

**Aleksi Riihiaho**

CAD-ohjelmaan liitetyn kehärakenteiden lujuuslaskentamoduulin käyttöliittymän suunnittelu ja toteutus

Diplomityö

|  |  |
| --- | --- |
| Tarkastaja: professori Reijo Kouhia Tarkastaja ja aihe hyväksytty  Teknisten tieteiden tiedekuntaneuvoston kokouksessa x. xkuuta 2016 |  |
|  |  |

TIIVISTELMÄ

**ALEKSI RIIHIAHO**: CAD-ohjelmaan liitetyn kehärakenteiden lujuuslaskentamoduulin käyttöliittymän suunnittelu ja toteutus

Tampereen teknillinen yliopisto

Diplomityö, XX sivua, YY liitesivua

Xkuu 2016

Konetekniikan diplomi-insinöörin tutkinto-ohjelma

Pääaine: Koneiden ja rakenteiden analysointi

Tarkastaja: professori Reijo Kouhia

Avainsanat: FEM

Tiivistelmä

ABSTRACT

**ALEKSI RIIHIAHO**: Heading

Tampere University of Technology

Master of Science Thesis, XX pages, YY Appendix pages

X 2016

Master’s Degree Programme in Mechanical Engineering

Major: Analysis of Machines and Structures

Examiner: Professor Reijo Kouhia

Keywords: FEM

The basics in English in Master’s theses.

ALKUSANAT

Tämä dokumenttipohja on laadittu TTY:n opinnäytetyöohjeen vuoden 2014 version mukaan edellistä pohjaa muokkaamalla. Työryhmä haluaa kiittää kaikkia ohjeen päivitykseen osallistuneita.

Alkusanoissa esitetään opinnäytetyön tekemiseen liittyvät yleiset tiedot. Tapana on myös esittää kiitokset työn tekemiseen vaikuttaneille henkilöille ja yhteisöille. Alkusanat eivät kuulu arvioinnin piriin, mutta niissä ei silti ole sopivaa moittia tai kritisoida ketään. Alkusanojen pituus on enintään 1 sivu. Alkusanojen lopussa on päivämäärä, jonka jälkeen työhön ei ole enää tehty korjauksia.

Tampereella, x.x.2016

Aleksi Riihiaho

SISÄLLYSLUETTELO

[1. Johdanto 1](#_Toc447467489)

[2. Kehärakenteiden elementtimenetelmä 2](#_Toc447467490)

[2.1 Tekninen taivutusteoria 2](#_Toc447467491)

[2.2 Laskentamalli 3](#_Toc447467492)

[2.2.1 Elementit 4](#_Toc447467493)

[2.2.2 Koordinaatistot 4](#_Toc447467494)

[2.2.3 Solmumittausjärjestelmän kierto 5](#_Toc447467495)

[2.2.4 Ratkaisukaavat 6](#_Toc447467496)

[2.3 Tietokoneavusteinen laskenta 7](#_Toc447467497)

[2.3.1 Aktiivisarakeratkaisija 7](#_Toc447467498)

[3. Käyttöliittymäsuunnittelun periaatteita 8](#_Toc447467499)

[3.1 Käyttäjäkokemus 8](#_Toc447467500)

[3.2 Käyttäjätarpeiden kartoitus 9](#_Toc447467501)

[3.3 Käyttäjäkeskeinen suunnittelu 9](#_Toc447467502)

[3.3.1 Normanin suunnitteluperiaatteet 9](#_Toc447467503)

[3.3.2 Nielsenin heuristiikat 11](#_Toc447467504)

[3.3.3 Shneidermanin kahdeksan kultaista sääntöä 11](#_Toc447467505)

[4. Graafisen käyttöliittymän ohjelmointi Windows-ympäristössä 12](#_Toc447467506)

[4.1 Olio-ohjelmointi 12](#_Toc447467507)

[4.1.1 Oliot ja luokat 12](#_Toc447467508)

[4.1.2 Periytyminen 14](#_Toc447467509)

[4.2 Suunnittelumallit 15](#_Toc447467510)

[4.2.1 MVC-arkkitehtuuri 15](#_Toc447467511)

[4.2.2 Visitor pattern -malli 15](#_Toc447467512)

[4.3 Windows Ribbon Framework 15](#_Toc447467513)

[4.4 OpenGL-piirtotyökalut 15](#_Toc447467514)

[5. Vaatimukset laskentamoduulille 16](#_Toc447467515)

[5.1 Asiakasvaatimukset 17](#_Toc447467516)

[5.2 STAFRA 19](#_Toc447467517)

[5.3 Avaruuspalkkielementti 19](#_Toc447467518)

[5.4 Yhteensopivuus Vertexin eri ohjelmistojen kanssa 20](#_Toc447467519)

[6. Käyttöliittymän toteutus 21](#_Toc447467520)

[6.1 Tietorakenne 21](#_Toc447467521)

[6.2 Puunäkymä 21](#_Toc447467522)

[6.3 Ribbon-valikko 21](#_Toc447467523)

[6.4 Käyttötapaukset 21](#_Toc447467524)

[6.4.1 Tutkimusten luominen ja poistaminen 21](#_Toc447467525)

[6.4.2 Tuentojen ja kuormitusten asettaminen 21](#_Toc447467526)

[6.4.3 Tulosten tarkastelu 21](#_Toc447467527)

[7. Jatkokehitystarpeet 22](#_Toc447467528)

[8. Yhteenveto 23](#_Toc447467529)

[Lähteet 24](#_Toc447467530)

LIITE A:

LYHENTEET JA MERKINNÄT

CC-lisenssi Creative Commons -lisenssi

LaTeX ladontajärjestelmä tieteelliseen kirjoittamiseen

SI-järjestelmä ransk. Système international d’unités, kansainvälinen mittayksikköjärjestelmä

TTY Tampereen teknillinen yliopisto

URL engl. Uniform Resource Locator, verkkosivun osoite

*a* kiihtyvyys

**F**voima

*m* massa

Työssä käytetyt lyhenteet ja merkinnät määritellään ja selitetään kootusti aakkosjärjestyksessä työn alussa ja kun ne esiintyvät tekstissä ensimmäisen kerran Lyhenteiden kanssa käytetään tällöin sulkeita. Selitetekstin tyyli on tässä *Symbol description*. Tämän sivun lopussa on *Section Break*, jotta sivunumerointi menee oikein. Lisäksi johdannon yläotsakkeen (header) asetus *Link to Previous* on pois päältä, ja lisäksi sivunumeron muotoilusta on valittu *Start at 1* (eikä *Continue*).

.

# Johdanto

Vertex

Vertex-ohjelmistot

Miksi tehdään sisäänrakennettu FEM-laskentaosio Vertexin ohjelmistoihin?

Käyttöliittymän suunnittelu ja toteutus FEM-laskentaosiolle

Minkälaisia ongelmia on tarkoitus laskea? Asiakastarve

Vertexin aiemmat FEM-laskentamoduulit, Numerola

# Kehärakenteiden elementtimenetelmä

Elementtimenetelmä (*Finite element method*, FEM) on matemaattinen menetelmä differentiaali- ja osittaisdifferentiaaliyhtälöiden reuna-arvotehtävien numeeriseen ratkaisemiseen. Elementtimenetelmällä voidaan muuntaa osittaisdifferentiaaliyhtälö algebralliseksi yhtälösysteemiksi, joka on helposti ratkaistavissa. Muunnos tapahtuu diskretoimalla kenttäfunktion äärettömän suuri tuntematon arvojoukko äärelliseksi määräksi solmuarvoja. Diskretoinnin jälkeen solmuarvojen ratkaisu saadaan yhtälösysteemistä. Eri tekniikan aloilla tämänlaisia tehtäviä joudutaan ratkaisemaan paljon, joten elementtimenetelmä onkin nykyään levinnyt muun muassa mekaniikan, termodynamiikan, virtausmekaniikan, murtumismekaniikan, akustiikan ja sähkötekniikan aloille. [7][12]

EPÄLIN, LIN?

