OHJELMISTOJEN LAADUN JA KOON MITTAAMINEN

Mitat ja mittaus

- ”You can’t control what you can’t measure” – Tom DeMarco, 1982.
- DeMarcon toteama on kaikkien mittausspesialistien motto: ilman mittausta ei ole ohjausta.
  - Väite on tietenkin liioiteltu. Kaikkia merkittäviä tietoja ei osata, voida tai haluta mitata. Mutta: **mittaamalla ohjaus helpottuu.**

Mitat ja Mittaus

- Yrityksen laatutavoitteet voidaan esittää erityisten indikaattorien tavoitearvojen avulla (haluttu taso)
  - *Mittausohjelma* määrittelee indikaattorit ja mittaukset
- Esimerkiksi *asiakastyytyväisyyden* ja - *uskollisuuden mittaaminen* usealla indikaattorilla
  - Mielipidekyselyt
  - ”Net Promoter Score”
  - Häiriöilmoitusten ja tukipyyntöjen lukumäärän kehitys (suhteessa käyttäjämäärään)
- Mittareiden halutaan *arvioida*, *ennustaa* tai *tilastoida* tuotteita, tuotantoprosesseja, resursseja ym. ohjauksen ja kehittämisen tarpeisiin

Mitat ja Mittaus

- Vaikka mittauksia ei tehtäisikään täydessä laajuudessaan (tai tehdään ne vain kertaluonteisesti), mittausohjelma toimii *kommunikaation* ja *sitouttamisen* välineenä
  - Mittausohjelman toimeenpanoa tärkeämpää on saada kaikki ymmärtämään ja hyväksymään mitä tavoitellaan
- Huom! - mittausten käyttö *palkitsemisen* tai *henkilökohtaisen suoriutumisen arvioinnin* perusteena kääntyy kuitenkin helposti itseään vastaan
  - Voi olla monta tapaa saada mittarit näyttämään haluttuja arvoja, eivätkä kaikki ole tarkoituksenmukaisia
  - Kehittäjät vierastavat henkilöön kohdistuvia mittauksia ja vaihtavat työpaikkaa

Mitta, mittaus, mittari

- *Mitta* tai *metriikka* (measure, metric) on luku tai symboli, joka karakterisoi kiinnostuksen kohteen (entity) tiettyä ominaisuutta (attribute)
- *Mittaus* (measurement) on prosessi, joka liittää luvun tai symbolin jonkin tosimaailman olion ominaisuuteen sen kuvailemiseksi selvästi määritellyn säännön mukaan
- *Mittari* tarkoittaa yleensä mittaa, tai mittaa ja sen mittauksessa käytettyä menetelmää (~mittauksen määrätty suoritustapa)
  - Tekniikassa mittarilla tarkoittaa yleisesti myös *mittalaitetta*, jolla tietty mittaus tehdään ja josta mittauksen tulos voidaan suoraan lukea

Katso myös: Mittaaminen, http://fi.wikipedia.org/wiki/Mittaaminen

Mittaus

- Edellä annettu mittauksen määritelmä sopii tilanteisiin, joissa on olemassa ”luonnonlain” omainen, kokeellisesti vahvistettu *syy-seuraussuhde* prosessissa mitatun *ominaisuuden* ja *mitan* välillä (fysikaaliset systeemit)
  - Esimerkiksi hyvin pienten ja hyvin suurten lämpötilojen mittaamiseen käytetään aivan erilaisia menetelmiä, jotka on kuitenkin tieteellisesti (teorian ja kokeiden avulla) todettu *päteviksi* ja *luotettaviksi*

Mittaus

- Kuten myöhemmin näemme, ohjelmisto- ja ohjelmistoprosessimittareissa syy-seuraussuhteet eivät ole aina kovin vahvasti perusteltuja
- Kaner ja Bond[1] korostavat syy-seuraussuhteen merkitystä ohjelmistomittareissa määrittelemällä *mittauksen* seuraavasti
  - "Measurement is the empirical, objective assignment of numbers, according to a rule derived from a model or theory, to attributes of objects or events with the intent of describing them"
  - Taustalla on siis oltava *malli* tai *teoria*, joka selittää uskottavasti kohteen ominaisuuden ja sen mitan välisen kausaalisen yhteyden, ei pelkästään tilastollinen korrelaatio

[1]KANER, Cem, et al. Software engineering metrics: What do they measure and how do we know?. In: *In METRICS 2004. IEEE CS*. 2004.

