Ohjelmistojen vaatimusmäärittely

Jukka Paakki
Helsingin yliopisto
Tietojenkäsittelytieteen laitos

(Perustuen Juha Tainan luentomateriaaliin
keväältä 2010)

1. Johdanto

- *Vaatimusmäärittely* (Requirements Engineering) on yksi ohjelmistojärjestelmien kehitystyön perustehtävistä. Se on jossakin muodossa läsnä lähes kaikissa ei-triviaaleja ohjelmistojärjestelmiä toteuttavissa prosesseissa ja –projekteissa.
- Ohjelmistojen vaatimusmäärittelyä on tehty yhtä kauan kuin on tehty ohjelmistoja, mutta tieteenä sitä on tehty 1980-luvun puolestavälistä ja kurinalaisesti 1990-luvun alkupuolelta lähtien.
  - Alkukohdaksi voidaan katsoa vuosi 1993, jolloin pidettiin ensimmäinen vaatimusmäärittelykonferenssi. Toki sitä ennen on jo tehty yksittäistä alan tutkimusta.

Vaatimusmäärittelyn faktat ja vaihtoehdot

- Vaatimusmäärittely tarkoittaa eri ihmisille eri asioita. Oikeastaan vain seuraavista asioista ollaan yleisesti samaa mieltä:
  - Järjestelmälle asetetaan (toiminnallisia ja ei-toiminnallisia) vaatimuksia, jotka sen on toteutettava.
  - Osa vaatimuksista on järjestelmää koskevia rajoitteita.
  - Vaatimukset tulevat *sidosryhmiltä* (stakeholders). Kukin sidosryhmä on jollain lailla tekemisissä järjestelmän kanssa.
  - Vaatimukset on kuvattava sopivalla tavalla järjestelmän ominaisuuksiksi.

Vaatimusmäärittelyn luonne

- Vaatimusmäärittelyssä *kartoitetaan* eli *kartutetaan* (elicitation), *arvioidaan* (evaluation), *määritellään* (specification), *dokumentoidaan* (documentation), *analysoidaan* (analysis) *ja muutetaan* (evolution) järjestelmään kohdistuvia *tavoitteita* (objectives) ja *oletuksia* (assumptions) sekä sen *toiminnallisuutta* (functionality), *laatuominaisuuksia* (qualities) ja *rajoituksia* (constraints).
- Edellinen laaja määritelmä voidaan tiivistää:
  - Vaatimusmäärittelyssä selvitetään, mitä järjestelmältä vaaditaan ja miten löydetyt vaatimukset saadaan kuvatuksi jatkokehitykseen soveltuvalla tavalla.
  - Osa vaatimuksista koskee järjestelmän ohjelmistolla toteutettavaa osaa, jolloin on kyse *ohjelmistovaatimuksista* (software requirements).

Vaatimusmäärittelyn alkuajat

- Ohjelmistotuotannon alkuaikoina vaatimuksia ei sen kummemmin määritelty. Ohjelmistot olivat pieniä, määrittely oli melko helppoa ja sidosryhmiä oli vähän.
- Ohjelmistojen monimutkaistuessa niiden laatu alkoi kärsiä kehnosti tehdystä vaatimusmäärittelystä.
- Jo 1976 julkaistiin empiirinen tutkimus: ”Software requirements: Are they really a problem?” Sen mukaan riittämättömät, epätäydelliset, epäyhtenäiset tai moniselitteiset vaatimukset ovat yleisiä ja vaikuttavat huomattavasti ohjelmistojen laatuun (Bell ja Thayer, 1976).
- Bell ja Thayer totesivat, että vaatimukset eivät ilmesty tyhjästä. Niitä täytyy tuottaa, tarkistaa ja korjata.

Vaatimusmäärittely tänään

- Vaatimusmäärittely on edistynyt paljon 1970-luvun jälkeen, mutta siitä huolimatta se on ehkä kehittymättömin ohjelmistokehitystyön perustehtävistä.
- Myös nykyisin puuttuvat ja virheelliset vaatimukset ovat yksi suurimmista projektien epäonnistumisen syistä.
  - Vaatimusmäärittely on vaikeaa. Vaatimusten kartoitus, yhteensovittaminen ja tulkitseminen puhtaasti epäformaalista luonnollisesta kielestä äärimmäisen formaaliksi ohjelmakoodiksi on kaukana triviaalista ongelmasta.
  - Lisäksi modernit ohjelmistojärjestelmät ovat äärimmäisen monimutkaisia. Vaatimuksia ja sidosryhmiä on paljon, ja vaatimuksilla on huomattavasti keskinäisiä riippuvuuksia.

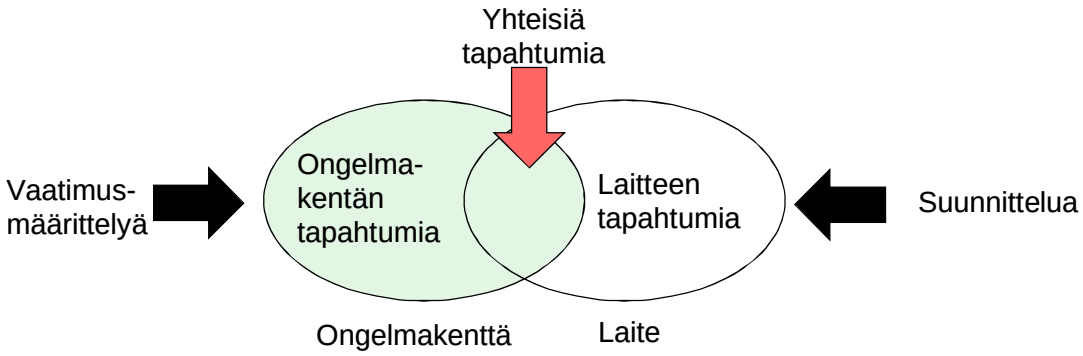
2. Vaatimusmäärittelyn perusteet

- Ennen kuin voimme toteuttaa jonkin ongelman ratkaisevan (ohjelmisto)järjestelmän, meidän on ymmärrettävä ja määriteltävä, mikä tarkkaan ottaen on ratkaistava ongelma.
- Meidän täytyy keksiä, ymmärtää, muotoilla, analysoida ja päättää kolme asiaa:
  - *Mikä* on ratkaistava ongelma?
  - *Miksi* ongelma pitää ratkaista?
  - *Kenen vastuulla* on ongelman ratkaisu?
- Vaatimusmäärittelyssä siis vastataan kolmeen kysymykseen: *Mitä* halutaan? *Miksi* halutaan? *Kuka* ottaa vastuun?

Ongelmakenttä

- Ratkaistava ongelma on osa laajempaa kokonaisuutta, ongelmakenttää.
- *Ongelmakenttä* (problem domain, oppikirjassa ”problem world”) tarkoittaa sellaista teknistä, fyysistä tai organisaatioympäristöä, jossa ongelma esiintyy.
- Ohjelmistoprojektin tehtävänä on tehdä ongelmakentässä toimiva *laite* (machine), joka ratkaisee kyseisen ongelman.
  - Laite sisältää ohjelmiston lisäksi laitteiston, jossa ohjelmisto toimii. Laitteistovaatimukset ovat osa ohjelmistojärjestelmän vaatimuksia mutta eivät ohjelmiston vaatimuksia.
  - Kurssilla laite ja ohjelmistojärjestelmä ovat synonyymeja.

Vaatimusmäärittely vs. suunnittelu



- Vaatimusmäärittelyssä ollaan kiinnostuneita ongelmakentän ominaisuuksista ja laitteen vaikutuksesta ongelmakenttään.
- Suunnittelussa ollaan kiinnostuneita laitteesta.
- Ongelmakentän ja laitteen leikkaus on *laitteen rajapinta.*

Järjestelmä

- *Järjestelmä* (system) tarkoittaa joukkoa (järjestelmä)komponentteja, jotka toimivat yhteistyössä täyttääkseen jonkin tavoitteen.
  - Järjestelmä on laajempi käsite kuin ohjelmistojärjestelmä. Se sisältää mm. käyttäjät.
- Koska vaatimusmäärittelyssä olemme aikeissa ratkaista ongelmakentän ongelman, tarvitsemme itse asiassa kaksi järjestelmää:
  - Nykyjärjestelmä (system-as-is), joka ratkaisi ongelman ennen kehitettävää laitetta.
  - Tuleva järjestelmä (system-to-be), joka otetaan käyttöön, kun laite on valmis.

Nykyjärjestelmä

- Nykyjärjestelmä on käytännössä aina olemassa.
  - Nykyisen järjestelmän ei tarvitse olla tietokonepohjainen. Esimerkiksi ennen TKTL:n ilmoittautumisjärjestelmää labroihin ilmoittauduttiin fyysisesti jonottamalla ja laittamalla nimi ilmoittautumislistalle. Tämäkin oli järjestelmä.
  - Jos ongelmakenttä on täysin uusi, niin nykyistä järjestelmää ei löydy. Esimerkiksi ongelmakenttä ”pysyvä tukikohta kuussa” ei sisällä nykyistä järjestelmää, vaan korkeintaan osia siitä.

Tuleva ohjelmisto

- Laitteeseen kehitettävä ohjelmisto on vain yksi tulevan järjestelmän komponenteista. Sitä kutsutaan *tulevaksi ohjelmistoksi* (software-to-be).
- Muut komponentit vaikuttavat ongelmakenttään. Ne muodostavat tulevan ohjelmiston toimintaympäristön.
- Muita komponentteja ovat esimerkiksi
  - ihmiset ja liiketoimintayksiköt
  - fyysiset laitteet, kuten anturit ja datakaapelit
  - yhteistyössä olevat toiset osajärjestelmät

Ongelmakentän ymmärrys

- Ymmärtääksemme ongelmakentän meidän on ymmärrettävä nykyjärjestelmän tavoitteet, toimintaa säätelevät lait, rajoitteet ja heikkoudet.
- Ymmärtääksemme tulevan ohjelmiston vaatimukset meidän on ymmärrettävä ongelmakentän lisäksi tulevalle järjestelmälle asetettavat vaatimukset.
- Usein ongelmakenttä ei ole stabiili, vaan ratkaisun vaatimukset muuttuvat ajan myötä. Tällöin kehitetään yksi tai useampi välivaihe: seuraava järjestelmä (system-to-be-next).

Ongelmakentän ositus

- Ongelmakenttä voidaan osittaa kolmelle tasolle (vertaa kalvo 7):
  - Miksi tulevaa järjestelmää tarvitaan? (WHY?)
    - Nykyistä järjestelmää tutkimalla löydetään nykyisiä ongelmia, keksitään vaihtoehtoisia ratkaisuja ja ymmärretään ongelmakenttää.
    - Tulevalle järjestelmälle selvitetään tavoitteet, jotka sen tulee täyttää.
  - Mitä tarpeita sen avulla täytetään? (WHAT?)
    - Tulevan järjestelmän tavoitteet toteuttavat palvelut sekä järjestelmän oletukset ja reunaehdot selvitetään.
  - Kuka tulevassa järjestelmässä osallistuu tarpeiden täyttämiseen? (WHO?)
    - Tuleva ohjelmisto, ihmiset, laitteistot ja olemassa olevat (osa)järjestelmät toteuttavat tulevan järjestelmän.

Miksi? Eli Why-taso

- Tulevan järjestelmän kehitystyön taustat tulee selvittää, jotta ongelmakenttää voidaan ymmärtää kunnolla.
  - Miksi nykyjärjestelmä ei kelpaa? Mitkä ovat sen heikkoudet?
  - Mitä tulevalta järjestelmältä odotetaan?
  - Miten tuleva järjestelmä suhtautuu asiakkaan liiketoimintamalleihin ja –tavoitteisiin?
  - Miten tuleva järjestelmä suhtautuu nykyisin käytössä oleviin järjestelmiin?
- Vastausten selvittäminen on yleensä osa *kelpoisuusselvitystä* (feasibility study), jossa katsotaan, kannattaako projektia ylipäänsä aloittaa. Yhtä lailla se voi olla osa vaatimusmäärittelyä.

Ristiriitaiset tavoitteet

- Tulevan järjestelmän tavoitteet saattavat vaihdella sen mukaan, kuka niitä esittää. Eri sidosryhmien tavoitteet saattavat olla ristiriitaisia.
  - Esimerkiksi ongelmakenttä ”luotain toiselle tähdelle” vaatii järjestelmän, joka toimittaa luotaimen perille mahdollisimman nopeasti (fyysikot ja insinöörit) tai mahdollisimman edullisesti (poliitikot).
- Tavoitteiden väliset ristiriidat on selvitettävä ennen kuin projektissa voidaan edetä. Tämä vaatii kompromisseja eri osapuolilta.

