1. Johdanto

- Termi *Ohjelmistotuotanto* (Software Engineering) esiteltiin ensimmäistä kertaa 1968 pidetyssä NATO:n konferenssissa. Termi määriteltiin näin:

  The establishment and use of *sound* engineering principles in order to obtain *economically* software that is *reliable* and works *efficiently* on *real machines*.

  (P. Naur, R.Randell (eds.): Software Engineering: A Report on a Conference Sponsored by the NATO Science Committee, 1968)

Ohjelmistotuotannon piirteitä

- Määritelmä oli aikaansa edellä, sillä se pätee erittäin hyvin edelleen. Ohjelmistotuotanto
  - on kurinalaista insinöörityötä,
  - pyrkii tarjoamaan keinot valmistaa *laadukkaita* ja *tehokkaita* ohjelmistoja mahdollisimman edullisesti ja
  - tarjoaa tehtäväjaon ja työkalut kaikkiin ohjelmistojen valmistuksen työvaiheisiin.

Ohjelmisto ja järjestelmä

- *Ohjelmisto* (software) on kokoelma yhteistyössä toimivia tietokoneohjelmia, ohjelmien käyttämiä tiedostoja ja niihin liittyviä dokumentteja.
- *(Tietokonepohjainen) järjestelmä* (computer-based system) on joukko toisiinsa liittyviä komponentteja.
  - laitteisto- ja ohjelmistokomponentteja,
  - käyttäjät ja käyttöympäristö.

Osajärjestelmät ja käyttäjät

- Yleensä järjestelmä koostuu useasta osajärjestelmästä.
  - Kukin osajärjestelmä on itsenäinen kokonaisuus, joka toimii yhteistyössä muiden osajärjestlemien kanssa.
  - Osajärjestelmä rakentuu laitteisto- ja ohjelmistokomponenteista, jotka toteuttavat yhteistyössä osajärjestelmän tehtävät.
  - Käyttäjät käyttävät järjestelmää osajärjestelmien kautta (käyttöliittymä).

Järjestelmän ja ohjelmiston laadinta

- Järjestelmää ja ohjelmistoa on laadittava yhteistyössä:
  - Ohjelmistotuotantoa ei voi tehdä ilman tietoa ympäristöstä, johon ohjelmistoa ollaan tekemässä.
  - Yhteistyö edellyttää ohjelmiston laatijoilta valmiutta kommunikoida sovellusalueen asiantuntijoiden kanssa.
  - Ohjelmistoa on helppo syyttää, jos jokin menee pieleen, vaikka syy olisi muualla.

Mitä vaaditaan hyvältä ohjelmistolta?

- Hyvällä ohjelmistolla on seuraavat keskeiset ominaisuudet:
  - *ylläpidettävyys:* tuotetta voidaan muokata vastaamaan muuttuviin vaatimuksiin.
  - *luotettavuus:* tuote ei aiheuta fyysisiä eikä taloudellisia vahinkoja missään tilanteessa.
  - *tehokkuus:* tuote ei tuhlaa resursseja.
  - *käytettävyys:* tuote on looginen ja helppokäyttöinen.

2. Ohjelmistotuotannon historiaa

- Ensimmäinen kaupallinen ohjelmisto tehtiin niinkin aikaisin kuin 1951 Englannissa, tekijänä J. Lyons company.
- Alkuaikoina ohjelmistojen teko oli suoraviivaista ja ohjaamatonta. Silti jo 50-luvulla tehtiin kohtuullisen monipuolisia ohjelmistoja.

1960-luku ja ohjelmistokriisi

- 1960-luvun alussa määriteltiin jo aika moderneja asioita, kuten testattavuus ja rajapinnat. Silti ei vielä voitu puhua varsinaisesta ohjelmistotuotannosta.
- Vuosikymmenen edetessä ohjelmistojen koot kasvoivat ja ongelmat pahenivat. Projektit alkoivat myöhästellä. Tuli ns. *60-luvun ohjelmistokriisi.*
- Suoraviivainen koodaus ei enää riittänyt

Ensimmäiset ohjelmistotuotannon konferenssit

- Lopulta 1968 NATO organisoi jo aiemmin mainitun konferenssin, jossa esiteltiin termi Software Engineering.
- Seuravana vuonna NATO organisoi toisen Software Engineering –konferenssin. Yhdessä nämä konferenssit määrittelivät ohjelmistotuotannon perusteet.

Ensimmäisten konferenssien tulos

- Konferenssien tulokset eivät ehkä ratkaisseet ohjelmistokriisiä:
  - jotkut ovat sitä mieltä, että se ei ole vieläkään ratkennut.
- Sen sijaan konferensseissa esitellyt ideat rakensivat perustan, jolle nykyaikainen ohjelmistotuotanto on rakennettu.

1970-luku ja prosessimallit

- 1970-luvulle tultaessa määriteltiin ensimmäinen ohjelmistotuotannon *prosessimalli*: vesiputousmalli.
  - Prosessimalli on mahdollisimman yleisesti sovellettavissa oleva ohjeisto ohjelmistojen tuottamiseen.
- Vesiputousmallin pohjana käytettiin muiden insinööritieteiden *järjestelmäsuunnittelumallia.*

Lisää prosessimalleja

- Järjestelmäsuunnittelumallin avulla voidaan suunnitella vaikkapa siltoja. Malli kopioitiin suoraan ohjelmistoille.
- Varsin pian huomattiin, että vesiputousmalli ei sovi kaikenlaisten ohjelmistojen tekoon. Niinpä 70- ja 80-luvulla kehitettiin lukuisia muita ohjelmistotuotannon prosessimalleja.

Ohjelmointikielten alkuaikoja

- Ensimmäiset ohjelmat kirjoitettiin assemblerilla, mutta ensimmäiset korkean tason ohjelmointikielet kehitettiin jo 50-luvulla.
- Vasta 60-luvulla kehitetyt *Cobol* ja *PL/1* olivat riittävän tehokkaita.
- 60-luvun lopussa kehitetty *C-kieli* (tietenkin B-kielestä jatkokehitetty) oli myös merkittävä askel.

1980-luku ja oliopohjaisuus

- 1980-luvun suurin mullistus oli *oliopohjaisuus*. Ensimmäinen oliokonferenssi OOPSLA pidettiin 1986.
- Oliomaailman periaatteet oli toki esitelty aiemmin, mutta vasta ensimmäiset oliokielet kokosivat ne yhteen.
- Ohjelmointikielistä oliopohjaisuus levisi vähitellen koko ohjelmistotuotantoon.

1990-luvun alku ja henkilökohtainen tietojenkäsittely

- 90-luvulle tultaessa ohjelmistotuotanto oli aika vakiintunutta. 80-luvulla alkanut *henkilökohtainen tietojenkäsittely* (jokaisella oma tietokone) mullisti ohjelmistojen tarpeen, mutta ei niiden valmistuksen tekniikkaa.
- Ohjelmistotuotanto eli *työkalujen aikaa*. CASE-työkalut (Computer-Aided Software Engineering) kehittyivät.

1999 ja ketterät prosessimallit

- Toistaiseksi viimeinen ohjelmistotekniikan murros tapahtui 1999. Tällöin esiteltiin ensimmäinen *ketterä prosessimalli* Extreme Programming (XP).
- Ketterät prosessimallit saavuttivat nopeasti erityisesti ohjelmoijien ja PK-yritysten suosion, sillä ne suosivat ohjelmointia suunnittelun sijaan.

Hajaantunut nykyhetki (2006)

- Tällä hetkellä (2006) ohjelmistotuotantoa kuvaa parhaiten hajaantuminen ja erikoistuminen.
- Vaikka perusperiaatteet, joita opetetaan mm. tällä kurssilla, ovat säilyneet ennallaan, niin yritykset käyttävät itselleen kehitettyjä prosessimalleja, menetelmiä ja työkaluja.

Erikoistunut nykyhetki (2006)

- Myös tehtävät ohjelmistot eroavat toisistaan enemmän kuin koskaan.
  - Yhtäältä meillä on kännyköihin ja mobiililaitteisiin tehtäviä pieniä ohjelmia.
  - Toisaalta meillä on valtavia ohjelmia, joiden suorituskyky ja käyttöturvallisuus on määritelty erittäin korkealle.
- Näin ohjelmistoyritysten täytyy erikoistua säilyttääkseen markkinansa.

Tulevaisuus

- Tulevaisuudessa ohjelmistotuotannolla on kolme haastetta:
  - Epäyhtenäisyyden haaste (heterogeneity challenge).
  - Toimitusajan haaste (delivery challenge).
  - Luottamuksen haaste (trust challenge).
- Haasteet eivät ole riippumattomia. Osa voi olla jopa keskenään ristiriitaisia.

Epäyhtenäisyyden haaste

- Ohjelmistopohjaisten järjestelmien täytyy toimia hyvin erilaisissa ympäristöissä eri järjestelmien kanssa.
- Ohjelmistojen pitää myös toimia vanhojen järjestelmien kanssa.
- Vanhoihin järjestelmiin täytyy usein kehittää uutta toiminnallisuutta.
- Vanhoja järjestelmiä tekohengitetään.

Toimitusajan haaste

- Ohjelmistojen teko ohjelmistotuotannon menetelmin vaatii paljon aikaa. Aika taas on kallis resurssi.
- Liiketoiminnassa taas täytyy olla dynaaminen ja valmis muutoksiin.
- Näin myös ohjelmistojen täytyy valmistua ja mukautua muutokseen nopeasti ilman että laatu kärsii.

Luottamuksen haaste

- Ohjelmistot vaikuttavat aina vain enemmän elämäämme. Mitä enemmän ohjelmistot hallitsevat, sitä enemmän meidän pitää voida luottaa niihin.
- Luottamus ei kuitenkaan synny tyhjästä. Ohjelmistoista pitää voida nähdä, että ne toimivat halutulla tavalla.
- Tällä hetkellä luottamus kärsii mm. roskapostin, virusten ja matojen takia.

3. Ohjelmistotuotantoprosessi

- *Ohjelmistotuotantoprosessi* (software process) on joukko toimintoja, jotka johtavat ohjelmiston valmistumiseen.
- Prosessi on ohjeisto. Se kertoo työvaiheet ja niiden väliset suhteet.
- Prosessin ilmentymä on *projekti*. Siinä toteutetaan prosessin työvaiheet.
- Käytännössä jokainen ohjelmistoyritys käyttää itselleen räätälöityjä prosesseja.

Prosessimalli

- *Prosessimalli* (process model) on abstrakti kuvaus prosessista.
  - Prosessimallista voidaan johtaa monta erilaista prosessia eri organisaatioihin ja sovellusalueille.
- Prosessimalli kuvaa yleisperiaatteen, ei yksityiskohtia. Näin prosessimalli määrittelee kehykset, joihin itse prosessit rakennetaan.

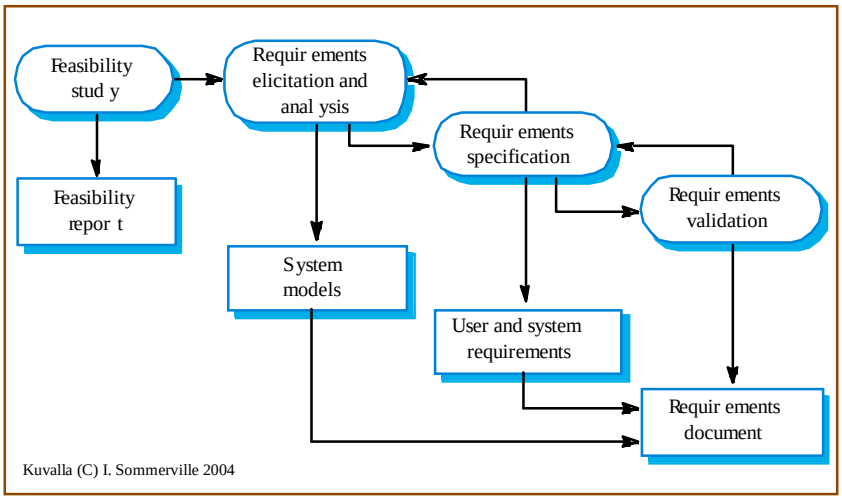
Prosessimallien perustehtävät

- Jokaiseen prosessimalliin sisältyy tavalla tai toisella seuraavat *perustehtävät*:
  - Vaatimusten keruu ja analyysi (vaatimusmäärittely),
  - Ohjelmiston suunnittelu,
  - Toteutus ja yksikkötestaus,
  - Integrointi ja järjestelmätestaus,
  - Käyttö ja ylläpito.

Vaatimusmäärittely

- Mitä vaatimuksia ohjelmistolle asetetaan:
  - mitä toimintoja sen pitää sisältää
  - liittyykö toimintaan laadullisia vaatimuksia (esim. nopeuden tai käyttötavan suhteen)
- Mitä rajoituksia ohjelmiston toimintaan liittyy:
  - esim. yhteistoiminta muiden ohjelmistojen kanssa

Vaatimusmäärittelyn tehtäväjako



Ohjelmiston suunnittelu

- Suunnittelussa vaatimuksista lähtien jalostetaan toteutuskelpoinen kuvaus ohjelmistosta.
- Suunnittelu aloitetaan tekemällä arkkitehtuurisuunnitelma: ”ohjelman piirustukset”.
- Arkkitehtuurisuunnitelmaa ositetaan ja tarkennetaan halutulle tasolle.

Suunnittelun tasot

- Suunnitteluprosessi vaatii useita työvaiheita, jotka käsittelevät ongelmaa eri tarkkuustasoilla. Tasot voivat olla esimerkiksi seuraavat:
  - Arkkitehtuurisuunnittelu
    - Tehdään järjestelmästä korkean tason kuva, josta selviävät osajärjestelmät ja niissä käytettävät komponentit.

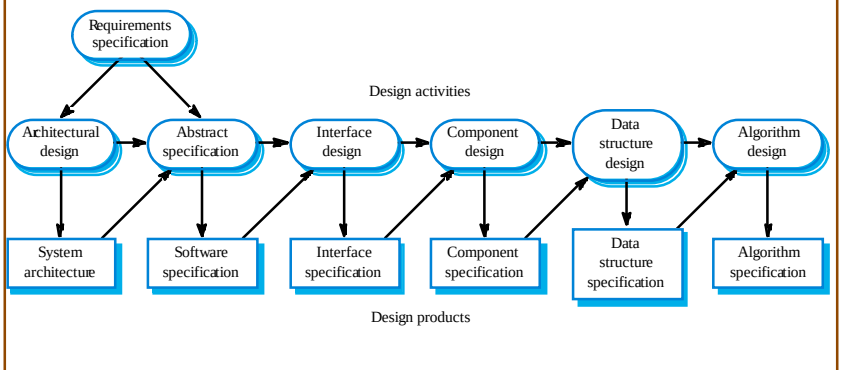
Suunnittelun tasot 2

- Suunnittelun tasot jatkuvat:
  - Abstrakti määrittely
    - Määritellään kunkin osajärjestelmän tarjoamat palvelut ja rajoitteet.
  - Rajapintasuunnittelu
    - Määritellään kunkin osajärjestelmän tarjoamat rajapinnat muille osajärjestelmille.
  - Komponenttisuunnittelu
    - Määritelmään osajärjestelmittäin palvelut toteuttavat komponentit ja suunnitellaan ne.

Suunnittelun tasot 3

- Suunnittelun tasot jatkuvat:
  - Tietorakenteiden suunnittelu
    - Suunnitellaan järjestelmän toteutuksessa välttämättömät tietorakenteet.
  - Algoritmien suunnittelu
    - Suunnitellaan järjestelmän toteutuksessa käytettävät algoritmit.
- Komponentit voivat olla esimerkiksi olioluokkia tai luokkaryhmiä.

Suunnittelun tasojen kaaviokuva



Kuvalla (C) I. Sommerville 2004

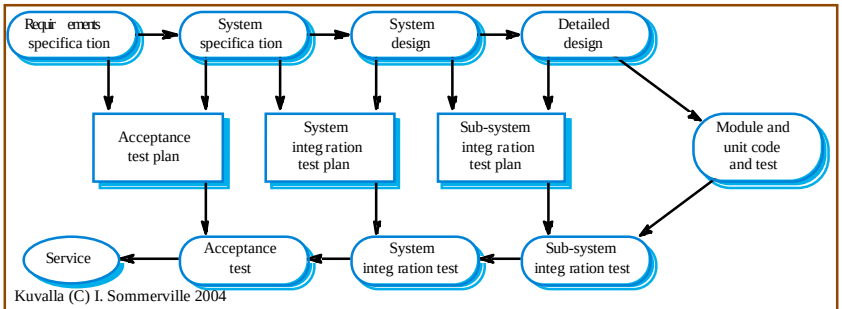
Toteutus ja yksikkötestaus

- Toteutuksessa suunnitelman perusteella tehdään komponentteja.
  - Ohjelmakomponentti: yhden asian kunnolla tekevä ohjelman osa, jolla on selkeä rajapinta ulospäin.
  - Komponentti voi olla kooltaan mitä tahansa olioluokasta kokonaiseen ohjelmaan.
- Yksikkötestauksessa varmistetaan, että komponentit toimivat oikein.