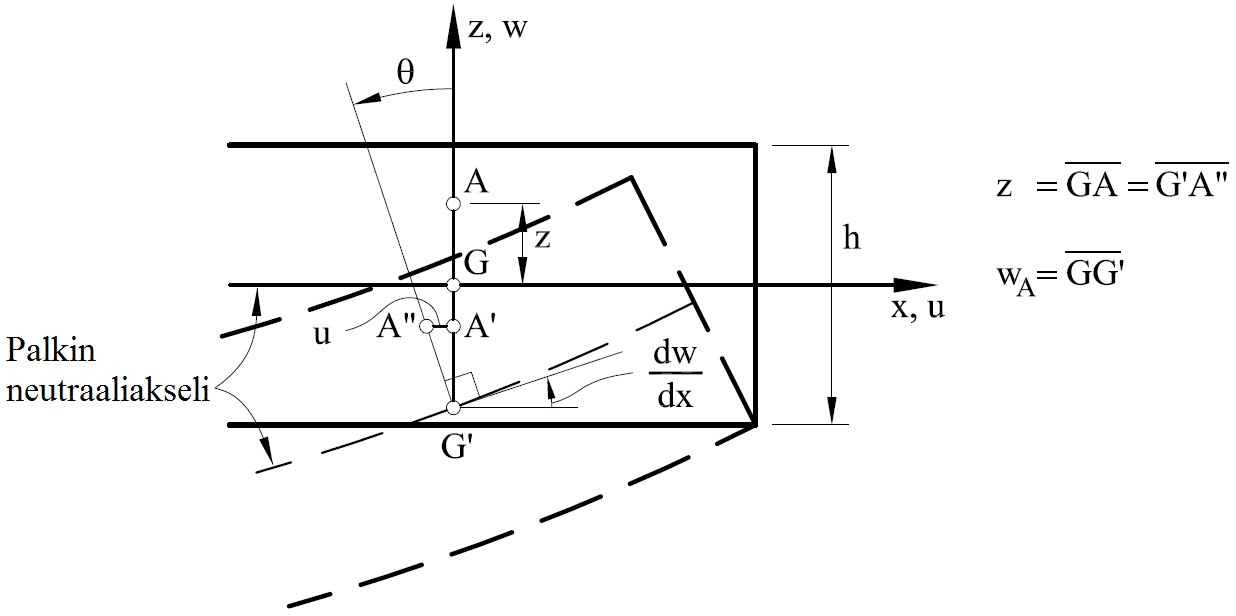
Kehärakenteiden elementtimenetelmässä rajoitutaan ratkaisemaan yksiulotteisista perusrakenneosista koostuvia rakenteita. Tällaisia rakenneosia ovat esimerkiksi sauvat, palkit ja pilarit. Tämän johdosta kehärakenteiden elementtimenetelmä voidaan perustaa suoraan rakenteiden mekaniikan analyyttisten ratkaisumenetelmien teorialle. Tämä tarkoittaa myös sitä, että kehärakenteiden elementtimenetelmä johtaa teknisen taivutusteorian puitteissa tarkkaan ratkaisuun diskretoidun mallin solmukohdissa. [12]

Tämä luku käsittelee kehärakenteiden elementtimenetelmään liittyvää teoriaa. Tämän teorian tietämystä tarvitaan tässä työssä käsiteltävän lujuuslaskentamoduulin käyttämisessä.

## Tekninen taivutusteoria

Teknistä taivutusteoriaa kutsutaan joskus myös klassiseksi palkkiteoriaksi tai sen kehittäjiensä mukaan Euler-Bernoullin palkkiteoriaksi. Se on yksinkertaisin palkin käyttäytymistä kuvaava malli. Teknisen taivutusteorian perusoletukset ovat seuraavat: [7]

1. Palkin poikkileikkaus säilyy tasona taivutuksessa.
2. Palkin poikkileikkaus ei veny taivutuksessa.
3. Palkin poikittainen leikkausmuodonmuutos on merkityksetön.



1. Teknisen taivutusteorian mukainen palkki. Kuva muokattu lähteestä [10].

Kuva 1 ja perusoletusten mukaisesti voidaan kirjoittaa siirtymille *u* ja *w* lausekkeet

ja (1)

, (2)

joissa on palkin neutraaliakselin pisteen *x* Z-suuntainen siirtymäkomponentti ja *θ* palkin poikkileikkauksen kiertymäkulma. Tekninen taivutusteoria olettaa palkin taipumat pieniksi, jolloin voidaan merkitä ja . Tällöin siirtymien likimääräisiksi lausekkeiksi saadaan

ja (3)

. (4)

Kolmannesta oletuksesta seuraa, että palkin poikkileikkauksen kiertymäkulma on sama kuin palkin neutraaliakselin kulmakerroin, jolloin siirtymän *u* lausekkeen voi kirjoittaa seuraavasti:

. (5)

## Laskentamalli

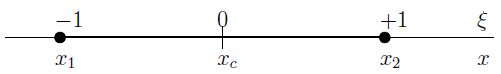
Rakenteiden mekaniikassa elementtimenetelmän perusajatuksena on jatkuvan materiaalikontinuumin diskretisoiminen äärelliseen määrään elementtejä, jotka liittyvät toisiinsa niin sanotuissa solmukohdissa. Rakenteen muodostamaa elementtijoukkoa kutsutaan laskentamalliksi. [12]

### Elementit

Elementtityypin määrittelyyn vaaditaan seuraavat asiat: [7]

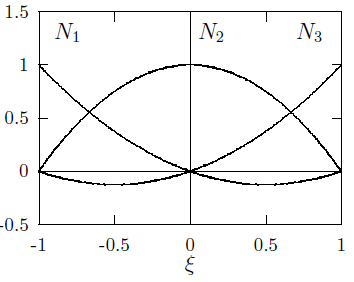
* Elementin geometria
* Muotofunktiot
* Vapausasteiden määrittely

Elementin geometria voi olla lähes mikä tahansa geometrinen kuvio tai kappale, esimerkiksi jana, nelikulmio tai tetraedri. Kehärakenteiden elementtimenetelmässä elementtien geometria on yksiulotteinen, jolloin rajoitutaan janasta tai suljetusta käyrästä muodostuviin elementteihin. Kuva 2 on esitetty yksinkertainen janaelementti, jonka paikallinen koordinaatti määritellään välillä .



1. Yksiulotteinen janaelementti. [7]

Elementtimenetelmällä muodostetun yhtälöryhmän ratkaisu antaa tuloksia vain laskentamallin solmuissa. Jotta saataisiin tuloksia myös solmujen välillä, eli elementeillä, täytyy elementeille olla määritelty muotofunktiot. Muotofunktiot interpoloivat haluttua suuretta elementin alueella. Tavallisesti muotofunktioina käytetään yksinkertaisia polynomeja. on esitetty paraboliset Lagrangen muotofunktiot, jotka ovat esimerkki yleisesti elementtimenetelmässä käytetyistä muotofunktioista. [7]

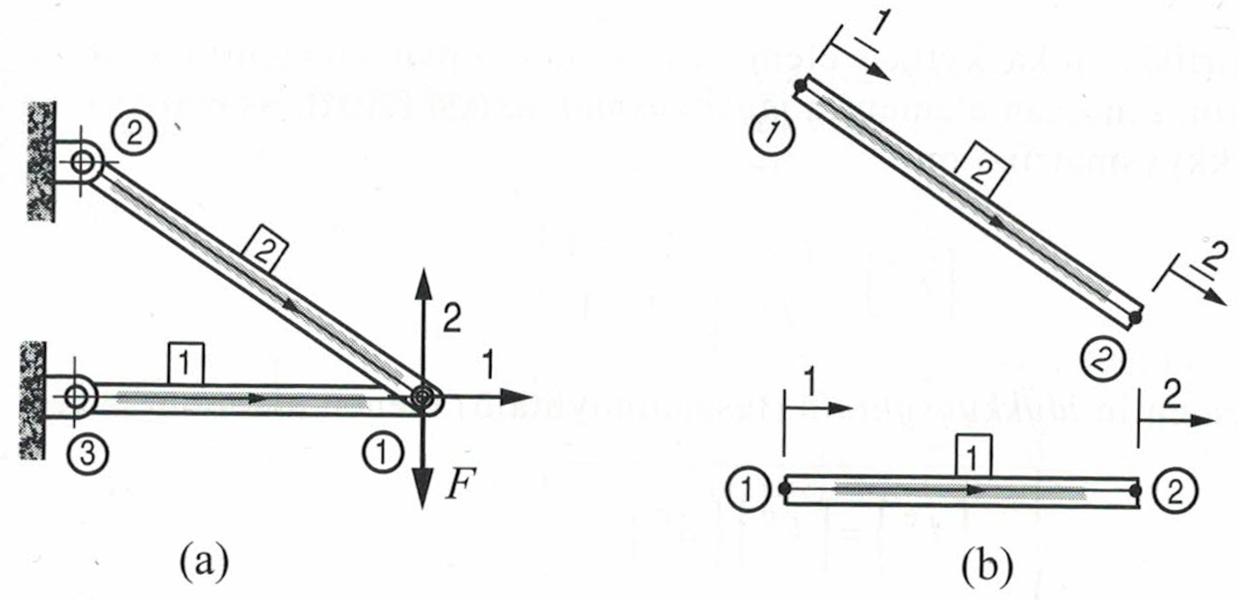


1. Paraboliset Lagrangen muotofunktiot. Kuva muokattu lähteestä [7].

### Koordinaatistot

Elementtien väliset solmut voidaan luokitella kahteen kategoriaan: paikallissolmuihin ja rakennesolmuihin. Paikallissolmuilla tarkoitetaan yksittäisen elementin päissä tai nurkissa olevia solmuja, joilla elementit liittyvät toisiin elementteihin. Rakennesolmuilla taas tarkoitetaan koko laskentamallin solmuja. Rakennesolmut yksilöidään yleensä juoksevalla numeroinnilla, mutta numeroinnin järjestyksellä ei ole laskennan kannalta väliä. Elementin suunnistus määrää sen alku- ja loppupään. Yksittäisen elementin paikallissolmut numeroidaan järjestyksessä pienimmästä suurimpaan alkupäästä lähtien.

