Vesiputousmallin soveltuvuus tietovuoparadigmaa käyttäviin projekteihin

Mikko Pitkänen

Pro gradu –tutkielma



Tietojenkäsittelytieteen laitos

Tietojenkäsittelytiede

Ajassa X

ITÄ-SUOMEN YLIOPISTO, Luonnontieteiden ja metsätieteiden tiedekunta, Kuopio

Tietojenkäsittelytieteen laitos

Tietojenkäsittelytiede

Opiskelija, Mikko Pitkänen: Ohjelmistokehitysmallien vertailu tietovuoparadigmaa käyttävissä projekteissa

Pro gradu –tutkielma

Pro gradu –tutkielman ohjaajat: FT Niina Päivinen FT Isaac Afara

Kuukausi X 201X

TIIVISTELMÄ

AVAINSANAT

ACM-luokat (ACM Computing Classification System, 1998 version): ks. tutkielman laatimisohje

UNIVERSITY OF EASTERN FINLAND, Faculty of Science and Forestry, Kuopio

School of Computing

Computer Science

Opiskelija, Mikko Pitkänen: Comparative study of various software development models in projects that use dataflow paradigm

Master’s Thesis

Supervisors of the Master’s Thesis: PhD Niina Päivinen and PhD Isaac afara

Mont X 201X

Abstract

Keywords:

CR Categories (ACM Computing Classification System, 1998 version):

Esipuhe

Haluaisin kiittää:

Lista ihmisten nimistä tähän.

Lyhenneluettelo

OCT Optical coherence tomography

VI LabView kielellä toteutettu ohjelma

Coming soon

Sisällysluettelo

[Johdanto 2](#_Toc416879579)

[Rinnakkaislaskenta 3](#_Toc416879580)

[1.1 Rinnakkaislaskenta 3](#_Toc416879581)

[2 Suoritusmallit ja rinnakkaisuus 4](#_Toc416879582)

[2.1 Von Neumann malli 4](#_Toc416879583)

[2.2 Tietovuoparadigma 5](#_Toc416879584)

[2.3 Tietovuoparadigman erikoispiirteet 7](#_Toc416879585)

[2.4 Tietovuokielen vahvuudet ja heikkoudet 9](#_Toc416879586)

[2.4.1 Vahvuudet 9](#_Toc416879587)

[2.4.2 Heikkoudet 11](#_Toc416879588)

[Vesiputousmalli 13](#_Toc416879589)

[2.5 Vaatimusten määrittely 14](#_Toc416879590)

[2.6 Suunnittelu 15](#_Toc416879591)

[2.6.1 Käyttötapauskaavio 16](#_Toc416879592)

[2.6.2 Tietovuokaavio 16](#_Toc416879593)

[2.7 Toteutus 17](#_Toc416879594)

[2.7.1 LabView ohjelmointi 17](#_Toc416879595)

[2.8 Testaus 19](#_Toc416879596)

[2.9 Ylläpito 19](#_Toc416879597)

[Tavoitteet 20](#_Toc416879598)

[Materiaalit ja metodit 21](#_Toc416879599)

[2.10 LabView 21](#_Toc416879600)

[2.11 Tapaustutkimus 21](#_Toc416879601)

[Uusi teoreettinen tulos 22](#_Toc416879602)

[Pohdintaa 23](#_Toc416879603)

[Yhteenveto 24](#_Toc416879604)

[Viitteet 25](#_Toc416879605)

# Johdanto

Rinnakkaislaskennan määrän odotetaan Mark & Marty (2008) mukaan lisääntyvän, sillä tulevaisuuden prosessoriarkkitehtuurit tulevat sisältämään jopa tuhansia ytimiä. Tämänkaltainen laitteistotason muutos aiheuttaa haasteita sovelluskehityksessä, sillä tällä hetkellä ei ole olemassa malleja näin suuren rinnakkaisuuden implementoimiseen. Aiheen tutkiminen on täten ajankohtaista, jotta olemassa olevia ratkaisuja pystytään kehittämään ongelman ratkaisemiseksi.

Tutkimuksen eksperimentaalinen osuus toteutettiin sovelletun fysiikan laitoksella Itä-suomen yliopistossa BBC-ryhmän alaisuudessa. Ryhmä tarvitsi käyttöönsä järjestelmän, jolla on mahdollista näyttää tarpeen mukaan eri videosyötteitä ruston mekaanisten ominaisuuksien mittaamisen aikana. Projekti toteutettiin vesiputousmallia käyttäen, mitä käytetään referenssinä muita ohjelmistokehitysmalleja vertaillessa. Ohjelmointikielenä käytettiin National Instrumentsin Labview-kieltä, joka on visuaalinen tietovuokieli.

# Rinnakkaislaskenta

## Rinnakkaislaskenta

Skillicorn & Talia (1998) mukaan rinnakkaislaskennasta ajankohtaisen tekee prosessorien kehityksen suunnan muutos, joka on olosuhteiden pakosta joutunut tukemaan moniydinarkkitehtuureja. Tämä johtuu siitä, että pienempiä komponentteja kehittäessä joudutaan ratkaisemaan atomaariseen mittakaavaan liittyviä ongelmia, mikä tekee rinnakkaislaskennasta kustannustehokkaan vaihtoehdon. Useammalla ytimellä voidaan nostaa laskentatehoa, koska eri osiin jaettua ongelmaa voidaan ratkaista samanaikaisesti, mikä tekee rinnakkaislaskennasta imperatiivisia ratkaisuja tehokkaamman.