Integrointi ja järjestelmätestaus

- Ohjelmakomponentit kootaan yhteen.
- Komponenttien välinen yhteistyö testataan integrointitestauksessa.
- Komponenteista kootaan osajärjestelmiä (itsenäisiä osia).
- Osajärjestelmien yhteistyö testataan.
- Järjestelmätestauksessa koko ohjelmisto testataan käyttöympäristössä

Testauksen kaaviokuva



Kuvalla (C) I. Sommerville 2004

- Kaaviossa testaus on alalaidassa ja suunnittelu ylälaidassa.
- Testauksen kaaviokuva kuvaa *V-mallin*. Siitä puhutaan lisää myöhemmin.

Ohjelmiston käyttö ja ylläpito

- Ohjelmistoa on usein tarvetta muuttaa myöhemminkin:
  - Käytössä paljastuu virheitä.
  - Käyttäjien vaatimukset täsmentyvät tai muuttuvat.
  - Käyttöympäristö ja laitteisto muuttuvat.
  - Käyttötavat muuttuvat.
- Ylläpito = ohjelmiston muuttaminen käyttöönoton jälkeen.

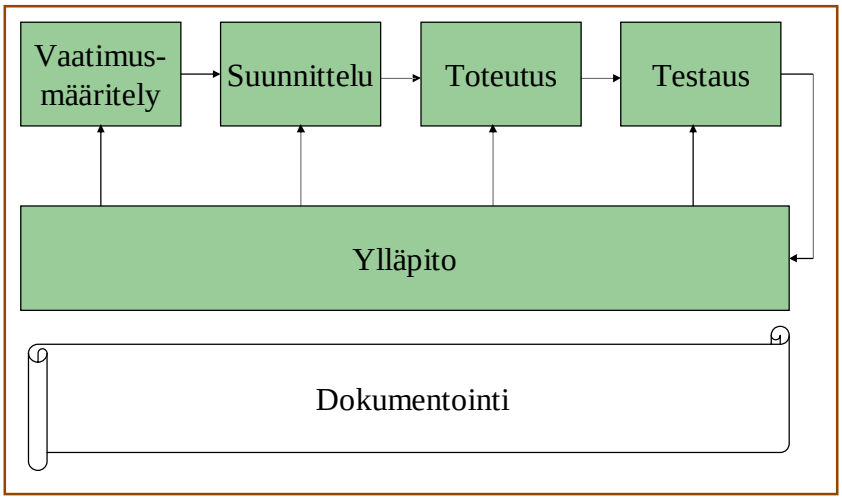
3.1. Yleisimmät prosessimallit

- Yleisimmät prosessimallit ja niiden tyypillisimmät ominaisuudet ovat:
  - Vesiputousmalli: lineaarisuus.
  - Evoluutiomalli: prototyypit.
  - Komponenttimalli: valmiiden osien käyttö.
  - Lisäävä malli: tehdään vähän kerrassaan.
  - Spiraalimalli: toistetaan samat tehtävät eri mittakaavassa lisättäessä toimintoja.
  - Ketterät prosessimallit: dynaamisuus.

Vesiputousmalli

- Vesiputousmallissa on lineaarinen lähestymistapa:
  - Jo päätettyyn vaiheeseen ei enää palata myöhemmin.
  - Jokainen vaihe saatetaan loppuun, sen tulokset hyväksytään ja jäädytetään ennen seuraavaan vaiheeseen siirtymistä.
- Mallin vaiheet ovat suoraviivaisesti prosessimallin perustehtävät.

Vesiputousmallin tehtäväjako



Vesiputousmallin vaiheet

1. Vaatimusmäärittely
- mitkä ovat järjestelmän
  - palvelut
  - rajoitukset
  - tavoitteet ?
- laaditaan ohjelmistoa kuvaava malli, jonka avulla voidaan tarkistaa vaatimusten järkevyys

2. Suunnittelu
- tarkennetaan vaatimus-määrittelyssä laadittua mallia:
  - osajärjestelmät ja
  - niiden välinen yhteistyö
- tarkennetaan jokainen osajärjestelmä halutulle abstraktiotasolle asti
  - riittävä tarkkuus toteutuksen pohjaksi

Vesiputousmallin vaiheet 2

3. Toteutus (+yksikkötestaus)
   - suunnitelma toteutetaan osa kerrallaan
   - eri osat voidaan tehdä eri ohjelmointikielillä
   - kukin osa (yksikkö) testataan toteutuksen jälkeen
   - yksikkötestauksen menetelmät

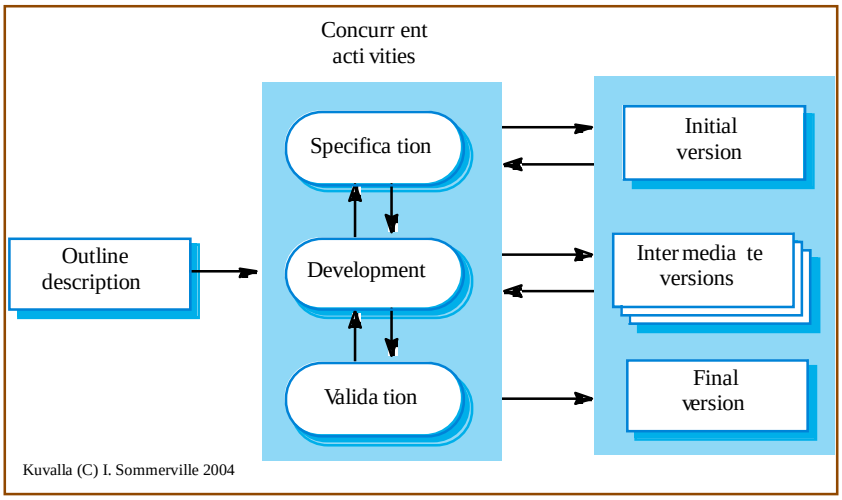
Malli perustuu vahvasti hyvään dokumentaatioon.

4. Integrointi ja testaus
   - toimivista osista kootaan osajärjestelmiä ja näistä edelleen koko järjestelmä
   - integrointitestaus: rajapintojen toimivuus
   - koko järjestelmän verifiointi (selvitetään, että se toimii) ja validointi (selvitetään, että se tekee mitä halutaan)
5. Ylläpito
   - käyttöönotto
   - korjaukset, laajennukset

Evoluutiomalli

- Tavoitteena on ohjelmiston kehittäminen väliversioiden kautta:
  - Väliversiot ovat joko tuotantokäytössä olevia ohjelmia tai prototyyppejä.
  - Aloitetaan parhaimmin ymmärretyistä osista.
  - Käyttäjä antaa palautetta väliversioista.
  - Palautteen perusteella jatketaan kehitystä.

Evoluutiomallin vaiheet



Evoluutiomallin edut

- Asiakkaan tarpeet huomioidaan hyvin:
  - Vaatimusten täsmällinen määrittäminen on usein vaikeaa.
  - Kun asiakas näkee toimivan järjestelmän, hänen on helpompi kertoa, mitä hän tahtoo
  - Väärinkäsitykset korjataan nopeammin, jolloin korjaus ei tule niin kalliiksi.
- Käyttöliittymä on koko ajan näkyvillä:
  - Optimitilanteessa ohjelmiston käyttö on helppo oppia.

Evoluutiomallin ongelmat

- Etenemisen seuranta:
  - Milloin tuote oikeastaan on valmis?
- Rakenteelliset ongelmat:
  - Jatkuvat muutokset voivat rikkoa suunnitellun arkkitehtuurin.
- Työvälineiden tarve (sovelluskehitin):
  - Prototyyppien tekeminen vaatii sopivat välineet (ja taidon käyttää niitä).

Mitä prototyypille tehdään

- Prototyypin hylkääminen
  - Tarkoituksena tarkentaa asiakkaan toiveita ja vaatimuksia
- Prototyypin kehittäminen eteenpäin
  - Tarkoituksena esitellä asiakkaalle ydinohjelmisto tai jokin ydinohjelmistoon liitetty aputoiminto.
- Asiakkaan voi olla vaikeaa ymmärtää, miksi prototyyppi ei kelpaa lopputuotteeksi.
  - Prototyyppi on esitoteutus.
  - Prototyyppi ei täytä laadukkaalle ohjelmistolle asetettavia vaatimuksia.

Komponenttimalli

- Tavoitteena on käyttää mahdollisimman paljon jo valmiita komponentteja.
- Valmiita komponentteja saadaan
  - omista aiemmista projekteista,
  - ostamalla ja
  - ilmaisista Free Software –projekteista (vaarallisia, koska ylläpidosta ei ole takeita)
- Uudelleenkäytettäviä komponentteja pidetään komponenttikirjastossa.

Komponenttimallin työvaiheet

- Komponenttimallin työvaiheet ovat:
  - Vaatimusmäärittely
    - Etsitään ja analysoidaan vaatimuksia.
  - Komponenttianalyysi
    - Etsitään komponenttikirjastosta vaatimusanalyysin mukaiseen tuotteeseen sopivia komponentteja.
  - Vaatimusten muokkaus
    - Täydennetään vaatimuksia ottaen huomioon löydetyt komponenttiehdokkaat.

Komponenttimallin työvaiheet II

- Työvaiheet jatkuvat:
  - Suunnittelu ja komponenttien eristys
    - Eristetään komponenttikirjastoon omia kelvollisia komponenttiehdokkaiden suunnitelmia.
  - Toteutus ja komponenttien eristys
    - Eristetään komponenttikirjastoon omia kelvollisia komponenttiehdokkaiden toteutuksia.
  - Järjestelmän validointi
    - Varmistetaan, että tehtiin oikea järjestelmä.

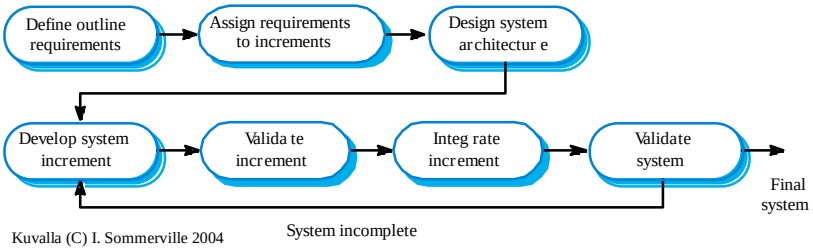
Lisäävä malli

- Vesiputousmallissa ohjelma määritellään ja suunnitellaan yhtenä kokonaisuutena. Tämä ei usein toimi:
  - vaatimuksia ei tunneta kunnolla,
  - vaatimukset ovat epäselviä tai
  - vaatimukset muuttuvat.
- Lisäävässä mallissa ohjelma määritellään ja toteutetaan iteratiivisesti sykleissä: yleisestä erikoiseen.

Lisäävä malli 2

- Ohjelmisto jaetaan osiin, jotka toteutetaan omina sykleinään.
  - Ensin toteutetaan ydin, johon seuraavien syklien tulokset voidaan liittää.
- Vain yhden syklin vaatimukset käsitellään kerrallaan.
- Toteuttamattomien syklien vaatimukset voivat muuttua.

Lisäävän mallin strategia ja edut



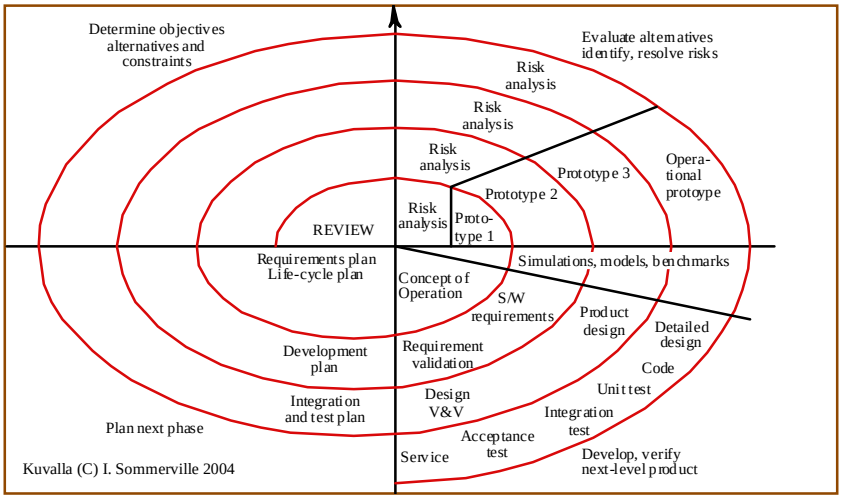
Kuvalla (C) I. Sommerville 2004

- Mallin etuja:
  - Alusta asti käyttökelpoinen tuote.
  - Tärkeimmät piirteet toteutetaan ensin.
  - Virheelliset määritykset huomataan ajoissa.

Spiraalimalli

- Kukin spiraalin kierros esittää yhtä prosessin työvaihetta
- Neljä sektoria:
  - tehtävän asetus = tavoite ja aikataulu
  - riskianalyysi = mitkä ovat tärkeimmät riskit
  - kehitystyö ja validointi
  - suunnittelu = tuloksen arviointi ja seuraavan vaiheen suunnittelu
- Ei kiinteitä tehtäviä
- Kussakin kierroksessa voidaan käyttää muita prosessimalleja
- Esim.
  - 1. kierros = vaatimus-analyysi prototyyppejä käyttäen
  - 2. kierros suunnittelu vesiputousmallina
- Sopii myös ylläpitoon

Spiraalimalliesimerkki



Ketterät prosessimallit

- 1980- ja 1990-luvun prosessimalleissa korostettiin
  - huolellista projektisuunnittelua,
  - formalisoitua laadunvalvontaa,
  - yksityiskohtaisia analyysi- ja suunnittelumenetelmiä,
  - CASE-työkalujen käyttöä ja
  - täsmällistä tarkasti ohjattua ohjelmistoprosessia.

Ketterien prosessimallien synty

- Vaatimukset tukivat erityisesti laajojen, pitkäikäisten ohjelmistojen kehitystyötä, mutta pienten ja keskisuurten ohjelmistojen tekoon ne osoittautuivat usein turhan raskaiksi.
- Ristiriidan seurauksena syntyi joukko *ketteriä prosessimalleja* (agile process models), jotka korostivat itse ohjelmistoa yksityiskohtaisen suunnittelun ja dokumentaation sijaan.
  - Näistä tunnetuimmat ovat *eXtreme Programming* (XP) ja *Scrum.*

Ketterien prosessien luonne

- Kaikki ketterät prosessimallit ovat lisäävän mallin tapaan iteratiivisia.
- Yksi iteraatiokierros on vain muutaman viikon mittainen. Sen aikana käydään läpi kaikki ohjelmistotuotannon perustehtävät.
- Lyhyen syklin johdosta asiakas on jatkuvasti mukana kehitystyössä.

Ketterien prosessien periaatteet

- Asiakkaan osallistuminen
  - Asiakas osallistuu aktiivisesti kehitystyöhön. Asiakas esittää ja priorisoi uusia järjestelmän vaatimuksia ja evaluoi syklin aikana saadut tulokset.
- Lisääntyvät ominaisuudet
  - Kullakin syklillä ohjelmistoon lisätään uusia ominaisuuksia, joiden vaatimukset saadaan asiakkaalta.

Ketterien prosessien periaatteet 2

- Ihmiset prosessia tärkeämmät
  - Projektiryhmäläisten taidot ja toimintatavat ovat tärkeämmät kuin ohjailtu prosessi
- Muutoksen hyväksyminen
  - Vaatimukset muuttuvat, joten järjestelmä pitää suunnitella tukemaan muutosta.
- Yksinkertaisuuden ylläpito
  - Sekä ohjelmisto että prosessi tulee pitää mahdollisimman yksinkertaisena.

Extreme programming (XP)

- Ohjelma kehitetään hyvin pieninä sykleinä.
  - Aluksi tehdään pieni ydin.
  - Jatkossa lisätään sykli kerrallaan mahdollisimman pieni järkevä joukko ominaisuuksia.
  - Toteutusjärjestys määräytyy asiakkaiden tarpeista
- Projektia suunnitellaan mahdollisimman lyhyt aika eteenpäin.
- Dokumentaatio karsitaan minimiin. Toimiva tuote on tärkeämpi kuin raskas dokumentti-kirjasto. Koodi on tärkein dokumentti.
- Malliin kuuluu iso joukko ohjeita.