Suorat ja johdetut mittarit

- Mittarit voivat olla *suoria* (direct) tai *johdettuja* (indirect)
  - Suorien mittareiden mittaukset kertovat sellaisenaan kohteen tilasta (luontaisesti valideja). Esimerkiksi lämpötila on suora mittari. f(ominaisuus) → arvo
  - Johdettujen mittareiden mittaukset saadaan yhden tai useamman suoran mittarin mittausten funktiona. Esimerkiksi kuukauden keskilämpötila on johdettu mittari. h($f(o_1)$,$g(o_2)$,…) → arvo
  - Monien ohjelmistomittarien kohdalla voidaan kuitenkin kysyä, kuinka suoria ja valideja "suorat" mittarit ovat – esim. mitkä asiat oikeasti luokitellaan vioiksi tai kuinka tarkkaan vikojen korjaamisesta todella pidetään kirjaa
- *Sijaismittaa* (surrogate metric) käytetään, kun itse ominaisuuden mittaaminen on mahdotonta
- Mittarien avulla halutaan *arvioida*, *ennustaa* tai *tilastoida* mittaajan kannalta mielenkiintoisia kohteen ominaisuuksia

Ohjelmistomittarit

- Ohjelmistotuotannossa on paljon suoria ja johdettuja mittareita, *ohjelmistomittareita* (software metrics).
- Ohjelmistomittareilla ja laatutekijöillä on yhteys. Useimpien laatutekijöiden sisältä voidaan löytää osatekijöitä, joiden toteutumista voidaan arvioida mittareilla.
- Kun laatutekijän osatekijöiden mittarit näyttävät, että osatekijät toteutuvat ohjelmistossa, laatutekijä toteutuu ohjelmistossa *ainakin osittain*.
  - Miksi vain osittain?
    - laatutekijä on yleensä enemmän kuin osatekijöidensä summa
    - kaikille osatekijöille ei ole varmoja mittareita
    - Mittareiden taustalla olevissa teorioissa ja niiden kokeellisessa varmennuksessa on puutteita

LAATUMITTARIT

Laatumittarit

- Laatumittarit mittaavat ohjelmistotuotteen tai ohjelmistoprosessin laatua.
  - Laatumittari on funktio, jonka syötteenä on ohjelmistoon liittyvä data ja tuloksena dataa kuvaava numeerinen arvo.
  - Laatumittarin numeerinen arvo (mittauksen tulos) kertoo, missä määrin mitattu data täyttää mittarilla kuvattavan laatuattribuutin (quality attribute)
    - Laatuattribuutti tarkoittaa jotakin laadun ominaisuutta. Esimerkiksi laatutekijät ja -osatekijät (characteristics) ovat laatuattribuutteja
    - Arvon tulkinta vaatii usein sen suhteuttamista *kokeellisesti* tai *kokemusperäisesti* määriteltyyn tavoitetasoon

”Onks kymppi paljon vai vähän?”