Mitä? Eli What-taso

- Kun tulevan järjestelmän tavoitteet tiedetään, voidaan selvittää, mitä *palveluja* (services) järjestelmältä odotetaan näiden tavoitteiden täyttämiseksi.
- Osa palveluista toteutetaan tulevan ohjelmiston avulla. Muut palvelut toteutetaan järjestelmän muiden komponenttien avulla, myös mahdollisesti manuaalisesti.
- Palvelujen ja vaatimusten välillä on monta-moneen-suhde. Palvelu voi ratkaista monta vaatimusta ja yhden vaatimuksen toteuttamiseksi voidaan tarvita useita palveluja. (Yleensä tosin yksi riittää.)
- Palvelujen lisäksi täytyy selvittää järjestelmää koskevat rajoitteet (constraints, esimerkiksi hinta) ja oletukset (assumptions, esimerkiksi käyttäjäprofiili).

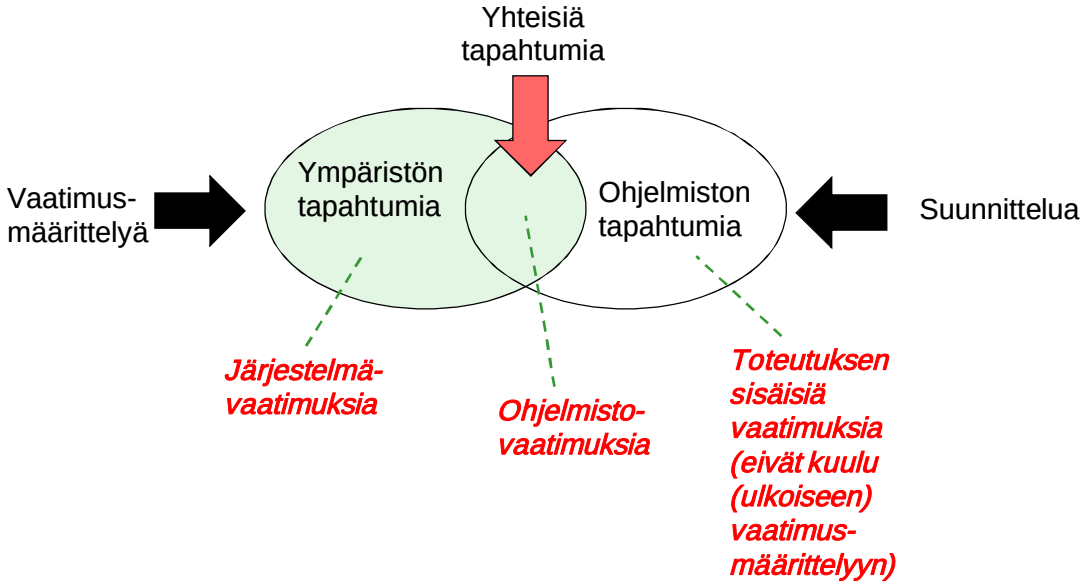
Kuka? Eli Who-taso

- Tulevan järjestelmän vastuut on selvitettävä. Tämä koskee sekä järjestelmässä toimivia ihmisiä että järjestelmän osajärjestelmiä.
- Ilman toimivaa vastuunjakoa kehittynytkin järjestelmä saattaa toimia väärin tai hitaasti.
  - Vastuunjako ymmärretään yleensä ihmisten väliseksi, mutta sitä on yhtä lailla osajärjestelmien välillä.
- Vastuut voidaan jakaa usealla tavalla.
  - Hyvä vastuunjako pienentää What-tason epäonnistumisen riskiä.

Ohjelmiston toimintaympäristö

- Samalla lailla kun ongelmakenttä ja laite eroteltiin toisistaan, myös tulevan ohjelmiston toimintaympäristö ja ohjelmakoodi erotetaan toisistaan.
- Vaatimusmäärittelyssä selvitetään, millä ehdoilla tuleva ohjelmisto toimii (toimintaympäristö) ja miten ohjelmisto kommunikoi ulkomaailman kanssa (ohjelmiston ja toimintaympäristön rajapinta).
- Toimintaympäristön vaatimukset ovat *järjestelmävaatimuksia* (system requirements)
- Rajapinnan vaatimukset ovat *ohjelmistovaatimuksia* (software requirements).

Ohjelmiston toimintaympäristön kaavakuva



Järjestelmävaatimukset

- Järjestelmävaatimus tarkoittaa sellaista toimintaympäristön vaatimusta, jonka tuleva ohjelmisto toteuttaa yhteistyössä muiden järjestelmän komponenttien kanssa. Nämä vaikuttavat suoraan ongelmakenttään.
- Esimerkiksi seuraava on järjestelmävaatimus:
  - Luotain aloittaa jarrutuksen, kun sen etäisyys x tähdestä on välillä 0,5 AU ≤ x ≤ 1,5 AU (AU = maan keskietäisyys auringosta).

Ohjelmistovaatimukset 1

- Ohjelmistovaatimus tarkoittaa sellaista toimintaympäristön vaatimusta, jonka ohjelmisto toteuttaa yksin. Nämä vaikuttavat välillisesti ongelmakenttään rajapinnan kautta.
- Esimerkiksi seuraava on ohjelmistovaatimus:
  - Muuttujan ’etäisyysTähdestä’ arvo on enintään *1,5* aina silloin kun muuttujalla ’jarrutus’ on arvo *True*.
- Ohjelmistovaatimukset ovat myös järjestelmävaatimuksia, koska ne ovat osa ongelmakenttää. Järjestelmävaatimukset eivät usein ole ohjelmistovaatimuksia.

Ohjelmistovaatimukset 2

- Ohjelmistovaatimukset tulevat (myös) ohjelmistokehittäjien käyttöön, joten ne on kuvattava heidän ymmärtämällään tavalla.
- Ohjelmiston sisäisiä tapahtumia ei voi havaita toimintaympäristössä.
  - Täten toteutuksen sisäiset vaatimukset ovat ainoastaan ohjelmistokehittäjille tarkoitettuja.
  - Esimerkiksi: muuttujalla 'virhekoodi' voi olla arvot *1* (laskutoimituksen ylivuoto) ja *2* (laskutoimituksessa nollalla jako).

Reunaehdot ja oletukset

- *Reunaehto* (domain property) on ongelmakentän ominaisuus. Se on aina voimassa, eikä sitä voi kiertää mitenkään.
- Reunaehdot liittyvät yleensä fysiikan lakeihin. Esimerkiksi seuraava on reunaehto:
  - Luotaimen lähettämät radiosignaalit kulkevat enintään valonnopeudella.
- *Oletus* (assumption) on väite, jonka oletetaan pätevän ongelmakentässä ja joka muotoillaan ongelmakentän tapahtumien avulla. Esimerkiksi seuraava on oletus:
  - Ohjelmistossa laskettu luotaimen etäisyys tähdestä on sama kuin luotaimen todellinen etäisyys tähdestä.

Järjestelmä- ja ohjelmistovaatimusten sykli

- Järjestelmä- ja ohjelmistovaatimusten roolit on pidettävä selvinä. Oikein määritelty ohjelmisto ohjaa seuraavaa sykliä:
  1. Tuleva ohjelmisto saa syötteitä syötelaitteilta, kuten syöteantureilta.
  2. Syötteet vaikuttavat ohjelmiston toimintaan ohjelmistovaatimusten mukaisesti.
  3. Ohjelmisto tuottaa vaatimusten mukaisia tulosteita tulostuslaitteille, kuten ohjausantureille.
  4. Ohjausanturit ohjaavat laitteistokomponentteja, jolloin ohjelmiston toimintaympäristö muuttuu. Muutokset ovat järjestelmävaatimusten mukaisia.
  5. Syötelaitteet havaitsevat toimintaympäristön muutoksen. Sykli alkaa alusta.

Ohjelmistovaatimusten luokat

- Vaatimukset on perinteisesti jaettu kahteen luokkaan: *toiminnallisiin vaatimuksiin* (functional requirements) ja *ei-toiminnallisiin vaatimuksiin* (non-functional requirements).
- Toiminnalliset vaatimukset liittyvät tulevan ohjelmiston tarjoamiin palveluihin (”mitä tekee”).
- Ei-toiminnalliset vaatimukset määrittelevät, mitä rajoituksia tulevan ohjelmiston palveluilla on (”miten tekee”).
- Reunaehdot ovat sukua ei-toiminnallisille vaatimuksille. Myös reunaehdot rajoittavat järjestelmän toimintaa, mutta erona ei-toiminnallisille vaatimuksille on, että niistä ei voi neuvotella.

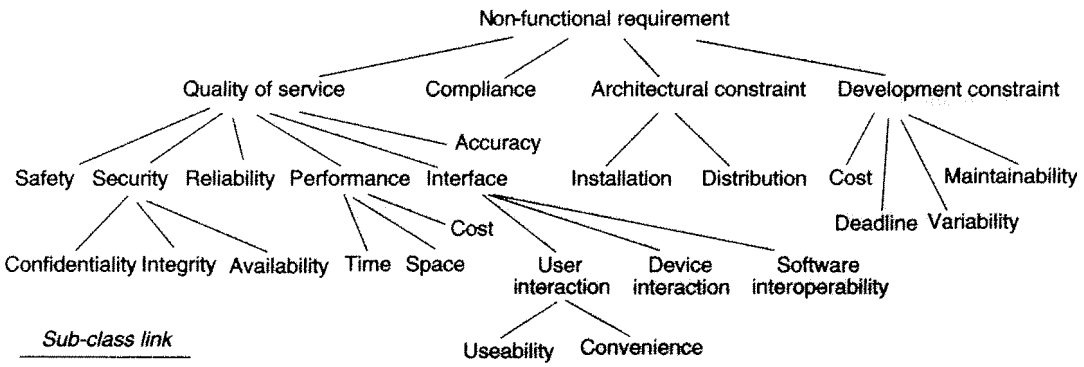
Toiminnalliset vaatimukset

- Toiminnalliset vaatimukset määrittelevät, miten tuleva ohjelmisto vaikuttaa ympäristöönsä. Ne vastaavat aiemmin mainittuun Mitä-kysymykseen (What?). Esimerkki:
  - Luotaimen hallintaohjelmisto lähettää pyydettäessä raportin omasta tilastaan.
- Toiminnalliset vaatimukset voivat myös viitata toimintaympäristön tilaan, joka vaikuttaa tulevan ohjelmiston toimintaan. Esimerkki:
  - Luotaimen hallintaohjelmisto kytkee aurinkopaneelit pois päältä, kun paristojen varaustaso x toteuttaa ehdon $95\ \% < x \leq 100\ \%$.

Ei-toiminnalliset vaatimukset

- Ei-toiminnalliset vaatimukset määrittelevät ehtoja sille, miten tulevan ohjelmiston pitää täyttää toiminnalliset vaatimukset tai (harvemmin) miten tulevan ohjelmiston kehitystyötä pitää tehdä. Esimerkkejä:
  - Kaikki luotaimen radioliikenne on salakirjoitettua.
  - Luotaimen hallintaohjelmistosta tehdään samoilla määrityksillä versio sekä Windowsiin että Linuxiin.
- Ei-toiminnallisia vaatimuksia voi toisinaan olla vaikea havaita, mutta onneksi niistä on tehty useampikin kattava luokittelu.

Ei-toiminnallisten vaatimusten luokittelu



Kuvalla (c) Axel van Lamsweerde, 2009

Ei-toiminnallisten vaatimusten luokkia

- Laatuvaatimukset (quality (of service))
  - Kuvaavat ohjelmiston laatuun liittyviä ominaisuuksia vastaten kysymykseen ”Miten hyvin” (How well). Näillä on yhteys ohjelmiston laatutekijöihin (quality factors, quality attributes).
- Mukautuvuusvaatimukset (compliance)
  - Kuvaavat ohjelmiston suhdetta kansallisiin ja kansainvälisiin lakeihin, sosiaalisiin sääntöihin, poliittisiin tai kulttuurisiin rajoituksiin, standardeihin jne.
- Arkkitehtuurivaatimukset (architectural constraint)
  - Kuvaavat, miten tuleva ohjelmisto liittyy toimintaympäristöönsä.
- Kehitystyön vaatimukset (development constraint)
  - Kuvaavat, mitä ohjelmistotekniikan prosesseja ja menetelmiä on käytettävä, mitkä ovat kehitystyön sallitut kustannukset jne. Nämä eivät liity ohjelmiston käyttöön.