4. Ohjelmistojen vaatimusmäärittely

- Ehkä kaikkein merkittävin ohjelmistotuotannon ongelma on selvittää asiakkaalta järjestelmälle asetettavat *vaatimukset* (requirements)
  - Kaikki helposti määriteltävät järjestelmät on jo määritelty, joten ongelma vain pahenee.
- Vaatimukset kuvaavat sen, minkälaisen järjestelmän asiakas haluaa auttamaan tai helpottamaan jotain toimintaa.

Vaatimusmäärittelyn tehtävät

- *Vaatimusmäärittely* (requirements engineering) tarkoittaa prosessia, jossa selvitetään järjestelmälle asetettavat vaatimukset.
- Vaatimusmäärittelyssä selvitetään
  - mitä järjestelmän pitää tarjota (toiminnot),
  - toivotut ja vaaditut koko järjestelmää koskevat ominaisuudet (laatuvaatimukset),
  - järjestelmän rajoitukset ja reunaehdot.

Vaatimusten moniselitteisyys

- Termi vaatimus on moniselitteinen.
- Yhtäältä se tarkoittaa hyvin yleistä kuvausta järjestelmän toiminnasta, toisaalta se tarkoittaa erittäin yksityiskohtaista rakenteista esitystä pienestä järjestelmän yksityiskohdasta.
- Molempia ääripäitä tarvitaan, sillä vaatimusmäärittelyn tuloksia käytetään ohjelmistoprosessin eri vaiheissa.

Korkean ja matalan tason vaatimukset

- Korkean tason vaatimuksia käytetään
  - kilpailutettaessa ohjelmistoyrityksiä ohjelmistosopimuksesta,
  - kuvattaessa järjestelmää loppukäyttäjille,
  - kuvattaessa järjestelmää omalle johdolle.
- Matalan tason vaatimuksia käytetään
  - sopimuksena asiakkaan ja ohjelmistoyrityksen välillä,
  - suunnittelun lähtökohtana.

Käyttäjä- ja järjestelmävaatimukset

- Jos vaatimusmäärittelyssä ei tehdä eroa korkean ja matalan tason vaatimusten välillä, niin vaatimusten kuvaustarkkuus voi olla väärä tiettyyn tehtävään.
- Niinpä vaatimukset jaetaan korkean tason *käyttäjävaatimuksiin* (user requirements) ja matalan tason *järjestelmävaatimuksiin* (system requirements).

Käyttäjävaatimukset

- Käyttäjävaatimuksissa kuvataan luonnollisella kielellä ja kaavioilla,
  - mitä palveluja järjestelmän odotetaan tarjoavan ja
  - mitä rajoituksia ja reunaehtoja järjestelmällä on.
- Käyttäjävaatimusten lukijat eivät ole ohjelmistotuotannon ammattilaisia, joten kuvausten pitää olla helppolukuisia.

Järjestelmävaatimukset

- Järjestelmävaatimukset määrittelevät yksityiskohtaisesti ja rakenteisella yhtenäisellä kuvaustekniikalla
  - järjestelmän tarjoamat palvelut,
  - järjestelmän toiminnot,
  - järjestelmän toiminnan rajoitteet.
- Järjestelmävaatimukset määrittelevät täsmälleen, mitä järjestelmä tekee ja mitä ehtoja sille asetetaan.

Vaatimusesimerkki

- LIBSYS shall keep track of all data required by copyright licensing agencies in the UK and elsewhere. (Käyttäjävaatimus).
- LIBSYS copyright licensing: (Järjestelmävaatimus)
  1. On making a request for a document from LIBSYS, the requestor shall be presented with a form that records details of the user and the request made.
  2. LIBSYS request forms shall be stored on the system for five years from the date of the request.
  3. All LIBSYS request forms must be indexed by user, by the name of the material requested and by the supplier of the request.
  4. LIBSYS shall maintain a log of all requests that have been made to the system.
  5. For material where authors' lending rights apply, loan details shall be sent monthly to copyright licensing agencies that have registered with LIBSYS.

Vaatimusten kohderyhmät

- Käyttäjä- ja järjestelmävaatimukset pitää erottaa toisistaan, sillä eri käyttäjäryhmät lukevat vaatimuksia eri tavoin.
  - Käyttäjävaatimusten lukijat ovat usein päälliköitä ja loppukäyttäjiä, jotka ovat kiinnostuneita kokonaisuudesta ilman hankalia yksityiskohtia.
  - Järjestelmävaatimusten lukijat tarvitsevat yksityiskohtia voidakseen suunnitella järjestelmän tai selvittääkseen, miten järjestelmä sopii asiakasyrityksen liiketoimintaan.

Vaatimusten luokittelu

- Käyttäjä- ja järjestelmävaatimukset luokitellaan kolmeen ryhmään:
  - *Toiminnalliset vaatimukset* (functional requirements) kuvaavat järjestelmän tarjoamat palvelut, miten järjestelmä reagoi annettuihin syötteisiin ja miten järjestelmä toimii määritellyissä tilanteissa.
    - Joskus toiminnalliset vaatimukset määrittelevät myös, mitä järjestelmä ei missään tapauksessa tee.

Vaatimusten luokittelu 2

- *Ei-toiminnalliset vaatimukset* (non-functional requirements) ovat järjestelmän toiminnoille asetettavia rajoituksia ja reunaehtoja.
- Ei-toiminnalliset vaatimukset koskevat koko järjestelmää. Kaikki järjestelmän toiminnalliset vaatimukset toteuttavat järjestelmän ei-toiminnalliset vaatimukset.
- Jotkut ei-toiminnalliset vaatimukset voivat koskea myös järjestelmän kehitystyötä.

Vaatimusten luokittelu 3

- *Toimintaympäristövaatimukset* (domain requirements) ovat vaatimuksia, jotka järjestelmän täytyy täyttää toimiakseen yhteistyössä muun maailman kanssa.
- Toimintaympäristövaatimukset (tai lyhyemmin ympäristövaatimukset) liittyvät esimerkiksi muihin yhteistyössä toimiviin järjestelmiin.
- Toimintaympäristövaatimukset voivat olla toiminnallisia tai ei-toiminnallisia.

Toiminnalliset vaatimukset

- Toiminnalliset vaatimukset annetaan käyttäjä- ja järjestelmävaatimuksina.
- Toiminnalliset vaatimukset voidaan kuvata
  - palveluina: mitä kaikkea järjestelmä tekee,
  - toimintoina: mitä komentoja järjestelmä tottelee,
  - palveluina ja toimintoina: mitkä toiminnot toteuttavat mitkä palvelut (ristiviittaus).

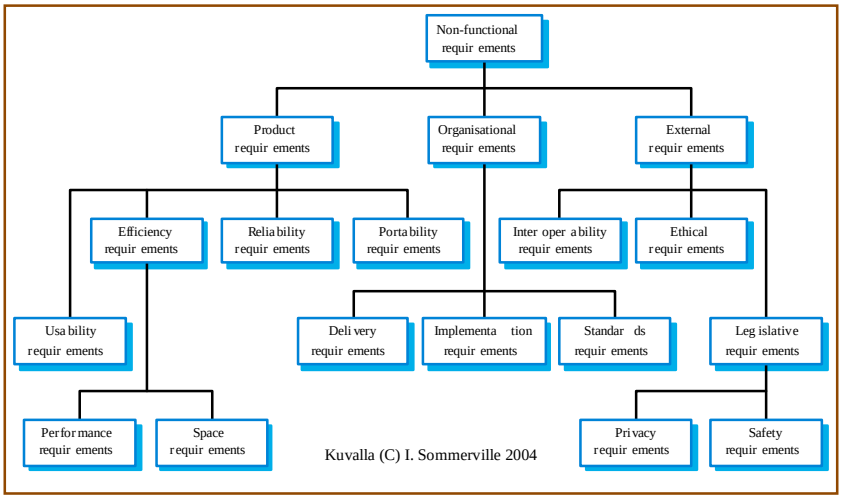
Toiminnalliset vaatimukset 2

- Huono toiminnallisten vaatimusten määrittely johtaa huonoon toteutukseen.
  - Suunnittelija menee yli sieltä, missä aita on matalin, vaikka asiakas olisi halunnut jotain muuta.
- Näin toiminnallisten vaatimusten on oltava
  - täydellisiä: kaikki palvelut määritellään,
  - yhdenmukaisia: määrittelyssä ei ole ristiriitaisuuksia.

Ei-toiminnalliset vaatimukset

- Ei-toiminnalliset vaatimukset ovat yhtä tärkeitä kuin toiminnalliset vaatimukset. Ne kertovat sen, miten käyttäjä kokee järjestelmän.
- On mahdollista, että ei-toiminnallisen vaatimuksen pettäessä koko järjestelmä on käyttökelvoton.
- Kehitystyöhön liittyvät ei-toiminnalliset vaatimukset vaikuttavat esim. ylläpitoon.

Ei-toiminnallisia vaatimuksia



Käyttöympäristövaatimukset

- Käyttöympäristövaatimukset ovat tärkeitä, sillä ne määrittelevät ehdot, jotka järjestelmän tulee täyttää, jotta se voi toimia yhteistyössä muiden järjestelmien tai fysiikan lakien kanssa.
- Käyttöympäristövaatimuksia ovat esimerkiksi järjestelmälle määriteltävät rajapinnat muihin järjestelmiin tai luonnonlaeista seuraavat rajoitteet.

Käyttöympäristövaatimusesimerkkejä

- ” Junan hidastuvuus lasketaan
  $D_{juna} = D_{ohjaus} + D_{gradientti}$
  missä $D_{gradientti}$ on $9.81ms^2$ * korvattu gradientti/alfa ja missä $9.81ms^2$ /alfa tunnetaan eri junatyypeille.”
- “Järjestelmä käyttää standardia Oracle 10.X:n tietokantaliittymää standardoidun ajurin kautta.”

Vaatimusdokumentti

- *Vaatimusdokumentti* (software requirements document tai software requirements specification, SRS) kuvaa yksiselitteisesti kaikki ohjelmistolle asetetut vaatimukset ja kehitetyt mallit.
- Dokumentin rakenne riippuu tuotettavan ohjelmiston tyypistä.
  - Esimerkiksi tietojärjestelmän dokumentti eroaa upotetun järjestelmän dokumentista.

Vaatimusdokumentin rakenne

- Vaatimusdokumentin rakenteen pitää olla sellainen, että sitä voidaan käyttää
  - sopimuksena asiakkaan ja ohjelmistoyrityksen välillä,
  - suunnittelun lähtökohtana,
  - johdolle esiteltävänä yleisenä kuvauksena tehtävästä tuotteesta,
  - testauksen lähtökohtana jne.

Vaatimusdokumentin sisältö

1. Esipuhe (preface)
   - Selvittää kenelle dokumentti on tarkoitettu, dokumentin versiohistorian ja yhteenvedon edellisen version jälkeen tehdyistä muutoksista.
2. Johdanto
   - Sisältää järjestelmän yleiskuvauksen, tärkeimmät tehtävät ja yhteistyön muiden järjestelmien kanssa. Johdanto voi sisältää myös lyhyen vanhan järjestelmän kuvauksen ja viitteet vanhan järjestelmän dokumentaatioon.

Vaatimusdokumentin sisältö 2

3. Sanasto
   - Määrittelee dokumentissa käytetyt termit. Dokumentin tulee olla myös alan sanastoa tuntemattoman henkilön luettavissa.
4. Käyttäjävaatimukset
   - Kuvaa toiminnalliset ja ei-toiminnalliset käyttäjävaatimukset luonnollisella kielellä ja kaavioilla. Myös mahdolliset käyttäjävaatimuksiin laskettavat ympäristövaatimukset luetellaan täällä.

Vaatimusdokumentin sisältö 3

5. Järjestelmäarkkitehtuuri
   - Antaa yleiskuvan ohjelmiston rakenteesta.
   - Näyttää palvelujen jakautumisen osajärjestelmiin ja komponentteihin.
   - Sisältää myös tiedot jo valmiina olevista uudelleenkäytettävistä komponenteista.
6. Järjestelmävaatimukset
   - Kuvaa yksityiskohtaiset toiminnalliset ja ei-toiminnalliset vaatimukset.
   - Kuvaustapa on yhteinen kaikille vaatimuksille.

Vaatimusdokumentin sisältö 4

7. Järjestelmämallit
   - Sisältää yksityiskohtaisemmat mallit järjestelmän osajärjestelmistä, komponenteista ja niiden välisistä suhteista.
   - Järjestelmämallit ovat suunnittelun perustana. Ne kuvataan yhdellä tai useammalla kuvaustekniikalla.
8. Järjestelmän elinkaari
   - Selittää odotetut laitteiston, käytön ja ohjelmiston vaatimusten muutokset.

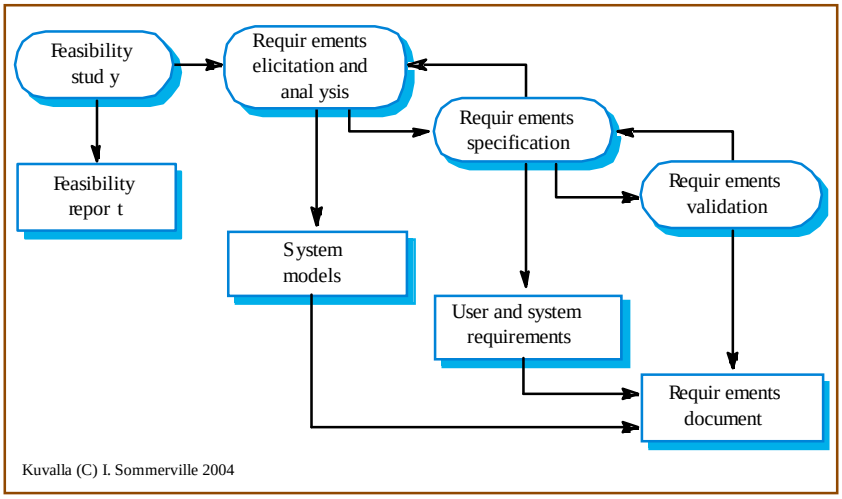
Vaatimusdokumentin sisältö 5

9. Liitteet
   - Sisältää sellaiset olemassaolevat dokumentit tai viitteet, jotka vaikuttavat tuotteeseen, mutta joita ei ole määritelty vaatimusmäärittelyssä.
   - Esimerkiksi käytettävän tietokannan hallintajärjestelmän kuvaus voi olla tällainen liitedokumentti.
10. Hakemisto

4.1. Vaatimusmäärittelyprosessi

- Kuten yleisemmissäkin prosesseissa, myös vaatimusmäärittelyprosesseissa on paljon vaihtelua. Käytännössä kaikista prosesseista löytyvät kuitenkin samat perustehtävät:
  - *kelpoisuusselvitys* (feasibility study),
  - *kartoitus ja analyysi* (elictation and analysis),
  - *määrittely* (specification) ja
  - *validointi* (validation).

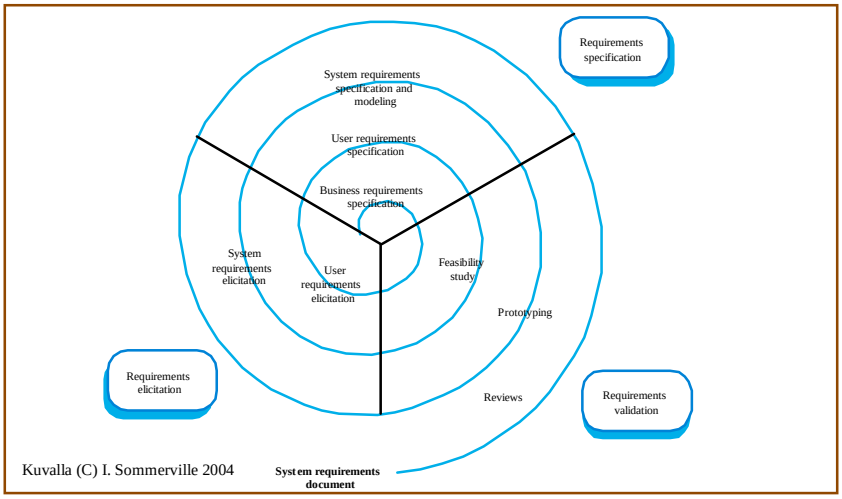
Vaatimusmäärittelyn tehtäväjako (sama kuin kalvo 27)



Vaatimusmäärittelyn spiraalimalli

- Vaatimusmäärittelyyn voidaan ottaa myös spiraalimainen näkökulma.
- Tällöin saadaan mukaan näkökulma, minkä mukaan vaatimuksia ei saada kokoon kerralla vaan iteratiivisesti.
- Malli sallii myös vaatimusmäärittelyn rinnalla tehtävän ohjelmiston suunnittelun/toteutuksen.

