Projektinhallinta: kustannusarvio

Ohjelmiston kustannusarviot

- Yleensä jo projektin tarjouksen osana on jonkinlainen kustannusarvio
- Projektin tärkeimmät kustannustekijät:
  - työvoimakustannukset (ylivoimaisesti suurin menoerä):
    - Palkat ja sosiaalikulut
    - Työtilat ym.
  - laitteisto- ja ohjelmistokulut
  - matkat ja koulutukset

Kustannusten arvioinnin ongelma

- Kustannusarvio joudutaan laatimaan aikaisin, sillä sitä tarvitaan neuvoteltaessa tarjouksesta
- Kokonaistyöpanoksen tarve riippuu olennaisesti laadittavasta ohjelmatuotteesta:
  - Minkälaisista osajärjestelmistä tuote koostuu?
  - Minkälaista ammattitaitoa tekijöiltä edellytetään?
  - Kuinka paljon työtä – kuinka kauan työ kestää?

Kustannusten arvioinnin tapoja

- Aiempaan kokemukseen perustuva arviointi
  - Ohjelmistotekniikan ja sovellusalueen asiantuntijat, analogiaperiaate
- Kustannusten mallintaminen
  - Matemaattinen malli, jolla ennustetaan kustannuksia tiettyjen lähtökohtatietojen ja arvioiden perusteella
  - Lukuisia malliin sisältyviä suureita kuitenkin arvioitava subjektiivisesti kokemukseen perustuen

Huomioitavaa

- Parkinsonin laki:

  “Work expands so as to fill the time available for its completion”

→Sovellus: mitä tahansa arvioidaan, projekti kuluttaa vähintään sen verran

→Budjetointi vaikuttaa suoritukseen

- Voidaan lähteä käytettävissä olevasta budjetista ja ”katsoa minkä verran toiminnallisuutta saadaan aikaisin”

Tuotos ja työpanos ohjelmistotyössä

- Työaika vaikuttaa kohtalaisen suoraviivaisesti kustannuksiin (palkat, tilakustannukset)
- Tarve arvioida tarvittavat tuotokset ja tietyn suuruiseen tuotokseen tarvittava työpanos (aika)
- Tuotoksen määrää mitaan tyypillisesti
  - ohjelmatuotteen **kokona** (koodirivejä) tai
  - ohjelmatuotteen sisältämän **toiminnallisuuden määränä**

Koodimäärään perustuva arviointi

- Tavallisin tuotoksen mitta on koodin määrä
  - Konkreettinen → Helposti miellettävissä, helposti (jälkikäteen) laskettavissa
- Toisaalta:
  - Koodin laatiminen on vain osa työstä
  - Koodirivin ilmaisuvoima riippuu käytetystä ohjelmointikielestä
  - Koodirivien määrän arviointi ei ole helppoa projektin alkuvaiheessa

Toiminnallisuuteen perustuva koon arviointi

- Tuotosten suuruutta voidaan arvioida myös ohjelmiston toiminnallisuudella

→Korkeampi abstraktiotaso

- Toiminnallisuuden arvion tuloksena yksittäinen luku, mielekkyys, tulkinta?
- Ohjelmointikielestä riippumaton mitta
- Arviointi enemmän konkretiaan sidottu kuin koodirivien määrän arviointi
- Arvioita voidaan myös käyttää ennustamaan koodirivien lukumäärää

Toimintopisteanalyysi (function point analysis)

- Toiminnallisten vaatimusten analysointi
- Toiminnallisuutta kuvaavat alkiot
  - Ulkoiset syötteet ja tulosteet
  - Ulkoiset liittymät (external interfaces)
  - Interaktiot käyttäjän kanssa
  - Järjestelmän käyttämät tiedostot
- Kunkin kompleksisuus arvioidaan→ painot → summataan → toimintopisteiden lukumäärä
- Modifioidaan vielä erilaisilla kompleksisuus-tekijöillä (mm. uudelleenkäyttö, hajautus)

Oliopisteanalyysi

- *Oliopisteanalyysi* (object point analysis) on uudempi tekniikka, jossa on sama perusidea
- Helpompi analysoida korkean tason järjestelmäkuvauksesta kuin toimintopisteitä
- Oliopisteitä lasketaan
  - Näytöistä (number of separate screens),
  - Tuotettavista raporteista (number of reports produced)
  - Toteutettavista moduuleista (number of modules to be developed)

