josta löytyi kokemusta usean vuoden ajalta. C++:lle löytyi myös tarvittavat kirjastot, joilla kommunikointi muun muassa LabJackin kanssa onnistuisi. Kehitystyökaluiksi valikoitui Qt ja QML.

Qt on myös alustariippumaton ja on kirjoitettu tavallisella C++:lla. Qt sisältää runsaasti C++ kirjastoluokkia, mutta on rikastettu useilla omilla helppokäyttöisillä funktioilla ja makroilla. Qt:n kirjastoluokat ovat pääsääntöisesti lisensoitu GPLv2:n alle (8.). (9.)

QML (Qt modeling language) on Qt:n päälle rakennettu graafisiin käyttöliittymien suunnitteluihin soveltuva merkintäkieli. QML on suunniteltu helppokäyttöiseksi ja alustariippumattomaksi. Qt Quick on QML:n standardikirjasto, joka tarjoaa kaiken tarpeellisen graafisen käyttöliittymän toteutukseen. (9.)

5.2 Ohjelmointi

Ohjelmoinnin toteutus jaettiin kahteen osaan, backend- ja frontend-osiin. Backend tekee ohjelmiston toiminnan sekä tiedon käsittelyn. Frontend määrittää käyttäjärajapinnan eli graafisen käyttöliittymän. Myös backend ja frontend jaettiin useimpiin pienempiin kokonaisuuksiin.

Backendin kirjoitus tapahtui C++-ohjelmointikielellä noudattaen C++17-standardia (10, s. 21.). Toteutus suoritettiin kappaleessa 4.1 esitellyn arkkitehtuurimäärittelyn mukaisesti. Jokaisesta testauslaitteen fyysisestä komponentista luotiin oma ohjelmistoyksikkö tai luokka. Lisäksi luotiin yleisiä apufunktioita esimerkiksi tallentamaan testaustuloksia muistiin ja myöhemmin niistä koostettuja raportteja tiedostoon. Jakamalla ohjelmisto pienempiin luokkiin ja funktioihin, tulee ohjelmistosta helpommin luettava ja ymmärrettävä. Tämä helpottaa myöhemmässä vaiheessa tehtävien yksikkötestien suunnittelua ja toteutusta.

Myös testattavan laitteiden testivaiheet jaettiin pienempiin testitapahtumiin, joiden toteutus noudatti kuvan 9 esimerkin mukaista toimintaa. Aluksi asetetaan haluttu komponentti tiettyyn tilaan ja luetaan mittaustulos. Lopuksi mittaustulos tallennetaan muistiin myöhempään käyttöä varten. Lisäksi jokaisen testitapahtuman jälkeen lähetetään käyttöliittymälle signaali testien edistymisestä.

```
/** 2.001 V3V3 */
measurementNumber = "2.001";
measurementName = "V3V3";
labJack->setDigitalPinState(CPTCommon::DigitalState::Low, LabJackCommon::FIO_6);
labJack->readAnalogInput(&measuredValue, LabJackCommon::AIN_7);
testInfo->appendResult(testLimitGroup, measurementNumber, measurementName, measuredValue, TestInfo::MeasurementUnit::V);
```

KUVA 9. Koodiesimerkki testitapahtumasta.