Vaatimusmäärittelyn spiraalí



Kelpoisuusselvitys

- Kelpoisuusselvitys on lyhyt esivaihe vaatimusmäärittelylle. Siinä katsotaan
  1. tuoko kehitettävä järjestelmä lisäarvoa asiakkaalle.
  2. voidaanko järjestelmä toteuttaa nykyisellä teknologialla, projektille varatulla aikataululla ja budjetilla.
  3. voidaanko järjestelmä integroida jo olemassaoleviin järjestelmiin.

Kelpoisuusselvitys 2

- Lisäksi, vaikka Sommerville ei sitä listaa:
  4. kannattaako järjestelmä toteuttaa, vai voidaanko vastaava järjestelmä ostaa valmiina. Onko ostettava järjestelmä sovitettavissa asiakkaan tarpeisiin?
- Tuloksena saadaan päätös siitä, kannattaako järjestelmän kehitystyötä jatkaa.

Vaatimusten kartoitus ja analyysi

- Vaatimusten kartoitus ja analyysi – työvaiheessa etsitään yhteistyössä asiakkaan ja järjestelmän loppukäyttäjien kanssa,
  - mitä palveluja järjestelmältä vaaditaan,
  - mitä järjestelmän suorituskyvyltä vaaditaan,
  - mitä laitteisto- ja ympäristörajoituksia on huomioitava jne.
- Työvaiheessa huomioidaan *sidosryhmät.*

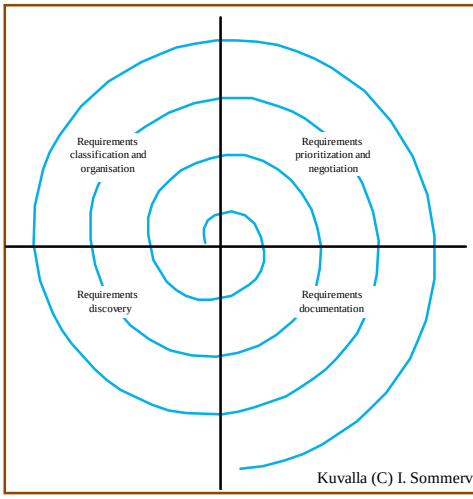
Sidosryhmät

- *Sidosryhmä* (stakeholder) on henkilö tai ryhmä, joka suoraan tai välillisesti on tekemisissä kehitettävän järjestelmän kanssa.
- Eri sidosryhmillä on erilaisia tarpeita kehitettävälle järjestelmälle: ristiriitoja.
- Sidosryhmät eivät välttämättä osaa kertoa, mitä kaipaavat järjestelmältä.

Kartoituksen ja analyysin vaiheet

- Vaatimusten kartoitus ja analyysi voidaan osittaa seuraaviin tehtäviin:
  - *Vaatimusten esiinkaivelu* (req. discovery)
  - *Vaatimusten luokittelu* (req. classification and organization)
  - *Vaatimusten priorisointi* (req. priorization)
  - *Vaatimuksista neuvottelu* (req. negotiation)
  - *Vaatimusten dokumentointi* (req. documentation)

Kartoituksen ja analyysin spiraali



- Spiraali kuvaa sitä, että vaatimusten kartoitus ja analyysi on iteratiivista toimintaa.
- Työ aloitetaan isoista vaatimuksista ja edetään kohti pieniä yksityiskohtia.

Kuvalla (C) I. Sommerville 2004

Vaatimusten esiinkaivelu

- Vaatimusten esiinkaivelussa sidosryhmiltä selvitetään, mitä he odottavat järjestelmältä.
- Selvittäminen voi olla suoraviivaista (esim. haastattelut) tai se voi epäsuoraa (dokumentaatioon tutustuminen, loppukäyttäjien työympäristöön tutustuminen).

Esiinkaivelun tekniikoita

- Esiinkaivelussa voidaan käyttää esimerkiksi seuraavia tekniikoita:
  - Näkökulmat (viewpoints): sidosryhmät ovat joko suoraan, välillisesti tai ympäristön kautta sidoksissa järjestelmään.
  - Haastattelut (interviews): sidosryhmiltä selvitetään sopivin kysymyksin, mitä he odottavat järjestelmältä.

Esiinkaivelun tekniikoita 2

- Lisää tekniikoita:
  - Skenaariot (scenarios) ja käyttötapaukset (use cases): kirjataan suoraviivaisia kuvauksia tulevan järjestelmän käytöstä, skenaarioita. Tarvittaessa toisiinsa sidoksissa olevat skenaariot ryhmitellään yhteen käyttötapauksiksi.
  - Etnografia (ethnography): mennään paikan päälle katsomaan, miten loppukäyttäjät toimivat nykyisin.

Vaatimusten määrittely

- Vaatimusten määrittelyssä vaatimuksen kartoituksessa ja analyysissa löydetyt vaatimukset lajitellaan käyttäjä- ja järjestelmävaatimuksiksi.
- Tarvittaessa vaatimuksia muokataan ja tarkennetaan. Käyttäjävaatimuksista voidaan johtaa järjestelmävaatimuksia.
- Tuloksena saadaan koko järjestelmää esittävät priorisoidut käyttäjä- ja järjestelmävaatimukset.

Vaatimusten validointi

- Vaatimusten validoinnissa pyritään osoittamaan, että vaatimuksilla kuvattu järjestelmä on juuri sitä, mitä asiakas haluaa.
- Validointi on erittäin tärkeää, sillä virheelliset vaatimukset heijastuvat koko tuotteen elinkaaren ajan.
  - Mitä myöhemmin virheellinen vaatimus keksitään, sitä kalliimpaa sen korjaus on.

Vaatimusten tarkastus

- Vaatimuksista tarkastetaan
  - Verifioitavuus:onko vaatimus testattavissa?
  - Ymmärrettävyys: ymmärtävätkö tekijät ja loppukäyttäjät vaatimukset?
  - Seurattavuus: onko vaatimuksen esittäjä ja sidosryhmä tiedossa?
  - Mukautuvuus: Voidaanko vaatimusta muuttaa ilman suuria vaikutuksia muihin tunnistettuihin vaatimuksiin?

Muuttuvat vaatimukset

- Mitä isommasta järjestelmästä on kyse, sitä todennäköisemmin vaatimukset muuttuvat:
  - Sidosryhmien tarpeet vaihtelevat.
  - Käyttöönoton jälkeen löytyy uusia vaatimuksia.
- Vaatimusmäärittelyprosessissa täytyy huomioida muuttuvien vaatimusten vaikutus aikatauluun ja tuotteeseen.

5. Ohjelmistojen suunnittelu

- Suunnittelussa päätetään ohjelmiston loogisesta rakenteesta.
- Rakenne kuvataan eri *abstraktiotasoilla.*
  - Mitä korkeampi abstraktiotaso, sitä paremmin selviää yleiskuva, mutta sitä vähemmän näkyy yksityiskohtia.
  - Korkeimmalla abstraktiotasolla on arkkitehtuurisuunnitelma, matalimmalla on pseudokoodi.

Suunnittelun luonne

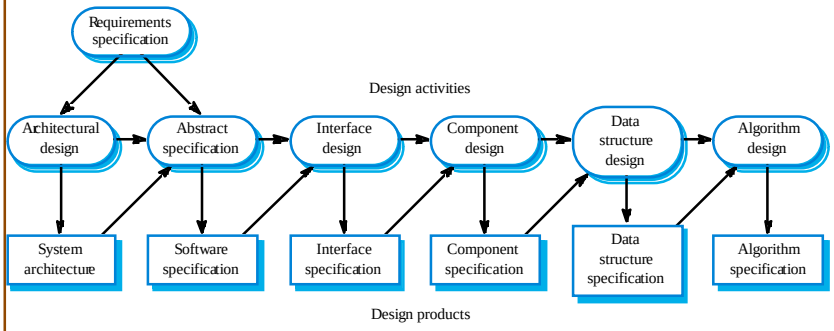
- Suuunnittelu on luovaa toimintaa, jota ei kannata rajoittaa liikaa prosesseihin.
  - Tämä on oppikirjan ja allekirjoittaneen kanta. Monet ovat kuitenkin myös sitä mieltä, että tarkasti ohjatut suunnittelu-prosessit antavat parhaat tulokset.
- Ei ole helppoa suoraviivaista tapaa suunnitella ohjelmisto. Jokainen suunnittelu on ainutkertainen.

Suunnittelun aloitus

- Vaikka suunnittelu on luovaa toimintaa, jotain osviittaa hyvästä suunnittelusta saadaan abstraktiotasoista.
- Aivan kuten vaatimusmäärittely aloitettiin suurista kokonaisuuksista, myös suunnittelu kannattaa aloittaa korkeista abstraktiotasoista. Näin ohjelmistosta säilyy koko suunnittelun ajan selkeä kokonaiskuva.

Suunnittelun runko

- Suunnittelun runkona voidaan käyttää kalvolla 32 ollutta kaaviokuvaa:



Kuvalla (C) I. Sommerville 2004

5.1. Arkkitehtuurisuunnittelu

- Suuret järjestelmät ositetaan aina keskenään yhteistyötä tekeviksi osajärjestelmiksi.
- Osajärjestelmien tunnistaminen ja niiden välisen viestinvälityksen määrittely on arkkitehtuurisuunnittelua.
- Arkkitehtuurisuunnittelun lopputuloksena saadaan järjestelmän arkkitehtuurikuvaus.

Osajärjestelmät ja ei-toiminnalliset vaatimukset

- Osajärjestelmiin jako riippuu ei-toiminnallisista vaatimuksista:
  - Jos *suorituskyky* on tärkeää, kannattaa käyttää mahdollisimman pientä lukumäärää laajoja osajärjestelmiä.
  - Jos *turvallisuus* on tärkeää, kannattaa käyttää sisäkkäisiä arkkitehtuureja, joista kaikkein sisimmällä tasolla ovat turvallisuuskriittisimmät toiminnot.
  - Jos *saatavuus* on tärkeää, samat palvelut kannattaa toistaa monessa osajärjestelmässä.
  - Jos *ylläpidettävyys* on tärkeää, kannattaa käyttää suurta määrää pieniä osajärjestelmiä.

Arkkitehtuurisuunnittelun luonne

- Arkkitehtuurisuunnittelu on luovaa toimintaa, jossa pyritään kehittämään ohjelmistolle sellaiset kehykset, että niihin voidaan suunnitella sekä toiminnalliset että ei-toiminnalliset vaatimukset toteuttava järjestelmä.
- Työvaiheen tehtävät vaihtelevat vahvasti sen mukaan, minkälaista järjestelmää ollaan kehittämässä.

Arkkitehtuurisuunnitelma

- Arkkitehtuurisuunnittelun lopputuloksena saadaan järjestelmän arkkitehtuurisuunnitelma. Se sisältää
  - koko järjestelmän kuvauksen,
  - järjestelmästä tehdyt kaaviokuvat selityksineen,
  - järjestelmän jaon osajärjestelmiin ja osajärjestelmittäin komponentteihin,
  - osajärjestelmien välisen yhteydenpidon.

Osajärjestelmin ositus

- Arkkitehtuurisuunnittelun jälkeen kukin osajärjestelmä ositetaan *moduuleiksi* (module).
  - Moduuli on järjestelmän komponentti, joka käyttää muiden moduuleiden palveluja ja tarjoaa muille moduuleille palveluja.
  - Moduuli ei ole itsenäinen,kuten osajärjestelmä, vaan tiiviissä yhteistyössä muiden moduulien kanssa toteuttaa osajärjestelmän toiminnot.

Moduulijako

- Oliopohjaisessa suunnittelussa kukin moduuli on joukko yhteistyötä tekeviä olioita.
- Moduuli tarjoaa palvelurajapinnan muille moduuleille, ja vaatii palveluita muilta moduuleilta näiden palvelurajapintojen kautta.
- Moduulijako tuo uuden abstraktiotason suunnitteluun.

5.2. Rajapintasuunnittelu

- Rajapintasuunnittelu on ehkä suunnittelun tärkein työvaihe. Siinä jokaisen osajärjestelmän tai moduulin rajapinta muihin osajärjestelmiin tai moduuleihin suunnitellaan ja dokumentoidaan yksityiskohtaisesti.
- Rajapintojen pitää olla sellaiset, että tarjottuja palveluja voidaan käyttää tietämättä niiden toteutuksesta mitään.

Rajapintojen määrittely

- Rajapintojen määrittelyt tulee tehdä yksiselitteisiksi ja ristiriidattomiksi.
  - Luonnollinen kieli ei moniselitteisyytensä vuoksi riitä rajapintojen määrittelyyn.
  - Ohjelmointikielikään ei välttämättä ole paras tapa määritellä rajapintoja, sillä ohjelmointikieli menee helposti turhan matalalle abstraktiotasolle.

Rajapintaformalismi

- Rajapinnan määrittelyyn voidaan käyttää jotain formalismia, kuten Sommervillen luvussa 10.2.
- Toisaalta formalismia tärkeämpää on valittu abstraktiotaso. Kuvauksesta täytyy selvitä tarvittavat tiedot ilman turhia toteutusriippuvia yksityiskohtia.

Rajapinnasta tarvittavia tietoja

- Rajapinta voidaan kuvata esim. näin:
  - Nimi ja kuvaus : Rajapinta tunnistetaan.
  - Tarjotut palvelut : kustakin palvelusta
    - nimi ja kuvaus : palvelu tunnistetaan.
    - sisään otettavat parametrit arvojoukkoineen.
    - ulos annettavat parametrit arvojoukkoineneen.
    - poikkeustilanteet : mihin on varauduttu.
    - alkuehdot : mitä ehtoja kutsujalta vaaditaan.
    - loppuehdot : mitä kutsuttava takaa tulokselta.

Rajapintaesimerkki

- Nimi: Juuri
- Kuvaus: Toisen asteen yhtälön ratkaisu.
- Palvelu Juuri
  - Kuvaus: Toteuttaa rajapinnan Juuri
  - Parametrit sisään: a,b,c : R (reaalilukuja)
  - Parametrit ulos: r_1,r_2 : R (reaalilukuja)
  - Poikkeustilanteet:
    - a = 0 (ei toisen asteen yhtälö)
  - Alku- ja loppuehdot: Ei ole

5.3. Oliopohjainen suunnittelu

- Edellinen suunnitteluprosessi (kalvo 106) sopii sellaisenaan olipohjaiseen suunnitteluun. Malli on kuitenkin geneerinen, jota voidaan tarkentaa esimerkiksi oliopohjaisessa lähestymistavassa.
- Sommerville listaa viisi työvaihetta, jotka sopivat erityisesti oliopohjaiseen suunnitteluun.

Oliopohjaisen suunnittelun perustehtävät

1. Järjestelmän toiminnan ja sen erilaisten käyttötapojen ymmärtäminen.
   - Siis vaatimusten ymmärtäminen.
2. Järjestelmäarkkitehtuurin suunnittelu
   - Arkkitehtuurisuunnittelua, kokonaiskuvan määrittely.
3. Tärkeimpien olioiden tunnistus
   - Oliot voivat olla eritasoisia osajärjestelmästä palvelun tai toiminnon yksityiskohtaan.
4. Järjestelmämallien kehitys
   - Osajärjestelmä- ja komponenttisuunnittelua.
5. Olioiden rajapintojen määrittely
   - Rajapintasuunnittelua.

Oliomalli ja yleinen suunnittelumalli

- Yleiseen suunnittelumalliin verrattuna oliosuunnittelumalli painottaa olioiden roolia.
- Oliot ovat käteviä mm. ulkomaailman mallintamisessa, mutta silti on järkevää pitää lisäksi komponenttitaso kokoamassa yhteistyössä toimivat oliot yhteen.
- Olioita voidaan tunnistaa järjestelmämalleista ja –kuvauksista. Näitä olioita käyttämällä ja uusia lisäämällä saadaan tarkennettua arkkitehtuuria ja kehitettyä järjestelmämalleja.

6. Verifiointi ja validointi

- Verifiointi ja validointi (V&V) on ohjelmistotuotannon työvaihe, missä varmistetaan, että
  - ohjelmisto täyttää sille asetetut implisiittiset ja eksplisiittiset vaatimukset ja
  - ohjelmisto täyttää sen tilanneen asiakkaan ohjelmistolle asettamat tarpeet.
- V&V:ta tehdään koko ohjelmistoprosessin elinkaaren ajan.