Paikalliskoordinaatistolla tarkoitetaan yksittäisen elementin sisäistä koordinaatistoa, jossa X-akseli osoittaa palkin suuntaan alkupäästä loppupäähän päin. Y- ja Z-akselit valitaan yleensä palkin poikkileikkauksen pääsuuntien mukaan. Koko rakenteelle yhteiselle koordinaatistolle käytetään nimitystä rakennekoordinaatisto. Solmumittausjärjestelmällä tarkoitetaan kussakin tilanteessa voimassa olevaa koordinaatistoa, jossa solmumittaus suoritetaan.



1. Laskentamalli (a) ja yksittäiset elementit (b). Kuva muokattu lähteestä [12].

Kuva 4 vasemmalla on esitetty yksinkertaisen sauvarakenteen laskentamalli. Laskentamallin solmumittaus rakennekoordinaatistossa on merkitty nuolilla 1 ja 2. Oikean puoleisessa kuvassa on laskentamallin yksittäiset elementit ja niiden paikalliset solmumittausjärjestelmät. Laskentamallia luodessa yksittäisten elementtien paikalliset solmumittausjärjestelmät tulee muuttaa rakenteen solmumittausjärjestelmän mukaiseksi. Muutos toteutetaan koordinaatistoa kiertämällä.

### Solmumittausjärjestelmän kierto

Palkkielementtien solmumittausjärjestelmää joudutaan usein kiertämään, koska palkit voivat osoittaa rakennekoordinaatistossa mielivaltaiseen suuntaan. Jotta sijoittelusummaus voidaan suorittaa, tulee koko rakenteessa olevien yksittäisten palkkielementtien solmumittausten olla samansuuntaiset. Koordinaatiston kierto elementin paikalliskoordinaatistosta rakennekoordinaatistoon ja toisin päin tehdään niin sanotulla koordinaatiston kiertomatriisilla. Kiertomatriisi kootaan siten, että paikalliskoordinaatiston kantavektoreilla **i**, **j** ja **k** lausutun vektorin **v** sekä rakennekoordinaatiston kantavektoreilla **i**, **j** ja **k** lausutun vektorin **v** välillä on lineaarinen yhteys:

. (6)

Esimerkiksi koordinaatistoa kulman *θ* verran X-akselin ympäri vastapäivään kääntävä kiertomatriisi *T* voidaan kirjoittaa seuraavasti:

. (7)

### Ratkaisukaavat

Yksittäisen elementin tasapainoyhtälöistä on johdettavissa kerroinmatriisi, joka kytkee elementin solmuille kohdistuvat voimat vastaaviksi solmusiirtymiksi. Tätä kerroinmatriisia kutsutaan elementin jäykkyysmatriisiksi. Koko laskentamallin jäykkyyttä kuvaa rakenteen jäykkyysmatriisi. Se kootaan laskentamallin muodostavien elementtien yksittäisistä jäykkyysmatriiseista sijoittelusummauksella. Sijoittelusummauksessa yksittäisten elementtien jäykkyysmatriisien alkiot sijoitetaan rakenteen jäykkyysmatriisin vastaaville paikoille rakenteen solmunumeroinnin mukaisesti. Eri elementeistä tulevat samaan rakennesolmuun liittyvät alkiot summataan yhteen. Sijoittelusummaus edellyttää, että yksittäisten elementtien paikallinen solmumittausjärjestelmä on yhdensuuntainen rakenteen solmumittausjärjestelmän kanssa. Sijoittelusummaus on helposti ohjelmoitavissa ja tehokas tapa koota rakenteen jäykkyysmatriisi tietokonelaskennan kannalta. [12]

Rakenteen laskentamallin kuormitukset kootaan rakennesolmujen kuormitusvektoriin

, (8)

jossa on suoraan rakennesolmuille annettu ulkoinen solmukuormitusvektori ja rakennesolmujen ekvivalenttinen solmukuormitusvektori. Rakenteeseen kohdistuva ulkoinen kuormitus jakaantuu usein elementin alueelle, eikä suoraan solmuille, jolloin nämä kenttäkuormitukset täytyy muuntaa solmuvoimiksi. Solmuvoimiksi muunnetut kenttäkuormitukset muodostavat ekvivalenttisen solmukuormitusvektorin. Yksittäisten elementtien ekvivalenttisista solmukuormitusvektoreista saadaan koottua rakennesolmujen ekvivalenttinen solmukuormitusvektori sijoittelusummauksella.

Rakenteen laskentamallin siirtymät saadaan ratkaistua rakennesolmujen tasapainoyhtälöstä

, (9)

jossa on rakenteen jäykkyysmatriisi ja rakennesolmujen siirtymävektori.

## Tietokoneavusteinen laskenta

Matriisien tallennus skyline-muotoon

### Aktiivisarakeratkaisija

Suurille matriiseille

# Käyttöliittymäsuunnittelun periaatteita

Asd

## Käyttäjäkokemus

Käyttäjäkokemuksella tarkoitetaan käyttäjän subjektiivista vaikutelmaa tuotteesta, joka syntyy käyttäjän ollessa vuorovaikutuksessa sen kanssa. Käyttäjäkokemus on siis tunnereaktioita, uskomuksia ja mieltymyksiä, jotka ilmenevät ennen tuotteen käyttöä, käytön aikana ja käytön jälkeen. Täsmällinen määritelmä käyttäjäkokemukselle löytyy ISO 9241-210 -standardista: ”Henkilön havainnot ja vasteet, jotka ovat seurausta tuotteen, järjestelmän tai palvelun käytöstä ja/tai ennakoidusta käytöstä”. [15]

Käyttäjäkokemukseen voidaan vaikuttaa luonnollisesti tuotteen käytettävyyttä parantamalla. Käytettävyyden kannalta oleellisinta on usein käyttöliittymän hyvä ja huolellinen suunnittelu. Myös käytettävyydelle on luotu standardin mukainen määritelmä (ISO 9241-11) ja se kuuluu näin: ”Tarkoituksenmukaisuus, tehokkuus ja tyytyväisyys, joilla tuotteen määritellyt käyttäjät saavuttavat määritellyt tavoitteet tietyissä käyttöympäristöissä”. Käytettävyyden lisäksi käyttäjäkokemukseen vaikuttavat myös tuotteen esteettiset ominaisuudet kuten esimerkiksi visuaalisuus ja auditiivisuus. Tuotteeseen liittyvillä oheistarvikkeilla ja -palveluilla on myös vaikutuksensa käyttäjäkokemuksen syntymisessä. Tällaisia voivat olla esimerkiksi tuotteen käyttöohje tai tukipalvelu. [15]

Tuotteen käyttöympäristö vaikuttaa merkittävästi käyttäjäkokemuksen syntymiseen. Käyttöympäristö, eli käyttökonteksti, määrittelee millaisessa ympäristössä ja olosuhteissa tuotetta käytetään. Sen voidaan katsoa koostuvan tehtäväkohtaisesta, fyysisestä, sosiaalisesta ja teknisestä ympäristöstä. Teknisestä ympäristöstä esimerkkinä käy vaikkapa laite, jolla kehitettävää ohjelmistoa on suunniteltu käytettävän. Ohjelmiston käyttöliittymää suunniteltaessa on tiedettävä tarkkaan, millaisilla laitteilla loppukäyttäjät tulevat tuotetta käyttämään. Sosiaalisia käyttöympäristöjä ovat esimerkiksi koti ja työpaikka. Kotona tunnelma on usein rentoutunut kun taas työpaikalla ilmapiiri voi olla virallisempi. [15]

Käytettävyydeltään hyvän käyttöliittymän ominaisuuksia



1. Käyttäjäkokemuksen syntyminen käyttäjän ja tuotteen välisessä vuorovaikutuksessa [15]

Miksi vaalia käyttäjäkokemusta? Ohjelmistonäkökanta

## Käyttäjätarpeiden kartoitus

## Käyttäjäkeskeinen suunnittelu

Prosessi, jolla hyvä (ISO 9241-11) tuote suunnitellaan

Näkökulma, joka pyrkii ymmärtämään ihmisen käyttäytymistä sekä taitoja ja hyödyntää niitä tuotesuunnittelussa.