Painokertoimet

- Elementin pisteytykseen vaikuttaa se, miten hankalaksi elementin toteutus katsotaan
  - Yksinkertainen näyttö: 1op
  - Keskivaikea näyttö: 2op
  - Vaikea näyttö: 3op
  - Yksinkertainen raportti: 2op
  - Keskivaikea raportti: 5op
  - Vaikea raportti: 8op
  - Moduuli/komponentti: 10op

Kustannusmallit

- Kustannusestimaatti saadaan tuotetta, projektia ja prosessia kuvaavien attribuuttien arvojen funktiona
- Asiantuntijat arvioivat mainittujen attribuuttien arvot
- Kustannusmallien peruskaava

$$Effort = A \times Size^{B} \times M$$

  - *A* vakio (ainakin periaatteessa organisaatiokohtainen)
  - *Size* = ohjelmiston koko
  - *B* suurten projektien kompleksisuuslisä
  - *M* kuvaa tuotteen, prosessien ja kehittäjien laatua

COCOMO II

- Empiirinen malli: perustuu kokemukseen suuresta määrästä projekteja
- Hyvin dokumentoitu
- “Riippumaton”: ei sidoksissa ohjelmistoyrityksiin
- Pitkä historia alkaen v. 1981, nykyisin COCOMO 2 –malli
- Alun perin vesiputousmallin kustannusarvioinnin tueksi
  - Nykyisin laajempi, mm. uudelleenkäytettävät komponentit
  - *tarkentuvat osamallit*: lisäinformaation myötä lisääntyvä tarkkuustaso kustannusten estimoinnissa

COCOMO II:n osamallit

- Application composition model
  - käytetään, kun ohjelmisto tehdään suurelta osalta valmiista komponenteista
- Early design model
  - käytetään, kun vaatimusmäärittely on tehty, mutta suunnittelu ei ole alkanut
- Reuse model
  - käytetään laskettaessa uudelleenkäytettävien komponenttien adaptoimisen ja integroinnin vaatimaa työmäärää
- Post-architecture model
  - käytetään kun järjestelmäarkkitehtuuri on valmis ja järjestelmästä on yksityiskohtaista tietoa

Application composition model

- Malli on tarkoitettu *komponenteista koottavien ohjelmien arviointiin*, mutta sitä voidaan kyllä käyttää arvioitaessa karkeasti mitä tahansa ohjelmistoa

$$PM = (NAP \times (1 - \%reuse/100)) / PROD$$

- PM= vaadittu työmäärä (henkilötyö-kk)
- NAP = oliopisteiden lukumäärä,
- %reuse = uudelleen käytettävien komponenttien osuus
- PROD=tuottavuus

Tuottavuuskerroin (PROD)

- Tuottavuuskerroin huomioi alla taulukossa esitetyn mukaisesti
  - kuinka kokenut tiimi tekee tuotetta
  - kuinka hyviä kehitystyökaluja on käytössä
- PROD keskiarvo (kokemus+tools/2)

Tiimin kokemus	Hyvin matala	Matala	Keskiverto	Korkea	Hyvin korkea
Kehitystyö-kalujen taso	Hyvin matala	Matala	Keskiverto	Korkea	Hyvin korkea
	4	7	13	25	50

Early design model

- Vaatimusmäärittelyn jälkeen
- Perustuu algoritmisten mallien peruskaavaan

  $PM = A \times Size^B \times M$

  – M on seitsemän tekijän tulo (ei-toiminnalliset vaatimukset, ympäristö, henkilöstön kyvyt/kokemus)
  – A estimoitu suuren datamäärän perusteella, A=2.94 alkuperäisessä kalibroinnissa
  – Size= tuhatta koodiriviä (KLOC)
  – B vaihtelee välillä 1.1...1.24
    - projektin “innovatiivisuus”, joustavuus, riskienhallinta, prosessin kypsyys

Post-architecture level

- M lasketaan 17 kertoimen perusteella (early design –mallissa 7 kerrointa)
- Koodin määrässä huomioidaan
  - reuse model –osamallin perusteella muodostettuja arvioita uudelleenkäytettävän koodin määrästä + muokkaustarpeesta sekä arvioidaan vaatimusten muuttumisesta seuraavia muutostarpeita
- Eksponentin määräämisessä viisi skaalaustekijää
  - Lähdetään eksponentin arvosta 1.01 ja lisätään siihen skaalaustekijöiden summa jaettuna 100:lla

