Ohjelmistojen koon mittaaminen eri tyyppisissä kehityshankkeissa

Pekka Forselius,
STTF Oy

Kansallinen ja kansainvälinen tilanne

Kun kerromme merkittävänä tutkimustuloksena että "**suurten ohjelmistojen kehittäminen edellyttää projekteilta suurempaa työmäärää kuin pienten**", useimmat kuulijat hymähtävät tälle itsestään selvyydelle. Niinpä niin. Kuitenkaan nämä samat ihmiset eivät käytä määrämuotoista ohjelmiston koon laskentaa apuna työmäärien arvioinnissa. Outoa, vai mitä! Miksi? Luulisi riippuvuuden olevan yhtä itsestään selvää myös käänteisesti. Ajattelevatko he kenties oheisen kuvan mukaisesti?

Suomalaisen perusluonteen tuntien on vaikeata uskoa, että ohjelmistojen koon laskennan laiminlyönnissä kyseessä olisi välinpitämättömyys. Mehän hoidamme asiat tunnollisesti direktiivejä noudattaen, ainakin niin hyvin kuin osaamme. Kyseessä täytyy siis olla osaamattomuus. Ohjelmistojen koon laskennan tärkeys on kyllä huomattu yleisemminkin. European Software Institute (ESI) on valinnut koon mittaamisen yhdeksi yhdeksästä huippukäytännöstä, jotka liittyvät ohjelmistotyön standardeihin ja työmenetelmiin. Vuonna 1995 tehdyssä tutkimuksessaan he kuitenkin havaitsivat, että juuri tätä käytäntöä ei oltu omaksuttu hyvin yhdessäkään Euroopan maassa eikä myöskään millään toimialalla. Oheisessa taulukossa kyseisen tutkimuksen kansallisen vertailun tulokset:

Huippukäytäntö:	**Maat, joissa omaksuttu hyvin**	**Maat, joissa sovelletaan heikosti**
Riskien ja toteuttamiskelpoisuuden arviointi	**Suomi**, Iso-Britannia, Belgia	Kreikka, Itävalta, Sveitsi
Tilannekatsausten suorittaminen määräajoin	Iso-Britannia, Hollanti, Espanja	Ruotsi, Portugali
Suunnitelmien ja koodin tarkastaminen ja läpikäynti	Iso-Britannia, Sveitsi, **Suomi**	Portugali, Ranska, Espanja, Belgia
Yleisten ohjelmointistandardien soveltaminen	Iso-Britannia, Sveitsi, Itävalta	Belgia, Tanska
Ohjelmiston koon arviointi määrätyn menetelmän avulla		**Kaikki maat**, erityisesti Ruotsi ja Portugali
Määrämuotoinen työmäärien, aikataulun ja kustannusten arviointi	**Suomi**, Itävalta, Sveitsi	Ruotsi, Belgia, Saksa
Välitulosten siirtäminen ryhmältä toiselle määrämuotoisena	Iso-Britannia, Ranska	Belgia, Tanska, **Suomi**
Testitapausten suunnitteleminen ennen ohjelmointia	Iso-Britannia, Sveitsi, Irlanti	Portugali, Belgia, Kreikka, Itävalta
Testaaminen riippumattoman laadunvarmistusryhmän suorittamana	Iso-Britannia, Irlanti, Ranska	Espanja, Belgia, Italia

Me suomalaiset emme siis ole osaamattomuutemme kanssa yksin, mikä varmaankin hieman lohduttaa. Se ei kuitenkaan oikeuta meitä unohtamaan asiaa kokonaan, jos haluamme olla todella hyviä. Pohjoismaisessa sarjassa erotumme toki jo nykyisin eduksemme, mikä on helppoa havaita taulukkoa tarkasteltaessa. Tanska ja erityisesti Ruotsi näkyvät oikeanpuoleisessa sarakkeessa, jos näkyvät. Norja ei erotu minkään käytännön osalta erityisen hyvänä, muttei myöskään huonoimpana. Meillä ei kuitenkaan ole mitään syytä tyytyä tähän sarjatasoon, sillä määrätietoisen ja pitkään kestäneen kehitystyön tuloksena maassamme on huomattavan paljon osaamista ohjelmistojen koon mittaamisesta. Se on vain aika ottaa käyttöön yritysten käytännön työssä.