Asiakaslähtöisyys

Käyttäjäkokemuksen suunnittelu on tuotteen käyttäjälle ilmeneviä laatuominaisuuksia

Paperikäyttöliittymällä prototypointi

Seuraavissa luvuissa 3.3.1 – 3.3.3 käsitellään erilaisia hyvän käyttäjäkeskeisen suunnittelun periaatteita, jotka ovat sovellettavissa käyttöliittymäsuunnitteluun. Suunnitteluperiaatteet soveltuvat myös käytettävyyden heuristiseen arviointiin. [15]

### Normanin suunnitteluperiaatteet

Käyttäjäkeskeisen suunnittelun alalla uraa tehnyt professori Donald A. Norman kertoo kirjassaan ”The Design of Everyday Things” (2013) [9] kuinka jokapäiväisten esineiden käytettävyyttä voitaisiin parantaa. Näitä ohjeistuksia on helppo soveltaa myös ohjelmistojen käyttöliittymien suunnitteluun. Norman on koonnut ohjeensa viideksi suunnitteluperiaatteeksi:

1. Loogiset kytkennät

Käyttäjälle tulisi selvitä, mikä ohjaimen ja toiminnan välinen kytkentä on. Kytkentä tulisi olla selvä jo ennen kuin käyttäjä on ensimmäistä kertaa käyttänyt ohjainta. Loogisia kytkentöjä suunniteltaessa kannattaa käyttää hyväksi fyysisiä ja tilaan liittyviä analogioita. Esimerkiksi ylöspäin suuntautuva liike ja ylöspäin osoittava nuoli merkitsevät usein lisäystä kun taas alaspäin suuntautuva liike ja alaspäin osoittava nuoli yhdistetään usein vähentymiseen. Tällaista logiikkaa voidaan käyttää hyväksi esimerkiksi lukumäärää tai mittasuhteita kuvaavissa ohjaimissa.

1. Mahdollisuudet ja rajoitteet

Kun ihminen muodostaa käsitystä tuotteen toiminnasta, hän tekee näkemästään rakenteesta tulkintoja rakenteen mahdollisuuksien ja rajoitteiden perusteella. Mahdollisuudet tarkoittavat eri toimintavaihtoehtoja, joita ovat esimerkiksi käyttöliittymän nappulat, jotka ovat painettavissa. Rajoitteilla puolestaan tarkoitetaan toiminnalle asetettuja rajoituksia, joita ovat esimerkiksi tekstikenttään syötettävän merkkijonon maksimipituus tai harmaannutetut, ei-käytössä olevat käyttöliittymän nappulat.

1. Näkyvyys

Näkyvyydellä tarkoitetaan käyttäjän kannalta oleellisten tietojen ja toimintojen esille tuomista. Käyttöliittymän tulee olla visuaaliselta ilmeeltään sellainen, että käyttäjän on helppo ymmärtää, kuinka tuotetta on tarkoitus käyttää toivotun lopputuloksen saavuttamiseksi. Oikeiden elementtien on oltava käyttöliittymässä näkyviä, jotta käyttäjä saa oikean käsityksen tuotteen toimintatavasta.

1. Palaute

Palaute on käyttäjälle annettavaa tietoa tuotteen toiminnasta. Palautteesta ilmenee, mitä on tehty ja mitä tuloksia saavutettu. Palaute on oleellista onnistuneen toiminnan jälkeen, mutta erityisesti myös virhetilanteissa palaute on tärkeää. Olennaista palautteessa on vasteaika. Palautteen on tapahduttava heti käyttäjän toiminnan jälkeen, jotta se osataan yhdistää kyseiseen toimintaan.

1. Virheiden käsittely

Käyttöliittymäsuunnittelun tavoitteena on toki tehdä käyttäjän tekemät virheet mahdottomiksi, mutta se harvoin onnistuu käytännössä. Tämän takia virhetilanteiden käsittely ja niistä palautuminen on keskeistä käyttöliittymäsuunnittelussa. Käyttäjän ei tulisi koskaan syyttää itseään virheistä, sillä se voi johtaa turhautumiseen ja jopa tuotteen hylkäämiseenkin. Käyttäjälle tulee tarjota virhetilanteen sattuessa selkeät ohjeet miten tilanteesta pääsee pois.

### Nielsenin heuristiikat

Käytettävyysasiantuntija Jakob Nielsen esittelee kirjassaan ”Usability Engineering” (1993) [8] kymmenen käytettävyysheuristiikan listan, jonka avulla voi arvioida tuotteen tai käyttöliittymän käytettävyyttä. Lista soveltuu myös uuden tuotteen tai käyttöjärjestelmän suunnittelun periaatteiksi. Nielsenin heuristisen arvioinnin listaan kuuluvat:

1. Järjestelmän tilan näkyvyys
2. Järjestelmän ja todellisen maailman yhteys
3. Käyttäjän kontrolli ja vapaus
4. Johdonmukaisuus
5. Virheiden välttäminen
6. Tunnistettavuus ennen muistamista
7. Käytön joustavuus ja tehokkuus
8. Yksinkertaisuus
9. Virheiden käsittely
10. Opasteet ja dokumentaatio

### Shneidermanin kahdeksan kultaista sääntöä

Tietojenkäsittelytieteiden professori Ben Shneidermanin kirjassa ”Designing the user interface: strategies for effective human-computer interaction” (1998) [13] esitellään kahdeksan kultaista sääntöä erityisesti ohjelmistojen käyttöliittymien dialogien suunnitteluun ja arviointiin. Säännöt ovat kuitenkin sovellettavissa ohjenuoriksi myös koko järjestelmän suunnitteluun [15]. Shneidermanin kahdeksan kultaista sääntöä ovat:

1. Noudata yhtenäisyyttä toimintaketjuissa ja toimintatavoissa
2. Tarjoa edistyneille käyttäjille oikoteitä
3. Tarjoa informatiivista palautetta
4. Suunnittele dialogit siten, että ne johtavat lopputulokseen
5. Tarjoa yksinkertaista virheenkäsittelyä
6. Auta käyttäjää toipumaan virhetilanteista
7. Tue käyttäjän kontrollin tunnetta
8. Rajoita käyttäjän lyhytkestoisen muistin kuormitusta

# Graafisen käyttöliittymän ohjelmointi Windows-ympäristössä

Tässä luvussa käsitellään ohjelmistotekniikkaan liittyviä käsitteitä ja malleja, jotka koskevat erityisesti graafisen käyttöliittymän ohjelmointia Windows-ympäristössä C++-ohjelmointikielellä. Luvussa painotetaan erityisesti niitä asioita, joiden ymmärtäminen on tässä diplomityössä suunniteltavan ja toteutettavan graafisen käyttöliittymän kannalta oleellisia.