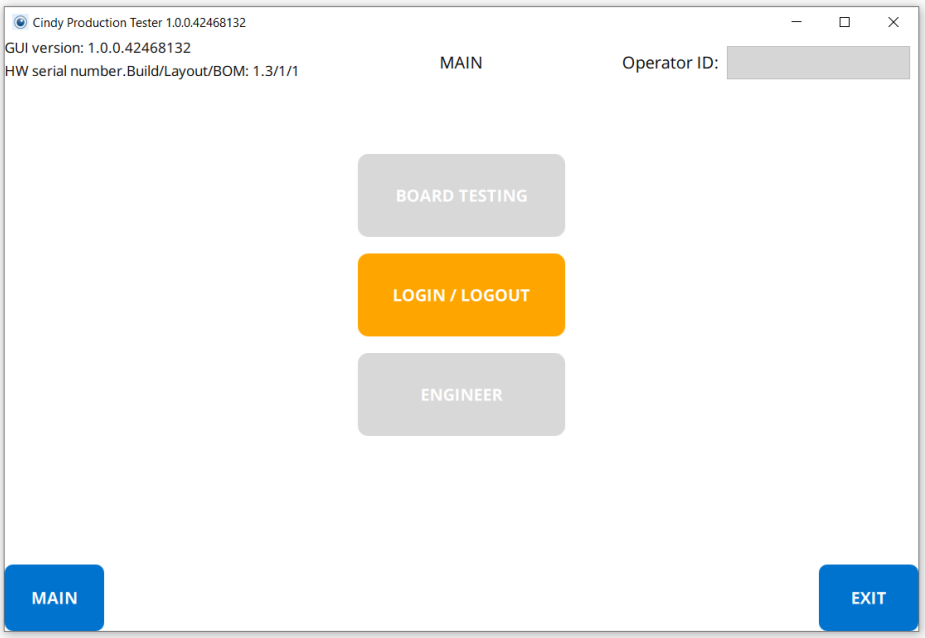
Graafinen käyttöliittymä suunniteltiin QML-ohjelmointikielellä. Käyttöliittymän tuli olla selkeä ja helppokäyttöinen, jotta testaus nopeutuu ja käyttäjän ei tarvitse käyttää paljoa resursseja sen käytön opetteluun.

Jokaisesta näkymästä ja sivusta luotiin oma kustomoitu komponentti. Lisäksi luotiin jokaisesta näkymän tai sivun elementeistä omat komponentit. Kuvassa 10 esitellään omatekoisen napin käyttöä koodissa.

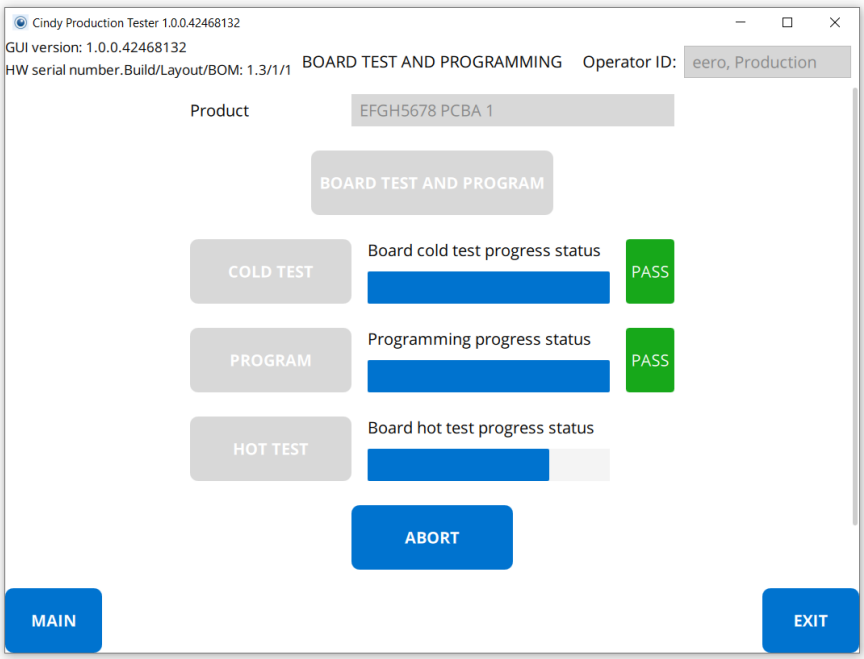
```
OMComp.BaseButton
{
    id: boardTestButton

    buttonType: OMComp.BaseButton.Type.Primary

    //: The text of the Board test button
    //% "Board testing"
    buttonText: qsTrId("board-test-button")

    buttonHeight: buttonHeight
    buttonWidth: buttonWidth
    Layout.alignment: Qt.AlignHCenter

    onButtonClicked:
    {
        requestAppState("TESTING");
    }
}
```

KUVA 10. Esimerkki omatekoisen komponentin käytöstä.