Mittaamismenetelmät

Ohjelmistojen kokoa voidaan laskea monella eri tavalla ja ilmaista monella eri yksiköllä. Seuraavissa kappaleissa esitellään yleisimmin tunnetut ja käytetyt menetelmät.

Lähdeohjelman rivimäärien laskeminen on toistaiseksi kaikkein ylei-

simmin käytetty tapa ilmaista ohjelman tai sovelluksen koko. Englanninkielisessä kirjallisuudessa vilahtelee usein sellaisia lyhenteitä kuin LOC (Lines of Code), SLOC (Source Lines of Code) ja KLOC (Kilo Lines of Code). Mittayksikkönä on siis ohjelmarivi. Ymmärrettävästi ohjelmarivi on vertailuissa aika ongelmallinen, jos emme tiedä käytettyä ohjelmointikieltä. Eri kielillä toteutettujen ohjelmien asettaminen suuruusjärjestykseen tuskin onnistuu järkevästi pelkkien rivimäärien avulla. Ajatellaanpa vaikka Assemblerilla ja Visual Basicilla toteutettujen lähdeohjelmien vertaamista keskenään. Myös eri ohjelmoijilla voi olla hyvinkin paljon toisistaan poikkeava tyyli kirjoittaa ohjelmia. Sen vuoksi ainakin eri organisaatioissa työskentelevien ihmisten kirjoittamien ohjelmien vertaaminen on hankalaa, ellei talokohtaisia standardeja ja ohjelmoinnin perusohjeita tunneta.

Monien ohjelmointikielten rivimäärä ei ole ollenkaan yksikäsitteinen luku, vaikka lähdeohjelmalistaus olisi luettavana. Kommenttilauseiden pois jättäminen on yleisesti katsottu tarpeelliseksi, mutta esimerkiksi erilaisten muuttujaesittelyjen huomioiminen tai huomiotta jättäminen vaatii laskijoilta lisäsopimuksia. Myös copy-kirjastojen käyttö aiheuttaa eroja, jos toisinaan koodi puretaan auki ja toisinaan ei.

Ohjelmiston loogisten lauseiden laskeminen on läheistä sukua rivimäärien laskennalle, mutta on mm. Capers Jonesin mukaan sitä parempi tapa. Laskenta tehdään tässäkin tapauksessa lähdeohjelmakirjastoista, mutta laskentaohjelman toteuttaminen on edellistä mittaustapaa vaativampaa. Laskenta pitää tehdä etsimällä koodista ohjelmakielen varattuja suorituskäskyjä. Mahdotonta se ei tietenkään ole, ja kerran toteutettua koonlaskentaohjelmaa voi sitten käyttää pitkän aikaa. Loogisten lauseiden määrää voi tietysti yrittää haarukoida rivimäärien perusteella, mutta tällöin menetelmä ei ole juurikaan luotettavampi kuin suoraan rivimäärien perusteella mitoittaminen. Jonesin havaintojen mukaan esimerkiksi 950 riviä pitkä cobol-ohjelman proseduuri-osuus sisältää keskimäärin 700 loogista lausetta.

Dokumenttisivujen määrään perustuva koon arviointi herättää ensi näkemältä hilpeyttä, mutta monet hyvinkin vakavasti otettavat organisaatiot käyttävät tätä menetelmää. Jos ohjelmistojen kuvausstandardit on käytettävää fonttikokoa myöten erittäin tarkasti määritelty, voidaan eri sovellusten kokoero selvittää vaikkapa punnitsemalla järjestelmäkuvaus vaa'alla. Eräiden maiden puolustusvoimissa ja vaikkapa turvallisuuskriittisten sovellusten parissa työskentelevissä organisaatioissa standardit ovat hyvin tiukkoja, joten niille tämäkin tapa soveltuu. Yleisesti eri organisaatioiden ohjelmistojen kokoja ei kuitenkaan pitäisi mennä vertailemaan keskenään, elleivät standardit satu olemaan täsmälleen yhteneväisiä.