Prosessoriarkkitehtuuri on siinä määrin kehittynyttä, että rinnakkaislaskenta komponenttitasolla ei vaadi toimenpiteitä, vaan ongelma on Skillicorn & Talia (1998) mukaan siirtynyt sovellustasolle. Sovellusten ohjelmoiminen rinnakkaislaskentaa tukevalla tavalla on haasteellista, sillä ihmisillä on tapana ajatella asioita imperatiivisesti, joten rinnakkaisuuden tunnistaminen voi olla haasteellista.

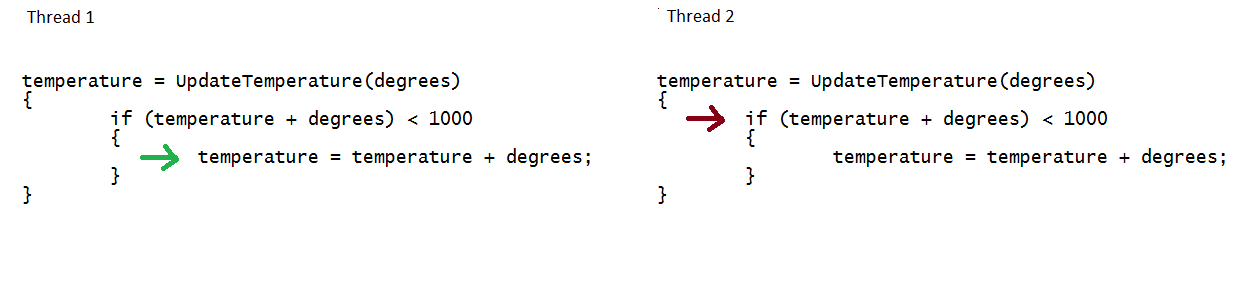
# Suoritusmallit ja rinnakkaisuus

## Von Neumann malli

Marttila-Kontion (2011) mukaan Von Neumann malli määritellään siten, että siinä käytetään globaalia muistiosoitejärjestelmää datan tallentamista ja muokkaamista varten. Yksittäinen ohjelmalaskuri on vastuussa muistiosoitteiden päivittämisestä, minkä takia mallia noudattavien ratkaisujen kutsutaan noudattavan imperatiivista paradigmaa.

Marttila-Kontion (2011) mukaan perinteisiksi mielletyt ohjelmointikielet kuten C, Java ja C++ noudattavat Von Neumann mallia, jossa rinnakkaisuuden implementoiminen on ongelmallista. Kaikissa mainituissa kielissä on kuitenkin olemassa jonkinlaisia rinnakkaisuuden mahdollistavia ratkaisuja, jotka eivät kuitenkaan ole täysin ongelmattomia.

Reinders (2007) mainitsee, että rinnakkaisuuden implementoiminen voi olla vaarallista, sillä rinnakkain toimivat säikeet voivat aiheuttaa väärin toteutettuna ikäviä tilanteita. Reindersin (2007) mukaan termiä ”thread safety” käytetään käsitellessä säikeistyksestä aiheutuvia ongelmia, jotka johtuvat algoritmien tuottamista sivuvaikutuksista. Algoritmi aiheuttaa sivuvaikutuksia, mikäli se vaikuttaa näkyvyysalueensa ulkopuolisiin tekijöihin, kuten globaaleihin tai staattisiin muuttujiin. Alla olevasta kuvasta voidaan nähdä esimerkki sivuvaikutuksesta, joka voi potentiaalisesti olla syynä vakavaan ohjelmointivirheeseen.



Kuva 1: Sivuvaikutuksia aiheuttava rinnakkaistettu funktio

Mikäli säie 1 ei ehdi päivittää muuttujan *temperature* arvoa, niin säikeen 2 if-lause palauttaa true. Täten säie 2 pääsee virheellisesti päivittämään *temperature* muuttujan arvoa. Virhettä kutsutaan kilpailutilanteeksi, jossa funktion palautusarvo on riippuvainen säikeiden suoritusjärjestyksestä. Yllä oleva funktio pitäisi kriittisten osien kohdalta lukita muilta säikeiltä, jotta virhettä ei pääsisi tapahtumaan.