Laatuvaatimukset 1

- Käyttöturvallisuus (safety): millä tavoin ohjelmiston on osaltaan estettävä ympäristössä tapahtuvat onnettomuudet yms.
- Turva (security): millä tavoin ohjelmiston on turvattava ympäristön resurssit ja tiedot niiden väärinkäytöltä.
  - Luottamuksellisuus (confidentiality): luottamuksellista tietoa saavat käyttää vain siihen valtuutetut.
  - Eheys (integrity): tiedon on oltava oikeellista ja sitä saavat tuottaa ja päivittää vain siihen valtuutetut.
  - Käytettävissä oleminen (availability): resurssin tai tiedon on oltava siihen valtuutettujen käytettävissä milloin tahansa.
- Luotettavuus (reliability): miten pitkään ohjelmiston on pysyttävä toimintakunnossa riippumatta ympäristössä tapahtuvista häiriöistä.
- Suorituskyky (performance): ohjelmistoa koskevat vasteaika-, tila-, kuormitus- ja kustannusrajoitteet.
  - Aika (time): paljonko palvelun suorittamiseen saa kulua tietokoneaikaa.
  - Tila (space): paljonko palvelun suorittamiseen saa kulua tietokoneen muistia.
  - Kustannussäästöt (cost): paljonko uuden ohjelmistoratkaisun tai sen yksittäisten palvelujen on tuotettava säästöä (rahassa tai työssä) verrattuna nykyjärjestelmään.

Laatuvaatimukset 2

- Liittymät (interface): millä tavoin ohjelmisto liittyy järjestelmän muihin komponentteihin.
  - Käyttöliittymä (user interaction): millä tavoin ohjelmiston palvelut tarjotaan järjestelmän käyttäjille.
    - Käytettävyys (useability): (oppikirjassa) palvelun dialogien, syötteiden ja tulosteiden muoto.
    - Käyttömukavuus (convenience): (oppikirjassa) millainen ohjelmiston on oltava, jotta käyttäjät kokisivat sen käyttämisen mukavaksi.
  - Laitteistoliittymät (device interaction): millä tavoin ohjelmisto on vuorovaikutuksessa järjestelmän sisältämien laitteistojen kanssa.
  - Ohjelmistojen välinen yhteentoimivuus (software interoperability): millä tavoin (syötteet, tulosteet, protokollat) ohjelmisto on vuorovaikutuksessa järjestelmän sisältämien muiden ohjelmistojen kanssa.
- Tarkkuus (accuracy): millä tarkkuudella ohjelmiston tuottamien tulosten on vastattava todellisia tulosarvoja.

Arkkitehtuurivaatimukset

- Asennettavuus (installation): millaiseen käyttöympäristöön ohjelmisto on kehitettävä (käyttöjärjestelmä, ohjelmakirjastot jne.).
- Hajautus (distribution): millaisten (maantieteellisesti) hajautettujen organisaatioiden, tietovarastojen ja muiden järjestelmien kanssa ohjelmiston on oltava yhteistoiminnassa.

Kehitystyön vaatimukset

- Kustannukset (cost): paljonko ohjelmiston kehittäminen saa maksaa.
- Takaraja (deadline): milloin ohjelmiston on oltava valmis.
- Monimuotoisuus (variability): missä määrin profiililtaan erilaisten käyttäjäryhmien on saatava ohjelmistolta haluamiaan erityispalveluita.
- Ylläpidettävyys (maintainability): missä määrin ja millä tavoin ohjelmiston kehitysprojektissa on varauduttava ohjelmiston myöhempään ylläpitoon ja jatkokehitykseen.

- Tähän luokkaan kuuluu muitakin ohjelmiston kehitystyötä ohjaavia laatuvaatimuksia, kuten ohjelmiston siirrettävyys (portability), uudelleenkäytettävyys (reusability) ja testattavuus (testability).

Vaatimusmäärittelyprosessi

- Vaatimusmäärittelyllä on prosessi, joka sisältää sen työvaiheet, toimijat ja tuotokset.
- Vaatimusmäärittelyn työvaiheet sisältävät tuotoksen tekemiseen tarvittavat tehtävät.
- Vaatimusmäärittelyn toimijat ovat tulevan järjestelmän (tai ohjelmiston) *sidosryhmiä* (stakeholders).
  - Sidosryhmä tarkoittaa henkilöä tai ryhmää, jonka toimintaan tuleva järjestelmä vaikuttaa, joka on vastuussa tulevan järjestelmän vaatimuksista tai joka osallistuu valmiin järjestelmän hyväksymiseen.
- Vaatimusmäärittelyn tuotos on *vaatimusdokumentti* (requirements document). Se kuvaa sekä löydetyt vaatimukset että tulevan ohjelmiston ja toimintaympäristön välisen rajapinnan.

Vaatimusmäärittelyn työvaiheet 1

- Ongelmakentän ymmärtäminen (domain understanding)
  - Tässä työvaiheessa tutustutaan nykyiseen järjestelmään ja ongelmakenttään mahdollisimman monelta suunnalta. Selvitettäviä asioita ovat:
    - Miten nykyjärjestelmä toimii osana ongelmakenttää. Millaista organisaatiota se palvelee.
    - Mitkä ovat nykyisen järjestelmän tavoitteet, komponentit, säännöt ja tehtävät. Mitä rajoituksia ja sääntöjä sillä on.
    - Mitkä ovat vaatimusmäärittelyn sidosryhmät.
    - Mitkä ovat sidosryhmien mielestä nykyisen järjestelmän vahvuudet ja heikkoudet.

Vaatimusmäärittelyn työvaiheet 2

- Vaatimusten kartutus (requirements elicitation)
  - Tässä työvaiheessa syvennetään tietämystä ongelmakentästä, hankitaan raakatietoa tulevasta järjestelmästä ja sen käyttöympäristöstä sekä etsitään potentiaalisia vaatimuksia:
    - Miten uudella teknologialla ja/tai muuttuneilla liiketoimintamalleilla voidaan korjata nykyisen järjestelmän heikkouksia menettämättä sen vahvuuksia.
    - Mitä nykyistä järjestelmää parantavia tavoitteita tulevalla järjestelmällä on ja mitä erilaisia vaihtoehtoja tavoitteiden täyttämiseen on.
    - Ympäristön ja tekniikan tulevalle järjestelmälle asettamat rajoitteet.
    - Mahdolliset vastuunjaot (Kuka-taso).
    - Tyypillisiä tulevan ohjelmiston ja sen toimintaympäristön välistä vuorovaikutusta kuvaavia skenaarioita.
    - Ongelmakentän ja toimintaympäristön ominaisuudet ja oletukset.
    - Edellisten kohtien kanssa yhteensopivat tulevan ohjelmiston vaatimukset.

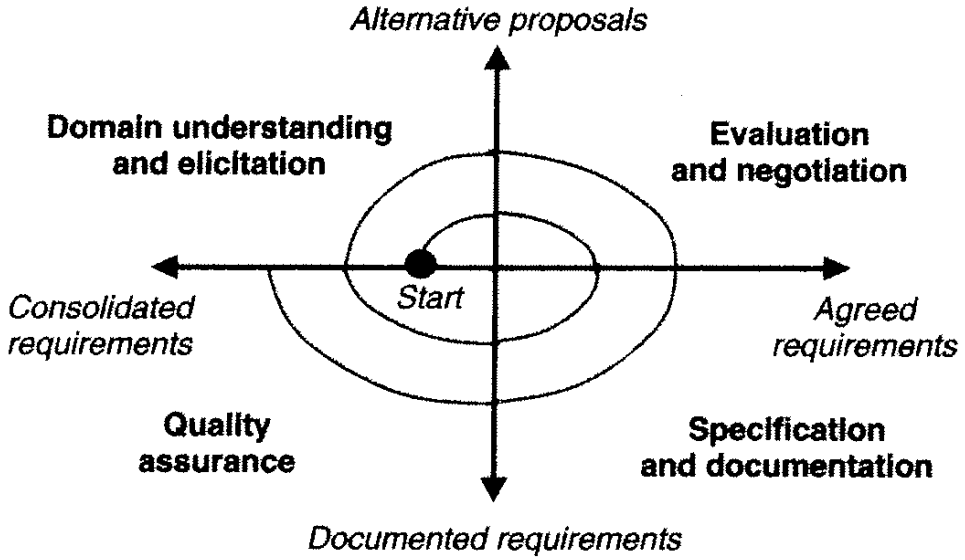
Vaatimusmäärittelyn työvaiheet 3

- Vaatimusten arviointi (evaluation)
  - Tässä vaiheessa ratkotaan kartutusvaiheessa esille nousseet kysymykset ja ongelmat:
    - Vaatimuksiin liittyvät ristiriidat ratkotaan. Eri näkökulmat ja sidosryhmät voivat nähdä ongelmakentän ja tulevan ohjelmiston eri tavoin, mikä voi johtaa ristiriitaisiin vaatimuksiin tai erilaisiin tulkintoihin.
    - Tulevan järjestelmän riskit kartoitetaan ja järjestelmälle tehdään riskianalyysi.
    - Kartutusvaiheessa esille tulleet vaihtoehtoiset ratkaisut arvioidaan. Vaihtoehdoista valitaan paras.
    - Vaatimukset priorisoidaan. Se helpottaa sekä vaatimusten evaluointia että toteutusaikataulun laadintaa.

Vaatimusmäärittelyn työvaiheet 4

- Spesifiointi ja dokumentointi (specification and documentation)
  - Tässä vaiheessa tarkennetaan, strukturoidaan ja dokumentoidaan arviointivaiheessa hyväksyttyjä tulevan järjestelmän ominaisuuksia.
  - Tuloksena saadaan vaatimusdokumentti, joka sisältää tulevan järjestelmän tavoitteet, määritelmät, ongelmakentän ominaisuudet, vastuut, järjestelmävaatimukset, ohjelmistovaatimukset ja käyttöympäristöoletukset. Dokumentti voi sisältää myös perusteluja tehdyille valinnoille, hyväksymistestitapauksia, kustannusarvioita yms.
- Vaatimusten vahvistaminen (requirements consolidation)
  - Lopuksi vaatimusdokumentista varmennetaan, että tuleva järjestelmä täyttää sidosryhmien tarpeet.
  - Samalla varmistetaan vaatimusdokumentin laatu: spesifikaatiot on kuvattu selkeästi ja yhtenevästi, ne ovat ristiriidattomat ja kattavat ongelmakentän.
  - Vahvistettu vaatimusdokumentti toimii syötteenä kehitystyölle.
- Vaatimusmäärittelyprosessi on iteratiivinen. Kaikkea ei selvitetä kerralla, vaan ongelmakentän ymmärrys ja vaatimukset tarkentuvat vaiheittain.

Vaatimusmäärittelyn spiraali



Kuvalla (c) Axel van Lamsweerde, 2009

Vaatimusmäärittelyn laatu (quality assurance) 1

- Hyvä vaatimusmäärittely ja erityisesti hyvä vaatimusdokumentti eivät synny itsestään. Niiden on täytettävä joukko laatutekijöitä:
  - Täydellisyys (completeness)
    - Kaikki tulevalle järjestelmälle asetetut tavoitteet on määritelty.
    - Kaikki mahdolliset tulevan ohjelmiston syötteet ja tulosteet on katettu.
    - Poikkeus- ja virhetilanteet yms. on katettu.
  - Yhdenmukaisuus (consistency)
    - Vaatimukset, oletukset ja reunaehdot ovat keskenään yhteensopivia ja ne on kuvattu yhtenäisesti.

Vaatimusmäärittelyn laatu 2

- Sopivuus (adequacy)
  - Vaatimukset kuvaavat tulevalle järjestelmälle asetettuja todellisia tarpeita.
  - Ohjelmistovaatimukset vastaavat niitä kuvaavia järjestelmävaatimuksia.
  - Reunaehdot kuvaavat ongelmakentän lainalaisuudet oikein.
  - Toimintaympäristöstä tehtävät oletukset ovat realistisia.
- Yksikäsitteisyys (unambiguity)
  - Vaatimusten, oletusten ja reunaehtojen kuvauksissa ei ole moniselitteisyyttä.
- Mitattavuus (measureability)
  - Vaatimusten pitää olla sellaisella tarkkuudella, että niitä voidaan arvioida ja vertailla.

Vaatimusmäärittelyn laatu 3

- Asianmukaisuus (pertinence)
  - Vaatimukset ja oletukset liittyvät tulevalle järjestelmälle asetettuihin tavoitteisiin, eivät toteutukseen.
- Kelpoisuus (feasibility)
  - Vaatimusten pitää olla toteutettavissa annetulla budjetilla, aikataululla ja teknologialla.
- Luettavuus (comprehensibility)
  - Vaatimukset, oletukset ja reunaehdot on kuvattu ymmärrettävästi.
- Hyvä rakenne (good structuring)
  - Vaatimusdokumentti on rakenteeltaan looginen ja yhtenäinen. Termejä ei käytetä ennen niiden esittelyä.