Skaalaustekijät (Sommerville 2010)

Scale factor	Explanation
Precedentedness	Reflects the previous experience of the organization with this type of project. Very low means no previous experience; extra-high means that the organization is completely familiar with this application domain.
Development flexibility	Reflects the degree of flexibility in the development process. Very low means a prescribed process is used; extra-high means that the client sets only general goals.
Architecture/risk resolution	Reflects the extent of risk analysis carried out. Very low means little analysis; extra-high means a complete and thorough risk analysis.
Team cohesion	Reflects how well the development team knows each other and work together. Very low means very difficult interactions; extra-high means an integrated and effective team with no communication problems.
Process maturity	Reflects the process maturity of the organization. The computation of this value depends on the CMM Maturity Questionnaire, but an estimate can be achieved by subtracting the CMM process maturity level from 5.

Esimerkki (Sommerville 2010)

- A company takes on a project in a new domain. The client has not defined the process to be used and has not allowed time for risk analysis. The company has a CMM level 2 rating.
  - Precedenteness - new project (4)
  - Development flexibility - no client involvement - Very high (1)
  - Architecture/risk resolution - No risk analysis - V. Low .(5)
  - Team cohesion - new team - nominal (3)
  - Process maturity - some control - nominal (3)
- Scale factor is therefore 1.17.

Laadunhallinta

Laadunhallinta

- Viimeisen 20 vuoden aikana ohjelmistojen laatu on parantunut huomattavasti
  - modernit prosessimallit opettavat entistä joustavammat ja silti hallitut työtavat,
  - modernit ohjelmistotuotannon tekniikat hallitaan useimmiten aika hyvin
  - ohjelmistojen *laadunhallinta* (quality management) on otettu merkittäväksi osaksi ohjelmistojen kehitystyötä

Mitä ohjelmiston laatu tarkoittaa?

- Ohjelmiston laatu = ohjelmiston vastaavuus määrittelynsä kanssa?
- Järjestelmävaatimukset eivät yksin riitä laadun perustaksi
  - Niiden laatu ei ole tarpeeksi hyvä, epätäydellisyys, ristiriitaisuus
  - ”Jännite” asiakkaan ja kehitystiimin näkökulmien välillä
  - Kaikkia laadun piirteitä ei osata määritellä yksiselitteisesti (esim. ylläpidettävyys)
  - Ohjelmisto ei välttämättä täytä asiakkaan tarpeita, vaikka se vastaisi määrittelyä

- Onnistunut laadunhallinta takaa, että tehtävät tuotteet ovat laadukkaita sekä asiakkaan että kehitystiimin kannalta
- Pienissä projekteissa keskeisintä on luoda *laatukulttuuri,* jossa laatu on *kaikkien vastuulla*
- Mitä suurempi organisaatio ja suuremmat järjestelmät, sitä tärkeämpää on tarkasti ohjattu laadunhallinta

Laadunhallinnan tehtävät

- Laadunhallinta
  - määrittelee tarvittavat laatustandardit ja laadun saavuttamiseksi tarvittavat menetelmät
  - varmistaa, että määriteltyjä standardeja ja menetelmiä myös seurataan käytännössä
  - pyrkii kehittämään laatukulttuurin, jossa laatu on kaikkien vastuulla

Prosessin ja tuotteen laatu

- Tuotantoprosessin laatu vaikuttaa tuotteen laatuun
- Tuotteen laadun seuranta voi olla vaikeaa
- Prosessien ja tuotteen laadun välisestä yhteydestä tarvitaan lisätutkimusta

Laatukomponentit

- Parhaiten laatu määritellään *laatukomponenteilla* (quality attributes)
- Laadukas tuote on sellainen, joka täyttää *riittävän hyvin* sille määritellyt laatukomponentit
- Laatukomponenteilla on läheinen yhteys ei-toiminnallisiin vaatimuksiin
  - Siksi ei-toiminnallisia vaatimuksia kutsutaan laatuvaatimuksiksi