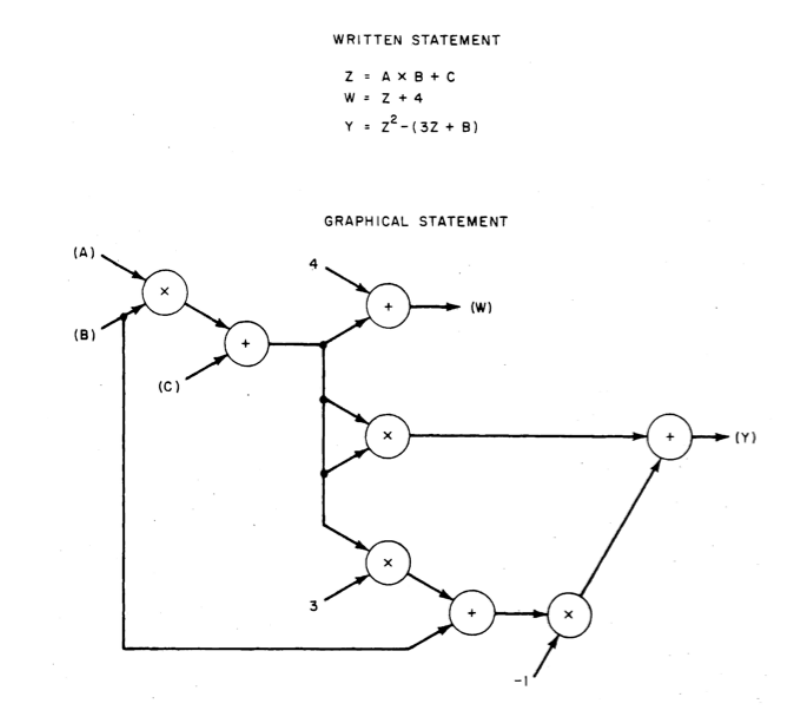
Säikeiden välinen kommunikaatio on Reindersin (2007) mukaan välttämätöntä virhetilanteiden välttämiseksi. Käytännössä tämä tarkoittaa sitä, että sivuvaikutuksia aiheuttavat rivit eristetään muilta säikeiltä kunnes kyseisessä vaiheessa oleva säie on saanut rivit suoritettua. Tämän jälkeen alue vapautetaan seuraavalle säikeelle. Tämäkään ei ole virheetön ratkaisu, sillä säikeitä lukitsemalla on mahdollista saavuttaa lukkiumatilanne, jossa säikeet eivät pääse jatkamaan suoritustaan. Tilanne syntyy kun joku mielivaltainen säie tarvitsee lukitussa käytössä olevan resurssin vapauttaakseen oman lohkonsa ja käyttää samalla resurssia, jota odotettava lohko tarvitsee.

Sousan (2012) mukaan säikeistäminen on yllämainituista syistä virheille altis ratkaisu rinnakkaislaskennan saavuttamiseksi. Tämän takia on kehitetty malleja, jotka välttävät suurimman osan säikeistämisen ongelmista samalla nostaen rinnakkaislaskennan abstraktion tasoa. Skillicorn & Talia (1998) kutsuvat näitä malleja yksinomaan rinnakkaisiksi malleiksi joiksi he mainitsevat mm. algoritmi luurangon ja tietovuon.

## Tietovuoparadigma

Sousa (2012) vertaa tietovuoparadigmaa tietovuokaavioon, jossa jokainen solmu on itsenäisesti toimiva lohko, jonka toiminta ei ole muista lohkoista riippuvainen. Lohko käynnistyy, kun se on saanut tarvittavat syöttöparametrit lohkossa olevien käskyjen suorittamiseen. Täten toimimalla tietovuoparadigma välttää yleisimmät rinnakkaislaskennassa tavattavat ongelmat, sillä lukkiutumista ei pääse syntymään, koska rinnakkain toimivat lohkot eivät vaikuta toistensa toimintaan. Sivuvaikutuksien puute onkin yksi tietovuoparadigman ominaispiirre, josta puhun lisää seuraavassa kappaleessa.

Toisistaan riippumattomien lohkojen avulla voidaan Sousan (2012) mukaan saavuttaa rinnakkaisuutta suuremmassa määrin imperatiivisiin ratkaisuihin verrattuna, sillä ohjelmoijan ei tarvitse erikseen määritellä tilanteita joissa rinnakkaisuutta käytetään. Alla olevasta kuvasta nähdään, että lyhyessäkin algoritmissa voi olla paljon rinnakkaisuutta, jota ei välttämättä pystyttäisi perinteisin ratkaisuin hyödyntämään.

Kuva 2:

Yllä olevasta kuvasta voidaan nähdä sama algoritmi esitettynä tekstuaalisessa ja visuaalisessa muodossa. Suunnatusta verkosta voidaan nähdä että samalla leveyssuunnan tasolla sijaitsevat lohkot voidaan suorittaa samanaikaisesti.

Edellä mainituista syistä johtuen voidaan nähdä, että tietovuo välttää imperatiivisten kielien rajoitukset sitomalla käskyjen suoritus saatavilla olevaan dataan. Perinteisillä kielillä tämän mittakaavan rinnakkaisuuden implementoiminen on Marttila-Kontion (2011) mukaan mahdotonta koska Von Neumann mallissa on vain yksi ohjelmalaskuri, mikä estää useamman käskyn samanaikaisen ajamisen.

## Tietovuoparadigman erikoispiirteet

Tietovuoparadigmalla on Ackermanin (1982) mukaan kuusi sitä yksilöivää ominaisuutta, joita ei nähdä samanaikaisesti muissa ohjelmointiparadigmoissa. Nämä ominaisuudet ovat:

1. Sivuvaikutuksien puute
2. Vaikutuksien paikallisuus
3. Muuttujien määrittelyn rajoittaminen
4. Iteraation outo notaatio kohdista 1 ja 4 johtuen
5. Proseduurit eivät muista edellisiä syöttöparametrejään

Ackermanin (1982) mukaan Tietovuoparadigma vaatii että käytettävät muuttujat määritellään vain kerran, koska käskyjen suoritus on kyseisistä muuttujista riippuvainen. Ohjelmien toiminnallisuus kärsisi perustavanlaatuisella tavalla, mikäli muuttujien määritteleminen useammin kuin kerran olisi mahdollista.

Oletetaan tilanne jossa muuttuja määritellään useammin kuin kerran

Pseudokoodi:

X = 5;

Y = mielivaltainen luku;

Z =mielivaltainen luku;

X = X + Y;

X = 10;

Z = X + Z;

Tietovuoparadigmaa käyttäen koodi toteutuisi alla olevan kuvan mukaisesti.