METATEKSTIÄ asian järjestyksestä

## Olio-ohjelmointi

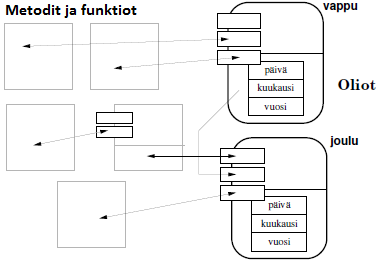
Ohjelmistotekniikan yhtenä vaikeimpana ongelmana pidetään suurten ohjelmistojen tekemisen monimutkaisuutta. Tietokonelaitteistojen tekniikoiden räjähdysmäinen kehitys on mahdollistanut entistä suurempien ja kompleksisempien ohjelmistojen suorittamisen. Ohjelmistojen tekemiseen tarvittavat työkalut ja menetelmät, kuten ohjelmointikielet, eivät ole kuitenkaan kehittyneet samaa vauhtia laitteistotekniikan kehityksen mukana. Tämän seurauksena on puhjennut niin sanottu ohjelmistokriisi, jolla tarkoitetaan juuri tätä tietokonelaitteistojen ja ohjelmointityökalujen kehityksien epävakaata suhdetta. [11]

Yhtenä ratkaisuna ohjelmistokriisin ratkaisemiseksi on tarjottu oliokeskeisiä menetelmiä, jotka ovatkin viime aikoina olleet suurimman huomion kohteena ohjelmistoteollisuudessa. Oliokeskeisiä menetelmiä käyttävässä ohjelmoinnissa, eli lyhyemmin olio-ohjelmoinnissa, perusperiaatteina ovat ongelmien jakaminen yhden ihmisen hallittaviin osakokonaisuuksiin sekä yksinkertaistaminen abstrahoimalla. Abstrahoinnin voidaan sanoa tarkoittavan toisiinsa liittyvien asioiden keräämistä yhdeksi järkeväksi toiminnalliseksi kokonaisuudeksi, joka voi olla puhtaasti ajatuksellinen eli abstrakti. [11]

### Oliot ja luokat

Olio-ohjelmoinnissa ongelmien jako yhden ihmisen hallittaviin osakokonaisuuksiin tarkoittaa ohjelmakoodin toiminnallisesti sekä tietorakenteellisesti yhteenkuuluvien osien kokoamista olioiksi. Yhden olion rakenne koostuu tietorakenteesta, sitä käsittelevästä ohjelmakoodista sekä olion tietorakenteen käsittelyyn tarkoitetusta julkisesta rajapinnasta. Oliorakenteen toteuttajan vastuulla on olion sisäinen toteutus, eli kuinka sen tietorakenne koostetaan ja kuinka sitä käsitellään. Oliorakenteen toteuttaja suunnittelee oliolle myös julkisen rajapinnan. Julkisen rajapinta tulisi olla mahdollisimman selkeä ja ymmärrettävä, vaikka sen käyttäjä ei tuntisi olion sisäistä toteutusta lainkaan. Julkinen rajapinta tarkoittaa metodeja tai funktioita, joilla oliota tai sen tilaa voidaan käsitellä sen ulkopuolelta muualla ohjelmakoodissa. Olio siis kätkee sisäänsä tietorakenteensa ja sen käsittelyn, jolloin olion käyttäjän vastuulle jää ainoastaan olion julkisen rajapinnan oikeaoppinen käyttö. Olion käytön kannalta epäoleellisen tiedon kapseloiminen sen sisälle on yksi tärkeimpiä abstrahoinnin työkaluja ohjelmoinnissa. [11]

C++-kielessä oliot luodaan luokkien avulla. Luokka on määrittely siitä muodostettavan olion tietorakenteelle, metodeille ja funktioille sekä julkiselle rajapinnalle. Luokka on siis yleinen rakenne, joka edustaa kaikkia kyseisestä luokasta luotuja olioita. Ohjelman ajon aikana ei ole olemassa luokkia, vaan ainoastaan luokan kuvauksen perusteella luotuja olioita. [11]



1. Oliot ja niiden tietorakenne sekä julkinen rajapinta. Kuva muokattu lähteestä [11].

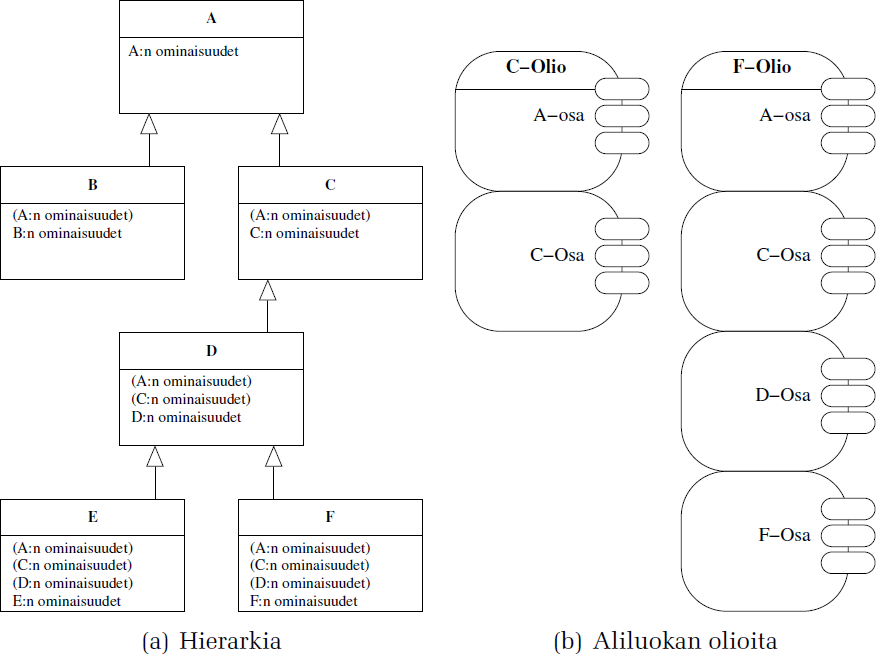
Kuva 6 on havainnollistettu tyypillisiä olio-ohjelmoinnissa käytettäviä olioita. Oliot ”vappu” ja ”joulu” ovat samasta luokasta tehtyjä instansseja, ja niiden tarkoitus on esittää päiväystä. Päiväystä esittävän olion tietorakenteeseen kuuluu siis tässä tapauksessa päivä, kuukausi ja vuosi, jotka voidaan ilmaista kokonaislukuina. Olioiden julkisessa rajapinnassa on metodeita ja funktioita, joiden avulla niiden tilaa pystytään muokkaamaan ja kontrolloimaan olioiden ulkopuolelta. Kuvassa olevien päiväystä esittävien olioiden tapauksessa metodit voisivat olla esimerkiksi päiväyksen kasvattaminen halutulla määrällä päiviä tai päiväyksen tulostaminen näytölle tietyllä syntaksilla.

Ohjelmistoja suunniteltaessa olio-ohjelmointi on oiva työkalu ohjelmointiongelman mallintamiseen siten, että se vastaisi mahdollisimman hyvin todellisen maailman ongelmaa. Kun ohjelmassa olevat oliot vastaavat mahdollisimman hyvin todellisen maailman olioita, on monimutkaisenkin tietokoneohjelman rakenne helpommin ymmärrettävissä. Toisaalta ohjelmistoihin suunniteltavien olioiden tulisi olla myös ohjelmoinnin kannalta katsottuna käytännöllisiä ja tehokkaita, jolloin ne eivät välttämättä täsmällisesti vastaa todellisen maailman olioita. Olio-ohjelmointi vaatii ohjelmoijalta hyvää abstraktista ajattelukykyä. [11]

### Periytyminen

Yksi tietokoneohjelmien rakenteiden hahmottamisen ja kategorisoimisen avuksi luotu mekanismi on oliokeskeisissä ohjelmointikielissä yleisesti mukana oleva periytyminen. Se tarkoittaa luokkien jaottelua kantaluokkiin ja aliluokkiin niiden yhteisten ominaisuuksien ja sukulaisuussuhteiden perusteella. Periytymisen avulla ohjelmiston olioille on mahdollista luoda hierarkkinen periytymisrakenne, joita esiintyy usein myös todellisessa maailmassa. Periyttämisen avulla voidaan luoda useita samankaltaisia olioita, joilla on hieman toisistaan poikkeava käyttäytyminen, kirjoittamatta kuitenkaan uudestaan olioiden yhtenevän käyttäytymisen toteuttavaa koodia. [11]

Käytännössä C++-ohjelmointikielessä periytymisellä tarkoitetaan sitä, että uusi luokka muodostetaan toisen olemassa olevan luokan pohjalta siten, että uusi luokka sisältää kaikki toisen luokan ominaisuudet. Tällöin alkuperäistä luokkaa, jonka ominaisuudet periytyvät uuteen luokkaan, kutsutaan kantaluokaksi. Uutta, periytettyä luokkaa, kutsutaan aliluokaksi.



1. Periytymishierarkia ja oliot [6].

Kuva 7 on esimerkki luokkien periytymishierarkiasta havainnollistavana kaaviona. Vasemman puolen hierarkiassa nuoli osoittaa aina aliluokasta kantaluokkaan päin. Esimerkin tapauksessa luokka A toimii kantaluokkana kaikille muille luokille. Luokkien ominaisuudet, eli julkisen rajapinnan tarjoamat metodit ja funktiot, periytyvät kuvan vasemman puolen kaavion mukaisesti. Aliluokista luotujen olioiden voidaan ajatella sisältävän kaikki sen periytymishierarkiassa yläpuolella olevien sukulaisluokkien informaation, kuten kuvan oikealla olevassa kuviossa on esitetty.