Ohjelmistojen laatukomponentteja

- Käyttöturvallisuus (safety)
- Tietoturvallisuus (security)
- Luotettavuus (reliability)
- Tehokkuus (efficiency)
- Käytettävyys (usability)
- Opittavuus (learnability)
- Joustavuus (resilience)
- Vakaus (robustness)
- Muutettavuus (adaptability)
- Virheettömyys (faultlessness)
- Monimutkaisuus (complexity)
- Modulaarisuus (modularity)
- Testattavuus (testability)
- Siirrettävyys (portability)
- Ylläpidettävyys (maintainability)
- Uuskäyttöisyys (reusability)

Laatuvaatimukset

- Ohjelmiston tyyppi vaikuttaa siihen, mitkä laadun komponentit pitää huomioida kehitystyössä

→ tuotteen laatuvaatimukset

→ ei-toiminnallisia vaatimuksia, jotka yleensä liittyvät koko järjestelmään

  – Joskus laatuvaatimus voi liittyä yksittäiseen toimintoon: esimerkiksi tietyn toiminnon vasteaika on laatuvaatimus

Laatukomponenttien valinta

- Laatukomponentit ovat usein ristiriidassa keskenään
  - Esimerkiksi tietoturvallisuus voi olla ristiriidassa käytettävyyden kanssa
- Kehitystyössä on päätettävä, mitkä laatukomponenteista ovat tärkeimmät ja mistä voidaan tinkiä.
  - Esimerkiksi edellä varmaankin tingittäisiin käytettävyydestä tietoturvallisuuden hyväksi

Laadunhallintaryhmä

- Laadunhallinta kannattaa antaa ohjelmistoprojekteista riippumattomalle *laadunhallintaryhmälle*
  - Laadunhallintaryhmä seuraa projekteja ja raportoi projektien johtoryhmille
- Laadunhallintaryhmän avulla projekteihin saadaan objektiivinen kehitystyön ulkopuolinen näkökulma

Standardit

- Standardit ovat sopimuksia, joiden avulla pyritään tasalaatuisuuteen
- Tiivistävät “parhaat käytännöt”
- Vältetään tehtyjen virheiden toistaminen
- Standardi voi koskea tuotetta tai prosessia:
  - tuotetta koskevat standardit sisältävät tuotteen osien rakenteet ja esitystavat
  - prosessia koskevat standardit sisältävät prosessin työvaiheet ja niissä käytettävät työtavat