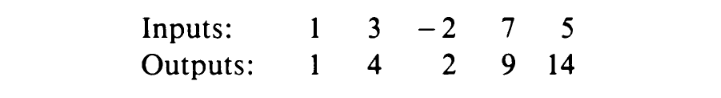
Kuva 3: Pseudokoodi

Kuvassa olevat algoritmit ovat tietovuoparadigman vastaisia, sillä särmissä oleva tieto voi esimerkin mukaan muuttua. Muuttujien määrittely on täten rajoitettava yhteen kertaan, jotta paradigma toimisi joka suorituskerralla samalla tavalla.

Tietovuoparadigman erikoispiirteeksi voidaan Johnston & alin (2004) mukaan laskea sivuvaikutuksien puute, minkä voi huomata esimerkiksi siten, että tietovuokielellä toteutetussa funktiossa ei voi muokata kyseisen funktion syöttöparametreja. Muuttujien määrittelyn rajoittaminen vain yhteen kertaan edellä mainitun ominaisuuden kanssa aiheuttaa Ackermanin (1982) mukaan sen, että tietovuolla toteutetun kielen toistorakenne on hieman omituinen, koska joka iteraatiolla on esiteltävä uusi muuttuja. Whiting & pascoe (1994) mainitsevat että tämän kaltainen käytäntö tekee toistorakenteesta muistiriippumattoman ja mahdollistaa muuttujan käyttämisen toistorakenteen ulkopuolella samanaikaisesti.

Tietovuoparadigmassa laskenta on Ackeramin (1982) mukaan paikallista, millä hän tarkoittaa sitä, että samaa muuttujaa voidaan muuttaa eri funktioissa vaikuttamatta kyseisten funktioiden toimintaan. Toisin sanoen muuttujien näkyvyysalue rajoittuu yksittäisen funktion sisälle, minkä takia yksittäistä muuttujaa muuttamalla ei vaikuteta koko ohjelman toimintaan.

Ackerman (1982) kertoo myös, että tietovuokielisillä funktioilla ei ole kykyä muistaa aikaisempia syötteitään. Ackerman (1982) esittää esimerkin funktiosta, jonka puhdas toteuttaminen tietovuokiellellä on vähintäänkin haasteellista:



Kuva 5: Ackermanin esittelemä funktio

Iteratiivisella kielellä tämän kaltaisen funktion kirjoittaminen on triviaalia, sillä funktio voi säilyttää edellisten syötteiden summan, jota päivitetään aina funktiota kutsuessa. Tietovuokielellä kuvan mukaiset ongelmat ratkaistaan datavirtoja käyttämällä, joita koskevat omat rajoitteensa.

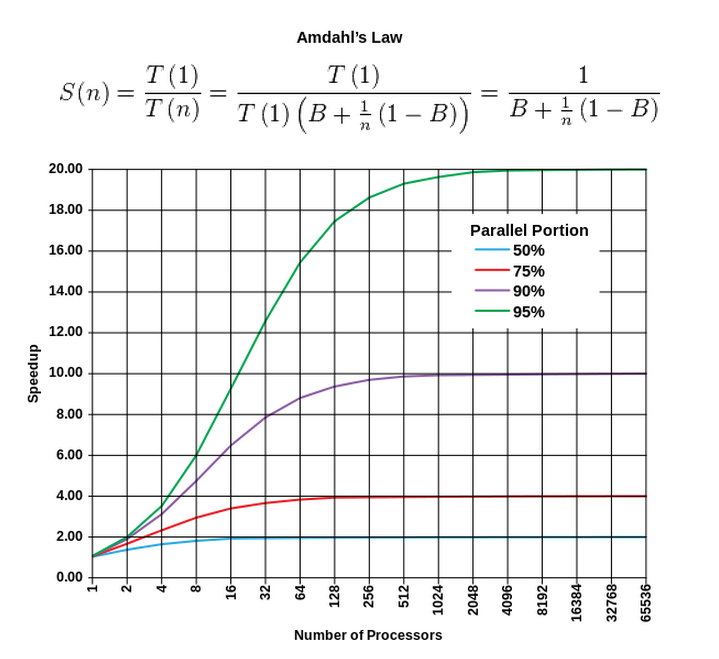
## Tietovuokielen vahvuudet ja heikkoudet

### Vahvuudet

Sousa (2012) kertoo että tietovuoparadigman avulla on mahdollista toteuttaa ohjelmointikieli, joka on laadultaan visuaalinen. Yksi tällainen kieli on LabView, joka on Marttila-Kontion (2011) mukaan saavuttanut suurta suosiota tutkijoiden keskuudessa. Visuaaliset kielet mielletään tekstuaalisia ohjelmointikieliä helpommiksi ymmärtää, sillä ohjelmoijan ei tarvitse keskittyä kieliopillisesti täydellisen koodin tuottamiseen. Ohjelmointirakenteet tulevat visuaalisessa ohjelmoinnissa yleensä valmiina ratkaisuina, joten ohjelmoijan ei tarvitse miettiä toistorakenteiden kaarisulkujen paikkaa tai for-silmukan syntaksia.

Helposti omaksuttava syntaksi on Sousan (2012) mukaan myös loppukäyttäjää helpottava ominaisuus, sillä koodia ymmärtävä käyttäjä voi korjata itse järjestelmässä esiintyviä vikoja. 1 lauseen kappale!

Suurin syy tietovuoparadigman käytölle on Sousan (2012) mukaan rinnakkaislaskennan määrä von Neumann malliin verrattuna. Hill & Marty (2008) mainitsevat, että potentiaalinen nopeusero voidaan laskea alla olevan Amdahli lain avulla, jonka mukaan suoritus kasvaa kertoimella S, jossa n on ytimien määrä ja B algoritmin rinnakkaistettava osuus.