Vaatimusmäärittelyn laatu 4

- Muunneltavuus (modifiability)
  - Vaatimusdokumenttia on mahdollista päivittää ja laajentaa paikallisesti siten, ettei kokonaisuus kärsi.
- Jäljitettävyys (traceability)
  - Jokaisesta vaatimuksesta, oletuksesta ja reunaehdosta tiedetään, mikä sidosryhmä sen on esittänyt ja miksi. Jokaisen vaatimuksen vaikutus tulevaan järjestelmään selviää dokumentista.
- Osa tärkeimmistä laatuvaatimuksista (esim. täydellisyys ja sopivuus) riippuu ongelmakentästä ja tulevalle järjestelmälle asetetuista (implisiittisistä) tavoitteista. Tällaisia vaatimuksia on vaikea arvioida.

Ketterä ohjelmistokehitys ja vaatimusmäärittely

- *Ketterä ohjelmistokehitys* (agile software development) on moderni tapa tehdä ohjelmistoja. Siinä ohjelmistoa rakennetaan iteratiivisesti ja inkrementaalisesti, pieni osa kerrallaan ja lyhyissä sykleissä.
- Ketterä ohjelmistokehitys ja vaatimusmäärittely eivät ole ristiriidassa keskenään. Myös vaatimusmäärittelyn spiraali (kalvo 40) on iteratiivinen prosessi.
- Ketterissä ohjelmistoprojekteissa jokainen sykli aloitetaan kokouksella, jossa asiakas ja kehitystiimi sopivat ko. syklissä toteutettavista vaatimuksista. (Suuri) osa vaatimuksista saadaan aiemmin tuotetusta työlistasta, (pieni) osa asiakkaalta lisäyksinä työlistaan.
- Vaatimusmäärittelyn näkökulmasta jokainen spiraalin sykli lisättynä lyhyellä toteutusvaiheella vastaa ketterän ohjelmistokehityksen sykliä.

Syklin lyhentämisen seurauksia

- Syklin äärimmäinen lyhentäminen johtaa keventyneeseen vaatimusmäärittelyyn:
  - Osa toiminnallisuudesta saadaan suoraan asiakkaalta mahdollisesti hyvin kevyellä arvioinnilla, spesifioinnilla ja laadunvarmistuksella tai jopa ilman näitä vaiheita.
    - Esimerkiksi testitapaukset vastaavat usein spesifikaatiota.
  - Yhdellä kerralla toteutetaan pieni osa tulevan ohjelmiston toiminnallisuudesta.
  - Asiakkaalta saadaan välitön palaute toteutetuista vaatimuksista.

Ketterän kehitystyön vaikutuksia

- Vaatimusmäärittelyn kannalta kaikki projektit eivät ole hyviä *ketteriä* projekteja. Oppikirja listaa seuraavat ehdot:
  - Kaikki sidosryhmät, myös loppukäyttäjät, supistuvat yhdeksi ”asiakkaan” rooliksi.
  - Asiakas on läsnä kehitystiimin työtilassa tai aina tavoitettavissa.
  - Projekti on riittävän yksinkertainen ja ei-kriittinen, jotta ei-toiminnalliset vaatimukset, ympäristön rajoitukset, taustalla olevat oletukset, vaihtoehtoiset toteutustavat ja tulevan järjestelmän riskit voidaan kuitata vähällä vaivalla tai jättää täysin huomioimatta.
  - Asiakas osaa antaa (toiminnalliset) vaatimukset nopeasti, yhtenäisesti (ei ristiriitoja) ja tärkeysjärjestyksessä (ei erillistä priorisointia).
  - Vaatimuksia ei tarvitse spesifioida tarkasti ennen toteutusta.
  - Tiheät julkaisut ovat tärkeämpiä kuin vaatimusten laatu.
  - Uudet tai muuttuvat vaatimukset eivät vaadi raskasta refaktorointia.
  - Kehitystiimi huolehtii myös tulevan ohjelmiston ylläpidosta.
- Onneksi ketteryys ei ole binäärinen ominaisuus. Mitä vähemmän edellisistä ehdoista toteutuu, sitä enemmän aikaa tulee tarjota kullekin vaatimusmäärittelyn syklille.

3. Ongelmakentän ymmärtäminen ja vaatimusten kartutus

- Ongelmakentän ymmärtäminen ja vaatimusten kartutus (OYVK) on vaatimusmäärittelyn spiraalin ensimmäinen työvaihe. Siinä on kaksi toisiinsa vahvasti sitoutunutta osaa:
  - Ongelmakentän ymmärtäminen: selvitetään, missä ympäristössä nykyjärjestelmä ja tuleva järjestelmä toimivat.
  - Vaatimusten kartutus: selvitetään, mitä tulevalta järjestelmältä odotetaan ongelmakentässä.
- Tämä on vaatimusmäärittelyn haastavin työvaihe:
  - Etsittävää tietoa on paljon.
  - Tulokset vaikuttavat kaikkiin ohjelmistokehityksen (ei pelkästään vaatimusmäärittelyn) työvaiheisiin.

Kerättävä tietämys

- OYVK-työvaiheessa kerätään tietämystä, jota tarvitaan tulevan järjestelmän ominaisuuksien kartoittamiseen.
- Työssä tarvitaan ainakin:
  - Tulevan järjestelmän käyttöorganisaation tietämystä: rakenne, liiketoimintatavoitteet, politiikat, roolit ja vastuut.
  - Ongelmakentän tietämystä: alan käsitteet, erityispiirteet, säännöt ja lait.
  - Nykyisen järjestelmän tietämystä: tavoitteet, toimijat, resurssit, tehtävät, työnkulut ja ongelmat.

OYVK:n tulos

- OYVK-työvaiheen tuloksena saadaan alustava raportti
  - nykyjärjestelmästä
  - käyttöorganisaatiosta
  - ongelmakentästä
  - nykyjärjestelmästä löydetyistä ongelmista
  - nykyjärjestelmän tarjoamista mahdollisuuksista (tulevan järjestelmän kannalta)
  - vaihtoehtoisista tavoista ratkaista nykyjärjestelmän ongelmat
  - vaatimuksista ongelmien ratkaisemiseksi

3.1. Sidosryhmien tunnistus ja vuorovaikutus

- Sidosryhmiä voi olla paljon, jolloin niitä kaikkia ei voi ottaa mukaan vaatimusmäärittelyyn. Sopiva otos voidaan valita esimerkiksi seuraavilla kriteereillä:
  - Asema organisaatiossa.
  - Päätäntävalta tulevasta järjestelmästä.
  - Ongelmakentän asiantuntijuus.
  - Altistuminen havaituille ongelmille (eli nykyisen järjestelmän käyttökokemus).
  - Järjestelmän hyväksymiseen vaikuttaminen (käyttöönotossa).
  - Henkilökohtaiset tavoitteet tulevalle järjestelmälle.
- Sidosryhmiä voidaan päivittää prosessin edetessä, kun löydetään uusia näkökulmia.

OYVK:n esteitä 1

- OYVK-prosessi ei ole helppo, vaan siinä on paljon sille ominaisia hankaluuksia. Niiden tunnistaminen helpottaa niiden välttämistä:
  - Sidosryhmien väliset konfliktit
    - Sidosryhmien tavoitteet järjestelmälle vaihtelevat.
  - Tietoon ei päästä käsiksi
    - Avainhenkilöt eivät ennätä osallistua OYVK-työvaiheeseen tai eivät pidä sitä tärkeänä.
    - Sidosryhmät eivät uskalla kertoa mielipidettään.
    - Pelätään tulevaan järjestelmään siirtymisen seurauksia.
    - Tarvittavaa dataa on hankala kerätä.

OYVK:n esteitä 2

- Kommunikaatiovaikeudet
  - Sidosryhmien taustat, terminologiat ja kulttuurit eroavat niin paljon, että yhteistä kieltä ei löydy.
- Hiljainen tieto (tacit information) ja piilossa olevat tavoitteet
  - Sidosryhmät eivät osaa kuvata ajatuksiaan sanoiksi.
  - Jätetään kertomatta tärkeää tietoa, jota luullaan itsestään selväksi.
  - Sidosryhmillä on epärealistisia odotuksia.
  - Sidosryhmät eivät tiedä, mitä oikeastaan haluavat.
- Yhteiskunnalliset tekijät
  - Poliittiset tekijät ja kilpailijat vaikuttavat päätöksiin.
  - Muutosvastarintaa esiintyy.
  - Aika-/kustannuspaineet rajoittavat työvaihetta.
- Epävakaat olosuhteet
  - Organisaatiomuutokset ja työntekijöiden vaihtuvuus vaikuttavat prosessiin.
  - Sidosryhmien tarpeet ja painotukset muuttuvat prosessin edetessä.

Kartutustekniikat

- Vaatimuksia voidaan kartuttaa kahdella tavalla:
  - *Epäsuorilla kartutustekniikoilla* (artefact-driven elicitation techniques) vaatimuksia etsitään tallennetuista tiedoista ilman suoraa kontaktia sidosryhmiin.
  - *Suorilla kartutustekniikoilla* (stakeholder-driven elicitation techniques) vaatimuksia etsitään vuorovaikutteisesti yhdessä sidosryhmien kanssa.
- OYVK-työvaihe kannattaa aloittaa epäsuorilla kartutustekniikoilla, koska niillä löydetyt tiedot auttavat suoria kartutustekniikoita.

3.2. Epäsuorat kartutustekniikat

- Yleisiä epäsuoria kartutustekniikoita:
  - Taustatutkimus
    - Kaivetaan vaatimuksia olemassa olevista dokumenteista.
  - Tiedon louhinta
    - Kaivetaan vaatimuksia olemassa olevien dokumenttien ulkopuolisesta datasta (esim. käyttötilastoista).
  - Kyselytutkimukset
    - Käytetään sidosryhmien täyttämää kyselylomaketta.
  - Kertomukset
    - Kuvataan järjestelmän toimintaa askel askeleelta.
  - Prototyypit
    - Tehdään kehitettävän järjestelmän malli sidosryhmille kokeiltavaksi.

Taustatutkimus

- Taustatutkimuksessa tutustutaan olemassa olevaan dokumentaatioon:
  - Organisaation dokumentaatio: organisaatiokaaviot, liiketoimintasuunnitelmat, toimintakäsikirjat, kokouspöytäkirjat, osavuosikatsaukset jne.
  - Ongelmakentän dokumentaatio: alan kirjat, tieteelliset julkaisut, säännöt, vastaavat järjestelmät jne.
  - Nykyjärjestelmän dokumentaatio: käyttöopas, tiedonkulku, työskentelytavat, liiketoimintasäännöt yms. Myös valitukset, viat, muutospyynnöt yms.
- Dokumentaatiota on yleensä pikemminkin liikaa kuin liian vähän. Tiedon louhinta auttaa etsimään tulevan järjestelmän kannalta keskeisiä tietoja.

Tiedon louhinta

- Taustatutkimusta voidaan täydentää keräämällä tietoja dokumentteihin tallentamattomasta datasta, kuten myynnistä, käyttötilastoista, tehokkuuskäyristä, käyttöasteesta jne.
- Tiedon louhinnalla saadaan parhaimmillaan dynaamisia tilastoja nykyisen järjestelmän käytöstä ja käyttäytymisestä.
- Tiedoilla voidaan kartuttaa myös ei-toiminnallisia vaatimuksia, kuten käytettävyyttä, suorituskykyä ja kustannuksia.
- Oikein tulkittuina tällaiset tiedot ovat erittäin tärkeitä tulevalle järjestelmälle.

Kyselytutkimukset

- Kyselytutkimukset ovat tehokas keino kerätä tietoja isolta joukolta sidosryhmiä. Niissä sidosryhmät vastaavat monivalinnoilla joukkoon lomakkeella olevia kysymyksiä.
  - Kyselytutkimuksella saadaan selvitettyä näppärästi ja melko edullisesti mielipiteitä.
  - Toisaalta kyselytutkimuksen antamat tulokset ovat yksipuolisia, koska lomake on melko rajoittunut tiedonvälityskanava.
- Kyselytutkimukset sopivat parhaiten haastatteluissa tarvittavien taustatietojen keruuseen.

Kertomukset

- *Kertomukset* (narratives) ovat erinomainen keino kuvata, miten nykyjärjestelmä toimii tai miten tulevan järjestelmän pitäisi toimia.
- Väljin muoto kertomuksista ovat *kuvakäsikirjoitukset* (storyboards). Niissä kertomus kerrotaan kuvasarjana.
  - Esimerkiksi nykyjärjestelmän käytöstä kertova sarjakuva on kuvakäsikirjoitus.
- Kuvakäsikirjoitukseen saa rakennetta lisäämällä siihen tarkentavaa tietoa:
  - Ketkä ovat mukana kuvakäsikirjoituksessa.
  - Mitä heille tapahtuu.
  - Miten mikin tapahtuu lyhyinä episodeina.
  - Miksi kaikki tapahtuu.
  - Entä jos jokin muu tapahtuma toteutuisi kesken episodin.
  - Mikä voi mennä pieleen.