Toimintopistelaskenta (Function Point Analysis) lienee pisimmälle kehitetty tapa määrittää ohjelmistojen kokoa sen toiminnallisuuden perusteella. Tässä yhteydessä ei pitäisi kuitenkaan puhua yhdestä ainoasta mittaustavasta, sillä nykyisin tunnetaan ainakin 35 erilaista toimintopistemenetelmää. Kaikki ne ovat kuitenkin perusperiaatteeltaan saman tyyppisiä. Perusajatuksena on kuvata ohjelmisto käyttäjän tunnistamien toiminnallisten osien avulla. Niitä voivat olla esimerkiksi raportit, kyselytehtävät, varastoitavat tietojoukot, tietojen ylläpitotehtävät ja automaattiset liittymät arvioitavan ohjelmiston ja muiden ohjelmistojen välillä. Mitattavalla ohjelmistolla toteutetut toiminnot jaetaan menetelmästä riippuen 2-10 luokkaan. Kokoa laskettaessa luetellaan ensin kuhunkin luokkaan kuuluvat toiminnot, sitten kunkin yksittäisen toiminnon monimutkaisuus arvioidaan ja sille annetaan tällä perusteella tietty pistemäärä. Yhteen lasketuista pistemääristä muodostuu ohjelmiston koko toimintopisteinä (fp). Seuraavassa taulukossa esitellään muutamia tunnetuimpia FPA-menetelmiä:

Menetelmän nimi:	**Käytön laajuus**:	**Lisähuomioita**:
IFPUG 4.0	Kaikkein laajimmalle levinnyt, etenkin suurissa amerikkalaisissa organisaatioissa	Perustuu täysin alkuperäiseen Alan Albrechtin menetelmään, samat 5 toimintotyyppiä ja 14 tasaustekijää
Experience 2.0	Suomessa kaikkein suosituin, muissa maissa muutama menetelmän käyttöön koulutettu	Laajennettu ja jonkin verran alkuperäisestä nykyaikaistettu versio, 6 toimintotyyppiä, algoritmit lisätty. Ei tasaustekijöitä
Mark II	Suosituin Isossa-Britanniassa ja Irlannissa, joitakin käyttäjiä Hong Kongissa ja Kanadassa	Vain 2 toimintotyyppiä joiden lisäksi lasketaan tiedostoviittauksia. 5 uutta tasaustekijää alkuperäisten lisäksi. Tosin kouluttajat nykyään suosittelevat tasaustekijöiden käytön laiminlyömistä
Feature Points	Ei laajassa käytössä, jonkin verran USAssa	Capers Jonesin laajennus alkuperäiseen menetelmään, toimintotyyppeihin lisätty algoritmit
Full Function Point	Kokeilukäytössä lähinnä Kanadassa ja USAssa	Erityisesti sulautettujen ohjelmistojen laskentaa varten kehitteillä oleva laajennus alkuperäiseen menetelmään, sisältää useita erilaisia algoritmisia toimintotyyppejä

Tässä esitellyt menetelmät ovat kaikki melko yleiskäyttöisiä. Näiden lisäksi yllä mainittuihin 35 menetelmään sisältyy joukko hyvin suppealle ohjelmistojoukolle soveltuvia murteita. Kansainvälinen standardointiorganisaatio ISO on jo useiden vuosien ajan ollut kehittämässä standardia ohjelmistojen toiminnallisen koon laskemiseen (functional sizing). On ilmeistä, että kaikki yllä mainitut menetelmät tulevat täyttämään standardin vaatimukset joko sellaisenaan tai hyvin pienin muutoksin. Työhön saattaa kuitenkin kulua vieläkin jopa vuosia, joten täyttä varmuutta asiasta ei voi olla kenelläkään.