Verifioinnin ja validoinnin ero

- Verifioinnissa varmistetaan, että ohjelmisto vastaa määrityksiään:
  - “Verification: Are we building the product right?”
- Validoinnissa varmistetaan, että ohjelmisto täyttää asiakkaan sille asettamat odotukset:
  - “Validation: Are building the right product?”

Verifiointi- ja validointitekniikat

- V&V:ssa käytetään enimmäkseen kahta tekniikkaa: tarkastuksia ja testausta.
  - Tarkastukset (software inspections).
    - Tarkastuksissa joukko ihmisiä analysoi ja tarkastaa järjestelmäkuvauksia, kuten vaatimusdokumentaatiota, suunnittelukaavioita ja ohjelmakoodia. Tarkastukset ovat staattinen tekniikka. Ne eivät vaadi suorituskelpoista ohjelmaa.

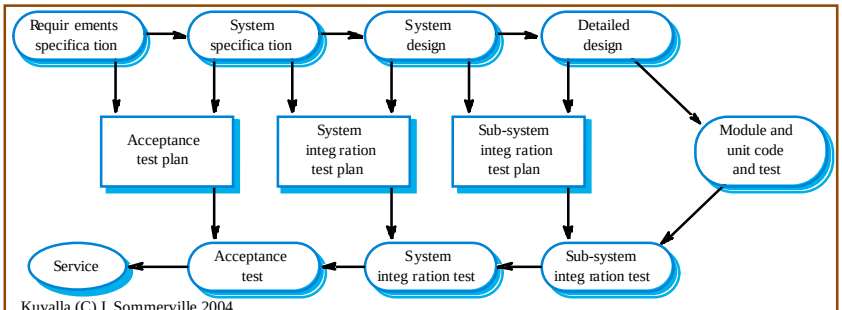
Verifiointi- ja validointitekniikat 2

  - Testaus (software testing).
    - Testauksessa ohjelmistoa tai sen osaa suoritetaan tietyillä testitiedoilla ja tuloksia analysoimalla ja ohjelmiston suoritusta seuraamalla selvitetään, että toiminta on odotettua. Testaus on dynaaminen tekniikka, sillä siihen tarvitaan suorituskelpoinen ohjelma.
- Tarkastuksia voidaan tehdä koko ajan, testausta vasta ohjelmakoodin kanssa.
- Molempia tekniikoita tarvitaan V&V:ssa.

Verifioinnin ja validoinnin tavoite

- V&V:n tavoitteena on varmistaa, että ohjelmisto täyttää sille asetetut tavoitteet.
- Ohjelmiston ei tarvitse olla virheetön, eikä se myöskään yleensä ole sitä.
- Ohjelmiston on sen sijaan oltava tarkoitettuun käyttöön ”riittävän hyvä”.
- Tavoitetaso riippuu sovellusalueesta.

V&V:n hallinnan V-malli



Kuvalla (C) I. Sommerville 2004

- V-malli kuvaa V&V-työvaiheen suhteen muihin prosessin työvaiheisiin.
- Tarkastuksia voidaan pitää missä tahansa työvaiheessa tai niiden välillä.

6.1. Tarkastukset

- Tarkastus (inspection) on kokous, jossa tarkastetaan jonkin työvaiheen tuotos, tai osa siitä, ja yritetään löytää siitä puutteita ja virheitä.
  - Puute = vajaa määritys tai puuttuva toiminta.
  - Virhe = väärin tehty määritys tai ei-toivottu toiminta.

Tarkastukset ja prosessi

- Tarkastuksia tehdään kaikissa prosessin työvaiheissa. Aina kun projektissa on saatu jotain valmiiksi, tulos kannattaa varmentaa tarkastuksella.
- Tarkastukset parantavat tuotteen laatua, sillä aikaisessa vaiheessa löydetty puute tai virhe on helpompi korjata kuin myöhemmin löydettynä.

Tarkastusten luonne

- Tarkastus on muodollinen tilaisuus, johon osallistuu 3-6 henkilöä.
  - Tilaisuudella on tarkka aikataulu.
  - Henkilöt edustavat eri sidosryhmiä asiakkaasta projektiryhmän jäseniin.
  - Tarkastuksessa kootaan löydetyt puutteet ja virheet.
  - Tarkastukseen valmistaudutaan etukäteen noin 2h ajan.

Tarkastukseen osallistujat

- Osallistujien roolit:
  - Puheenjohtaja (moderator): vastaa tarkastuksen aikataulusta ja ohjelmasta.
  - Sihteeri (scribe): kirjaa ylös löyd. asiat.
  - Alustaja (reader): kuvaa esitettävän asian.
  - Kirjoittaja (author/owner): edustaa dokumentin tekijöitä.
  - Tarkastaja (inspector): etsii dokumentista puutteita ja virheitä (kaikkien rooli).

Ennen tarkastusta

- Ennen tarkastustapahtumaa:
  - kaikki tarkastuksessa tarvittavat dokumentit ovat saatavilla,
  - osallistujilla on ollut aikaa tutustua dokumentteihin,
  - käytössä on tarkistuslistat yleisimmistä puutteista ja
  - tarkastettava dokumentti on sellaisella tasolla, että siinä ei ole ilmeisiä virheitä.

Tarkastustapahtuma

- Tarkastustapahtuma saa kestää korkeintaan kaksi tuntia. Siinä keskitytään yksinomaan löytämään puutteita ja virheitä.
- Tarkastukseen osallistujat eivät keskustele löydetyistä puutteista. Kun puute on havaittu, sihteeri kirjaa sen ylös ja siirrytään eteenpäin.

Tarkastuksen päätös

- Tarkastuksen lopuksi ryhmä äänestää tuotoksen hyväksymisestä:
  - Hyväksytään sellaisenaan: ei muutoksia.
  - Hyväksytään muutoksin: löydetyt puutteet ja virheet on korjattava, mutta tuotoksesta ei tarvita enää uutta tarkastusta.
  - Hylätään: löydetyt puutteet ja virheet on korjattava. Korjauksen jälkeen tuotoksesta käydään läpi uusi tarkastus.

Tarkastusten kultaiset säännöt

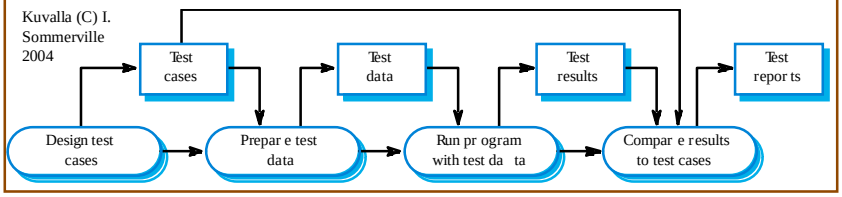
1. Arvioidaan tuotetta, ei tekijää.
2. Suunnitellaan aikataulu ja pidetään siitä kiinni.
3. Ei väittelyä.
4. Ei ratkota löydettyjä ongelmia.
5. Rajoitetaan osallistujien määrä 3-6 henkeen.
6. Valmistaudutaan huolellisesti tarkastukseen.
7. Käytetään tarkistuslistoja sekä valmistautuessa että tarkastuksessa.
8. Varataan riittävästi aikaa ja resursseja.
9. Koulutetaan osallistujat.
10. Pidetään tarkastuksessa kännykät kiinni!

6.2. Testaus

- Testauksella on kaksi tavoitetta:
  - Osoittaa sekä asiakkaille että projektille, että ohjelmisto täyttää sille asetetut vaatimukset. Tämä on *validointitestausta.*
  - Löytää ohjelmistosta puutteita ja virheitä, joiden johdosta ohjelmisto ei toimi, toimii väärin tai ei vastaa sille asetettuja määrittelyjä. Tämä on *määritysten ja syntaksin testausta* (Sommervillella termi on defect testing).

Testausprosessi

- Testausprosessi sisältää seuraavat tehtävät:
  - Testitapausten suunnittelu => testitapaukset
  - Testien valmistelu => testien syötteet + testiympäristö
  - Testien suoritus => testien tulokset
  - Tulosten analyysi => virheraportit + yhteenveto



Testauksen piirteitä

- Suunnittelu kannattaa tehdä V-mallin mukaisesti rinnan kehitystyön kanssa.
- Valmistelu on testiympäristön ohjelmointia ja datan keruuta, johon pätevät normaalit ohjelmistotekniikan säännöt.
- Suoritus pitää automatisoida, jotta testit voidaan suorittaa helposti uudestaan.
- Analysoinnissa tarvitaan *oraakkeli* (oracle), joka kertoo, oliko testin antama tulos oikea.

Täydellinen testaus mahdotonta

- Täydellinen testaus, missä ohjelma testataan kaikilla mahdollisilla syötteillä, syötekombinaatioilla ja ajoituksilla, ei ole käytännössä mahdollista.
  - Jo hyvin yksinkertaisilla ohjelmilla kaikkien testitapausten suoritus veisi vuosia.
- Tämän johdosta testauksessa valitaan osajoukko kaikista mahdollisista testitapauksista.

Testauksen oleellinen kysymys

- Miten valitaan sellainen testitapausten osajoukko, että
  - sen suorittaminen on mahdollista järkevässä ajassa ja että
  - sen avulla ohjelma tai sen osa saadaan testattua *riittävän hyvin*.
- Sommerville suosittelee vastaukseksi yrityskohtaista testauspolitiikkaa, jonka mukaan testitapaukset valitaan.

Testausvaiheet

- Sommerville jakaa testauksen kahtia:
  - *Komponenttitestauksessa* (Component testing) ohjelmaa testataan osina. Testatut osat kootaan yhteen ja testaan koottuina.
    - Komponenttitestaus kuuluu pääosin ohjelmiston kehittäjille (projektiryhmälle).
  - *Järjestelmätestauksessa* (System testing) testataan järjestelmää kokonaisuutena.
    - Järjestelmätestaus kuuluu pääosin ulkopuoliselle testausryhmälle.

Järjestelmätestaus

- Työvaihe on *liittävä:*
  - Kuhunkin osajärjestelmään liitetään komponentteja ja testataan komponenttien yhteistyö. Tämä on *integrointitestausta.*
  - Komponentit lisätään ja testataan yksi kerrallaan, kunnes kaikki osajärjestelmän komponentit on liitetty ja testattu.
  - Kun kaikki osajärjestelmät on koottu ja testattu, testataan vielä, että osajärjestelmät toimivat yhdessä oikein.

Integrointitestaus

- Integrointitestauksessa testataan valmiiden yksittäin toimivien komponenttien yhteistyö osajärjestelmässä.
- Integrointia voidaan tehdä ylhäältä alas:
  - Ensin tehdään osajärjestelmän runko ohjauskomponenteista, minkä jälkeen niihin liitetään toiminnat toteuttavat komponentit yksi kerrallaan.

Integrointitestaus 2

- Integrointia voidaan tehdä yhtä lailla alhaalta ylös:
  - aloitetaan toimintakomponenteista ja edetään kohti korkean tason komponentteja.
- Käytännössä tehdään ns. voileipätestausta, missä integrointitestataan sekä alhaalta ylöspäin että ylhäältä alaspäin.

Rasitustestaus

- Integrointitestauksen jälkeen järjestelmän kriittisiä ominaisuuksia voidaan testata:
  - suorituskykyä,
  - luotettavuutta ja vikasietoisuutta,
  - turvallisuutta.
- Rasitustestauksessa ohjelma viedään äärirajoille ja mielellään vielä niiden yli.

Hyväksymistestaus

- *Hyväksymistestaus* (Acceptance testing, Sommervillella Release testing) tehdään sen jälkeen, kun integrointiestaus on saatu valmiiksi.
- Hyväksymistestauksen tarkoituksena on varmistaa, että tuote on sellaisessa kunnossa, että se voidaan antaa asiakkaalle tuotantokäyttöön.

Yksikkötestaus

- Komponenttitestaus, tai *yksikkötestaus,* (Component testing / Unit testing) tehdään ennen järjestelmätestausta.
  - Kaikille komponenteille tai ohjelmiston osille ei tehdä yksikkötestausta. Tällaisia osia ovat esimerkiksi valmiina ostetut *COTS-komponentit* (Commercial Off the Shelf).
  - Yksikkötestaus ja koodaus ovat sidoksissa: koodatessa tehdään myös yksikkötestejä.

Yksikkötestauksen tasot

- Yksikkötestausta tehdään jakamattomille ohjelman osille:
  - *Olioille*.
    - Olioa ei kannata hajottaa erikseen testattaviksi metodeiksi, sillä olion metodit ovat yleensä vahvasti sidoksissa toisiinsa.
  - *Koosteisille olioille tai olioryppäille*.
    - Joskus oliot ovat niin vahvasti sidoksissa toisiinsa, että niitä ei voi testata erikseen.

Yksikkötestauksen tasot 2

- Komponenteille.
  - Yleensä komponentin toteuttavat oliot testataan erikseen ja integroidaan sen jälkeen komponentiksi. Tämän jälkeen komponentti voidaan vielä testata koosteisten olioiden tapaan jakamattomana kokonaisuutena.
  - Jos komponentit ovat kovin isoja, niiden toiminnallisuus kannattaa testata osina. Tällöin kannattaa erityisesti keskittyä komponentin rajapintoihin ja integrointiin.

Rajapintatestaus

- Komponenttitestauksen oleellinen osa on rajapintatestaus. Siinä etsitään virheitä, jotka johtuvat rajapintojen virheistä tai vääristä rajapintojen oletuksista.
- Rajapinnat ja rajapintatestaus ovat erittäin tärkeitä, sillä sekä olioiden että komponenttien palvelut määritellään niiden rajapintojen kautta.

Rajapinnat

- Rajapintoja:
  - Parametrien välitysrajapinnat.
  - Jaetun resurssin rajapinnat: muisti ym.
  - Proseduraaliset rajapinnat: palvelukutsut
  - Viestinvälitysrajapinnat.
- Havaittavia virheitä:
  - Rajapintaa käytetään väärin.
  - Rajapinnan toimintaa ei ymmärretä.
  - Rajapinnan käytön ajoitus on väärä.

7. Ohjelmiston evoluutio

- Ohjelmiston käyttöönoton jälkeen se on vasta elinkaarensa alkupäässä.
  - Käyttö voi kestää vuosikymmeniä.
- Käytön aikana
  - ohjelmistolle tulee uusia vaatimuksia,
  - vanhat vaatimukset muuttuvat,
  - ohjelmistosta täytyy korjata virheitä ja
  - ohjelmisto täytyy saada toimimaan uudessa järjestelmässä.

Ohjelmiston muutospaineet

- Ohjelmistoon syntyy muutospaineita. Nämä aiheuttavat *ohjelmiston evoluution* (software evolution).
- Prosessia, jossa ohjelmiston muutospaineita helpotetaan muokkaamalla ohjelmistoa, kutsutaan *ohjelmiston ylläpidoksi* (software maintenance).

Ohjelmiston evoluution kustannukset

- Suurin osa ohjelmistoon kuluvista resursseista tarvitaan ohjelmiston luovuttamisen jälkeen.
  - Ohjelmiston evoluution kustannus voi olla 50-90% kaikista kustannuksista.
- Näin ollen pääosa ohjelmistotuotannon kustannuksista tulee uusien tuotteiden kehitystyön sijaan vanhojen tuotteiden ylläpidosta.

Ylläpidon tekijät

- Optimitilanteessa ohjelmiston kehittänyt yritys jatkaa kehitystyötä luovutuksen jälkeen. Tällöin ohjelmiston kehitys jatkuu saumattomasti.
- Joskus kehitystyö jätetään osittain tai kokonaan asiakkaalle. Tällöin kehittäjillä ei ole käytössään kaikkea aiemmassa kehitystyössä tehtyä dokumentaatiota ja käytännön kokemusta.

Lehmanin lait

- Ohjelmiston evoluutio on muutoksen hallintaa. Työvaiheesta Lehman ja Belady kehittivät joukon lakeja, jota kutsutaan Lehmanin laeiksi.
- Lehmanin lait selittävät ohjelmiston muutospaineita ja niiden vaikutuksia ohjelmistoon ja sidosryhmiin.
- Lakeja ei ole todistettu, joten ne ovat oikeastaan teoreemoja (otaksumia).

Lehmanin 1. laki

- Jatkuvan muutoksen laki:
  - Todellisessa käytössä olevan ohjelmiston täytyy muuttua tai ajan kanssa menettää hyödyllisyyttään käyttöympäristössä.
- Lain mukaan ohjelmiston evoluutiota ei voida välttää. Ohjelmistot tulevat aina tarvitsemaan muutoksia elinkaarensa aikana.