Skenaariot

- *Skenaario* (scenario) on ehkä yleisimmin käytetty kertomustyyppi. Siinä kuvataan järjestelmälle ominainen komponenttien välinen toiminta.
- Skenaario sisältää rakenteisen kuva-käsikirjoituksen kohdat ”ketkä”, ”mitä” ja ”miten”.
- Skenaarioilla voidaan kuvata nykyistä järjestelmää tai tulevan järjestelmän vaatimuksia.
- Skenaariot sisältyvät joihinkin olioperustaisiin ohjelmistomallinnusmenetelmiin (esim. OMT).

Positiiviset ja negatiiviset skenaariot

- Skenaariot voivat olla positiivisia tai negatiivisia.
  - Positiivisessa skenaariossa kuvataan, mitä järjestelmässä tapahtuu.
  - Negatiivisessa skenaariossa kuvataan, mitä järjestelmässä ei saa tapahtua.
- Useimmat skenaariot ovat positiivisia. Sidosryhmät pitävät niistä, koska ne kuvaavat konkreettisia tapahtumia.
- Negatiiviset skenaariot ovat vastaesimerkkejä. Niitä tarvitaan, kun halutaan selvittää järjestelmän reunaehtoja.

Normaalit ja epänormaalit skenaariot

- Normaaleissa skenaarioissa kuvataan, miten järjestelmän onnistunut toiminta etenee.
- Epänormaaleissa skenaarioissa kuvataan poikkeustilanteita: mitä tapahtuu, kun asiat eivät mene odotetulla tavalla.
- Normaalit ja epänormaalit skenaariot ovat molemmat positiivisia skenaarioita. Järjestelmän on selvittävä myös poikkeustilanteista.
- Sidosryhmät keskittyvät useimmiten normaaleihin skenaarioihin, mutta epänormaalien skenaarioiden unohtaminen saattaa johtaa katastrofiin.

Skenaarioiden vahvuudet ja heikkoudet

- Skenaariot ovat selkeitä ja helposti ymmärrettäviä eri sidosryhmille. Niiden avulla on helppo kuvata sekä järjestelmän toivottua että sen ei-toivottua toimintaa. Niitä voi käyttää myös testitapausten suunnittelussa.
- Toisaalta skenaariot ovat vain esimerkkejä. Ne ovat osittaisia kuvauksia järjestelmän toiminnasta eivätkä siten anna täydellisiä vaatimuksia.
  - Tarkka järjestelmän kuvaus skenaarioina vaatisi kaikkien järjestelmän komponenttien kaikkien toimintatapausten kaikkien kombinaatioiden läpikäymisen.
  - Tuloksena olisi testauksesta tuttu kombinatorinen räjähdys.
- Skenaariot kuvaavat komponenttien vuorovaikutusta, mutta eivät sen taustalla olevia tavoitteita.

Prototyypit

- Joskus dokumentti ei ole paras tapa kuvata tulevan järjestelmän toimintaa sidosryhmille. Teksti ei välttämättä kerro, millä lailla tuleva järjestelmä vaikuttaa työskentelyyn.
- Tällaisissa tilanteissa järjestelmän tai ohjelmiston *prototyypit* (prototypes) auttavat. Prototyyppi on yksinkertaistettu järjestelmä tai ohjelmisto, joka näyttää oikealta, mutta toimii rajoitetusti.
- Parhaimmillaan prototyyppi helpottaa toiminnallisten vaatimusten löytämistä eri sidosryhmiltä.
- Toisaalta prototyypit eivät ole kovin käteviä ei-toiminnallisten vaatimusten kartutuksessa.

3.3. Suorat kartutustekniikat

- Suorat kartutustekniikat perustuvat sidosryhmien kanssa käytävään suoraan vuorovaikutukseen.
- Tärkeimmät suorat kartutustekniikat:
  - Haastattelut (interviews)
    - Kysytään sidosryhmiltä, mitä he haluavat.
  - Havainnointi (observation)
    - Käydään katsomassa nykyjärjestelmää paikan päällä.
  - Ryhmäistunnot (group sessions)
    - Pidetään OYVK:n sidosryhmien ryhmätapaamisia.

Haastattelut

- *Haastattelut* ovat ehkä tärkein vaatimusten kartutustekniikka. Niissä kysytään yhdeltä sidosryhmältä kerrallaan haluttuja tietoja.
- Haastattelut voivat olla rakenteisia, vapaita tai hybridejä.
  - *Rakenteisessa haastattelussa* (structured interview) käytetään ennalta laadittuja kysymyksiä.
  - *Vapaassa haastattelussa* (unstructured interview) keskustelu etenee vapaamuotoisesti ilman valmiita kysymyksiä.
  - *Hybridihaastattelu* alkaa rakenteisena mutta vaihtuu vähitellen vapaaksi.

Haastattelujen muistilista 1

- Haastattelut ovat tehokas kartutusmenetelmä, kun ne tehdään oikein. Seuraavassa oppikirjasta onnistuneen haastattelun muistilista:
  - Valitse sellaiset haastateltavat, joiden avulla saadaan kattava ja luotettava kuva ongelmakentästä.
  - Valmistaudu haastatteluun etukäteen. Keskity haastateltavan kannalta oleellisiin kysymyksiin. Pidä haastattelun ohjat omissa käsissäsi.
  - Pidä huoli siitä, että haastateltava tuntee olonsa mukavaksi haastattelun alusta alkaen. Pyydä erikseen lupaa nauhurin käyttöön. Esittele haastattelun aihe. Aloita helpoilla kysymyksillä.
  - Pidä haastattelun ilmapiiri tasa-arvoisena.

Haastattelujen muistilista 2

- Keskity haastateltavan työhön ja siihen liittyviin ongelmiin.
- Pysy asiassa. Kysy avoimet kysymykset haastattelun loppupuolella.
  - Avoin kysymys ei sisällä vastausehdotusta. Esimerkiksi ”Mitä kello on?” on avoin kysymys. ”Onko kello jo kaksi?” ja ”Valitse seuraavista kellonaika” ovat suljettuja kysymyksiä.
- Ole valmis vaihtamaan haastattelun suuntaa, jos saat odottamattoman ja mielenkiintoisen vastauksen.
- Kysy ”Miksi?”-kysymyksiä tehdyistä päätöksistä, ennalta vahvistetuista ratkaisuista ja muista kyseenalaisista ratkaisuista, MUTTA ole tarkkana, jotta et vaikuta hyökkäävältä.

Haastattelujen muistilista 3

- Vältä seuraavanlaisia kysymyksiä:
  - Mielipiteen värittämiä ja puolueellisia kysymyksiä.
  - Kysymyksiä, jotka sisältävät toivotun vastauksen (suljettujen kysymysten aliluokka).
  - Kysymyksiä, joihin haastateltava ei todennäköisesti osaa tai voi vastata.
  - ”Tyhmiä” kysymyksiä, eli sellaisia, joiden vastaus sinun pitäisi haastateltavan mielestä jo tietää. Nämä liittyvät yleensä ongelmakentän reunaehtoihin.
- Tee haastattelun puhtaaksikirjoitus heti haastattelun jälkeen, jolloin se on vielä kirkkaana mielessäsi.
- Sovi haastateltavan kanssa puhtaaksikirjoitetun haastattelun tarkistuksesta.

Haastattelujen vahvuudet ja heikkoudet

- Haastattelujen avulla saadaan parhaimmillaan kartutettua sellaista tietoa, jota ei saada millään muulla tavalla:
  - Vanhan järjestelmän todellinen toiminta käytännössä.
  - Parannusehdotuksia.
  - Henkilökohtaisia mielipiteitä.
  - Valituksia vanhasta järjestelmästä.
  - Käsityksiä, tuntemuksia.
  - Ennalta arvaamattomia näkökulmia.
- Toisaalta joskus haastateltavien vapaasti kertomia erityyppisiä asioita on vaikea yhdistää yhtenäiseksi kokonaisuudeksi.
- Haastattelun onnistuminen riippuu paljolti haastattelijasta ja käytetyistä kysymyksistä.

Havainnointi

- Havainnointitekniikat perustuvat oletukseen, jonka mukaan tehtävän havainnointi on tehokkaampi keino ymmärtää tehtävää kuin siitä annettu kuvaus.
- Havainnointi on erityisen tehokasta silloin, kun tehtävään liittyy useita toimijoita, jotka eivät ole täysin selvillä toistensa tekemisistä ja eri toimijoiden välisistä yhteyksistä.
- Havainnointi voi olla aktiivista tai passiivista.
  - Aktiivisessa havainnoinnissa havainnoija osallistuu tehtävään äärimmillään jopa työryhmän jäsenenä.
  - Passiivisessa havainnoinnissa havainnoija tarkkailee työryhmää kauempaa vaikuttamatta tehtävään suoraan.

Passiivisia havainnointitekniikoita

- *Ääneen ajattelussa* (protocol analysis) havainnoitava kertoo ääneen, mitä on tekemässä.
  - Ääneen ajattelua käytetään enemmän käytettävyystutkimuksessa, mutta sama tekniikka toimii myös vaatimusmäärittelyssä.
- *Etnografiassa* (etnographic studies) havainnoija seuraa havainnoitavia pidemmällä ajanjaksolla.
  - Toiminnan lisäksi havainnoidaan toimijoiden asenteita, reaktioita, eleitä, keskusteluja yms.

Havainnoinnin vahvuudet ja heikkoudet

- Havainnointi on erinomainen tekniikka, kun halutaan selvittää hiljaista tietoa. Havainnoimalla huomataan parhaimmillaan myös esimerkiksi sellaisia yksityiskohtia, joita haastateltava jättäisi mainitsematta itsestäänselvyyksinä.
- Toisaalta havainnointi on kallis tekniikka. Se vaatii paljon aikaa eikä sen avulla saada kerralla selville kuin pieni osa nykyjärjestelmän toiminnasta.
- Lisäksi havainnoinnin avulla ei juurikaan löydetä tulevan järjestelmän uusia ratkaisuja tai vaihtoehtoisia toimintatapoja.

Ryhmäistunnot

- Ryhmäistunnoissa vaatimuksia kartutetaan ryhmissä. Ajatuksena on, että ryhmä havaitsee, arvioi ja esittää ratkaisuja paremmin ja monipuolisemmin yhdessä kuin yksilöinä.
- Ryhmäistuntoja pidetään muutaman päivän mittaisissa työpajoissa (workshop), joiden aikana pidetään useita istuntoja.
- Työskentelyn apuna käytetään koko työtilaa, kuten tussitauluja, piirtoheittimiä, tietokoneita jne.

Rakenteiset ryhmäistunnot

- *Rakenteisessa ryhmäistunnossa* (structured group session) kullakin osallistujalla on ennalta määrätty rooli: puheenjohtaja, sihteeri, käyttäjä, ryhmäpäällikkö, ohjelmistokehittäjä jne.
- Istuntoon osallistuvat esittävät mielipiteensä roolinsa edustajina. Istunnoilla on tarkka ohjelma, jota seurataan huolellisesti.
- Rakenteiset ryhmäistunnot ovat parhaimmillaan, kun sidosryhmillä on ristiriitaisia odotuksia tulevasta järjestelmästä. Osa ristiriidoista saadaan yleensä ratkottua jo istunnoissa.

Rakenteettomat ryhmäistunnot

- *Rakenteettomassa ryhmäistunnossa* (unstructured group session) eli *aivoriihessä* (brainstorming) ei ole erityisiä rooleja. Työskentely tapahtuu vapaasti seuraavassa järjestyksessä:
  - Ensimmäisessä vaiheessa jokainen osallistuja generoi spontaanisti mahdollisimman monta ongelmaa lähestyvää ratkaisua. Myös hullut ideat kelpaavat tässä vaiheessa.
  - Toisessa vaiheessa osallistujat arvioivat yhdessä ehdotettuja ratkaisuja ja valitsevat niistä parhaat ennalta sovituilla kriteereillä (esim. tehokkuus, soveltuvuus, kustannukset).
- Aivoriihet ovat erinomainen kartutustekniikka, kunhan osallistujat hyväksyvät hullutkin ideat eivätkä arvioi idean hyvyydellä idean esittäjää.

Ryhmäistuntojen vahvuudet ja heikkoudet

- Ryhmäistunnoissa saadaan nykyjärjestelmän kuvauksen ja siihen liittyvien ongelmien lisäksi myös vaatimuksia tulevalle järjestelmälle.
- Lisäksi niissä syntyy monipuolisempi kuva järjestelmästä kuin haastatteluissa ja havainnoinnissa.
- Toisaalta ryhmäistunnot vievät runsaasti aikaa, joten niihin saattaa olla vaikea saada avainhenkilöitä.
- Lisäksi vahvat yksilöt saattavat dominoida liikaa.
- Rakenteettomissa ryhmäistunnoissa saattaa kulua liikaa aikaa tuottamattomaan pähkäilyyn.