Takaisinlaskenta (backfiring) on eräänlainen yhdistelmä toimintopistelaskennasta ja lähdeohjelman lausemäärien laskennasta. Se perustuu Capers Jonesin kokoamaan konversiotaulukkoon, jonka avulla lausemäärät voidaan muuttaa toimintopisteiksi. Nykyisin tässä taulukossa on mukana reilusti yli 500 ohjelmointikieltä. Jokaisesta kielestä kerrotaan nimi, kielen taso sekä kielen ilmaisuvoima. Viimeksi mainittu tarkoittaa lausemäärää yhtä toimintopistettä kohti. Taulukon tieteellistä luotettavuutta on jonkin verran kritisoitu, tai ainakin epäilty, mutta käytännössä se vaikuttaa toimivan ainakin kohtalaisesti.

Jos takaisinlaskentamenetelmän mittaustarkkuutta halutaan parantaa, se edellyttää pitkäjänteistä tietojen keruuta omassa organisaatiossa. Yleisen taulukon rinnalle voi periaatteessa kuka tahansa muodostaa oman konversiotaulukkonsa. Se edellyttää ainoastaan kaksinkertaista koon mittaamista samoista ohjelmistoista. Yleensä tämä on helpointa tehdä juuri uuden sovelluksen kehittämisprojektin päättyessä. Hyvin useissa organisaatioissa tämän vaiheen mittaaminen tehdään jo nykyisin toimintopistelaskennan avulla. Tämän rinnalle tarvitsee ottaa samanaikainen lähdeohjelmalauseiden laskenta. Näin saadaan muutaman päättyneen projektin jälkeen varmuus siitä, kuinka hyvin Jonesin taulukot sopivat omaan organisaatioon. Onneksi yksikään ohjelmistotalo ei tarvitse kaikkia Jonesin mittaamia kieliä ja kehittimiä. Tyypillisesti alle 10-rivinen taulukko riittää.

Oheiseen taulukkoon on poimittu muutama esimerkkikieli Jonesin taulukosta:

Kieli:	**Kielen nominaalinen taso**:	**Keskim. lausemäärä/fp**:
Assembler	1.00	320
C	2.50	128
COBOL	3.00	107
PL/I	4.00	80
C++	6.00	53
Visual Basic	10.00	32

On hyvä huomata, että Capers Jones on käyttänyt taulukkoa muodostaessaan toimintopistemenetelmänä IFPUG 4.0 menetelmää, mutta se ei vaikuta lainkaan taulukoiden käyttökelpoisuuteen sellaisissakaan taloissa, joissa toimintopistelaskenta tehdään jonkin muun edellä mainitun menetelmän mukaisesti. Myös viimeisen sarakkeen arvojen tarkkuudesta lienee hyvä todeta, että mitä pienempiin ohjelmistokokonaisuuksiin mittaamisessa mennään, sitä epävarmempia tulokset ovat. Yksittäisissä ohjelmissa ilmaisuvoima voi vaihdella jopa –40 - +40 % taulukossa esitetystä arvosta. Koska lähdeohjelmalauseiden laskenta on syöttöaineistona takaisinlaskentamenetelmässä, periytyvät ensin mainitun hyvät ja huonot ominaisuudet luonnollisesti myös jälkimmäiseen. Takaisinlaskentamenetelmän ehdottomana omana vahvuutena on kuitenkin se, että eri välineillä toteutettujen ohjelmisto-osien koot voidaan laskea yhteen.