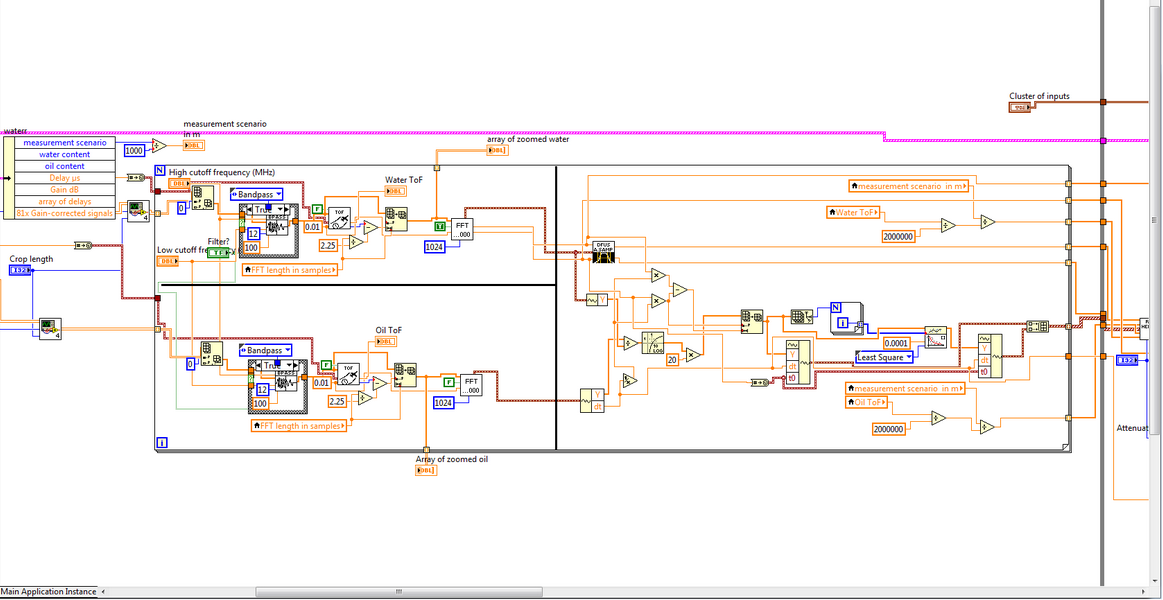


Kuva 6: Amdahlin laki

Kuvasta nähdään myös, että rinnakkaisuuden edut ovat pienet, mikäli algoritmia ei voida jakaa rinnakkain laskettaviin osiin. Tietovuoparadigma saavuttaa täten selvästi paremman prosessointinopeuden von Neumann malliin verrattuna, sillä rinnakkaisuutta voidaan hyödyntää paljon matalammalla tasolla.

### Heikkoudet

Sousan (2012) mukaan visuaaliset tietovuokielet, ovat etu niin ohjelmoijille kuin loppukäyttäjillekin, mutta niihin kohdistuu myös uniikkeja ongelmia joita ei tekstuaalisissa kielissä nähdä. Visuaalinen representaatio voi olla mittakaavaltaan ylitsepääsemätön, mikäli ohjelmakaavio on laajuudeltaan niin suuri, että se ei mahdu yhdelle näytölle. Ohjelmoijille tuttu tapa kommentoida koodia on myös hankalaa visuaalisessa ympäristössä, sillä tietovuolla ei ole samanlaista imperatiivista rakennetta kuin tektuaalisissa kielissä. Esimerkki vaikeasta tulkittavasta ohjelmakoodista on nähtävissä alla olevasta kuvasta.

Kuva 7: vaikeasti tulkittavaa koodia

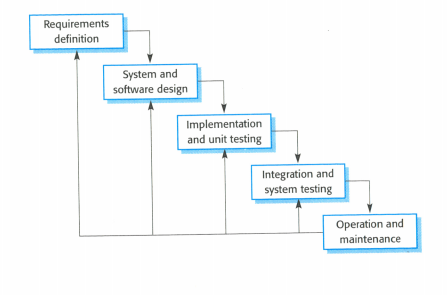
Vaikeasti tulkittava koodi ei välttämättä sinällään ole vakava asia, mutta se yhdistettynä dokumentoinnin vaikeuteen aiheuttaa suuria vaikeuksia sovelluksen ylläpidossa. Mikäli ainoa koodista selvää saava yksilö on koodin koodannut ohjelmoija, niin voidaan sanoa että sovellusta ei voida ylläpitää. Tämä tekee varsinkin ohjelmointivirheiden etsimisestä vaikeaa, mikä on Sousan (2012) mukaan hankalaa tietovuokielellä, koska jokaista rinnakkain tapahtuvaa tapahtumaa on pystyttävä seuraamaan samanaikaisesti. Ongelmien välttämiseksi perinteisen paperidokumentaation määrää on lisättävä.

Marttila-Kontio (2011) mainitsee, että tietovuolla ei ole sivuvaikutuksia, mikä helpottaa rinnakkaisuuden implementoimisessa, mutta vaikeuttaa samalla sovelluksen käyttäytymisen muokkaamista. Esimerkiksi von Neumann kielellä on mahdollista muuttaa globaalia tai staattista muuttujaa siten että ohjelman suoritus muuttuu perustavanlaatuisella tavalla.