Vielä OYVK:n tekniikoista

- Edellä kuvatut tekniikat eivät sulje toisiaan pois. Mikään tekniikka ei yksin riitä, vaan eri tekniikoita tulee käyttää yhdessä.
- Massamyyntiin tarkoitettujen järjestelmien kehitysprosessissa ei välttämättä ole mukana sidosryhmien edustajia. Tällöin suoria tekniikoita ei voi käyttää sellaisenaan (esimerkiksi ei löydy haastateltavia).
  - Tällaisissa tilanteissa epäsuoria tekniikoita voidaan täydentää esimerkiksi markkinatutkimuksilla ja liiketoimintasuunnitelmilla.

4. Vaatimusten arviointi

- OYVK:ssa kerätyt vaatimukset pitää arvioida:
  - Vaatimusten väliset ristiriidat pitää ratkoa etsimällä sopiva kompromissi.
  - Vaatimuksiin liittyvät riskit on tunnistettava ja niiden vaikutusta pienennettävä.
  - Vaihtoehtoiset ratkaisut on arvioitava ja arvion perusteella pantava paremmuusjärjestykseen.
  - Vaatimukset täytyy priorisoida.
- Vaatimusten arviointi on sidoksissa vaatimusten kartutukseen. Sitä mukaa kun vaatimuksia löydetään, niitä myös arvioidaan.

4.1. Epäyhdenmukaisuuksien hallinta

- Vaatimukset eivät yleensä ole keskenään yhdenmukaisia (consistent), koska niitä ovat esittäneet kovin eriluonteiset tahot.
- Epäyhdenmukaisuuksia on eri tyyppisiä:
  - Termistöyhteentörmäys (terminology clash)
  - Nimeämisyhteentörmäys (designation clash)
  - Rakenneyhteentörmäys (structure clash)
  - Vahva ristiriita (strong conflict)
  - Heikko ristiriita (weak conflict)

Epäyhdenmukaisuustyypit 1

- Termistöyhteentörmäys
  - Samasta käsitteestä käytetään eri vaatimuksissa eri nimeä.
  - Esim. ”Moottorit käynnistyvät kiihdytykseen” vs. ”Raketit sammuvat kiihdytyksen jälkeen”.
- Nimeämisyhteentörmäys
  - Sama nimi tarkoittaa eri asioita eri vaatimuksissa.
  - Esim. ”Viestinvälitin toimittaa luotaimelta lähtevät radioviestit” vs. ”Viestinvälitin huolehtii ohjelmiston komponenttien välisestä sanomanvälityksestä.”
- Rakenneyhteentörmäys
  - Samalle käsitteelle on määritelty eri rakenne eri vaatimuksissa.
  - Esim. ”Jarrutushetki tarkoittaa ajanjaksoa, jolloin raketit ovat käytössä” vs. ”Jarrutushetki tarkoittaa ajanjaksoa rakettien käynnistymisestä laskeutumiseen”.

Epäyhdenmukaisuustyypit 2

- Vahva ristiriita
  - Kaksi tai useampi väite ei voi missään tilanteessa olla yhdessä totta.
  - Esim. ”Ohjausjärjestelmä ohjaa kaikkia järjestelmiä” ja ”Yksikään järjestelmä ei ohjaa itseään”.
    - Mikä ohjaa ohjausjärjestelmää?
- Heikko ristiriita
  - Kaksi tai useampi väite ei voi joidenkin ehtojen vallitessa olla yhdessä totta.
  - Esim. ”Rakettimoottorit sammutetaan viimeistään viiden minuutin käytön jälkeen” vs. ”Rakettimoottorit sammutetaan, kun niitä ei enää tarvita.”
    - Ehdot ovat totta, kunhan moottoreita ei koskaan tarvita viittä minuuttia enempää.

Epäyhdenmukaisuuksien käsittely 1

- Termistö-, nimeämis- ja rakenne-yhteentörmäykset johtuvat määritelmien moniselitteisyydestä.
- Näistä ongelmista päästään eroon hyvin tehdyllä *sanastolla* (glossary of terms).
- Sanasto tarjoaa tarkan kaikkien sidosryhmien hyväksymän määritelmän jokaiselle käytetylle termille sekä mahdollisesti luettelon termin synonyymeista.
- Sanastoa rakennetaan vaatimusten kartutuksen aikana.

Epäyhdenmukaisuuksien käsittely 2

- Vahvojen ja heikkojen ristiriitojen ratkominen on edellisiä tyyppejä vaikeampaa.
- Ratkominen alkaa tunnistamalla ristiriitojen syyt:
  - Eri sidosryhmien tavoitteet ovat erilaiset, jolloin ne voivat olla ristiriidassa keskenään. Ristiriitaiset tavoitteet johtavat ristiriitaisiin vaatimuksiin.
    - Ristiriitaisten tavoitteiden ratkominen ratkoo samalla vastaavien vaatimusten väliset ristiriidat.
  - Eri vaatimustyyppien kesken on luonnostaan ristiriitaisuutta.
    - Ei-toiminnalliset vaatimukset ovat usein keskenään ristiriitaisia (esim. turvallisuus vs. käytettävyys).
    - Toiminnalliset vaatimukset saattavat olla ristiriidassa ei-toiminnallisten vaatimusten kanssa.

Ristiriitojen ratkominen

- Ristiriitojen ratkominen vaatii sidosryhmien välistä neuvottelua.
- Neuvottelussa huomioidaan:
  - Kunkin sidosryhmän asettamat tavoitteet tulevalle järjestelmälle.
  - Tavoitteiden väliset erot.
  - Tavoitteisiin liittyvät riskit ja epävarmuudet.
  - Tavoitteiden sidosryhmäkohtaiset prioriteetit.
- Neuvottelun tuloksena on saatava sellainen tavoitelista, johon kaikki sidosryhmät sitoutuvat.

Ristiriitojen hallintaprosessi

- Vaatimusten väliset ristiriidat ratkotaan näin:
  - Tunnistetaan päällekkäiset väitteet.
  - Tunnistetaan päällekkäisten väitteiden väliset ristiriidat.
  - Tuotetaan joukko kunkin ristiriidan ratkaisuja.
  - Evaluoidaan ratkaisut ja valitaan niistä paras.

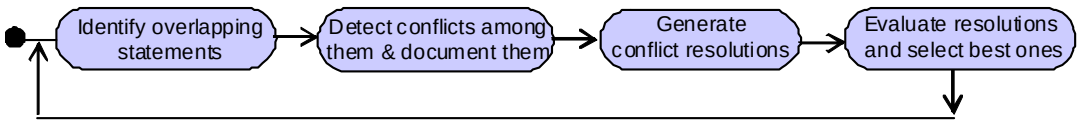


Figure 3.1 – Conflict management

Kuvalla (c) Axel van Lamsweerde, 2009

Tunnistetaan päällekkäiset väitteet

- Kaksi väitettä (vaatimusta, määritelmää, yms.) ovat päällekkäiset, jos ne kohdistuvat kokonaan tai osittain samaan ilmiöön (ongelmakentän alueeseen, laitteeseen, ohjelmistoon, yms.)
- Tunnistus voi olla syntaktinen, jolloin väitteiden avainsanoja verrataan sanaston avulla.
- Tunnistus voi olla semanttinen, jolloin väitteiden sisältämiä käsitteitä ja toimintoja verrataan toisiinsa.

Tunnistetaan päällekkäisten väitteiden väliset ristiriidat

- Epämuodollinen tunnistus
  - Ristiriitojen tunnistus perustuu asiantuntija-arvioihin ongelmakentästä.
- Heuristinen tunnistus
  - Ristiriitojen tunnistus perustuu johonkin yleiseen heuristiikkaan. Esimerkiksi ei-toiminnallisten vaatimusten ristiriidat voivat perustua ei-toiminnallisten vaatimusten luokitteluun (kalvo 29).
- Formaali tunnistus
  - Väitteet kuvataan matemaattiseen logiikkaan perustuvalla formaalilla kielellä. Ristiriidat tunnistetaan epäyhdenmukaisuustarkistuksilla (inconsistency checking), todistuksilla (theorem proving), raja-arvoanalyysilla (derivation of boundary conditions) tai vastaavalla menetelmällä.

Tuotetaan joukko kunkin ristiriidan ratkaisuja

- Ratkaisujen etsintään voidaan käyttää ainakin seuraavia keinoja:
  - Etsitään vaihtoehtoisia ratkaisuja kartutustekniikoilla (eli kysytään sidosryhmiltä).
  - Muokataan vaatimukset sellaisiksi, että ristiriidan aiheuttava raja-arvo ei toteudu.
  - Sallitaan ristiriitainen raja-arvotila, mutta minimoidaan sen kestoaika.
  - Heikennetään ristiriitaisia ehtoja siten, että ristiriita häviää.
  - Pudotetaan matalan prioriteetin vaatimuksia.
  - Erikoistetaan ristiriidan lähdettä tai kohdetta sellaiseksi erikoistapaukseksi, että ristiriita häviää.

Evaluoidaan ratkaisut ja valitaan niistä paras

- ”Parhaan” vaihtoehdon valinta ei välttämättä ole yksiselitteistä. Eri vaihtoehdot tarjoavat yleensä vaihtelevasti hyviä ja huonoja puolia.
- Päätöstä helpottamaan voidaan valita sopiva arviointikriteeristö, joka kertoo (yleensä subjektiivisen) valinnan hyvyysarvon.
- Arviointi voi esimerkiksi perustua ratkaisujen vaikutukseen kriittisimpiin ei-toiminnallisiin vaatimuksiin, niiden toteutuksen riskeihin tai sidosryhmien tarpeiden priorisointeihin.
- Työvaihe vaatii yleensä neuvottelua sidosryhmien kanssa.

4.2. Riskianalyysi

- Varsinkin vaatimusmäärittelyprosessin alkuvaiheessa sekä sidosryhmien edustajat että vaatimusmäärittelijät ovat herkkiä ylilyönneille. Esim. seuraavat ovat implisiittisiä oletuksia:
  - Tulevalle järjestelmälle asetetut odotukset ovat realistisia.
  - Ongelmakenttä toimii odotetulla tavalla.
  - Tuleva järjestelmä toimii odotetulla tavalla.
  - Kehitystyö sujuu suunnitellulla tavalla jne.
- Käytännössä edelliset kohdat eivät toteudu. Siirtyminen nykyjärjestelmästä tulevaan järjestelmään on täynnä riskejä.
- Tulevan järjestelmän vaatimukset jäävät vajaiksi, jos riskejä ei tunnisteta tai niihin ei varauduta.

Riski

- *Riski* (risk) on epävarmuustekijä, jonka toteutuminen voi aiheuttaa ongelmia tavoitteen saavuttamiselle. Riskillä on negatiivinen vaikutus tavoitteeseen.
- Riskillä on (*toteutumis)todennäköisyys* (likelihood of occurence) ja yksi tai useampia ei-toivottuja *seurauksia* (consequences).
- Jokaisella riskin toteutumisen seurauksella on oma todennäköisyys ja *vakavuus* (severity).
  - Riskin todennäköisyys on eri kuin seurauksen todennäköisyys. Riskillä ”Junan ovi aukeaa junan ollessa liikkeellä” on eri todennäköisyys kuin riskin seurauksella ”Matkustaja putoaa liikkuvasta junasta”.

Riskienhallinta

- *Riskienhallinta* (risk management) on oleellinen osa vaatimusten arviointia. Löydetyistä vaatimuksista, oletuksista ja reunaehdoista ei aina huomioida niihin liittyviä riskejä.
- Kaikkia riskejä ei tarvitse huomioida. Huomiointiin vaikuttavat riskin todennäköisyys, seurausten todennäköisyys ja seurausten vakavuus.
- Jos vaatimukseen liittyy huomioitavia riskejä, niin vaatimuksen tueksi tarvitaan *vastatoimia* (countermeasures). Kukin vastatoimi on uusi vaatimus, joka määrittelee, mitä järjestelmän pitää tehdä riskin toteutuessa (jotta riskin seuraukset eivät toteutuisi).

Riskienhallinnan keinoja

- Kaikki seuraavat keinot johtavat uusien vaatimusten esittelyyn:
  - Pienennetään riskin todennäköisyyttä.
  - Estetään riskin toteutuminen.
  - Pienennetään riskin seurauksen todennäköisyyttä.
  - Estetään tietyn seurauksen toteutuminen.
  - Pienennetään riskin seurauksen vaikutusta eli sen vakavuutta.
- Käytännössä uudet vaatimukset täydentävät riskialttiita vaatimuksia. Joskus uudet vaatimukset voivat myös korvata riskialttiin vaatimuksen.