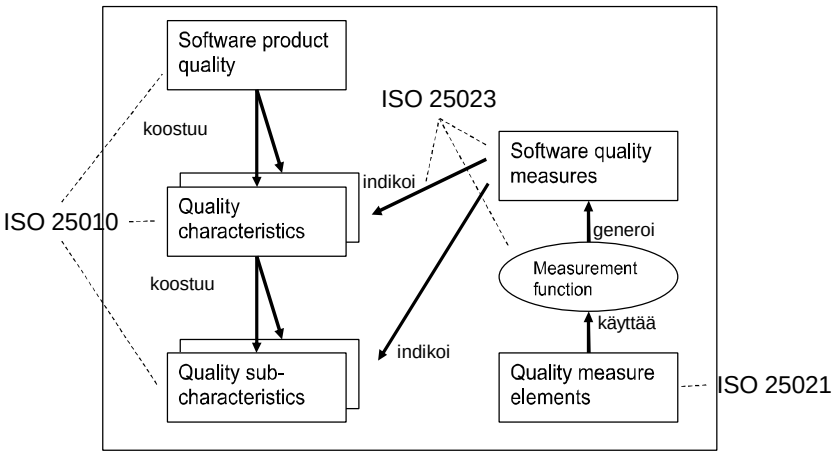
Laatumittarien tarkoitus

- Laatumittarit ovat laadunvarmistuksen työkaluja. Niiden avulla
  - varmennetaan, että ohjelmistoprojektit etenevät oikein:
    - verrataan toteutunutta laatua ja laatutason vaihtelua suunniteltuun laatuun
    - verrataan toteutunutta aikataulua ja budjettia suunniteltuun aikatauluun ja budjettiin
  - tunnistetaan, milloin kehitys- tai ylläpitoprosessia pitää parantaa:
    - kerätään historiatietoa toteutuneista projekteista ja tuotteista
- Mittarit voivat olla
  - prosessikohtaisia: miten hyvin prosessi ja sen ilmentymä (projekti) vastaavat toisiaan
  - tuotekohtaisia: miten hyvin tuote ja sen laatutekijät vastaavat toisiaan.

Laadun mittaus SQuaRE 2502n –standardien mukaan

- Laatumalli 25010 määrittelee käytön laadun ja tuotelaadun piirteet, joista valitaan kullekin projektille relevantit piirteet
- 25020 ja 25021 luo puitteet ohjelmistomittareiden määrittelylle
- 25022 ja 25023 antavat suosituksia ja esimerkkejä, mitä mittareita kunkin laatupiirteen kohdalla kannattaa käyttää
- Huom - prosessin mittaamiseen oma standardinsa
  - ISO/IEC 15504 SPICE http://www.sfsedu.fi/materiaalit

Laadun mittaus - tuotelaatu



Laadun mittaus

- Ohjelmiston laatuun liittyviä mitattavia ominaisuuksia standardi kutsuu kvantifioitaviksi ominaisuuksiksi (property to quantify, ei ed. kuvassa)
- Ominaisuus kvantifioidaan käyttämällä mittausmenetelmää (measurement method), joka on toimenpiteiden sarja, joka liittää ominaisuuteen tietyn tyyppisen (measurement scale) arvon
- Saatua ominaisuuden arvoa kutsutaan perusmitaksi (quality measure element)

Laadun mittaus

- Perusmitat kytketään laatupiirteisiin johdettujen laatumittareiden kautta (software quality measures)
  - Kuhunkin laatumittariin liittyy mittausfunktio (measurement function) eli algoritmi, joka (yleensä) yhdistää useamman perusmitan yhdeksi laatumittarin arvoksi
  - Samaan laatupiirteeseen liittyy tyypillisesti useita laatumittareita, joiden avulla laatu kvantifioidaan
  - Laatumittarin määrittelystä ilmenee, mikä on *peruste mittarin ja laatupiirteen yhteydelle*

Tuotelaadun mittarit

- ISO 25023 määrittelee kullekin laatupiirteelle yhden tai useamman mittarin, ja antaa suosituksia niiden käytöstä
  - Aksellilla ”Erittäin suositeltava – Suositeltava – Harkinnan mukaan”
  - Soveltuvuus ulkoisen ja/tai sisäisen laadun mittaamiseen
- Määrittelee myös perusmitat, joihin johdetut laatumittarit perustuvat sekä perusmittojen käyttämät mittayksiköt ja -asteikot (esim. koko, aika, työmäärä)
- Eri ohjelmistomittareita ja niiden suhdetta laatuun tarkastellaan lisää myöhemmin kurssilla!

Laatumittarien valinta

- Laatumittareita valittaessa määrä ei korvaa laatua. On parempi valita pieni joukko hyvin määriteltyjä mittareita kuin mitata kaikkea mahdollista
  - SQuaRE 25022, 25023 antavat suosituksia ja esimerkkejä
- Valittavat mittarit riippuvat siitä,
  - mikä on kiinnostavaa ja
  - mitä on ylipäänsä mahdollista mitata.
- Jotta mittarien käyttö olisi tehokasta, sen täytyy olla keskitettyä. Mittarit täytyy määritellä ylhäältä alaspäin (ensin tavoite, sitten siihen sopivat mittarit), sillä erilaisia mittareita on valtava määrä.
- Koska kaikkea ei voi eikä kannata mitata, valitut mittarit on perusteltava hyvin. Tähän voidaan käyttää *Goal Question Metric* -menetelmää (GQM, Basili & Rombach, 1988).