Lehmanin 2. laki

- Kasvavan monimutkaisuuden laki:
  - Ohjelmistoa muutettaessa sen rakenne muuttuu entistä monimutkaisemmaksi (ja siten virhealttiimmaksi). Kehitys voidaan pysäyttää, mutta se vaatii resursseja.
- Laki on eräänlainen ohjelmiston entropian laki. Sen mukaan ajan kanssa järjestelmän rakenne hajoaa, jos kehitykseen ei varauduta lisäresurssein.

Lehmanin 3. laki

- Ison ohjelmiston evoluution laki:
  - Ohjelmiston evoluutio ohjaa itse itseään. Ohjelmiston koko, julkaisuaika, raportoitujen virheiden määrä pysyvät samassa suuruusluokassa koko ohjelmiston elinkaaren ajan.
- Tämä laki on ylläpidon ajankäytön laki. Sen mukaan isojen ohjelmien ylläpito määräytyy jo kehitystyövaiheessa.

Lehmanin 4. laki

- Kehitystyöryhmän stabiilius:
  - Kehitystyön tehokkuus pysyy likimain vakiona ohjelmiston evoluutiovaiheen ajan.
- Lehmanin 4. lain mukaan ohjelmiston evoluutiota ei voida nopeuttaa.
- Yhdessä 3. ja 4. laki sanovat, että evoluutio on jokseenkin riippumaton hallinnan päätöksistä. Evoluutio on eräänlainen ohjelmistojen luonnonlaki.

Lehmanin 5. laki

- Säilytyksen laki:
  - Ohjelmiston elinkaaren aikana jokaisen uuden version esittelemät muutokset ovat likimain samansuuruiset verrattuna esittelytiheyteen.
- Lain mukaan isot muutokset ohjelmistoon synnyttävät paljon virheitä.
- Virheiden korjaamiseen menee aikaa; seuraava uusi versio tulee myöhemmin.

Lehmanin loput lait

- Viimeiset Lehmanin lait käsittelevät sitä, miten asiakkaat näkevät ohjelmiston evoluution. Ne voidaan tiivistää seuraavaan sanomaan:
  - mitä vähemmän ohjelmistoa ylläpidetään, sitä tyytymättömämmiksi ohjelmiston käyttäjät tulevat ajan kanssa.
- Lehmanin lait kuvaavat aika hyvin isojen räätälöityjen ohjelmistojen evoluutiota.

Ohjelmiston ylläpito

- Ylläpito on yleinen termi prosessille, jonka avulla jo luovutettua ohjelmistoa muutetaan sen elinkaaren aikana.
- Ylläpidosta on yleensä vastuussa erillinen ylläpitoryhmä.
- Ylläpidossa tehdään yleensä kohtuullisen pieniä muutoksia. Ohjelmiston arkkitehtuuri ei muutu (yleensä evoluutiossa se voi muuttua).

Ylläpitotyypit

- Ylläpitoa on kolmea eri tyyppiä:
  1. Ohjelmistovirheiden korjaus (korjaava ylläpito).
  2. Ohjelmiston sovittaminen uuteen ympäristöön (sovittava ylläpito).
  3. Ohjelmiston toimintojen muokkaus tai uusien toimintojen lisäys (täydentävä ylläpito).
- Usein ohjelmiston muutos vaatii kaikkia kolmea ylläpitotyyppiä.

Ylläpidon kustannukset

- Ylläpito voi maksaa 2-100 kertaa kehitystyön kustannusten verran:
  - Ylläpito on pitkäaikainen prosessi: jopa 25v
  - Ylläpitoryhmä ei välttämättä tunne kehitystyössä tehtyjä menetelmiä ja tuloksia, tai kehitystyöstä ei ole käytettävää dokumentaatiota
  - Ohjelmiston ikääntyessä sen arkkitehtuuri degeneroituu è ratkaisut vaikeutuvat.

Ylläpitotyyppien osuudet

- Tutkimusten mukaan ylläpitotyyppien osuudet ylläpidosta ovat seuraavaa suuruusluokkaa:
  - Ohjelmistovirheiden korjaus: 17%.
  - Ohjelmiston sovittaminen uuteen ympäristöön: 18%.
  - Ohjelmiston toimintojen muokkaus tai uusien toimintojen lisäys: 65%.

Evoluutioprosessi

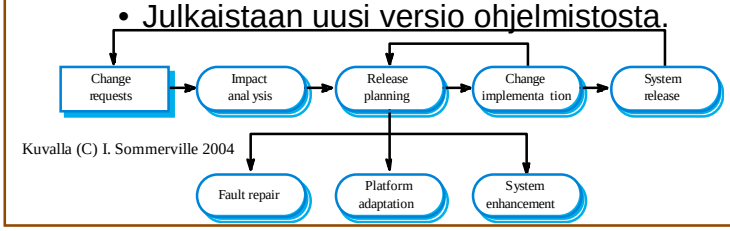
- Ohjelmiston evoluutioprosessi vaihtelee huomattavasti. Prosessiin vaikuttaa
  - ylläpidettävän tuotteen tyyppi,
  - kehitystyössä käytetty prosessi ja
  - ylläpitoon osallistuvien henkilöiden ammattitaito.
- Evoluutioprosessi voi vaihdella täysin epämuodollisesta prosessista hyvin tarkasti ohjattuun prosessiin.

Evoluutioprosessin tehtävät

- Evoluutioprosessiin liittyy
  - *vaikutusanalyysi* (impact analysis) (Sommervillella myös muutosanalyysi, change analysis)
    - Selvitetään, miten paljon muutos vaikuttaa ohjelmiston rakenteeseen ja mitä sen toteutus tulisi maksamaan.
  - *julkaisusuunnittelu* (release planning)
    - Päätetään, mitä muutoksia seuraavaan ohjelmistoversioon tehdään. Muutokset aiheuttavat jonkun kolmesta ylläpitotyypistä.

Evoluutioprosessin tehtävät 2

- Evoluutioprosessin tehtävät jatkuvat:
  - *päivityksen toteutus* (change implementation)
    - Toteutetaan päätetyt muutokset
  - *version julkaisu* (system release)
    - Julkaistaan uusi versio ohjelmistosta.



Kuvalla (C) I. Sommerville 2004

Evoluutio- ja kehitystyöprosessien erot

- Evoluutioprosessi eroaa kehitystyön prosessista seuraavissa kohdissa:
  - Evoluutioprosessi ei ala tyhjästä. Ensimmäisenä vaiheena on ymmärtää jatkokehitettävän ohjelmiston rakenne ja tehdyt ratkaisut.
  - Muutokset tulee tehdä sellaisiksi, että ne eivät riko olemassaolevia ratkaisuja. Vaihtoehtona on *ohjelmiston uudistaminen* (software re-engineering).

Ohjelmiston uudistaminen

- Jos näyttää siltä, että ohjelmiston ylläpidon kustannukset alkavat kohota taivaisiin eikä silti voida taata laadukasta lopputulosta, voi olla aika uudistaa ohjelmisto.
- Ohjelmiston uudistamisessa osa ohjelmistosta toteutetaan ja dokumentoidaan uudestaan sellaiseksi, että sen ylläpito helpottuu.

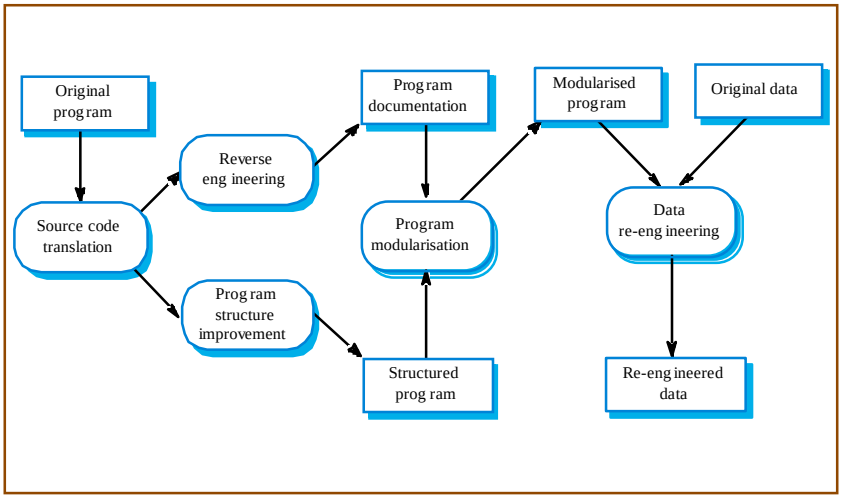
Uudistamisen merkitys

- Ohjelmiston uudistamisessa ohjelmiston toiminnallisuus ei muutu. Ainoastaan toiminnallisuuden toteuttavat ratkaisut muuttuvat. Loppukäyttäjälle ohjelmisto näyttää ja (yleensä) tuntuu samalta.
  - Ohjelmisto saattaa myös tehostua uudistamisessa, jolloin ohjelmiston rajoitukset ja reunaehdot lievenevät. Sen sijaan rajoitukset ja reunaehdot eivät saa koventua.

Uudistamisen ja kehitystyön ero

- Ohjelmiston uudistaminen eroaa kehitystyöstä siinä, että uudistettavasta ohjelmistosta tiedetään jo vaatimukset. Näin vaatimusmäärittely jää pois.
- Lisäksi vanhan järjestelmän suunnitelmasta voidaan käyttää osia, jolloin suunnittelu helpottuu.
  - Esimerkiksi ohjelmiston yleisarkkitehtuuri ei yleensä muutu uudistuksessa.

Uudistusprosessi



Perinnejärjestelmät

- *Perinnejärjestelmä* (legacy system) on vanhentunut ohjelmisto, jota käytetään edelleen, mutta jonka suunnittelusta ja toteutuksesta ei enää tiedetä juuri mitään.
- Perinnejärjestelmä on tavallaan mennyt yli elinkaarestaan. Järjestelmää käytetään vaihtelevista syistä kuitenkin edelleen.

Perinnejärjestelmän evoluutiostrategiat

- Jossain vaiheessa perinnejärjestelmää käyttävä yritys joutuu päättämään, mitä järjestelmälle tehdään:
  - Järjestelmä voidaan hylätä.
  - Järjestelmälle voidaan antaa tekohengitystä ylläpidon keinoin.
  - Järjestelmä voidaan uudistaa.
  - Järjestelmä voidaan korvata uudella vastaavalla järjestelmällä.

Perinnejärjestelmän evoluutiostrategiat 2

- Valittu strategia riippuu siitä, miten arvokas järjestelmä on yritykselle ja miten laadukas se on. Vaihtoehdot ovat:
  - Matala laatu, matala arvo: hylätään.
  - Matala laatu, korkea arvo: uudistetaan tai korvataan uudella järjestelmällä.
  - Korkea laatu, matala arvo: jatketaan ylläpitoa, jos se on kustannustehokasta
  - Korkea laatu, korkea arvo: jatketaan ylläpitoa.

8. Ohjelmistoprojektin hallinta

- Ohjelmistoprojektien koon kasvaessa on törmätty projektinhallinnan ongelmiin, kuten
  - jatkuva, osin huonosti hallittu kasvu,
  - myöhästymiset,
  - huono laatu,
  - budjettien ylitykset,
  - projektien epäonnistumiset jne.
- Muissa insinööritieteissä käytetyt projektin-hallinnan menetelmät eivät tunnu toimivan.

Projektinhallinnan tehtävät

- Projektinhallinnan tehtäviä:

- tehtäväkuvauksen (tarjouksen) laadinta
- projektisuunnitelman laatiminen ja ylläpito
- projektin aikataulun laatiminen ja ylläpito
- kustannusten arviointi ja seuranta
- projektin seuranta ja tarkastukset
- työntekijöiden valinta ja arviointi
- raportointi ja projektin esittely
- käytetyn prosessin kehittäminen

Tehtäväkuvauksen laatiminen

- Usein projekti täytyy “myydä” asiakkaalle tai esimiehille laatimalla tarjous:
  - mitä projektissa luvataan tehdä,
  - projektin kustannus- ja aikatauluarviot ja
  - miksi juuri meidän pitäisi saada tämä projekti.
- Hyvien projektitarjousten laatiminen voi ratkaista yrityksen koko liiketoiminnan:
  - saadaanko sopimus,
  - riittääkö yrityksellä hyviä projekteja = töitä.

Projektin seuranta

- Projektin etenemistä seurataan koko ajan
- Seuranta = verrataan suunnitelmia toteutuneeseen
  - aikataulu: työvaiheiden eteneminen,
  - kustannukset: jakautuminen ja kokonaiskertymä.
- Seurannan keinot:
  - seurantatyökalut,
  - keskustelut projektiryhmän jäsenten kanssa.
- Mitä tehdään, jos alkaa näyttää huonolta?
  - On tärkeää havaita ongelmat ajoissa.

Raportointi ja esittely

- Projektin tilasta raportoidaan sen kestäessä
  - asiakkaille,
  - esimiehille ja
  - laadunvalvontaryhmälle.
- Kirjalliset ja suulliset raportit:
  - mikä on projektin tämänhetkinen tilanne.
- Kommunikaatiotaito on tärkeää!

Projektin suunnittelu

- Projektin suunnittelun keskeinen apuväline on *projektisuunnitelma* (project plan):
  - laaditaan ennen varsinaisia projektin työvaiheita,
  - päivitetään ja täydennetään projektin aikana.
- Projektisuunnitelman tarkoitus on:
  - auttaa seuraamaan projektin etenemistä,
  - antaa mahdollisuudet saada projekti valmiiksi aikataulussa ja
  - antaa keinot huomata aikataulusta lipsumiset mahdollisimman pian.
- Tämä on projektin tärkein dokumentti!

Projektin seurannan vaiheet

1. Selvitetään projektin rajat (aika, henkilöt, budjetti).
2. Selvitetään projektin lähtötilanne (parametrit).
3. Määritellään projektin tarkistuspisteet ja tuotokset.
4. Toistetaan vaiheita 5-12, kunnes projekti päättyy tai keskeytetään:
5. Tehdään aikataulu.
6. Sijoitetaan tehtävät ja henkilöt aikatauluun.
7. Projektiryhmä toimii suunnitelman mukaan.
8. Tarkistetaan eteneminen.
9. Muutetaan tarvittaessa projektin parametreja.
10. Päivitetään tarvittaessa aikataulua.
11. Neuvotellaan tarvittaessa päivityksistä projektin rajoihin ja tuotoksiin.
12. Jos ilmenee ongelmia, tarkastetaan prosessi ja tarvittaessa korjataan projektisuunnitelmaa.

Projektisuunnitelman sisältö

- Välttämättömät tiedot:
  - projektin työvaiheet,
  - projektin tehtävät,
  - projektin aikataulu,
  - projektin riskit.
- lisäksi voi olla (ehkä eri dokumenteissa):
  - laaduntarkkailusuunnitelma,
  - validointisuunnitelma,
  - versionhallintasuunnitelma,
  - ylläpitosuunnitelma,
  - koulutussuunnitelma.

Projektisuunnitelman muutokset

- Suunnitelmaa päivitetään projektin ajan:
  - muutoksia tulee varmasti,
  - jotkut osat saattavat muuttua usein (esim. aikataulu, työnjako),
  - mieluummin väljä kuin tiukka suunnitelma.
- Päivityksen oltava suoraviivaista.
  - Kannattaa erottaa dokumentin pysyvämmät ja muuttuvammat osat.
- Näin projekti pysyy hallinnassa.

Projektisuunnitelma

1. Johdanto = projektin raamit
   - projektin tavoitteet,
   - ylärajat budjetille, ajoitukselle, resursseille ym.
2. Projektiorganisaatio
   - osallistujat ja heidän roolinsa,
   - osallistujien erityistaidot.
3. Riskianalyysi
   - projektin riskit: mikä voi mennä pieleen,
   - riskien todennäköisyys ja vakavuus,
   - riskien ratkaisustrategia.

Projektisuunnitelma 2

4. Laitteisto- ja ohjelmistovaatimukset
   - tarvittavat laitteisto- ja ohjelmistokomponentit,
   - uusien komponenttien kustannusarvio.
5. Työn ositus
   - toiminnot, tarkistuspisteet ja tuotokset.
6. Aikataulu
   - toimintojen väliset riippuvuudet,
   - tarkistuspisteiden vaatimat ajat,
   - henkilöiden työnjako.
7. Seuranta- ja raportointitavat

8.3 Projektin aikataulutus

- Projektin tarkistus-piste *(milestone)*
  - Päättää jonkin perustehtävän.
    - Perustehtävä = jakamaton projektin tehtävä.
  - Tarkistuspisteiden avulla seurataan projektin pysymistä aikataulussa.
- Projektin tuotos *(deliverable)*
  - Projektista saatava asiakkaalle merkittävä tulos.
    - Esim. vaatimus-dokumentti.
  - Kaikkiin tarkistus-pisteisiin ei liity tuotoksia.