4.3. Vaihtoehtojen arviointi ja päätöksenteko

- Vaatimusmäärittelyssä on tärkeää etsiä ja arvioida vaihtoehtoisia ratkaisuja:
  - Tulevan järjestelmän tavoite voidaan usein ratkaista erilaisilla yhdistelmillä alitavoitteita, ominaisuuksia ja oletuksia.
  - Tulevan järjestelmän vastuut voidaan jakaa usealla tavalla sen eri komponenteille.
  - Ristiriidat voidaan ratkaista usealla eri tavalla.
  - Riskiin voidaan varautua monella eri vastatoimella.
- Jokaista valintaa varten tarvitaan kriteerit, joilla arvioidaan vaihtoehtojen paremmuus.

Laadullinen arviointi

- Vaihtoehtojen paremmuutta voidaan arvioida laadullisilla (qualitative) arviointikriteereillä.
  - Laadullisissa arviointikriteereissä vaihtoehdot asetetaan paremmuusjärjestykseen ongelmakentän asiantuntijuuteen perustuvien päätösten avulla.
  - Vaihtoehtojen vaikutusta ei-toiminnallisiin vaatimuksiin voidaan esimerkiksi arvioida sopivalla (ei-numeerisella) asteikolla:
    - Esim. + = hyvä, ++ = erittäin hyvä, - = huono, -- = erittäin huono, +-=neutraali.
  - Vastaavalla tavalla voidaan arvioida vaihtoehtojen vaikutusta tulevan järjestelmän riskeihin.

Numeerinen arviointi

- Vaihtoehtoja voidaan arvioida myös numeerisesti (quantitatively). Tällöin käytetään yleensä *painotettuja matriiseja* (weighted matrices).
- Jokaiselle kriteerille annetaan painoarvo, joka esittää sen merkityksellisyyttä suhteessa muihin kriteereihin.
- Jokainen matriisisolu (kriteeri *k*, vaihtoehto *v*) kuvaa, millä todennäköisyydellä vaihtoehto *v* täyttää kriteerin *k*.
- Vaihtoehdoille lasketaan matriisista painotetut keskiarvot kriteerien painoarvojen ja vaihtoehtojen täyttymisen perusteella.

Numeerisen arvioinnin esimerkki 1

- Olkoon meillä kaksi vaihtoehtoista palvelua:
  1. Tarvittavat tiedot saadaan sähköpostitse.
  2. Tarvittavat tiedot saadaan esityslistalta.
- Olkoon meillä kolme kriteeriä (ei-toiminnallista vaatimusta):
  - Nopea vasteaika
  - Luotettava vastaus
  - Minimaalinen vaivannäkö
- Vaihtoehtojen matriisissa on 2x3=6 solua:

	Vasteaika	Luotettavuus	Vaivattomuus
Sähköposti	0,50	0,90	0,50
Esityslista	0,90	0,30	1,00

Numeerisen arvioinnin esimerkki 2

- Palvelujen toteutumisten todennäköisyydet arvioidaan kartutustietojen avulla. Esimerkiksi skenaariot auttavat tässä.
- Eri kriteereille määriteltävät painoarvot riippuvat sidosryhmien toiveista. On saatu seuraavat painoarvot:
  - Vasteaika: 0,30
  - Luotettavuus: 0,60
  - Vaivattomuus: 0,10
- Nyt vaihtoehdoille saadaan numeeriset arviot:
  - Sähköposti: 0,30*0,50+0,60*0,90+0,10*0,50 = 0,74
  - Esityslista: 0,30*0,90+0,60*0,30+0,10*1,00 = 0,55
- Annetuilla parametreilla sähköposti on siis parempi vaihtoehto.

4.4. Vaatimusten priorisointi

- Vaatimuksia täytyy arvioinnin lisäksi priorisoida. Prioriteetteja tarvitaan seuraavista syistä:
  - Järjestelmän kehitystyöhön varattu budjetti ja aikataulu eivät ehkä salli kaikkien löydettyjen vaatimusten toteuttamista.
  - *Lisäävässä kehitystyössä* (incremental development) järjestelmää toteutetaan osissa useana julkaisuna. Julkaisuun valittavat vaatimukset riippuvat niiden prioriteeteista.
  - Prioriteetteja tarvitaan ratkottaessa vaatimusten välisiä ristiriitoja.
- Prioriteettien perusteella päätetään, mitkä vaatimukset ovat pakollisia, mitkä toivottavia ja mitkä siirrettävissä myöhempään ajankohtaan.

Arvo-kustannus-vertailu

- Hyvä tapa priorisoida vaatimuksia on *arvo-kustannus-vertailu* (value-cost comparison):
  - Jokaisesta vaatimuksesta arvioidaan sen *arvo* (hyödyllisyys) järjestelmälle.
  - Jokaisesta vaatimuksesta arvioidaan sen kehitystyön *kustannus.*
  - Saadut arvot näytetään arvo-kustannus-kaaviossa.
- Arvojen ja kustannusten arviointiin käytetään sopivaa päätöksenteon heuristiikkaa.

Arvo-kustannus-kaavio: esimerkki

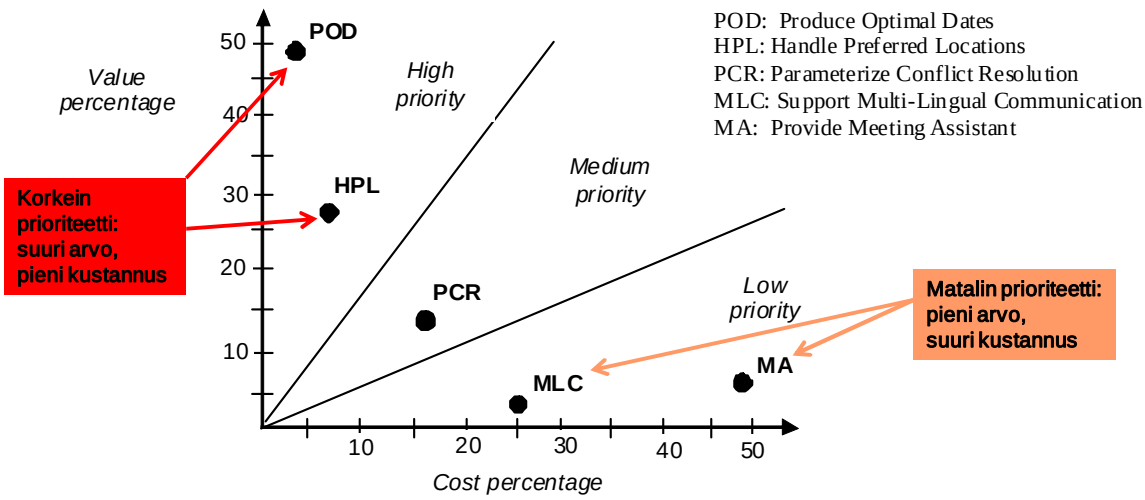


Figure 3.6 – Value-cost requirements prioritization for the meeting scheduler: outcome of the AHP process

Kuvalla (c) Axel van Lamsweerde, 2009

5. Vaatimusten spesifiointi ja dokumentointi

- Edellisissä työvaiheissa saadut tulokset on määriteltävä yksityiskohtaisesti sellaisella tarkkuudella, että tuleva järjestelmä voidaan suunnitella, toteuttaa ja testata niiden avulla.
- Tätä työvaihetta kutsutaan vaatimusten spesifioinniksi (ja dokumentoinniksi). Tuloksena saadaan uusi versio vaatimusdokumentista.
  - Koska vaatimusmäärittely on iteratiivinen prosessi, myös vaatimusdokumentti täydentyy asteittain.

5.1. Vaatimusdokumentin sisältö

- Kaikki tulevan järjestelmän sovitut ominaisuudet on määriteltävä yksikäsitteisesti:
  - Tavoitteet, käsitteet, ongelmakentän ominaisuudet, järjestelmä- ja ohjelmistovaatimukset, oletukset, vastuut.
  - Valittujen ratkaisujen perustelut.
- Sovitut ominaisuudet on kirjattava vaatimusdokumenttiin sellaisessa muodossa, että dokumentaatiota tarvitsevat sidosryhmät ymmärtävät sitä.

IEEE:n dokumenttipohja 1

- Vaatimusdokumentin sisällysluettelo IEEE:n standardista Std-830:

1. Johdanto
   - 1.1. Vaatimusdokumentin tarkoitus
   - 1.2. Tuotteen sovellusalue (ongelmakenttä, tulevan järjestelmän tarkoitus)
   - 1.3. Määritelmät ja lyhenteet
   - 1.4. Viitteet (kartutuksesta saatu dokumentaatio)
   - 1.5. Yleiskatsaus (dokumentin muiden lukujen sisältö)
2. Yleiskuvaus
   - 2.1. Tuotenäkökulma (ohjelmiston ja toimintaympäristön rajapinta, käyttäjärajapinnat, laitteet, muut ohjelmistot)
   - 2.2. Tuotteen toiminnot (tulevan järjestelmän toiminnallisuus)
   - 2.3. Käyttäjäpiirteet (käyttäjien oletettu profiili)
   - 2.4. Yleiset rajoitteet (kehitystyöhön kohdistuvat rajoitteet)
   - 2.5. Oletukset ja riippuvuudet (reunaehdot ja oletukset)
   - 2.6. Suhteutetut vaatimukset (valinnaiset ja lykätyt vaatimukset)

IEEE:n dokumenttipohja 2

3. Vaatimukset
   - 3.1. Toiminnalliset vaatimukset
   - 3.2. Ulkoisten rajapintojen vaatimukset (yhteentoimivuus)
   - 3.3. Suorituskykyvaatimukset (aika-, tila- ja kuormitusvaatimukset)
   - 3.4. Suunnittelun rajoitteet (arkkitehtuurivaatimukset)
   - 3.5. Laatuattribuutit (laatutekijät, kuten käytettävyys)
   - 3.6. Muut vaatimukset (turvallisuus, luotettavuus, ylläpidettävyys ym.)

Liitteet

Hakemisto

- Hyvä dokumenttipohja vähentää moniselitteisyyttä ja hälyä. Selkeä pohja parantaa dokumentin rakennetta. Vaatimusten luokittelu parantaa mitattavuutta.

Vaatimusdokumentin karikot 1

- Hyvän vaatimusdokumentin kirjoittaminen ei ole helppoa. Oppikirjassa on listattu seuraavat vaatimusdokumentin potentiaaliset heikkoudet:
  - Laiminlyönti (omission): ongelmakenttää ei ole katettu kotonaan, tulevan järjestelmän vaatimuksia puuttuu.
  - Ristiriita (contradiction): järjestelmän ominaisuuksia on kuvattu yhteensopimattomasti.
  - Riittämättömyys (inadequacy): ominaisuutta ei ole kuvattu riittävällä tarkkuudella.

Vaatimusdokumentin karikot 2

- Moniselitteisyys (ambiguity): ominaisuus voidaan tulkita keskenään ristiriitaisilla tavoilla.
- Mittaamattomuus (unmeasurability): ominaisuus on kuvattu tavalla, jota ei voi verrata sen vaihtoehtoihin tai jota ei voi verifioida valmiista tuotteesta.
- Häly (noise): vaatimusdokumentin osa ei kuvaa ongelmakenttää.
- Ylimäärittely (overspesification): vaatimusdokumentin osa ei liity ongelmakenttään vaan sen ratkaisuun (järjestelmän toteutukseen).
- Jälkilisäys (remorse): ominaisuus määritellään liian myöhään tai sattumalta.

Vaatimusdokumentin karikot 3

- Kelpaamattomuus (unfeasibility): ominaisuus ei ole toteutettavissa annetulla budjetilla, aikataululla tai toimintaympäristöllä.
- Vaikeatajuisuus (unintelligibility): ominaisuus on kuvattu sitä tarvitsevien sidosryhmien kannalta väärällä tavalla tai tarkkuudella.
- Kehno rakenne (poor structuring): ominaisuudet on ryhmitelty dokumentissa huonosti.
- Eteenpäinviittaus (forward reference): ominaisuus määritellään vasta sen käytön jälkeen.

Vaatimusdokumentin karikot 4

- Huono muokattavuus (poor modifiability): tietyn vaatimusdokumentin osan muokkaus vaikuttaa liian laajasti dokumentin muuhun sisältöön.
- Läpinäkyvyyden puute (opacity): ominaisuuden tavoite, esittäjä tai sidokset muihin ominaisuuksiin eivät selviä dokumentaatiosta.