Goal - Question - Metric

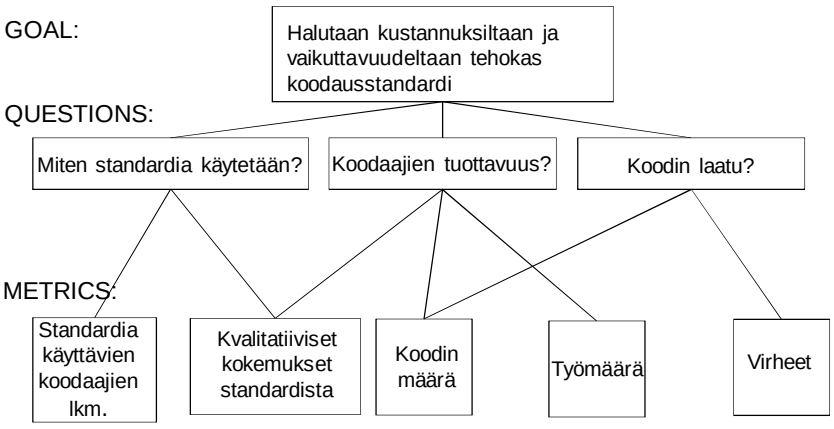
- GQM[1] perustuu näkemykseen, jonka mukaan järkevään mittaukseen yrityksen tarvitsee
  1. Määritellä yritys- ja projektitason *tavoitteet*
  2. Löytää ne yritys-, projekti- ja tuotetason *tiedot*, jotka määrittävät tavoitteiden toteutumista yrityksen *toiminnan* tasolla
  3. Suunnitella kehys, jonka avulla tiedot *kerätään* ja *tulkitaan* kuvaamaan tavoitteita ja niiden toteutumista
- Eli:
  1. Mitä halutaan tietää ja miksi?
  2. Mitä pitää selvittää yrityksen tuotteista ja prosesseista, jotta voidaan muodostaa tarvittava näkemys?
  3. Miten nämä tuotteita ja prosesseja koskevat tiedot käytännössä kerätään ja miten niitä pitää tulkita?
- GQM tarjoaa keinot määritellä kysymyksiin vastaava mittarijoukko.

[1] http://www.cs.umd.edu/~mvz/handouts/gqm.pdf

GQM-kolmitasomalli

- GQM on kolmitasoinen malli:
  - Käsitetaso (GOAL)
    - Selvitetään halutut tuote-, prosessi- ja resurssitavoitteet. Eli mittauksen kohde ja mitä siitä halutaan tietää tai muuttaa ja kenen näkökulmasta.
  - Toimintataso (QUESTION)
    - Tavoitteiden perusteella johdetaan joukko kysymyksiä, jotka kuvaavat kunkin tavoitteen toteutumista sitä luonnehtivien asiantilojen kautta
    - Kukin kysymys luonnehtii mittauksen kohdetta laatuun liittyvän ongelman tai faktan kannalta ja tietystä näkökulmasta.
  - Kvantitatiivinen taso (METRIC)
    - Kuhunkin kysymykseen etsitään sellainen (objektiivinen tai subjektiivinen) mittari/mittarit, jonka avulla kysymykseen saadaan kvantitatiivisia vastauksia.
- GQM:lla ei kannata tehdä liian kunnianhimoisia suunnitelmia. Aluksi on parempi etsiä perusmittarit ydinkysymyksiin ja sen jälkeen vähitellen parantaa mittarikehystä lisäämällä kysymyksiä ja niihin liittyviä mittareita.