Ohjelmistoprojektityypit

Ohjelmistojen koon mittaamisen haasteet ja tavoitteet vaihtelevat sen mukaan, millaisessa yhteydessä mittaaminen suoritetaan. Tarvittavien lähtötietojen saanti on välillä helppoa, välillä suorastaan mahdotonta. Ohjelmistokehitystyötä ja sen toimivuutta voidaan tarkastella pääsääntöisesti eri tyyppisten projektien kautta. Ahtaasti ottaen projekteilla tarkoitetaan vain tietyn kokoisia ja täysin ainutkertaisia hankkeita, mutta tässä yhteydessä käsite tulee ymmärtää mahdollisimman laajasti. Ainakin seuraavat 7 ohjelmistotoimintaan sisältyvää projektityyppiä voidaan tunnistaa:

- Uuskehitysprojekti
- Lisäävä ylläpito
- Muuttava ylläpito
- Konversio
- Vuotuinen kunnossapito
- Valmisohjelmiston hankinta
- Työn ulkoistaminen (outsourcing)

Uuskehitysprojektilla tarkoitetaan tässä kokonaisen uuden ohjelmiston kehittämishanketta. Ohjelmisto voi olla joko räätälöity tai tuotteeksi tarkoitettu. Se voi koostua yhdestä tai useammasta sovelluksesta ja kehitystyössä voidaan käyttää enemmän tai vähemmän valmiita komponentteja.

Lisäävä ylläpito on pienimuotoisempaa, ja kohdistuu aina jo olemassa olevan ohjelmiston laajentamiseen. Ohjelmiston toiminnallisuutta siis lisätään. Hyvin usein uuden tuoteversion tekemiseen liittyy lisäävää ylläpitoa. Räätälöityjen ohjelmistojen tapauksessa tähän hanketyyppiin sisältyvät käyttäjän toivomuksesta ohjelmistoon tehtävät uudet ominaisuudet.

Muuttava ylläpito tarkoittaa tässä yhteydessä pienimuotoista, muutamaan ohjelmiston toimintoon kohdistuvaa kehittämistä. Erotukseksi edellisestä, varsinaista uutta toiminnallisuutta ohjelmistoon siis ei tehdä, mutta yksittäisiä näyttöjä ja raportteja muutellaan ja ehkä joitakin piirteitä myös poistetaan.

Konversioprojektit voivat olla hyvinkin erilaisia keskenään. Tässä yhteydessä niillä tarkoitetaan hankkeita, joissa kokonainen ohjelmisto käydään läpi ja siihen kohdistuu jostakin syystä paljon saman tyyppisiä muutoksia. Esimerkkeinä Y2K-projektit ja Euro-muutosprojektit. Myös teknologian vaihtoon liittyvät hankkeet ovat usein tätä tyyppiä (esimerkiksi siirtyminen unix-ympäristöstä nt-maailmaan tai tietokannan hallintajärjestelmän vaihto).

Vuotuinen kunnossapito sisältää kaiken sen ohjelmiston ylläpitotyön, jota ei tilata erikseen edellä mainittuina hankkeina. Hyvin usein tähän sisältyy myös jonkin verran tuotannon hoitoa, vaikka se puhdasoppisesti pitäisikin putsata ohjelmistokehittämisestä kokonaan. Yleensä kyseessä on hyvin pienimuotoinen ylläpitotyö.

Valmisohjelmiston hankinta on myös oma projektityyppinsä. Ohjelmistot voivat olla hyvinkin erilaisia, mutta yleensä ostaja saa hankittavasta ohjelmistosta vain suoritettavan koodin. Lähdeohjelmat eivät sisälly toimitukseen. Tämän tyyppisiin hankkeisiin me ohjel-

mistoammattilaiset törmäämme sekä ostajina että myyjinä.

Ulkoistamisprojektit ovat yleensä suhteellisen laajaa sovellussalkkua koskevia. Ohjelmistojen ylläpitoon, käyttötoimintaan tai omistusoikeuteen liittyvät kysymykset ovat niissä keskeisiä. Pääosassa ovat valmiit sovellukset. Kehitteillä mahdollisesti olevat ohjelmistot muodostavat vain marginaalisen ongelman näissä tapauksissa.