Aikataulun laatiminen

1. Jaetaan projekti perustehtäviin.
2. Arvioidaan kunkin perustehtävän kesto.
3. Selvitetään perustehtävien riippuvuudet.
4. Selvitetään, mihin perustehtävään kukin tuotos tai tarkistuspiste liittyy.
   - perustehtävä joka tuottaa tuotoksen.
   - perustehtävä jonka päättyminen = tarkistuspiste
5. Yhdistetään perustoiminnot *toimintoverkoksi* (activity network).

Toimintoverkko

- Toimintoverkko kuvaa perustehtävien järjestyksen ja aikataulun.
- Yleensä projektissa on eräitä tarkistuspisteitä, jossa kaikki käynnissä olevat perustehtävät yhtyvät.
- Samalle projektille voi tehdä useita erilaisia toimintoverkkoja.
  - Yleensä projektipäällikkö tekee toimintoverkon.
  - Lopputulokseen vaikuttaa mm. varautuminen riskeihin.

Toimintoverkko 2

- Sisältää tehtävien keskinäiset riippuvuudet.
  - Joitakin tehtäviä ei voida aloittaa, ennen kuin eräät muut tehtävät on saatu päätökseen.
  - Riippuvuudet on otettava huomioon toiminto-verkossa.
- Tehtävät voivat olla rinnakkaisia.
  - Toisistaan riippumattomia tehtäviä voidaan hoitaa samanaikaisesti.
  - Rinnakkaisuuden määrä riippuu resursseista.
  - Rinnakkaisuus nopeuttaa projektia, mutta lisää aikataulun laadinnan vaatimaa työtä.

Kriittinen polku

- Jokaisella perustehtävällä on
  - aikaisin ajankohta, jolloin se voi alkaa,
  - myöhäisin ajankohta, jolloin sen täytyy alkaa,
  - aikaisin mahdollinen lopetusaika,
  - myöhäisin mahdollinen lopetusaika,
  - joustovara, jonka puitteissa työ ei myöhästy.
- Kaikilla tehtävillä ei ole joustovaraa. Tällaisista tehtävistä muodostuu *kriittinen polku* (critical path).
- Kriittisellä polulla olevat tehtävät eivät saa myöhästyä, tai koko projekti myöhästyy.

Ajoituskaavio

- Yleensä toimintoverkko kuvataan kompaktina *ajoituskaaviona* (activity bar chart).
- Ajoituskaavio sisältää:
  - kaikki perustehtävät kestoaikoineen,
  - tarkistuspisteet,
  - perustehtävien rinnakkaisuuden ja
  - perustehtävien ja tarkistuspisteiden päättymisaikojen joustovaran.

Ohjeita aikataulun laatimiseen

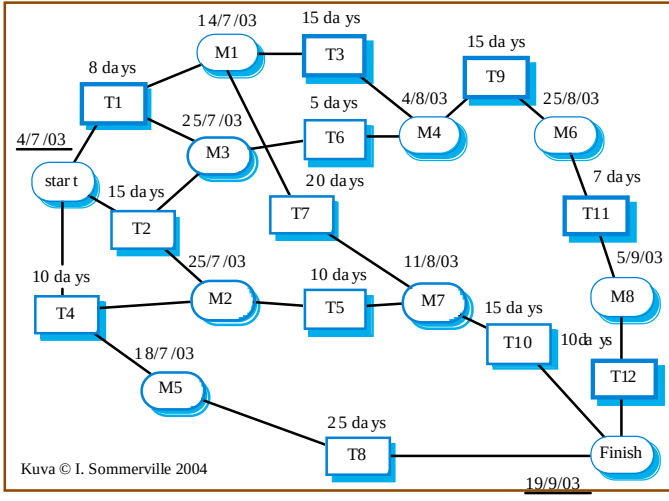
- Suuressa projektissa voi laatia ajoituskaavion kullekin isommalle työvaiheelle erikseen.
  - Työvaiheiden väliset riippuvuudet on huomioitava.
- Pienemmissä projekteissa riittää yksi yhteinen ajoituskaavio.
- Perustehtävien koko on suunnittelupäätös:
  - Hieno jaottelu: suunnittelun ja seurannan vaatima työmäärä on suuri.
  - Karkea jaottelu: poikkeamia ei ehkä havaita ajoissa
  - Pienimmät perustehtävät ~ 1-2 viikkoa.
  - Suurimmat perustehtävät ~ 8-10 viikkoa.

Tehtävien kestot ja riippuvuudet

Task	Duration (days)	Dependencies
T1	8	
T2	15	
T3	15	T1 (M1)
T4	10	
T5	10	T2, T4 (M2)
T6	5	T1, T2 (M3)
T7	20	T1 (M1)
T8	25	T4 (M5)
T9	15	T3, T6 (M4)
T10	15	T5, T7 (M7)
T11	7	T9 (M6)
T12	10	T11 (M8)

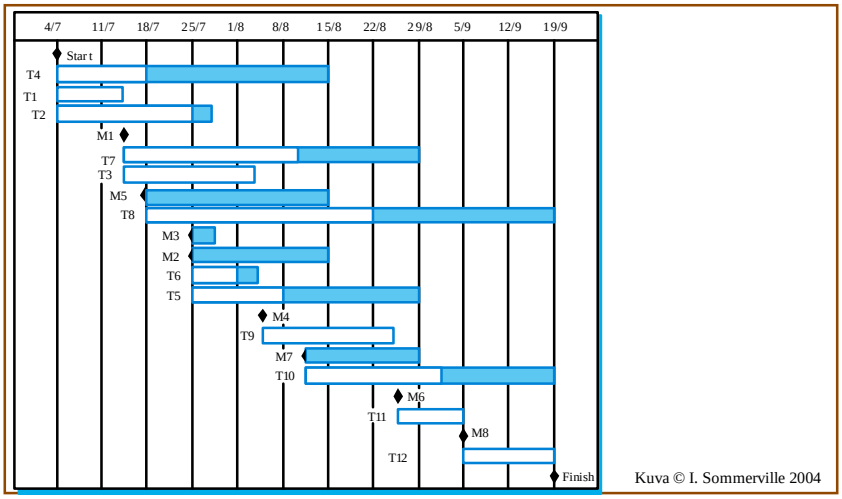
Kuva © I. Sommerville 2000

Toimintoverkko



Kuva © I. Sommerville 2004

Ajoituskaavio



Kuva © I. Sommerville 2004

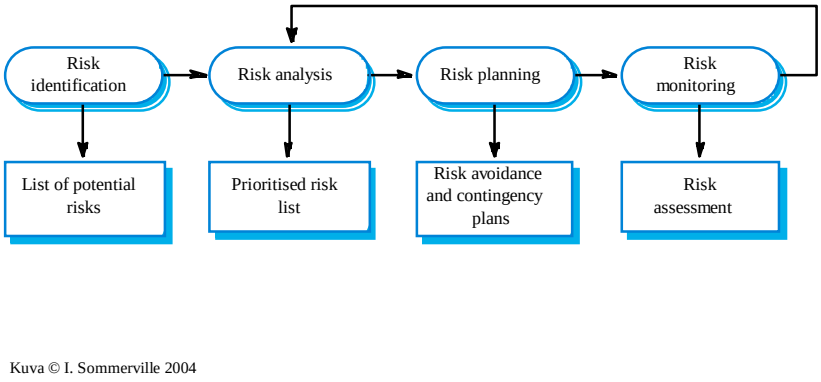
8.4. Riskienhallinta

- Projektin valmistuminen pyritään takaamaan myös tilanteissa, joissa tapahtuu jotakin ei-toivottua odottamatonta.
  - Tunnistetaan onnistumista uhkaavat riskit.
  - Analysoidaan tunnistetut riskit:
    - toteutumisen todennäköisyys,
    - toteutumisen vaikutukset.
  - Suunnitellaan vastatoimet.
  - Seurataan ja päivitetään riskejä projektin ajan.
    - Riskienhallinnan ylläpito.

Mikä on riski?

- Riski on tapahtuma, joka
  - on mahdollinen (todennäköisyys >0 mutta <1)
    - Todennäköisyys = 0: mahdoton tapahtuma.
    - Todennäköisyys = 1: projektin rajoite.
  - toteutuessaan vahingoittaa projektia
- Riski voi olla
  - Projektikohtainen:
    - vaikuttaa aikatauluun tai käytössä oleviin resursseihin,
  - Tuotekohtainen:
    - vaikuttaa kehitettävän tuotteen laatuun,
  - Yrityskohtainen:
    - vaikuttaa (tekijä- tai asiakas-)organisaatioon.

Riskienhallintaprosessi



Kuva © I. Sommerville 2004

Riskien tunnistus

- Pyritään löytämään kaikki riskit, jotka voivat vaikuttaa projektin onnistumiseen
  - Käytännössä unohdetaan kovin epätodennäköiset ja hyvin merkityksettömät riskit.
- Riskit voivat liittyä
  - käytettyyn teknologiaan,
  - henkilökuntaan,
  - organisaatioon,
  - käytettyihin työkaluihin,
  - vaatimuksiin,
  - kustannusten ja aikataulun arviointiin.

Riskien analysointi

- Mietitään kunkin riskin todennäköisyys ja vakavuus:
  - todennäköisyys = miten varmasti riski toteutuu
    - prosentteina tai luokiteltuna (esim. viisi luokkaa vähäisestä erittäin todennäköiseen)
  - vakavuus = miten merkittävä riski on projektille
    - tuhoisa, vakava, siedettävä vai vähäpätöinen
- Päätetään, miten riskeihin varaudutaan
  - mitkä riskit otetaan huomioon suunnitelmissa
  - yleensä on syytä ottaa huomioon ainakin kaikki tuhoisat ja kohtalaisen todennäköiset vakavat riskit

Riskien vastatoimet

- Jokaiselle valitulle riskille suunnitellaan vastatoimet = mitä tehdään jos riski toteutuu.
- Vastatoimet voivat olla
  - riskin välttämistä:
    - pienennetään toteutumisen todennäköisyyttä.
  - vaikutusten minimointia:
    - vähennetään toteutumisen haittavaikutuksia.
  - jatkosuunnitelmia:
    - mietitään riskin toteutuessa seurattavia toimintatapoja.

Riskien seuranta

- Riskien toteutumista seurataan koko projektin elinkaaren ajan.
  - Riskin toteutuessa ryhdytään suunnitelmien mukaisiin toimenpiteisiin
- Projektisuunnitelmaa voidaan joutua päivittämään:
  - Projektin kuluessa ilmenee uusia riskejä.
  - Jonkin tunnistetun riskin todennäköisyys tai vakavuus muuttuu.
  - Jokin sellainen riski toteutuu, jota varten ei ole suunniteltu vastatoimia.

8.5. Ohjelmiston kustannusarviot

- Yleensä jo projektin tarjouksen osana on jonkinlainen kustannusarvio
- Projektin tärkeimmät kustannustekijät:
  - laitteisto- ja ohjelmistokulut,
  - matkat ja koulutus ja
  - työvoimakustannukset (ylivoimaisesti suurin menoerä):
    - palkat,
    - työtilat,
    - sosiaalikulut jne.

Kustannusten arvioinnin ongelma

- Kustannusarvio joudutaan laatimaan aikaisin, sillä sitä tarvitaan neuvoteltaessa tarjouksesta
  - liian korkea arvio è projektia ei saada
  - liian matala arvio è projekti ei valmistu budjetin puitteissa
- Kokonaistyöpanoksen tarve riippuu olennaisesti laadittavasta ohjelmatuotteesta:
  - Minkälaisista osajärjestelmistä tuote koostuu?
  - Minkälaista ammattitaitoa tekijöiltä edellytetään?
  - Kuinka paljon työtä – kuinka kauan työ kestää?

Tuottavuus

- Teollisuudessa kustannusten arviointi perustuu yleensä tuottavuuteen:
  - kuinka paljon valmiita tuotteita tuotantolinjalta valmistuu aikayksikössä.
- tuottavuus = tuotoksen määrä/työpanos.
- Mikä on tuottavuus ohjelmistotyössä?
  - tuotos = kehitettävä ohjelmisto.
  - työpanos = projektin henkilötyö.

Tuotos ja työpanos ohjelmistotyössä

- Työpanosta voidaan mitata käytetyllä työajalla, sillä se vaikuttaa kohtalaisen suoraviivaisesti kustannuksiin (palkat, tilakustannukset).
- Tuotoksen määrää voidaan mitata
  - ohjelmatuotteen kokona (koodirivejä) tai
  - ohjelmatuotteen sisältämän toiminnallisuuden määränä.

Kustannusten arviointitekniikoita

- Kustannusmallit
  - kerätään tietoa aiemmista projekteista
  - kuvataan eri tekijöiden väliset riippuvuudet mallina
- Asiantuntija-arviot
  - sovellusalueen ja ohjelmistotekniikan asiantuntijat arvioivat työmäärän
- Analogiaan perustuva arviointi
  - perustuu aiempiin samantapaisiin projekteihin
- Parkinsonin laki
  - arvio = käytettävissä olevien resurssien kokonaismäärä
- Kilpailuun perustuva arviointi

Kokoon perustuva arviointi

- Tavallisin koon mitta on koodin määrä.
  - Ohjelmatuote koostuu toimivasta koodista.
  - Laaja ohjelmisto sisältää paljon koodia.
  - Koodin määrä on helposti mitattavissa.
- Toisaalta pätee:
  - Koodin laatiminen on vain osa työstä.
  - Koodirivin ilmaisuvoima riippuu käytetystä ohjelmointikielestä.
  - Koodirivejä ei voida laskea alkuvaiheessa.

Toiminnallisuuteen perustuva arviointi

- Kokoon perustuvan arvioinnin sijaan voidaan arvioida ohjelmiston toiminnallisuutta.
- Tällöin lasketaan, mitä kaikkea ohjelmiston halutaan tekevän, ja saadaan tuloksena ohjelmointikielestä riippumaton mitta ohjelmalle.
- Toiminnallisuuden laskun huono puoli on, että tulos ei sellaisenaan kerro mitään. Se on vain luku.

Toiminnallisuuteen perustuva arviointi 2

- Tunnetuin toiminnallisuuden mittari on *toimintopisteanalyysi (*function point analysis). Siinä ohjelmistoa arvioidaan laskemalla siitä
  - ulkoisten syötteiden ja tulosteiden lukumäärä,
  - käyttäjän toiminnan määrä (interaktio),
  - ulkoiset liittymät ja
  - tiedostot.

Oliopisteanalyysi

- Toimintopisteanalyysia uudempi tekniikka on *oliopisteanalyysi* (object point analysis). Siinä laskenta perustuu suunnitelmasta löydettyihin toiminnallisuutta kuvaaviin alkioihin.
- Tuloksena saadaan paljas luku, joka voidaan normalisoida kertomaan tuotteesta.
- OP = Σ [(alkioiden lukumäärä) * painokerroin]

Oliopisteanalyysi 2

- Oliopisteitä lasketaan kolmesta elementistä:
  - Näytöistä (number of separate screens),
  - Tuotetettavista raporteista (number of reports produced),
  - Toteutettavista moduuleista tai komponenteista (number of modules to be developed).
- Kukin elementti arvioidaan erikseen.

Oliopisteanalyysin elementtien arviointi

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

COCOMO

- *COCOMO* (COnstructive COst MOdel) on ohjelmatuotteen kokoon perustuva kustannusten arviointimalli. Sen ensimmäinen versio esiteltiin jo 1981.
- COCOMO perustuu projekteista kerättyyn dataan:
  - vesiputousmallin mukainen prosessi,
  - kokonaan uutta koodia,
  - kokomittana koodirivit.

COCOMO II

- Alkuperäinen COCOMO-malli jäi lopulta vanhanaikaiseksi. Tämän johdosta 1995 esiteltiin COCOMO II.
- COCOMO II toimii edeltäjäänsä paremmin modernissa ohjelmistotuotannossa:
  - spiraalimallin lähestymistapa,
  - tarkentuvat osamallit,
  - prototyypit, uudelleenkäyttö, jne.

COCOMO II:n mallit

- COCOMO II esittelee oikeastaan neljä arviointimallia. Valittava malli riippuu siitä, miten paljon tietoa meillä on kehitettävästä ohjelmistosta:
  - Application composition model
    - käytetään, kun ohjelmisto tehdään suurelta osalta valmiista komponenteista
  - Early design model
    - käytetään, kun vaatimusmäärittely on tehty, mutta suunnittelu ei ole alkanut

COCOMO II:n mallit 2

- Mallit jatkuvat:
  - Reuse model
    - käytetään laskettaessa uudelleenkäytettävien komponenttien vaatimaa työmäärää
  - Post-architecture model
    - käytetään kun järjestelmäarkkitehtuuri on suunniteltu ja järjestelmästä on yksityiskohtaista tietoa.