## Suunnittelumallit

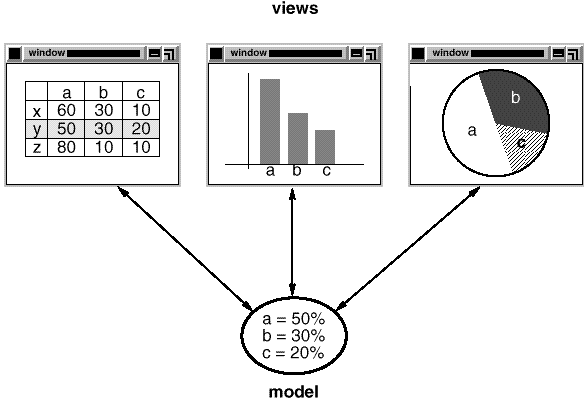
Olio-ohjelmoinnissa ohjelmiston toiminnallisuus muodostuu suuren oliojoukon yhteistoiminnan vaikutuksesta. Hyvin suunniteltu ohjelmisto sisältää sellaisia oliorakenteita, jotka ovat uudelleenkäytettäviä myös muissa kohteissa. Tällä tarkoitetaan suunnittelun geneerisyyttä. Kirjassa *”Design Patterns: Elements of Reusable Object-Oriented Software”* [4] on kerätty hyviksi havaittuja oliosuunnittelun käytäntöjä erilaisiin ohjelmointiongelmiin. Näistä käytännöistä käytetään kirjassa nimitystä suunnittelumalli. Kaikki kirjan käsittelemät suunnittelumallit ovat sellaisia, jotka on havaittu toimiviksi kahden tai useamman toisistaan riippumattoman ohjelmistoprojektihenkilöstön toimesta.

Seuraavissa aliluvuissa esitellään useammasta suunnittelumallista koostuvaa MVC-arkkitehtuuria sekä ”Visitor pattern” -suunnittelumallia. Niiden periaatteita on sovellettu tässä työssä toteutetussa käyttöliittymässä.

### MVC-arkkitehtuuri

MVC-arkkitehtuuri (*Model-view-controller)* on kolmesta erilaisesta luokasta koostuva ohjelmistoarkkitehtuurityyli. Se on yleinen suunnitteluperiaate monien käyttöliittymien toteutuksissa. MVC-arkkitehtuurin luokat ovat malli (model), näkymä (view) ja käsittelijä (controller). Sen etuna on käyttöliittymän näkyvän osan toteutuksen erottaminen sen esittämän tietomallin rakenteesta. Tällöin mallin tieto ei ole riippuvainen näkymästä, vaan saman mallin sisältö voidaan esittää usean erilaisen näkymän kautta. Malli ei ole riippuvainen myöskään käsittelijästä, joten se voidaan suunnitella täysin irrallaan käsittelijästä. Käsittelijä toimii käyttäjältä tulevien komentojen vastaanottajana muuntaen mallia ja näkymää niiden perusteella. [4]

Kuvassa 8 on havainnollistettu tyypillisen MVC-arkkitehtuurin rakennetta. Se sisältää kolmesta muuttujasta a, b ja c koostuvan mallin sekä kolme erilaista näkymään siihen. Yksinkertaistuksen vuoksi käsittelijä on jätetty kuvasta pois. Taulukko, histogrammi ja ympyräkaavio esittävät mallin tiedon eri tavoin. Malli viestii näkymille, kun siihen tulee muutoksia. Näkymät viestivät mallin kanssa saadakseen pääsyn näytettävään tietoon. Tällä tavoin malli ja näkymät pysyvät aina ajan tasalla. Käsittelijä, jota kuvaan ei ole piirretty, voisi toimia esimerkiksi näppäimistön ja hiiren painallusten tulkitsijana. Koska käsittelijä toteutetaan MVC-arkkitehtuurissa omana luokkanaan, se on helppo vaihtaa toiseen esimerkiksi hieman erilaiseen toteutukseen jotain toista käyttötarkoitusta varten.



1. MVC-arkkitehtuurin malli ja näkymät. [4]

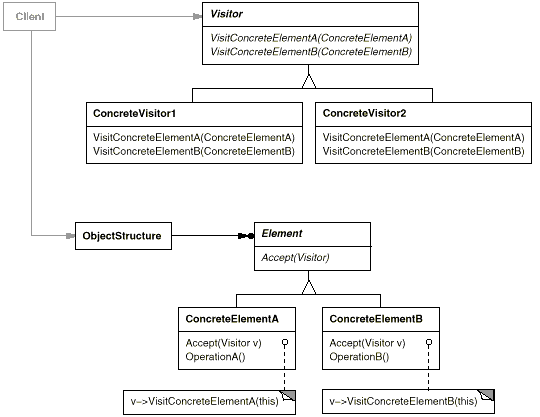
MVC-arkkitehtuuri koostuu itse asiassa kolmen erilaisen suunnittelumallin hyödyntämisestä:

* Observer-suunnittelumallista, jossa muutos yhden luokan oliossa voi vaikuttaa kuinka moneen tahansa muuhun olioon tarvitsematta mitään tietoa muiden olioiden toteutusyksityiskohdista. Esimerkiksi kuvan 8 tietomallin suhde näkymiin voi olla tällainen.
* Composite-suunnittelumallista, jossa yksittäisiä olioita käsitellään samalla tavoin kuin koosteolioita. Koosteolioilla on yhteinen kantaluokka yksittäisten olioiden kanssa, ja ne sisältävät ryhmän yksittäisiä olioita. Esimerkiksi kuvan 8 näkymät voisivat olla koosteolioita, jolloin niiden sisällä voisi olla vielä muita näkymiä.
* Strategy-suunnittelumallista, jossa jokin algoritmi toteutetaan luokkana. MVC-arkkitehtuurin tapauksessa algoritmina toimii käsittelijä-luokka, joka huolehtii käyttäjän antamien komentojen toteuttamisesta. Käsittelijä-luokkia voi olla useita ja niitä on mahdollista vaihtaa jopa ohjelman ajon aikana. Tällä tavoin saadaan muutettua näkymän vastetta käyttäjän syötteisiin helposti ainoastaan käsittelijä-oliota vaihtamalla.

### Visitor pattern -suunnittelumalli

Visitor pattern -suunnittelumalli mahdollistaa tietorakenteen ja siihen liittyvien operaatioiden irrottamisen toisistaan. Suurin käytännön hyöty tästä on se, että uusien tietorakennetta läpikäyvien operaatioiden toteuttaminen onnistuu helposti muuttamatta itse tietorakennetta. Visitor pattern -malli on hyödyllisin silloin, kun tietorakenteen tiedetään pysyvän kohtuullisen muuttumattomana, mutta tietorakenteeseen liittyviä uusia operaatioita halutaan mahdollisesti toteuttaa myöhemmin lisää. Se on erityisen käyttökelpoinen myös silloin, kun tietorakenne koostuu useista erilaiset rajapinnat sisältävistä luokista, joille kaikille halutaan tehdä saman tyyppisiä, mutta hieman toisistaan poikkeavia operaatioita. Operaatioita ei tällöin tarvitse toteuttaa tietorakenteen sisälle, vaan ne voidaan ulkoistaa omiksi luokikseen.

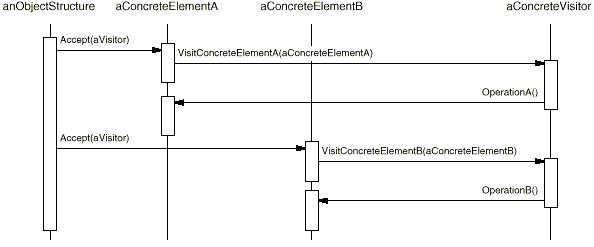
Visitor pattern -suunnittelumallin perusidea on se, että tietorakenne itse huolehtii alkioidensa läpikäymisen, ja erillinen Visitor-luokka suorittaa halutun operaation yhdelle tietorakenteen alkiolle. Kuvassa 9 on esitetty luokkakaavio, joka havainnollistaa suunnitteluperiaatteen toimintaa.