GQM-esimerkki 1



GQM-esimerkki 2

Tavoite	*Tarkoitus*	Parannetaan
	Kohde (prosessi)	muutospyyntöjen käsittelyn
	Asia/ongelma	ajallista täsmällisyyttä ja tehokkuutta
	Näkökulma	projektipäällikön näkökulmasta.
Kysymys 1		Mikä on nykyinen muutospyynnön käsittelynopeus?
Mittarit		• Muutospyynnön keskim. käsittelyaika • Keskipoikkeama • Käsittelyaikatavoitteen ylärajan ylittävien pyyntöjen osuus prosentteina
Kysymys 2		Tehostuuko käsittely parannusten myötä?
Mittarit		• Nykyinen keskimäär. käsittelyaika prosentteina valitusta perustasosta • Projektipäällikön oma subj. arvio

GQM

- Usealla GQM mallilla voi olla yhteisiä kysymyksiä ja mittareita
  - Ohjaa siihen, että mittauksen suorituksessa voidaan ottaa huomioon eri näkökulmat mitattavaan asiaan
  - Sama mittaus voi tuottaa erityyppisiä arvoja näkökulmasta riippuen

KOON MITTARIT

Ohjelmiston kokomittarit

- Huomattavassa määrässä ohjelmiston laatumittareita mitataan niiden osana ohjelmiston kokoa. Vaikka erilaisia kokomittareita on lukemattomia, kaksi on ylitse muiden:
  - KLOC (Kilos Lines Of Code): Kuinka monta tuhatta *koodiriviä* on ohjelmistossa.
  - FP (Function Points), toimintopisteet: Kuinka paljon *toiminnallisuutta* on ohjelmistossa.
- Muita kokomittareita:
  - Montako lausetta on ohjelmistossa (KLOC-variantti).
  - Montako luokkaa/metodia/funktiota on ohjelmistossa.
  - Montako *riippumatonta polkua* (independent path) on ohjelmistossa tai (yleensä) sen osassa.
    - Riippumaton polku on sellainen suoritusreitti ohjelmistossa, että sitä ei voi saada yhdistämällä muista riippumattomista poluista.

KLOC

- KLOC on yleisin ohjelmiston kokomittari. Sillä mitataan ohjelmiston kokoa koodiriveinä joko kommenttien kanssa tai ilman niitä.
- KLOC on (suhteellisen) selkeä mittari, joka kuvaa koon lisäksi yllättävän hyvin kompleksisuutta: mitä isompi ohjelmisto/ohjelma/luokka/metodi tms. on, sitä mutkikkaampi se yleensä on (ja esimerkiksi sitä hankalampi testata ja ylläpitää).
- KLOC:n heikkous on sen ohjelmointikieliriippuvuus. Eri ohjelmointikielillä toteutetuista ohjelmistoista lasketut KLOC-luvut eivät ole keskenään verrannollisia.
- Myös koodaustyylit vaikuttavat jonkin verran koodirivien määrään, mutta isoilla ohjelmistoilla erojen suhde koodirivien kokonaismäärään on melko pieni.
  - KLOC:n sijaan kannattaa laskea esimerkiksi puolipisteitä (so. lauseita), jos koodirivin pituuden vaihtelu osoittautuu ongelmaksi.

KLOC-ongelmia

- Koska KLOC riippuu käytetystä ohjelmointikielestä ja koodaustavasta, KLOC-arvot ovat monen, laadun kannalta vähemmän tärkeän muuttujan summa.
- Lisäksi KLOC:lla on ikävä ominaisuus: se on nimensä mukaisesti koodirivien määrä, joten **sitä ei voida laskea, ennen kuin koodi on kirjoitettu.**
- Miten lasketaan kirjastoista/kehyksistä *uudelleenkäytetyt* koodirivit??
  - Lähdekoodia ei edes usein saatavissa
- KLOC-arvoja voidaan arvioida, mutta arviot ovat summittaisia. Olisi parempi, jos jo määrittely- tai suunnitteluvaiheessa ohjelman koolle voisi antaa luotettavan arvion.
- Määrittelystä laskettava kokoarvio on mahdollista *toimintopisteillä* (function points). Niiden avulla jo melko yleisistä määrityksistä voidaan arvioida, miten paljon toiminnallisuutta ohjelmistossa on.
  - Arviot tarkentuvat projektin edetessä ja ohjelmiston vaiheittain valmistuessa