Graafisen käyttöliittymän toteutus noudatti tarkalleen kappaleessa 4.2 esitettyä käyttöliittymän määrittelyä. Alla esitellään toteutetun ohjelmiston päänäkymä (kuva 11) ja erään piirilevyn testaus suoritusvaiheessa (kuva 12).



KUVA 11. Ohjelmiston päänäkymä.



KUVA 12. Piirilevyn testaus suoritusvaiheessa.

5.3 Yksikkötestaus

Suunnitteluprosessin mukaisesti toteutuksen yhteydessä luotiin yksikkötestauksen suunnitelma ja ensimmäiset testit. Qt tarjoaa yksikkötestaukseen Qt Test -nimisen kehyksen, joka sisältää kaikki yksikkötestaukseen liittyvät toiminnot ja laajennuksen graafisen käyttöliittymän testaukseen (9.).

Qt Testin rakenne on yksinkertainen, jokainen yksityinen slot-funktio (private slots) määritellään testifunktioksi ja jokainen testifunktio suoritetaan, kun yksittäisen yksikön testi suoritetaan. Yksi testi voi sisältää useita testifunktioita. Lisäksi Qt Test esittelee muutaman funktion, joita ei määritellä testifunktioksi:

- initTestCase()-funktiota kutsutaan aina ennen, kun ensimmäinen testifunktio suoritetaan.
- initTestCase_data()-funktiota kutsutaan luomaan globaalin testidatataulukon.
- cleanupTestCase()-funktiota kutsutaan aina, kun viimeinen testifunktio on suoritettu.
- init()-funktiota kutsutaan aina ennen jokaisen testifunktion suorittamista.
- cleanup()-funktiota kutsutaan aina jokaisen testifunktion suorittamisen jälkeen.

Tässä opinnäytetyössä toteutetussa ohjelmistossa testataan jokaisen yksikön funktionaalinen toiminto sekä yksittäisten komponenttien ja luokkien toiminta. Testeissä testataan myös komponenttien virheellinen toiminto, jolla saadaan selvitettyä ohjelmiston käyttäytyminen virhetilanteissa. Esimerkki testifunktiosta löytyy kuvasta 13.

```
/**
 * @brief tst_ControllerBoard::selfTest
 * Performs controller board's self test
 */
void tst_ControllerBoard::selfTest()
{
    TestInfo *selfTestInfo = new TestInfo(measurementLimits);
    QSignalSpy testFinishedSpy(controllerBoard, &ControllerBoard::testFinished);

    labJackMock->setControllerBoardSelfTestEnabled(true);
    controllerBoard->selfTest(selfTestInfo);

    QCOMPARE(testFinishedSpy.count(), 1);

    selfTestInfo = qvariant_cast<TestInfo *>(testFinishedSpy.takeFirst().at(0));

    QCOMPARE(selfTestInfo->isPass(), true);
} /* selfTest */
```

KUVA 13. Pääohjauslaitteen itsetestauksen testifunktio.

6 JULKAISUTESTAUS, VALIDOINTI JA KÄYTTÖÖNOTTO

Toteutus- ja yksikkötestausvaiheen jälkeen vuorossa on virallinen julkaisutestaus. Virallisen julkaisutestauksen suoritti erillinen testaustiimi. Testauksessa vahvistettiin jokaisen ohjelmistovaatimuksen toiminta ja testauksen lopuksi luotiin varmennusraportti. Raportista kävi ilmi, että testauksen aikana löytyi muutama virhe, mutta ne eivät estäneet ohjelmiston käyttöönottoa ja ohjelmisto otettiinkin tuotannossa käyttöön.