Application-composition model

- Otetaan esimerkkinä COCOMOn malleista Application-composition model. Sen työmääräarvio perustuu oliopisteisiin:
  - PM = (NAP x (1- %reuse/100)) / PROD.
    - PM = person-months: henkilötyökuukausiarvio
    - NAP = oliopisteiden lukumäärä (COCOMO II puhuu sovelluspisteistä (application points)
    - %reuse = paljonko koodista on valmiina
    - PROD = tuottavuuskerroin (seuraava kalvo)

Tuottavuuskerroin

- Application-composition –mallin arvo riippuu myös seuraavista tekijöistä:
  - miten kokenut tiimi tekee tuotetta ja
  - miten hyviä kehitystyökaluja on käytössä.
- Näille on mallissa annettu kertoimet:

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

Application-composition model 2

- Lopullinen tuottavuuskerroin, PROD, saadaan laskemalla keskiarvo tiimin kokemuksen ja kehitystyökalujen hyvyyden kertoimista:
  - PROD = (tiimikerroin + työkalukerroin)/2
- Malli on tarkoitettu komponenteista koottavien ohjelmien arviointiin, mutta sitä voidaan kyllä käyttää arvioitaessa karkeasti mitä tahansa ohjelmistoa.

9. Laadunhallinta

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

Laadunhallinnan tehtävät

- Laadunhallinta
  - varmistaa, että tehtävät ohjelmistot täyttävät vaaditun laatutason,
  - määrittelee tarvittavat laatustandardit ja laadun saavuttamiseksi tarvittavat menetelmät,
  - varmistaa, että määriteltyjä standardeja ja menetelmiä myös seurataan käytännössä
  - pyrkii kehittämään *laatukulttuurin* (quality culture), jossa laatu on kaikkien vastuulla.

Mitä ohjelmiston laatu tarkoittaa?

- Lyhyesti ohjelmiston laatu tarkoittaa ohjelmiston vastaavuutta määrittelynsä kanssa. Tämä on kuitenkin ongelmallista:
  - Asiakkaan tarkoittama laatu (käytettävyys, luotettavuus, tehokkuus ym.) ei välttämättä ole sama kuin kehitystiimin tarkoittama laatu (ylläpidettävyys, uudelleenkäytettävyys jne.)
  - Kaikkia laadun piirteitä ei osata kuvata yksiselitteisesti (esim. ylläpidettävyys)
  - Vaikka ohjelmisto täyttää määrittelynsä, se ei välttämättä täytä asiakkaan sille asettamia tarpeita. Onko ohjelmisto tällöin laadukas?

Laadun ongelmia

- Edellä luetellut laadun määritelmät voidaan varmentaa vasta valmiista tuotteesta ja asiakkaan palautteesta. Tällöin laatu voidaan *varmentaa* mutta ei *varmistaa.*
- Onnistunut laadunhallinta takaa, että tehtävät tuotteet ovat laadukkaita sekä asiakkaan että kehitystiimin kannalta.

Laatukomponentit

- Parhaiten laatu määritellään *laatukomponenteilla* (quality attributes)
- Laadukas tuote on sellainen, joka täyttää *riittävän hyvin* sille määritellyt laatuattribuutit.
- Laatuattribuuteilla 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 ja miten. Tuotteella on *laatuvaatimuksia.*
- Laatuvaatimukset ovat ei-toiminnallisia vaatimuksia, jotka yleensä liittyvät koko järjestelmään.
  - Joskus laatuvaatimus voi liittyä yksittäiseen toimintoon: esimerkiksi tietyn toiminnon vasteaika on laatuvaatimus.

Laatukomponenttien valinta

- Yleensä laatukomponentit ovat ohjelmistossa 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.

Laadunhallinnan tehtävät

- Mitä isompi organisaatio ja mitä isommat järjestelmät, sitä tärkeämpää on tarkasti ohjattu laadunhallinta.
- Laadunhallinnalla on kolme tehtävää:
  1. *Laadunvarmistus* (quality assurance)
     - Vastataan organisaatiotason laatustandardeista ja –menetelmistä
     - Pidetään yllä *laatukäsikirjaa,* joka sisältää yrityksessä käytettävät laatustandardit ja laadunvarmistusmenetelmät.

Laadunhallinnan tehtävät 2

  2. *Laadun suunnittelu* (quality planning)
     - Selvitetään ja päätetään, mitlä laatuattribuutit ovat tärkeitä aloitettavassa projektissa.
     - Valitaan tiettyyn projektiin parhaiten sopivat laatukäsikirjan standardit ja menetelmät ja täydennetään niitä tarvittaessa.
  3. *Laadunvalvonta* (quality control)
     - Varmistetaan, että projekteissa seurataan sovittuja laatustandardeja ja –menetelmiä
- Pienissä projekteissa ei ehkä tarvita näin muodollista laadunhallintaa.

Laadunhallintaryhmä

- Laadunhallinta kannattaa antaa ohjelmistoprojekteista riippumattomalle *laadunhallintaryhmälle* (quality management team).
  - Laadunhallintaryhmä seuraa projekteja ja raportoi projektien johtoryhmille.
- Laadunhallintaryhmän avulla projekteihin saadaan objektiivinen kehitystyön ulkopuolinen näkökulma.

Standardit

- Standardit ovat sopimuksia, joiden avulla pyritään tasalaatuisuuteen.
- 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.

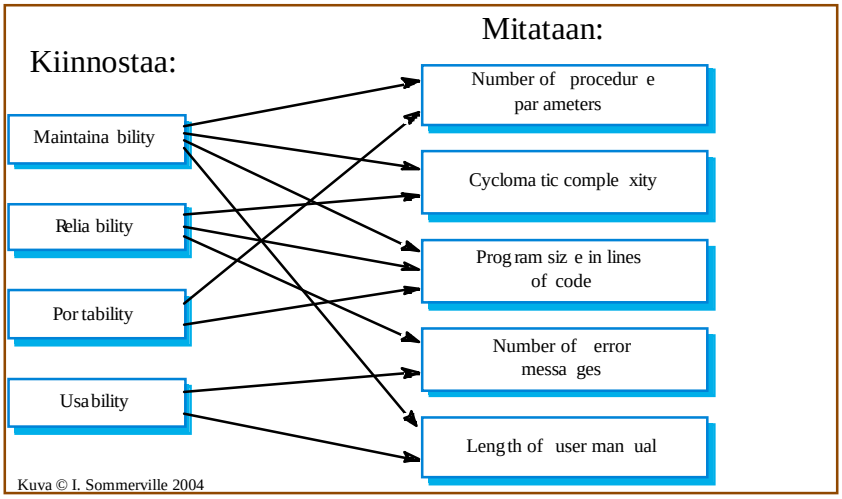
Mittaus

- *Mittaus* (measuring) tarkoittaa, että projektin kestäessä ja päättyessä sekä tuotteesta että prosessista lasketaan sitä kuvaavia arvoja.
  - Projektin aikana tehtävää mittausta käytetään seurattaessa ja ohjattaessa projektin etenemistä ja tuotteen laatua.
  - Projektin lopussa tehtävää mittausta käytetään historiatietona 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 saatujen näyttöjen määrä è arvioidaan käytettävyyttä.

Kiinnostavat ja mitattavat suureet



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 littyvät tuotteen rakenteellisiin ominaisuuksiin. Niillä voi olla välillinen yhteys laatuominaisuuksiin
  - Esim. laaja vaatimusmäärittely è 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.

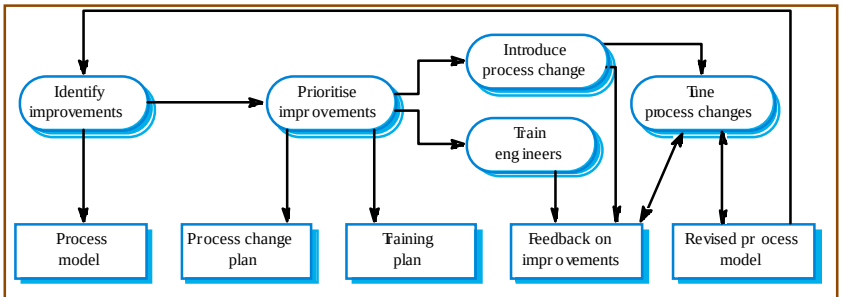
9.2 Prosessin parantaminen

- Prosessia voi olla tarpeen muuttaa:
  - työ- tai sovellusympäristö muuttuu,
  - työn tavoitteet muuttuvat,
  - tarjolla olevat menetelmät kehittyvät,
  - käytetty prosessi on vanhentunut.
- Tarkoituksellisen muutoksen tavoitteita:
  - tuotteen laadun parantaminen,
  - kustannusten vähentäminen,
  - työskentelyn tehostaminen.

Prosessin muutos

- Muutoksen pitää parantaa prosessia:
  - Se voi esitellä uusia menetelmiä, työtapoja ja työkaluja.
  - Siinä voidaan tehostaa työtehtäviä.
  - Se voi lisätä tai vähentää tuotoksia.
- Prosessia ei paranneta paljon kerralla. Uudistukset pitää ottaa kunnolla käyttöön ennen seuraavia parannuksia.

Prosessin parantamisprosessi



Kuva © I. Sommerville 2004

- Kyseessä on jatkuva iteratiivinen prosessi.
- Parantamisprosessin syötteenä ovat tuotantoprosessissa havaitut ongelmat.

Muutosten todentaminen

- Muutokset todennetaan mittaamalla:
  1. mitataan tilannetta kuvaavia suureita ennen muutosta.
  2. mitataan samat suureet muutoksen jälkeen
- Mitattavat suureet valitaan havaitun ongelman perusteella:
  - esim. ongelmana on tehoton testaus:
    - mitataan löytyneiden vikojen määrää, testatun koodin määrää ja käytettyä aikaa.

Prosessin kypsyys

- Prosessien vertailussa puhutaan prosessin *kypsyydestä* (maturity).
  - prosessien vertailuun tarkoitetut mallit luokittelevat prosesseja kypsyydeltään erilaisille tasoille
- Kypsyys kuvaa prosessin kyvykkyyttä:
  - tasaisesti laadukas tuotantolinja,
  - hyvin hallittu prosessi,
  - suunnitelmien ja aikataulun pitävyys,
  - sopeutuminen tarvittaviin muutoksiin.

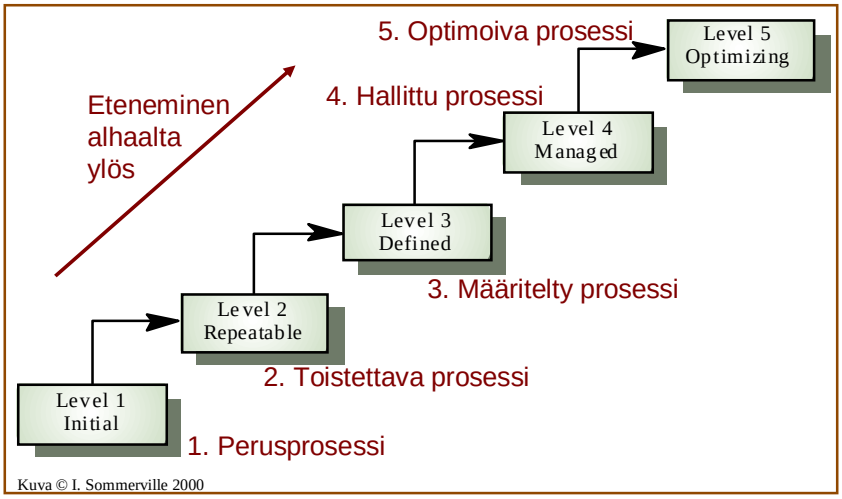
SEI CMM

- Tunnetuin kypsyystasomalli on *SEI CMM* (Software Engineering Institute Capability Maturity Model for software).
- Malli on kehitetty alkuaan Yhdysvaltain puolustusministeriön tarpeisiin. Sitä käytetään esimerkiksi alihankkijoiden tason arviointiin.
- SEI CMM on vaikuttanut useisiin muihin prosessien arviointimalleihin.

SEI CMM:n perusteet

- SEI CMM perustuu kypsyystasoihin. Mitä korkeammalla CMM:n kypsyystasolla prosessi on, sitä kehittyneemmäksi prosessi katsotaan.
- Kullakin tasolla on vaatimuksia, jotka tasolla olevan prosessin on täytettävä.
- Prosessia parannettaessa ei voida siirtyä suoraan ylimmälle tasolle, vaan siirtyminen pitää tehdä taso kerrallaan.

CMM: kypsyystasot



Kuva © I. Sommerville 2000

CMM:n tasot

1. Perusprosessi
   - ad hoc -toiminta
2. Toistettava prosessi
   - projektinhallinta
   - laadunvarmistus
   - tuotteenhallinta
   - voidaan toteuttaa vastaavia projekteja kuin ennenkin
   - ei omaa määriteltyä prosessimallia
3. Määritelty prosessi
   - oman organisaation tarpeisiin määritelty standardoitu prosessi
4. Hallittu prosessi
   - prosessia ja tuotetta mitataan
5. Optimoiva prosessi
   - jatkuva prosessin parantaminen

CMM:n painopistealueet

- Prosessin parantaminen ei voi tapahtua missä järjestyksessä hyvänsä, vaan kuhunkin tasoon liittyy joukko keskeisiä parannuskohtia (key process areas):
  - tietyt asiat täytyy saada kuntoon ennen kuin voi/kannattaa edetä kovin pitkälle muilla osa-alueilla.
- CMM sisältää konkreettisia parannustoimenpiteitä (key practices).

Kypsyystason arviointi

- SEI CMM on kehys, jonka avulla prosessien kypsyyttä voidaan verrata.
- Päätavoitteena on koko yrityksen (organisaation) kypsyyden arviointi
  - Yksittäiset projektit voivat olla tasoltaan paljon korkeammalla kuin yrityksen yleinen taso.
- Arvioinnin tekee ulkopuolinen ryhmä, jolla on arviointiin vaadittava koulutus.

Kypsyystasomallin käyttö

- Arvioinnin tavoitteena on saada
  - puolueeton kuva yrityksen kokonaistasosta ja
  - tärkeimmät korjattavat kohdat.
- Yritystä ei voi kuvata yhdellä numerolla, joten osa-alueisiin liittyvä tieto on olennaisinta prosessien parantamiselle.
- Kypsyystasomalli soveltuu parhaiten suuriin organisaatioihin.

10. Yhteenveto

- Ohjelmistotuotanto jakaantuu selkeästi seuraaviin osa-alueisiin:
  - projektin perustamiseen: aikataulu ym.,
  - ongelman määrittelyyn: mitä tehdään,
  - ohjelmistosuunnitteluun: miten tehdään,
  - toteutukseen: tehdään se,
  - verifiointiin ja validointiin: varmistetaan ja
  - ylläpitoon: pidetään huolta ohjelmistosta.

Osa-alueet ja perustehtävät

- Osa-alueet muodostavat myös ohjelmistotuotantoprosessin perustehtävät.
- Tehtäviä ei välttämättä suoriteta tässä järjestyksessä eikä edes peräkkäin. Ohjelmistotuotannossa on tilaa erilaisille prosesseille ja prosessimalleille. Kuitenkin nämä osa-alueet ovat aina läsnä kaikissa malleissa.

Projektin suunnittelu - älä aliarvioi

- Vaikka jokainen osa-alue on tärkeä, niistä kaikkein tärkeimmäksi nostaisin projektin sunnittelun.
- Suunnittelu ei ala tuotteesta, vaan prosessista. Ilman tarkkaa suunnitelmaa ei synny laadukasta tuotetta.
  - Tämä osa-alue pettää varsinkin ohtu-projekteissa pahiten.

Realistinen aikataulu

- Ennen ensimmäistäkään vaatimusmäärittelyn työvaihetta projektillanne on oltava selvä ja *realistinen* aikataulusuunnitelma, jota seurataan.
- Kun aikataulu on realistinen, työskentely on miellyttävämpää, tulokset ovat parempia ja työ valmistuu ajallaan.

Loppusanat

- Realistisuus on avainsana aikataulussa. On tuhat kertaa parempi saada aikaan pieni laadukas ohjelmisto, kuin tehdä iso ohjelmisto, joka on vain vähän sinne päin.
- Siitä se alkaa ja siihen se päättyy: hyvästä projektin suunnittelusta. Loput asiat voi kerrata vaikkapa tästä luentomonisteesta ja oppikirjasta.