# Vesiputousmalli

Boehm (1988) mukaan ohjelmistotuotanto voidaan nähdä joukkona erilaisia vaiheita jotka alkavat suunnittelusta ja päättyvät ylläpitoon. Vesiputousmalli on ohjelmistokehityksen kannalta katsottuna vanha, sillä sen on saanut alkunsa 70-luvulla. Huo & Al (2004) mainitsevat, että vesiputousmallia on tästä huolimatta käytetty menestyneesti pienissä ja suurissa projekteissa.

Pitkästä historiastaan huolimatta vesiputousmallia on Huo & al (2004) mukaan kritisoitu sen joustamattomuudesta, koska sitä noudattamalla on vaikea sopeutua muuttuviin tilanteisiin. Esimerkiksi mikäli projektin vaatimuksia muutettaisiin toteutus vaiheessa, niin tämä johtaisi projektin uudelleen aloittamiseen. Tässä kappaleessa tulen esittelemään vesiputousmallin vaiheet käyttämällä esimerkkinä toteutettua tapaustutkimusta, joka toteutettiin sovelletun fysiikan laitokselle Itä-Suomen yliopistossa.



Kuva 8: Vesiputousmalli

## Vaatimusten määrittely

Huo & al (2004) mainitsevat, että vesiputousmalli etenee lineaarisesti suunnittelusta valmiiseen tuotteeseen. Täten ensimmäisen vaihe on lopputuloksen kannalta tärkein vaihe, sillä mikäli vaatimukset on määritelty väärin, niin lopputuote ei vastaa asiakkaan odotuksia. Tarkkojen teknisten vaatimusten laatiminen on Sommervillen (2000: 98-99) mukaan hankalaa, sillä asiakkaalla on yleensä tapana kuvailla haluamansa tuote hyvin abstraktilla tasolla.

Sommerville (2000: 98-99) jakaa vaatimukset kolmeen kategoriaan:

1. Käyttäjävaatimukset
2. Järjestelmävaatimukset
3. Spesifikaatio

Jokainen vaatimuksen taso on suunniteltu palvelemaan tiettyä segmenttiä. Esimerkiksi käyttäjävaatimukset ovat selvempiä asiakkaalle kuin ohjelmoijalla, ja spesifikaatio on hyödyllisempi ohjelmoijalle kuin asiakkaalle. Alla olevasta spesifikaatiosta on nähtävillä BBC-ryhmälle toteutetun järjestelmän vaatimukset.

|  |  |  |  |
| --- | --- | --- | --- |
| **Requirement** | **Comments** | **Priority** | **Date Rvwd** |
| **New system requirements** |  |  | **6.5.2014** |
| Dynamic video feed (arthoscopy, otc etc..). ONLY ONE SCREEN! | Voice commands activate the videostream. This means that only relevant information is presented to the user | 1 |  |
| Video is recorded on demand | Voice command "record" | 1 |  |
| ICRS score is calculated from the video feed |  | 1 |  |
| Generates a color map from the measured area | Uses ICRS. It might be necessary to customise the map. Map updates automatically based on the measurement data. | 1 |  |
| Voice commands | .NET | 1 |  |
| Error correction | Escape command for the user. | 1 |  |
| Tooltips for the surgeon | sound and/or text | 3 |  |
| Videos linked to colormap | After the measurements are made the user can see the recordings output data via the generated color map | 4 |  |
| Saves audio | Audio is saved when the video is saved | 4 |  |
| Summary option | "Finding" command. User can add a summary as audio on the measured part. | 3 |  |
| **Old system requirements** |  | **1** | **16.4.2014** |
| Calculates the cartilage thickness |  |  |  |
| Calculates cartilage stiffness |  |  |  |
| Calculates the change in thickness when pressure is applied on said cartilage |  |  |  |
| Calculates sound reflection |  |  |  |

Taulukko 1: Vaatimukset

BBC-ryhmälle toteutetun järjestelmän käyttäjävaatimusten teossa auttoi ortopedi Antti Joukainen, jonka työskentelyä seurattiin sairaala Tarinassa. Kahden polvileikkauksen seuraamisen tuloksena muodostettiin alustavat järjestelmävaatimukset.

Sommerville (2000: 100-105) mukaan spesifikaation vaatimukset jaetaan yleensä toiminnallisiin ja ei-toiminnallisiin vaatimuksiin. Tapaustutkimuksen järjestelmän toiminnallisiin vaatimuksiin voidaan laskea esimerkiksi videon tallennus, ja ei toiminnallisiin tuki värisokeille. Toiminnalliset vaatimukset määritellään siten, että ne vaikuttavat jotenkin olennaisella tavalla järjestelmän toimintaa ja kuvaavat yleensä asiakkaan vaatimia toimintoja. Ei-toiminnaliset vaatimukset sen sijaan vastaavat laadusta, standardeista ja ulkoisista vaatimuksista.

## Suunnittelu

Suunnittelu vaiheessa luodaan haluttavan tuotteen perustavanlaatuinen runko, jota jalostetaan yksityiskohtaisemmaksi suunnitelmaksi suunnittelu vaiheen edetessä (Sommerville 2000). Toteutetun projektin aikana päädyin käyttämään ER- ja käyttötapaus kaavioita, sillä ne olivat mielestäni asiakkaalle helpoiten ymmärrettävissä.