Toimintopisteet

- Toimintopisteiden laskemisen pääidea on tuloksen riippumattomuus toteutusteknologiasta: lasketaan *toiminnallisuuden* määrä
  - Ei väliä koodataanko C:llä vai Javalla, toiminnallisuus on sama
  - Tuo esiin *tuottavuuserot* -> kuinka paljon koodia ja työtä tarvitaan tietyn kokoisen toiminnallisuuden toteuttamiseen
- Toimintopisteet on esitelty niinkin aikaisin kuin vuonna 1979 (Albrecht). Vaikka menetelmää on tämän jälkeen kehitetty eteenpäin, on alkuperäinen malli edelleen yllättävän toimiva.
  - Tällä hetkellä (alkuperäisestä) toimintopistemalleista huolehtii kokonainen organisaatio: International Function Point Users Group, IFPUG (www.ifpug.org).
- Toimintopistestandardeja on useita, esimerkiksi suomalainen FiSMA –menetelmä (ISO/IEC 29881:2010, myös suomenkielisenä SFS –standardina)
  - http://www.fisma.fi/toiminta/menetelmat/ ja http://www.fisma.fi/wp-content/uploads/2006/09/fisma_fsm_method_11.pdf

Toimintopisteiden laskenta 1

Perinteiset toimintopisteet lasketaan kolmessa vaiheessa:

1. Lasketaan *raakapisteet*
   - Arvioidaan ohjelmiston ”komponenttien” määrä:
     - syötteiden määrä, kuten lomakkeet
     - tulosteiden määrä, kuten raportit ja virheilmoitukset
     - kyselyiden määrä, kuten generoitavat näytöt
     - loogisten tiedostojen määrä, kuten tietokannan taulut
     - ulkoisten liittymien määrä, kuten käyttöliittymät, liittymät muihin järjestelmiin ja liittymät muihin sovelluksiin
   - Määritellään kullekin komponentille painokerroin sen mukaan, miten työlääksi sen toteuttaminen arvioidaan (helppo/yksinkertainen, keskimääräinen, vaikea/monimutkainen):
     - syötteet: 3 (yksinkertainen), 4 (keskimääräinen), 6 (vaikea)
     - tulosteet: 4, 5, 7
     - kyselyt: 3, 4, 6
     - loogiset tiedostot: 7, 10, 15
     - ulkoiset liittymät: 5, 7, 10
   - Lasketaan komponenttimäärien ja painokertoimien tulojen summa.

Toimintopisteiden laskenta 2

2. Lasketaan ohjelmiston *kompleksisuuskerroin* (complexity factor)
   - Kompleksisuuskerroin saadaan arvioimalla 14 kysymystä asteikolla 0-5, missä
     - 0=ei vaikutusta, 1=satunnainen, 2=kohtalainen, 3=keskimääräinen, 4=merkittävä, 5=olennainen vaikutus.
   - Saadut arvot lasketaan yhteen.

3. Lasketaan toimintopisteet FP kaavalla

   FP = CFP*(0,65 + 0,01* RCAF), missä

   - CFP = raakapisteet
   - RCAF = kompleksisuuskerroin

Kompleksisuuskertoimen kysymykset

Kompleksisuuskertoimen RCAF kysymykset:

- Tarvitaanko luotettavaa tietojen varmistus- ja palautusmenettelyä?
- Tarvitaanko tietoliikenneominaisuuksia?
- Onko hajautettua prosessinhallintaa?
- Onko kyseessä suorituskykykriittinen ohjelmisto?
- Käytetäänkö järjestelmää raskaasti kuormitetussa ympäristössä?
- Tarvitaanko interaktiivista tietojen syöttöä ohjelmiston suoritusaikana?
- Täytyykö interaktiivinen tietojen syöttö synkronoida usealle näytölle tai operaatiolle?
- Päivitetäänkö tiedostoja interaktiivisesti ohjelmiston suoritusaikana?
- Ovatko syötteet, tulosteet, tiedostot tai kyselyt monimutkaisia?
- Onko ohjelmiston suorittama laskenta monimutkaista?
- Onko koodi tarkoitettu uudelleenkäytettäväksi?
- Ovatko ohjelmiston muunnokset ja asennus mukana suunnittelussa?
- Onko ohjelmisto suunniteltu toimivaksi useina versioina eri organisaatioissa?
- Onko ohjelmiston käytettävyys keskeisessä roolissa?