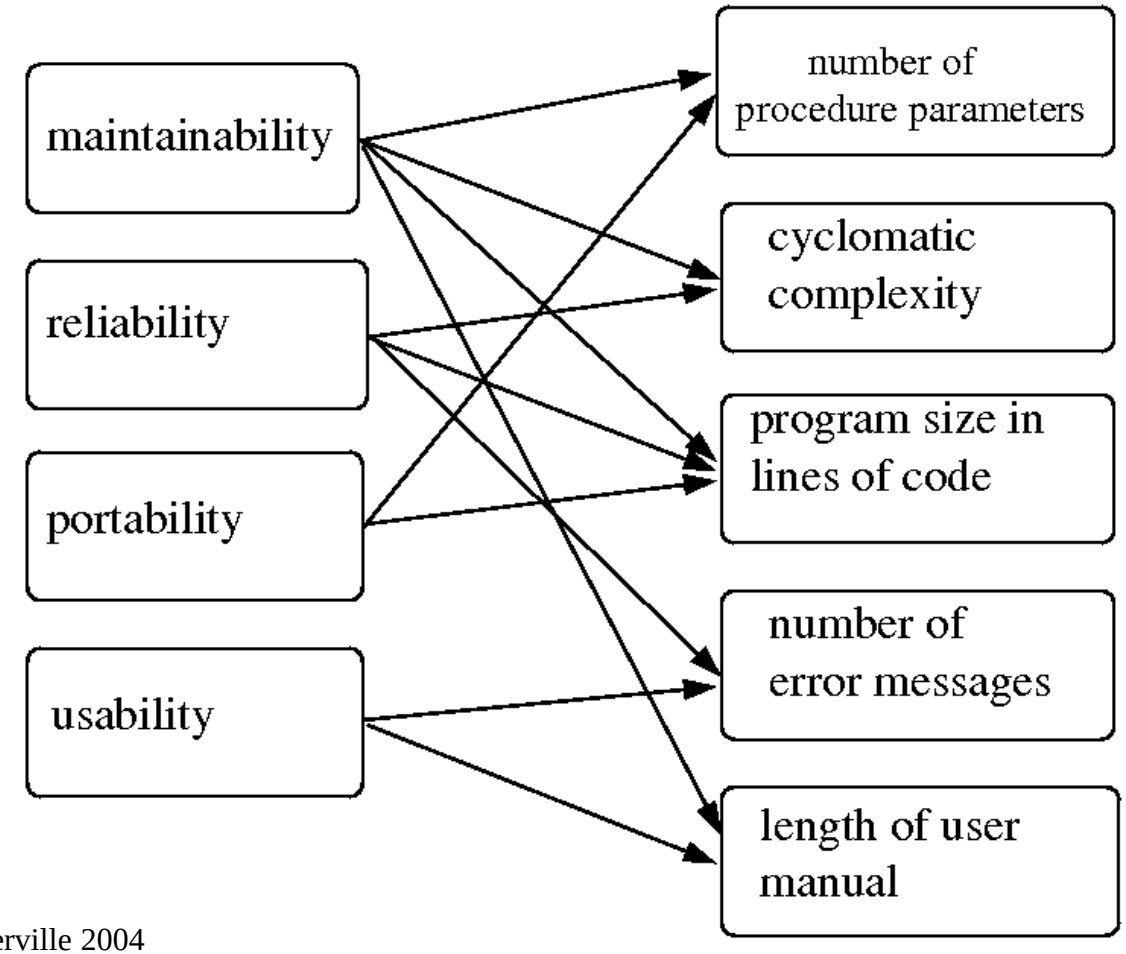
Mittaukset

- Mittaus (measuring)
  - projektin kestäessä ja päättyessä sekä tuotteesta että prosessista lasketaan sitä kuvaavia arvoja
- Mitattavat suureet (software metrics)
- Projektin aikana tehtävää mittausta käytetään seurattaessa ja ohjattaessa projektin etenemistä ja tuotteen laatua
- Projektin lopussa tehtävää mittausta käytetään tulevia projekteja varten

Kiinnostavat ja mitattavat suureet

- Mitattavat suureet eivät yleensä ole sellaisenaan kiinnostavia
- Saadut arvot pitää tulkita
- Tulkinnassa mitattavasta suureesta johdetaan tieto, joka kertoo jostain kiinnostavasta suureesta:
  - esim. mitataan vaatimusmäärittelyssä näyttöjen määrä ➔ arvioidaan käytettävyyttä

Kiinnostavat ja mitattavat suureet



Kuva © I. Sommerville 2004

Tuotteen mitat

- Staattiset mitat:
  - Kerätään mittaamalla projektin tuotoksia:
    - Suunnitelmia
    - Koodia
    - Dokumentteja
  - Kerättävissä projektin alusta alkaen
- Dynaamiset mitat:
  - Kerätään mittaamalla toimivaa ohjelmaa
  - Mitan arvo riippuu myös siitä, miten ohjelmaa käytetään:
    - Eri toimintojen käyttö
    - Syötteet
  - Kerättävissä vasta, kun on jotain, joka toimii

Mitä mitat kuvaavat?

- Staattiset mitat liittyvät tuotteen rakenteellisiin ominaisuuksiin
- Niillä voi olla *välillinen* yhteys laatuominaisuuksiin
  - Esim. laaja vaatimusmäärittely ➔ vaikeasti ylläpidettävä ohjelmisto
- Dynaamiset mitat liittyvät tuotteen käyttäytymiseen. Niillä on yleensä *suora* yhteys laatuominaisuuksiin.
  - Esim. suoritusaika, toipuminen jne.

Prosessin mitat

- Prosessin mittoja käytetään prosessin seurantaan ja parantamiseen:
  - aikamitat, kuten tiettyyn työvaiheeseen kulunut aika
  - resurssimitat, kuten käytettyjen henkilötyöpäivien määrä, koneaika
  - tapahtumamitat, kuten testauksessa löytyneiden vikojen lukumäärä, muutospyyntöjen lukumäärä.

Mittatulosten analysointi

- Mitattavaan suureeseen vaikuttaa yleensä monta samanaikaista tekijää
- Tulosten tulkintaan liittyy epävarmuutta
  - Esimerkiksi jos testauksessa löytyi vain pieni määrä vikoja, syy voi olla hyvässä koodauksessa, huonossa testauksessa, taitavassa suunnittelussa, huolellisissa tarkastuksissa, runsaassa uudelleenkäytössä ym.