### Käyttötapauskaavio

Kuvasta 9 voidaan nähdä käyttötapauskaavio, joka kuvaa hyvin yleisellä tasolla ohjelman toimintaa. Ortopedi suorittaa ohjelmalla mittauksen, jota on mahdollista seurata tallenteelta mittausten jälkeen. Käyttötapauskaavio ei välttämättä ole paras mahdollinen dokumentti tietovuoparadigman kuvaamiseen, koska Sommerville (2000) mainitsee, että sitä käytetään yleensä oliopohjaisen järjestelmän esittämiseen. Käyttötapauskaavio kuitenkin näyttävät myös miten eri tahot käyttävät järjestelmää, joten sen käyttäminen on perusteltua.

Kuva 9: Käyttötapauskaavio

### Tietovuokaavio

Tietovuokaavion käyttäminen tietovuoparadigmaa käyttävän järjestelmän kuvaamisessa on luonnollista, sillä tietovuokieli on vain yksityiskohtaisempi esitys tietovuokaaviosta. Suunnitteluvaiheessa tietovuokaavion avulla voidaan kuvata miten tietoa käsitellään järjestelmän eri osissa. Kuvasta 10 voidaan nähdä alustavanlaatuinen suunnitelma BBC-ryhmälle toteutetun järjestelmän toiminnasta. Kaaviossa nähdään eri virtuaali-instrumenttien välinen interaktio, jonka toteuttamiseen on nähtävä tavallista enemmän vaivaa, sillä LabView kielellä eri instrumenttien välinen tiedonsiirto on vaivalloista. Jokaiselle instrumentille on luonnollisesti tehtävä oma tietovuokaavionsa, mikäli niitä halutaan tutkia lähdekoodia korkeammalla tasolla.

Kuva 10: Tietovuokaavio

Tietovuokaavioparadigmaa käyttävän järjestelmän suunnittelu on järkevä aloittaa eri instrumenttien välisen kommunikaation kuvaamisesta, sillä se kuvaa hyvin korkean tason toimintaa. Sommerville (2000) mainitseekin, että tietovuodokumentaatiota tulee suunnitella ylhäältä alas periaatteella, jossa suunnitellaan ensin korkeamman abstraktio tason tiedonsiirto ennen alifunktioiden toiminnan kuvaamista.

## Toteutus

Sommerville (2000) tiivistää toteutuksen seuraavanlaisesti: ”Tässä vaiheessa toteutettu suunnitelma toteutetaan ohjelmalla tai joukolla ohjelmia. Kukin ohjelma testataan jotta voidaan varmistaa, että ne täyttävät niille määrätyn spesifikaation”.

### LabView ohjelmointi

Elliot & al (2007) mukaan LabView on graafinen ohjelmointikieli, joka on alun perin julkaistu vuonna -1986 tiedon hankinta ja automaatio tarpeisiin. labView kielen rakenne perustuu särmiin ja solmuihin, jotka asetetaan lohkokaaviolle. Lohkokaaviolle sijoitettua ohjelmaa kutsutaan virtuaali-instrumentiksi, joka koostuu Elliot & al (2007) mukaan kolmesta pääkomponentista:

1. Etupaneeli, joka sisältää kontrollit ja indikaattorit, jotka päivittyvät ajon aikana.
2. Lohkokaavio, jossa lähdekoodi sijaitsee.
3. Liitinpaneeli, jonka avulla on mahdollista määrittää ali-instrumenttien syöttö ja palautusparametrit.

Ohjelmointi koostuu valmiina saatavilla olevien käskyjen lisäämisestä funktiopaletista lohkokaaviolle. Käskyt liitetään toisiinsa särmillä, jotka välittävät tietoa edelleen seuraaville käskyille tietovuoparadigman mukaisesti. Ohjelmointirakenteet kuten while ja for silmukka esiintyvät lohkokaaviolla kehyksinä joidenka sisällä olevat toiminnot suoritetaan rakenteen määräämällä tavalla.

Elliot & al (2007) mainitsevat, että LabView toimii monessa mielessä tavallisen ohjelmoinkielen tavoin, sillä se tukee perinteisiä tietotyyppejä, sekä pystyy käyttämään 3-osapuolen ohjelmistokomponenttikirjastoja. Lohkokaavio esityksessä eri tietotyypit havainnollistetaan eri värein ja muodoin, jotta esityksen ymmärtäminen helpottuisi.

Virtuaali-instrumentin modulaarisuutta voidaan lisätä jakamalla instrumentti useisiin ali-instrumentteihin, jotka näyttäytyvät tavallisen käskyjen mukaisesti lohkopaneelissa. Kullakin ali-instrumentilla on oma ikoninsa ja särmillä liitettävät syöttö ja tulostus portit

Lohkokaaviolle asetetut kontrollit ja indikaattorit esiintyvät etupaneelissa käyttöjärjestelmäelementteinä. Etupaneelille asennetut käyttöjärjestelmäelementit ilmestyvät taasen lohkokaaviolle joko indikaattoreina tai kontrolleina. Tämänkaltainen toiminta tekee käyttöjärjestelmän implementoinnista automaattista. . Ali-instrumentit sisältävät myös etupaneelin, jonka avulla on mahdollista simuloida eri syöttöparametrien toimintaa, mikä helpottaa testaamista.

Kaupallisessa käytetään yleisesti jotain versionhallinta järjestelmää, koska useamman ohjelmoijan samanaikaista työpanosta on muuten vaikea koordinoida. Elliot & al (2007) mainitsevat että LabView kielelle on olemassa versionhallintajärjestelmä Visual SourceSafe, joka mahdollistaa suuremmankin mittakaavan projektin toteuttamisen.