FP-esimerkki

- Olkoon meillä sovellus, jolle on tehty seuraavat kokoarvot:
  - 1 yksinkertainen syöte, 1 monimutkainen
  - 2 keskivertoa tulostetta, 1 monimutkainen
  - 1 helppo kysely, 1 keskiverto, 1 monimutkainen
  - 1 yksinkertainen tiedosto, 1 monimutkainen
  - 2 monimutkaista ulkoista liittymää
- Kompleksisuuskertoimeksi on saatu 41 (14 kysymystä, vajaa 3 pistettä/kysymys).
- Näillä arvoilla saadaan:
  - CFP = (1*3+1*6)+(2*5+1*7)+(1*3+1*4+1*6)+(1*7+1*15)+(2*10) = 9+17+13+22+20 = 81
  - RCAF = 41
  - FP = CFP*(0,65+0,01*RCAF) = 81*(0,65+0,41) = 81*1,06 = 85,86

FP:n edut ja haitat

- Edut:
  - Arviointia voidaan tehdä jo ennen projektin alkua, joten FP:n arvoja voidaan käyttää projektin aloituspäätöksen tukena.
    - ”Pikalaskurit”
  - FP ei riipu käytettävistä ohjelmointikielistä tai työkaluista -> eri teknologioilla toteutettujen projektien vertailtavuus
  - FP on osoittautunut melko luotettavaksi toiminnallisuuden mittariksi.
  - FP:n laskeminen mahdollistaa *yksikkökustannusperustaisen* projektin ohjauksen ja laskutuksen -> laskutus perustuu *toimitetun toiminnallisuuden* määrään, ei tehtyihin tunteihin
    - Vrt. ”Running Tested Features” ketterän kehityksen metriikka[1]

[1] http://xprogramming.com/articles/jatrtsmetric/

FP:n edut ja haitat

- Kritiikkiä:
  - Saadut FP-arvot ovat *osittain subjektiivisia* (menetelmästä riippuen). Ne riippuvat käytetyistä kertoimista, käytetystä FP-variantista ja arvioijasta. Myös *käsityönä tehtävän laskennan kalleutta* on moitittu (laskentaa on pyritty automatisoimaan).
  - FP on tarkoitettu dataintensiivisille sovelluksille. Se antaa liian pieniä arvoja algoritmi-intensiivisille sovelluksille. Tämä tosin voidaan korjata käyttämällä modernimpia FP-variantteja.
- Huomaa, että toimintopisteiden (toiminnallisuuden) määrä on vain *yksi* projektin työmäärään ja kustannuksiin vaikuttava tekijä[1, s. 51]
  - Esimerkiksi FiSMA:n ND21 tilanneanalyysimenetelmä uuskehitykseen ottaa huomioon 21 vaikuttavaa tuottavuus-tekijää, joista tuotteen laatuvaatimukset ovat yksi osuus
  - http://www.fisma.fi/toiminta/menetelmat/

[1] Pekka Forselius: Onnistunut tietojärjestelmän hankinta. Talentum, 2013.

FP:n ja KLOC:n suhde

- Koska sekä KLOC että FP mittaavat ohjelmiston kokoa, luonteva kysymys on, onko mittareilla yhteys.
- Tällainen yhteys on olemassa. Tietyllä ohjelmointikielellä yhden toiminnallisuuspisteen toteutus vie sovelluksesta riippumatta likimain saman verran koodirivejä (”backfired” function point count).
- Seuraavassa muutamia suhdelukuja (LOC/FP) (1 KLOC = 1000 LOC). Luvut ovat suuntaa antavia (täytyy kokeellisesti vahvistaa aika ajoin):
  - Assembler: 320 (esimerkki: 85,86*320=27475 riviä koodia)
  - C: 130-90
  - Cobol: 100
  - C++: 65-55
  - Java: 45 (esimerkki: 85,86*45=3863 riviä koodia)
  - Visual Basic: 30
  - Power Builder (sovelluskehitin): 15
  - SQL: 10
- Mitä ilmaisuvoimaisempi ohjelmointikieli on kyseessä, sitä pienempi on sen LOC/FP-suhdeluku.