1. Visitor pattern -suunnittelumalli. [4]

Visitor pattern -suunnittelumallin voidaan katsoa koostuvan seuraavista viidestä komponenttityypistä:

* **Visitor -luokka,** joka esittelee vastaavan visit-operaation jokaiselle yksittäiselle tietorakenteen alkion luokalle. Visitor-luokka toimii kantaluokkana ConcreteVisitor-luokille.
* **ConcreteVisitor -luokka,** joka toteuttaa suoritettavan operaation yksittäiselle tietorakenteen alkiolle. Jokaista tietorakenteen alkion luokkaa kohden on myös oma funktionsa, joiden toteutus voi olla erilainen. ConcreteVisitor-luokan olio voi myös säilöä tietoa itseensä käydessään läpi tietorakennetta.
* **Element -luokka,** joka esittelee accept-operaation. Accept-operaatio saa parametrinaan Visitor-tyyppisen olion. Element-luokka toimii kantaluokkana kaikille tietorakenteen alkioiden luokille.
* **ConcreteElement -luokka,** joka **v**astaa yksittäisen tietorakenteen alkion luokkaa. Tämä luokka toteuttaa accept-operaation, joka ikään kuin päästää Visitor-tyyppisen olion tekemään operaationsa ConcreteElement-tyyppiseen olioon.
* **Tietorakenne,** joka koostuu Element-luokasta periytetyistä olioista. Siihen kohdistuvat operaatiot ovat Visitor-luokasta periytettyjen ConcreteVisitor-olioiden toteuttamia. Tietorakenteen muoto on täysin vapaa. Se voi olla esimerkiksi lista, puu tai graafi.



1. Visitor pattern -suunnittelumallin mukainen tietorakenteen läpikäynti. [4]

Kuva 10 havainnollistaa Visitor pattern -suunnittelumallin mukaista tietorakenteen läpikäyntiä. Kuvan tietorakenne on yksinkertaistettu siten, että se koostuu vain kahdesta alkiosta ConcreteElementA ja ConcreteElementB. Tietorakenteen alkiot kutsuvat Accept-funktiossa parametrina saadun Visitor-tyyppisen olion visit-funktiota. Visit-funktio taas saa parametrikseen käsiteltävänä olevan tietorakenteen alkion, jolle Visitor-olio suorittaa itse operaation.

## Windows Ribbon Framework

## OpenGL-piirtotyökalut

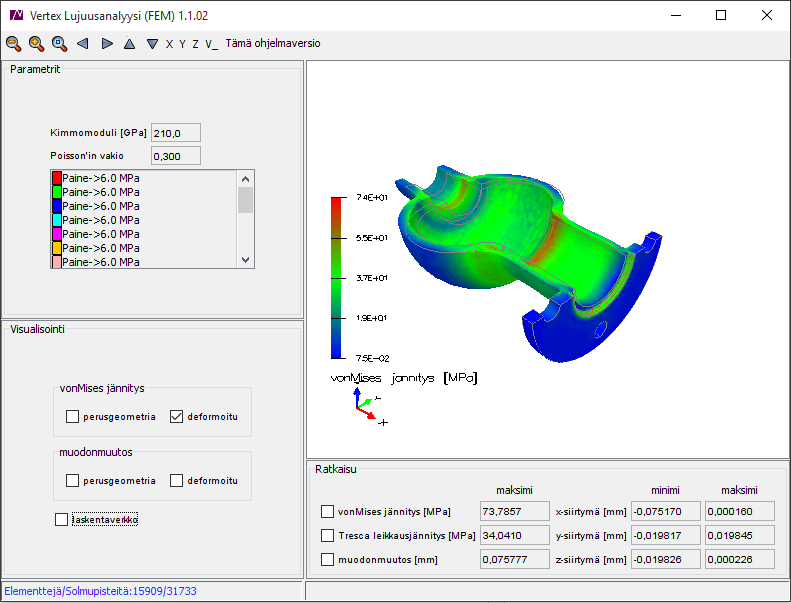
# Vaatimukset laskentamoduulille

Vertex-ohjelmistoihin on aiemminkin sisällytetty lujuuslaskentamoduuleja, joihin tässä diplomityössä käsiteltävä kehärakenteiden lujuuslaskentamoduuli on jatkoa. Vertex BD -rakennussuunnitteluohjelmistossa on ollut käytössä mistä lähtien kattoristikoiden normin mukainen mitoituslaskenta. mille normeille. Lisäksi Vertex G4 -mekaniikkasuunnitteluohjelmassa on ollut käytössä Numerola Oy:n kanssa yhteistyössä toteutettu lujuuslaskentamoduuli yksittäisille tilavuusmalleille.

KUVA Vertex BD Truss engineering

Voimat asetetaan z-suunnassa. Voimat asetetaan piirustukseen mielivaltaiseen kohtaan. BD:n kuva-mallipari.

Ongelma: muodostettua FEM-mallia (STAFRA:an menevää) ei nähdä missään muodossa, eikä sitä voi muokata



1. Vertex G4:n yksittäisen tilavuusmallin lujuusanalyysi.

KUVA Vertex G4 Numerola

Tällä hetkellä Numerola: ei verkota oikein, kankea käyttöliittymä, ei ole yhtenäinen osa Vertexiä.

3D mesh, Spatial

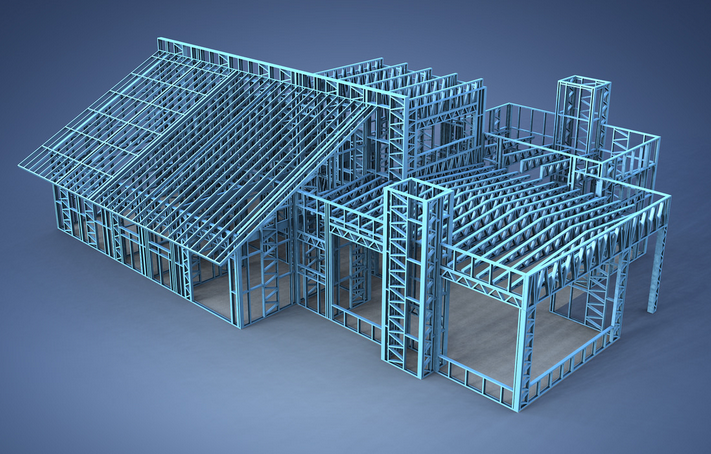
Tällä hetkellä STAFRA BD: ainoastaan ristikkolaskentaan räätälöity, ei voida laskea esimerkiksi aukkopalkkeja (lintel-rakenteita)

Laskentamallin luominen, tärkeys, mallin tehokkuus, yksinkertaistaminen, Hibbler Idealization [4]

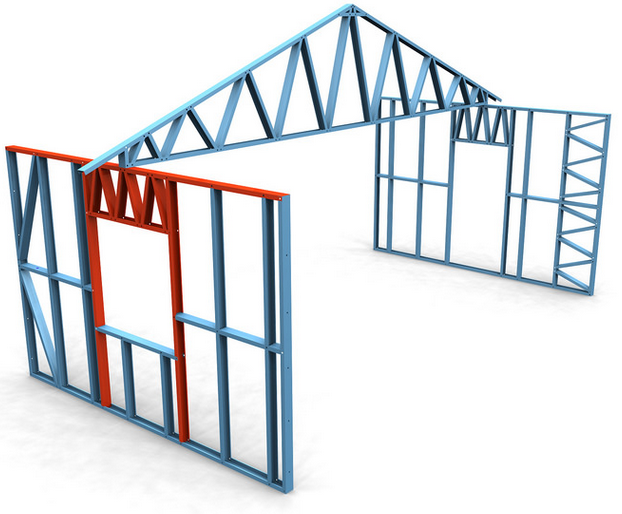
Bathe, laskennan vaiheet

## Asiakasvaatimukset

Vertex yksi suurimmista CFS-profiilirakentamisen suunnittelutyökaluista. Tällä hetkellä voidaan laskea vain ristikoita. Ei mahdollisuuksia laskea esimerkiksi seiniä, aukkopalkkeja (lintel-rakenteet) tai muutakaan. BD: Tarve laskea lintel-rakenteita (palkki pilarien päällä, esim. ovi- tai ikkuna-aukko). Suuret ristikot ja avaruuskehät, joissa todella paljon osia. Teräsrankarakentaminen Australiassa, CFHS-palkit



1. Esimerkki tyypillisestä Vertex BD -ohjelmalla suunnitellusta teräsrankarakenteesta.



1. CFS-profiilirakenteinen ikkunan aukkopalkki.

Vertex G4:ssä nykyään oleva Numerola Oy:n kanssa yhteistyössä tehty lujuuslaskentamoduuli tullaan myöhemmin korvaamaan Vertexin omalla yksittäisiä tilavuusmalleja ja mahdollisesti myös kokoonpanomalleja tukevalla lujuuslaskentamoduulilla. Vertexin asiakaskunnalta kyseltiin, minkälaisia kokemuksia heillä oli nykyisestä lujuuslaskentaoptiosta. Asiakkailta saatiin paljon kehitysehdotuksia ja toiveita tuleviin versioihin. Alla on listattu eniten esiin nousseita toiveita:

* Kehärakenteiden siirtymien, rasitusten ja tukivoimien laskenta
* Nurjahdusanalyysit
* Putkistojen jännitysanalyysit
* 3D- ja FEM-mallin välinen keskinäinen synkronointi malleja päivitettäessä
* FEM-mallin tekeminen ilman 3D-mallia
* Ohutlevyosien analysoiminen kuorielementeillä
* Kokoonpanomallien analysoiminen solidielementeillä
* Väsymisanalyysi

Seuraaviin Vertexin versioihin tuleva kehärakenteiden lujuuslaskentamoduuli tulee tyydyttämään asiakkaiden tarpeet kehärakenteiden siirtymien, rasitusten ja tukivoimien laskennan osalta. Osa asiakkaista oli yrittänyt laskea pienimuotoisia putkipalkkirakenteita tekemällä kokoonpanomallista yhden tilavuusmallin ja syöttämällä sen nykyiseen lujuusanalyysiin. FEM-malli luotiin tällöin solidielementeillä, mikä ei palkkirakenteita analysoitaessa ole hyvä ratkaisu. Palkkielementtejä käyttämällä FEM-malli saadaan luotua huomattavasti pienemmällä määrällä elementtejä ja laskenta on silloin paljon tehokkaampaa.

Muihin asiakaskunnalta saatuihin toiveisiin ei vielä seuraavissa Vertexin versioissa pystytä vastaamaan. Ne tullaan ottamaan kuitenkin huomioon tulevia kehitysaskeleita suunniteltaessa. Yksittäisiä tilavuusmalleja sekä kokoonpanomalleja tukevan solidielementtejä käyttävän lujuuslaskentamoduulin laskentamoottorin prototyyppiä on Vertexillä kehitetty jo varsin pitkälle.

## STAFRA

STAFRA *(Static analysis of frames)* on Lujuustekniikka Oy:n (nykyisin Vertex Systems Oy) kehittämä avaruuskehien statiikan ongelmien ratkaisuun tarkoitettu laskentamoottori. Se perustuu luvussa 2 esiteltyyn kehärakenteiden elementtimenetelmään ja sen elementit ovat Euler-Bernoulli -palkkiteorian mukaisia palkkeja. Palkkiteoriasta poiketen STAFRA ottaa huomioon myös leikkausjännityksistä aiheutuvat liukumat tehollisten leikkauspinta-alojen avulla. Teholliset leikkauspinta-alat palkin paikalliskoordinaatiston y- ja z-suunnissa saadaan kaavoilla

ja (10)

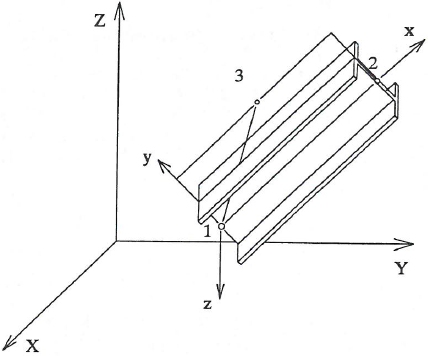
palkkielementin paikalliskoordinaatiston x-akselin yhtyessä poikkileikkausten pintakeskiöihin sekä y- ja z-akselit poikkileikkausten pääneliöakseleihin. Kertoimet ja aiheutuvat leikkausjännityksen epätasaisesta jakaantumisesta poikkileikkauksen alueelle ja ne ovat riippuvaisia poikkileikkauksen muodosta. STAFRA rajoittuu elementtien materiaalien osalta homogeenisiin, lineaarisesti kimmoisiin ja isotrooppisiin materiaaleihin. [14]

STAFRA on käytössä Vertex BD:n nykyisessä kattoristikoiden mitoituslaskennassa.

Aktiivisarakeratkaisu (Lohkomaisenakin)

## Avaruuspalkkielementti

Avaruuspalkkielementti on 2-solmuinen elementti, jonka molemmilla solmuilla on 6 vapausastetta. Kuvassa 11 on esitetty avaruuspalkkielementti, sen paikalliskoordinaatisto ja rakennekoordinaatisto. Elementin suunnistus kulkee solmusta 1 solmuun 2. Palkin yläpuolella oleva solmu 3 toimii suuntasolmuna, joka määrittää elementin paikalliskoordinaatiston y-akselin suunnan.



1. Avaruuspalkkielementti ja sen paikalliskoordinaatisto. [14]

## Yhteensopivuus Vertexin eri ohjelmistojen kanssa

Toteutettavan kehärakenteiden lujuuslaskentamoduulin tulee toimia yhdenmukaisella tavalla sekä Vertex G4 että Vertex BD -ohjelmissa. Tämä aiheuttaa haasteita laskentamoduulin toteutuksessa, sillä G4 ja BD -ohjelmien kehityshistoriat ovat varsin erilaiset. Tämän vuoksi joudutaan tekemään osittain erilaisia toteutuksia samoille laskentamoduulin toiminnallisuuksille.

Vertexin profiilit, KUVA periytymishierarkia

G4, G4 Plant, BD

# Käyttöliittymän toteutus

Asd

## Tietorakenne

Luokkakaavio

Datainterface: attribuuttitekstit-virtuaalinen oliomalli. Historialliset syyt takana, miksi tallennetaan osien attribuuttiteksteihin

## Puunäkymä

Valmiina olleet käyttöliittymäkomponentit puunäkymälle

studytree\_cont

## Ribbon-valikko

## Käyttötapaukset

### Tutkimusten luominen ja poistaminen

### Tuentojen ja kuormitusten asettaminen

### Tulosten tarkastelu

### Mitoitus

# Jatkokehitystarpeet

Käyttöliittymä solidielementtien laskennalle

Kuorielementit (ohutlevyosien laskenta)

Epälineaaristen ongelmien ratkaisu

Iteratiiviset ratkaisumenetelmät

# Yhteenveto

Asd

Lähteet

1. K.J. Bathe, *Finite Element Procedures in Engineering Analysis*, Prentice-Hall, Englewood Cliffs, New Jersey, 1982, 735 p.
2. M. Bordegoni, C. Rizzi, *Innovation in Product Design: From CAD to Virtual Prototyping*, Springer-Verlag London Limited, 2011, 188 p.
3. R.D. Cook, D.S. Malkus, M.E. Plesha, R.J. Witt, *Concepts and Applications of Finite Element Analysis*, John Wiley & Sons, Inc, 1989, 630 p.
4. E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns – Elements of Reusable Object-Oriented Software,* Addison-Wesley, 1995, 386 p.
5. R.C. Hibbeler, *Structural Analysis*, Pearson Prentice Hall, 2012, 695 p.
6. M. Hirz, W. Dietrich, A. Gfrerrer, J. Lang, *Integrated Computer-Aided Design in Automotive Development*, Springer-Verlag Berlin Heidelberg, 2013, 466 p.
7. K. Koskimies, *Oliokirja*, Talentum Oyj, 2000, 422 s.
8. R. Kouhia, M. Tuomala, *Johdatus mekaniikan ja sähkömagnetiikan numeerisiin menetelmiin*, luentomoniste, Tampereen teknillinen yliopisto, 2014, 417 s.
9. J. Nielsen, *Usability engineering,* Academic Press, 1993, 362 p.
10. D. Norman, *The Design of Everyday Things – Revised and Expanded Edition*, Basic Books, 2013, 347 p.
11. E. Oñate, *Structural Analysis with the Finite Element Method. Linear Statics. Volume 2. Beams, Plates and Shells*, Springer, 2013, 864 p.
12. M. Rintala, J. Jokinen, *Olioiden ohjelmointi C++:lla*, Talentum, 2005, 466 s.
13. T. Salmi, K. Kuula, *Rakenteiden Mekaniikka*, Pressus Oy, Tampere, 2012, 464 s.
14. B. Shneiderman, *Designing the user interface: strategies for effective human-computer interaction*, Addison-Wesley, 1998, 639 p.
15. STAFRA-3D -manuaali, Insinööritoimisto Lujuustekniikka Oy, 1980, 57 s.
16. K. Väänänen, Käyttäjäkokemuksen perusteet -kurssin luentomoniste, Tampereen teknillinen yliopisto, 2015, 68 s.

Liite A: M

Lopuksi muutamia pohjaa päivittän