## Testaus

Sommervillen (2000) mukaan testausvaiheessa eri ohjelmanpalaset integroidaan, jotta koko järjestelmää voidaan testata kokonaisuutena. Mikäli järjestelmä toimii spesifikaation mukaisesti, niin järjestelmä toimitetaan asiakkaalle.

## Ylläpito

Ylläpidon aikana korjataan testauksen aikana huomaamatta jääneitä virheitä ja parannetaan järjestelmän luotettavuutta ja suorituskykyä (Sommerville 2000).

## Tavoitteet

Tämän tutkielman tarkoituksena on tutkia vesiputousmallin soveltuvuutta tietovuoparadigmaa käyttäviin projekteihin. Tutkielmassa keskitytään tietovuoparadigman erikoispiirteiden esiintymiseen vesiputousmallin eri vaiheissa.

Tutkimuksen tutkimuskysymykset ovat:

1. Mitä tietovuoparadigman erikoispiirteitä on otettava huomioon suunnitellessa sitä käyttävää projektia?
2. Miten vesiputousmalli soveltuu tietovuoparadigmaan?
3. Mikä ohjelmistokehitysmalli soveltuu tietovuoparadigmalle parhaiten?

Kysymyksiin pyritään vastaamaan käyttämällä hyväksi aiheesta löytyvää tieteellistä kirjallisuutta, sekä toteuttamalla tietovuoparadigmaa käyttävä projekti. Projekti toteutetaan Itä-Suomen yliopistossa BBC-ryhmän valvonnan alaisena. Projektin teknisestä aspektista vastuussa on tohtori Isaac Afara.

# Materiaalit ja metodit

## LabView

LabView (Laboratory Virtual Instrumentation Engineering Workbench) on Nation lnstrumentsin kehittämä visuaalinen tietovuo-ohjelmointikieli. Projektia varten LabView valittiin, koska sen avulla on helppo implementoida spesifikaation vaatimia mittalaitteita haluttavaan järjestelmään.

Labview ohjelmointikieltä ei voida Marttika-Kontion (2011) mukaan pitää puhtaana tietovuokielenä, sillä se tarjoaa kontrollirakenteen, joka toimii imperatiivisesti. Kyseinen kontrollirakenne käyttö ei kuitenkaan ole hyvän ohjelmointitavan mukaista, vaan sen käyttö tulisi rajata tilanteisiin joissa tietovuon käyttäminen ei ole järkevää.

## Tapaustutkimus

Gagnon (2010) mukaan tapaustutkimuksen avulla on mahdollista saavuttaa tarkka yksityiskohtainen ymmärrys tutkittavasta aiheesta. Tapaustutkimusta voidaan käyttää uusien tieteellisten teorioiden kehittämiseen tai jo olemassa olevien teorioiden vahvistamiseen.

Tapaustutkimusta pidetään Gagnonin(2010) mukaan aikaa vievänä prosessina verrattuna muihin tutkimusmenetelmiin, koska siinä tuotetaan hyvin yksityiskohtaista tietoa paneutumalla tutkittavaan aiheeseen. Tämä tutkimusmetodi valittiin, koska maisterivaiheen harjoittelun aikana toteutettiin tietovuoparadigmalla projekti, joten tapaustutkimusta pystyttiin toteuttamaan työn ohessa.

# Uusi teoreettinen tulos

# Pohdintaa

## Yhteenveto

Coming soon

Viitteet

Aakkosjärjestys ja viitteet muutenkin kuntoon!

Johnston, W. M., Hanna, J. R., & Millar, R. J. (2004). Advances in dataflow programming languages. ACM Computing Surveys (CSUR), 36(1), 1-34

Iannucci, R. A. (1988). Toward a dataflow/von Neumann hybrid architecture (Vol. 16, No. 2, pp. 131-140). IEEE Computer Society Press.

Skillicorn, D. B., & Talia, D. (1998). Models and languages for parallel computation. ACM Computing Surveys (CSUR), 30(2), 123–169.

Ackerman, W. B. (1982). Data flow languages. Computer, 15(2), 15-25.

Whiting, P. G., & Pascoe, R. S. (1994). A history of data-flow languages. Annals of the History of Computing, IEEE, 16(4), 38-59.

Sousa, T. B. (2012). Dataflow Programming Concept, Languages and Applications. In Doctoral Symposium on Informatics Engineering.

Su, H. M., & Yew, P. C. (1989). On data synchronization for multiprocessors(Vol. 17, No. 3, pp. 416-423). ACM.

James Reinders, Intel Threading Building Blocks: Outfitting C++ for Multi-core Processor Parallelism. Publisher: O’Reilly Media, July 2007.

Maija Marttila-Kontio (2011), Visual flow languages challenges and opportunities.

Sun, X. H., & Chen, Y. (2010). Reevaluating Amdahl’s law in the multicore era.Journal of Parallel and Distributed Computing, 70(2), 183-188.

Hill, M. D., & Marty, M. R. (2008). Amdahl's law in the multicore era. *Computer*, (7), 33-38.

Boehm, B. W. (1988). A spiral model of software development and enhancement. *Computer*, *21*(5), 61-72.

Huo, M., Verner, J., Zhu, L., & Babar, M. A. (2004, September). Software quality and agile methods. In Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International (pp. 520-525). IEEE.

Ian Sommerville (2000) Software Engineering

Yves-C Gagnon (2010) The Case study as Research method A Practical Handbook

Elliott, C., Vijayakumar, V., Zink, W., & Hansen, R. (2007). National instruments LabVIEW: a programming environment for laboratory automation and measurement. *Journal of the Association for Laboratory Automation*, *12*(1), 17-24.