Kuten yllä olevista määritelmistä voi päätellä, kaikkiin hankkeisiin eivät erilaiset koon mittaamismenetelmät sovellu yhtä hyvin. Seuraavassa taulukossa tarkastellaan eri menetelmien sopivuutta eri projektityyppeihin. Näkökulma painottuu etukäteisarviointiin. Jos se kohdistettaisiin jälkikäteen tehtävään tuottavuuden mittaamiseen, tulos näyttäisi huomattavasti positiivisemmalta. Taulukossa olevat merkinnät tarkoittavat seuraavaa:

++
tarkoittaa, että menetelmä sopii erinomaisesti

+
tarkoittaa, että menetelmää voidaan käyttää

O
tarkoittaa, että rajatuissa tapauksissa menetelmää voidaan käyttää

-
tarkoittaa, että menetelmä ei sovellu

Koon mittausmenetelmien soveltuminen eri tyyppisten hankkeiden etukäteisarviointiin

Projektityyppi:	Rivimäärien laskeminen	Lausemäärien laskeminen	Toimintopiste-laskennat	Takaisin-laskenta	Dokumentti-sivumäärät
Uuskehitysprojektit	-	-	++	-	-
Lisäävä ylläpito	-	-	++	-	-
Muuttava ylläpito	+	++	+	++	+
Konversioprojektit	O	O	+	++	O
Vuotuinen kunnossa-pito	O	O	+	++	+
Valmisohjelmiston hankinta	-	-	+	-	O
Ulkoistamisprojektit	O	O	+	++	O

Koonmittausmenetelmien vertailua

Järjestäytyneet FPA-entusiastit kaikkialla yrittävät vakuuttaa, että heidän menetelmänsä on ainoa oikea ja sitä on sovellettava kaikissa tapauksissa, vaikka se olisi kuinka vaikeata. Kiistatta toimintopisteiden laskenta on kuitenkin liian aikaa vievää ja hankalaa monissa tapauksissa. Vanhojen sovellusten osalta riittää useimmiten jonkin lähdeohjelmalauseisiin perustuvan menetelmän käyttö.

Tässä artikkelissa ei vertailla eri toimintopistelaskentamenetelmiä keskenään, vaikka siitäkin riittäisi melko paljon asiaa. Totuus on kuitenkin, että kun on oppinut yhden menetelmistä kunnolla, ei muiden oppiminen ole kovinkaan vaikeaa. Sertifikaatteja osaamisesta myönnetään muutaman

menetelmän osalta, mutta osaaminen on toki

todistuksia tärkeämpää. Ohjelmistotyyppi vaikuttaa jonkin verran parhaan toimintopistemenetelmän valintaan, mutta perusohjeeksi voi sanoa, että kannattaa käyttää mahdollisimman paljon vain yhtä menetelmää. Mikäli eri FPA-menetelmien väliset konversiosäännöt tunnetaan, voi tietenkin käyttää useampiakin. Konvertointi vain on valitettavan usein mahdollista ainoastaan toiseen suuntaan.

Summa summarum

Kehitettävän tai käyttöön otettavan ohjelmiston koon laskenta on hieno juttu, mutta parhaimmillaankin se kertoo vain osan totuudesta. Päättäjät ovat ensisijaisesti kiinnostuneita projektien työmääristä, kestoista ja työn tuottavuudesta, sekä tietenkin näiden perusteella laskettavista kustannuksista ja tuotoista. Ohjelmiston koon lisäksi pitäisi tuntea aina projektin olosuhteet verrattuna muihin hankkeisiin. Myös riskien arviointi ja jopa niiden vaikutuksen mittaaminen ovat usein tarpeen. Joissain projektityypeissä myös uudelleenkäytön mittaaminen on nousemassa entistä tärkeämpään rooliin. Kokemushistoriaa tarvitaan lisäksi aina. Ilman sitä ei ennuste ole arvausta parempi, riippumatta siitä onko edellä mainittuja analyyseja tehty vai ei. Koon mittaaminen on siis tärkeätä, mutta se on pidettävä omalla paikallaan, nostamatta sitä jalustalle. ”Silver bulletteja” ei ole mittaamiseenkaan tarjolla.

Pekka Forselius,
STTF Oy