7 POHDINTA

Optomedin suunnitellessa ja valmistaessa lääketieteellisiä laitteita on erittäin tärkeää, että tuotannontestaus on laadukasta ja tehokas. Myös testauslaitteet ja ohjelmistot tulisi olla suunniteltu ja toteutettu nykyaikaisilla menetelmillä.

Opinnäytetyön tavoitteena oli suunnitella ja toteuttaa käyttöliittymäohjelmisto tuotannon testauslaitteen ohjaukseen. Työn toteutus tuli noudattaa ja täyttää lääketieteellisen ohjelmiston standardin vaatimuksia.

Konsepti- ja toteuttavuustutkimusta tehdessäni todettiin, että vanhaa ohjelmistoa ei enää kannattanut aloittaa korjaamaan ja kehittämään uusien laitteiden osalta, joten todettiin tarpeelliseksi suunnitella ja toteuttaa uusi ohjelmisto tuotannon testauslaitteen ohjaukseen.

Ohjelmistoa lähdettiin suunnittelemaan ja toteuttamaan suunnitteluprosessin mukaisesti. Kaikkien työtä ohjaavien dokumenttien valmistuttua toteutuksen tekeminen oli selkeää ja nopeaa. Toteutuksen aikana huomattiin, että testauslaitteella ei kyetä testaamaan kaikki haluttuja toimintoja, johtuen testauslaitteen virranmittauspiirien epätarkkuudesta. Näiden selvittely aiheutti aikatauluun viivästyksiä. Viivästyksien johdosta osa matalan prioriteetin testeistä jouduttiin jättämään tässä vaiheessa tekemättä.

Työn aikana sain kattavan osaamisen lääketieteellisen ohjelmiston standardin vaatimuksista ja soveltamisesta ohjelmistoprossiin. Huolimatta muutamasta haasteesta käyttöliittymäohjelmisto tuotannon testauslaitteen ohjaukseen saatiin valmiiksi ja otettua käyttöön tuotannossa.

Seuraavassa kehitysvaiheessa olisi tarkoitus lisätä puuttuvat testit ja toteuttaa ohjelmistomuutos korvatuille virranmittauspiireille.

LÄHTEET

1. Optomed Oyj 2024. Tilinpäätöstiedote Q4 2023. Optomed lyhyesti. Oulu. Hakupäivä 7.3.2024. https://www.optomed.com/wp-content/uploads/2024/02/Optomed-tilinpaatostiedote-Q4-2023-FIN.pdf.
2. Mattila, Heikki 2014. T723303 Testausmenetelmät, 3 op. Opintojakson oppimateriaali. Oulu: Oulun ammattikorkeakoulu, tietotekniikan osasto.
3. IEC 2006. IEC 62304:2006 Medical device software - Software life cycle processes. Hakupäivä 5.2.2024. https://www.iso.org/standard/38421.html
4. Rintala, Matti & Jokinen, Jyke 2003. Olioiden ohjelmointi C++:lla. Helsinki: Talentum Oyj.
5. Cyganek, Boguslaw 2021. Introduction to Programming with C++ for Engineers. Krakova: Wiley.
6. Hamill, Paul 2004. Unit Test Frameworks: Tools for High-Quality Software Development. First Edition. Sebastopol: O'Reilly Media.
7. Doxygen 2023. Documentation generator tool, version 1.10.0. Hakupäivä 11.5.2024. https://www.doxygen.nl/
8. Free Software Foundation Inc 2024. GNU General Public License, version 2. Hakupäivä 17.3.2024. https://www.gnu.org/licenses/gpl-2.0.html
9. Lazar, Guillaume & Penea, Robin 2018. Mastering Qt 5: Create stunning cross-platform applications using C++ with Qt Widgets and QML with Qt Quick. Second Edition. Birmingham: Packt Publishing.
10. Stroustrup, Bjarne 2013. The C++ Programming Language. Fourth Edition. Boston: Addisson-Wesley.