- Erityisen vaarallisia heikkouksia ovat laiminlyönnit, ristiriidat, riittämättömyydet, moniselitteisyydet ja mittaamattomuudet, jotka vaikuttavat suoraan tulevan järjestelmän laatuun. Loput heikkoudet vaikuttavat laatuun välillisesti lisäämällä epäonnistumisen riskiä ja aiheuttamalla turhaa työtä.

5.2. Spesifikaatiokielet

- Kaikki vaatimusdokumentin kuvaukset on kirjoitettava sopivalla *spesifikaatiokielellä* (spesification language).
- Spesifikaatiokielet voivat olla luonnollisen kielen osajoukkoja (epäformaaleja), predikaattilogiikkaan perustuvia täsmällisesti määriteltyjä kieliä (formaaleja) tai osittain määriteltyjä (semi-formaaleja).
  - Mitä formaalimpi spesifikaatiokieli on, sitä paremmin sillä vältetään moniselitteisyyttä.
  - Mitä epäformaalimpi spesifikaatiokieli on, sitä joustavampi ja helpommin hallittava se on.
  - Spesifikaatiokielen on oltava kaikkien sidosryhmien ymmärtämä, bisnesmiehistä aina koodaajiin saakka.

5.2.1. Luonnollinen kieli

- Luonnollinen (puhuttu) kieli on ilmeisin vaihtoehto spesifikaatiokieleksi.
- Luonnollinen kieli on äärimmäisen joustava. Sillä voidaan kuvata mikä tahansa tulevan järjestelmän ominaisuus. Kaikki sidosryhmät ymmärtävät luonnollista kieltä ilman koulutusta.
- Joustavuus on myös luonnollisen kielen huono puoli. Sillä kuvatut ominaisuudet kärsivät helposti ainakin moniselitteisyydestä, hälystä, eteenpäinviittauksista, jälkilisäyksistä, mittaamattomuudesta ja läpinäkyvyyden puutteesta.

Luonnollisen kielen vaatimusesimerkki 1

- ”Junan jarrut aktivoidaan hätäjarrua vedettäessä tai junan liikkuessa yli 100 km:n tuntinopeudella ja sen ollessa alle kilometrin päässä edellä kulkevasta junasta.”
  - Kun hätäjarrua vedetään, aktivoidaanko junan jarrut aina?
  - Kun hätäjarrua vedetään, aktivoidaanko junan jarrut vain, jos juna on alle kilometrin päässä edellä kulkevasta junasta?
- Kumpikin tulkinta on mahdollinen!

Luonnollisen kielen vaatimusesimerkki 2

- Ongelma johtuu luonnollisen kielen sanojen ”ja” ja ”tai” moniselitteisyydestä:
  - Säännön ”*Sitä* tai *Tätä* ja *Tuota*” voidaan tulkita tarkoittavan joko samaa kuin (1) ”(*Sitä* tai *Tätä*) ja *Tuota*” tai samaa kuin (2) ”*Sitä* tai (*Tätä* ja *Tuota*)”.
  - Jos *Sitä* on totta, vaaditaan vaihtoehdon (1) täyttymiseen, että myös *Tuota* on totta. *Tällä* ei enää ole väliä.
  - Jos *Sitä* on totta, pätee vaihtoehto (2) sen sijaan välittömästi riippumatta siitä, ovatko *Tätä* ja *Tuota* totta vai ei.
- Logiikassa on sovittu, että ”ja” ”sitoo” voimakkaammin kuin ”tai”, jolloin tulkinta (2) on oikea.
  - Edellisessä esimerkissä: Kun hätäjarrua vedetään, junan jarrut aktivoidaan aina.

5.2.2. Rakenteinen kieli

- Luonnollisen kielen puutteita voidaan korjata lisäämällä kieleen sovittuja sääntöjä. Näin saadaan rakenteinen kuvauskieli.
- Rakenteinen kuvauskieli sisältää suuren osan luonnollisen kielen ilmaisuvoimasta, mutta sääntöjen johdosta sen kuvaustapa on puhdasta luonnollista kieltä kurinalaisempi.
- Rakenteisuus voi kohdistua paikallisesti yksittäisiin lauseisiin, kuten vaatimuksiin, tai globaalisti koko vaatimusdokumentin sisältöön.

Tyylisääntöjen käyttö

- Yksinkertaisin keino saada luonnolliseen kieleen rakennetta on käyttää tyylisääntöjä (stylistic rules). Ne määrittelevät mallin, jonka mukaan vaatimusdokumentti kirjoitetaan.
- Tyylisääntöjen käyttö on samanlaista kuin yleisemmin teknisessä kirjoittamisessa. Tarkoituksena on yhtenäistää dokumentaatio sellaiseksi, että siihen tutustuminen vie mahdollisimman vähän aikaa.

Vaatimusmäärittelyn tyylisääntöjä

- Kirjoita oikealle kohderyhmälle.
- Aloita kertomalla, mistä on kysymys.
- Alkuun motivointi, loppuun yhteenveto.
- Määrittele kaikki termit ennen niiden käyttöä.
- Validoi tekstiäsi kysymyksillä tyyliin “Onko teksti luettavaa? Onko tämä oleellista? Onko asia selvitetty riittävän hyvin?”
- Kirjoita lyhyin lausein. Samassa lauseessa ei koskaan pidä kuvata enempää kuin yhtä vaatimusta, oletusta tai ongelmakentän ominaisuutta.
- Käytä verbiä “pitää” kuvaamaan pakollisia ominaisuuksia ja “pitäisi” kuvaamaan toivottuja ominaisuuksia.
- Vältä kapulakieltä ja turhia lyhenteitä.
- Käytä esimerkkejä selventämään vaikeita kohtia.
- Kokoa asioita kuviksi ja taulukoiksi.

5.2.3. Päätöstaulut

- Monimutkaiset ehdot on selkeää kuvata *päätöstauluina* (decision table).
- Päätöstaulu on kaksiulotteinen taulukko. Sen vaakarivit ovat atomisia ehtoja, jotka voivat olla tosia tai epätosia. Sen pystyrivit ovat vaihtoehtoisia atomisten ehtojen arvojen kombinaatioita.
- Kutakin arvokombinaatiota kohden on taulukon lopussa vaakariveillä yksi tai useampi seuraus (output condition). Seuraus toteutuu, kun todelliset atomisten ehtojen tulokset vastaavat pystyrivin arvokombinaatiota.

Päätöstauluesimerkki

- Junan jarrutuksen vaatimusta kuvaava päätöstaulu:

Päätöstaulujen vahvuuksia

- Päätöstaulut vähentävät moniselitteisyyttä. Oikein tehdyssä päätöstaulussa ovat mukana kaikki mahdolliset ehtojen kombinaatiot ja niiden seuraukset. Väärinkäsityksen mahdollisuutta ei ole.
- Päätöstaulujen *täydellisyys* (completeness) on helppo tarkistaa: jos päätöstaulussa on *n* atomista ehtoa, on siinä oltava 2^n kombinaatiota.
- Päätöstaulut ovat myös testauksen apuväline. Vaatimusmäärittelyssä tehtyjen päätöstaulujen avulla voidaan laatia (hyväksymistestauksen) testitapauksia ja mitata testauksen kattavuutta.

5.2.4. Rakenteiset lausemallit

- *Rakenteinen lausemalli* (structured statement template) tarkoittaa, että keskeiset asiat (vaatimukset, rajoitteet, reunaehdot ja ongelmakentän piirteet) kuvataan yhtenäisellä rakenteisella formaatilla.
- Rakenteinen lausemalli sisältää joukon kenttiä, jotka tarkentavat kuvattavaa asiaa.
- Mahdolliset kentät vaihtelevat. Usein mukana on ainakin seuraavia kenttiä:
  - tunniste, luokka, määritelmä, lisäehto, lähde, tavoite, vaikutus muihin ominaisuuksiin, prioriteetti ja stabiilius.

Rakenteisten lausemallien ominaisuuksia

- Rakenteiset lausemallit selkeyttävät ja yhtenäistävät vaatimusdokumenttia.
- Toisaalta rakenteisia malleja käyttämällä saadaan laveaa dokumentaatiota, koska jokaisesta vaatimuksesta täytetään useita kenttiä.
- Esimerkiksi lisäehto-kenttä voi tarkentaa vaatimusta liittämällä siihen numeerisia ehtoja:
  - Vaatimus: Valittujen kokousaikojen pitää sopia kutsutuille osallistujille.
  - Lisäehto: Ainakin 90 %:lla osallistujista pitää olla kalenterissa vapaata ainakin 80 %:ssa kokousajoista.
- Vaatimuksella voi olla useita tarkentavia lisäehtoja.

5.2.5. Semi-formaalit spesifikaatiokielet

- Luonnolliseen kieleen perustuvat spesifikaatiokielet voidaan korvata tai niitä voidaan täydentää *semi-formaaleilla* spesifikaatiokielillä.
- Semi-formaali tarkoittaa, että kielen käsittelemät alkiot ja niiden väliset yhteydet on määritelty formaalisti mutta niiden ominaisuudet on kuvattu luonnollisella kielellä.
- Yleensä semi-formaaleja kieliä havainnollistetaan sidosryhmille graafisina kaavioina, mutta niitä analysoidaan automaattisten työkalujen ymmärtämässä formaalisti määritellyssä muodossa.
- Esimerkiksi kaikki UML:n kuvauskielet ovat semi-formaaleja spesifikaatiokieliä.

Semi-formaalien spesifikaatiokielten esimerkkejä

- Semi-formaalit spesifikaatiokielet sopivat kuvaamaan nykyistä tai tulevaa järjestelmää jostakin tietystä näkökulmasta. Seuraavassa joitain tunnettuja vaihtoehtoja:
  - Ongelmakentän kuvaus: sisältö- ja ongelmakaaviot
  - Käsitteiden mallinnus: ER-kaaviot, luokkakaaviot
  - Tiedon ja toimintojen mallinnus: SADT-kaaviot
  - Tiedon kulku: tietovuokaaviot
  - Järjestelmän palvelut: käyttötapauskaaviot
  - Toimintaskenaariot: viestiyhteyskaaviot, sekvenssikaaviot
  - Komponenttien käyttäytyminen: tila(siirtymä)kaaviot

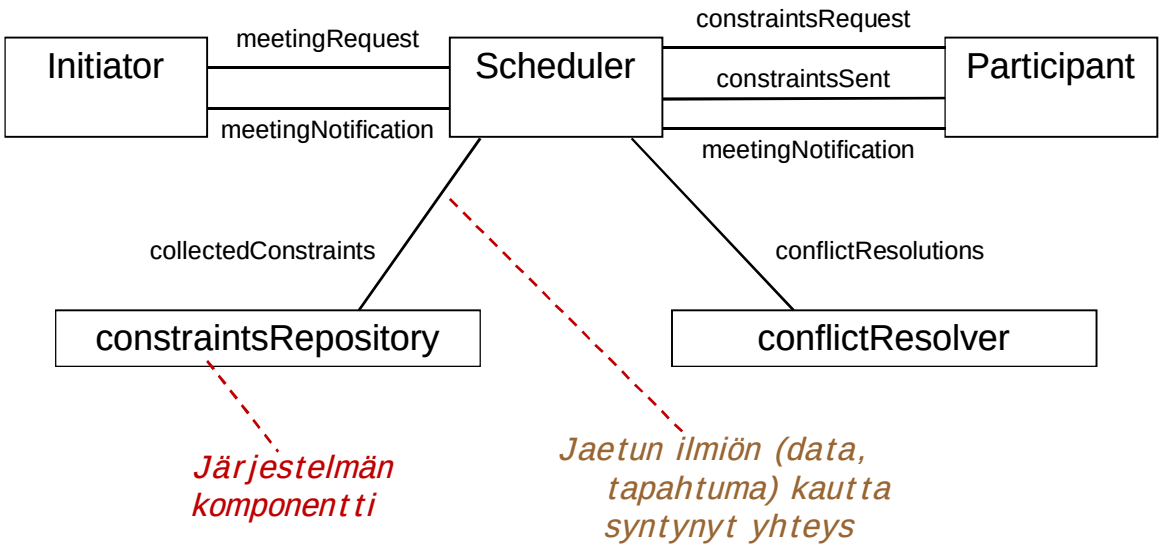
5.2.6. Sisältö- ja ongelmakaaviot

- *Sisältökaavio* (context diagram) on verkko, jossa solmut ovat järjestelmän komponentteja ja särmät komponenttien välisiä yhteyksiä ongelmakentässä.
- *Ongelmakaavio* (problem diagram) on sisältökaavion tarkennus. Siinä määritellään, mitkä komponentit hallitsevat komponenttien välisiä yhteyksiä, mitkä komponentit muodostavat tulevan laitteen ja mihin komponentteihin vaatimukset vaikuttavat.

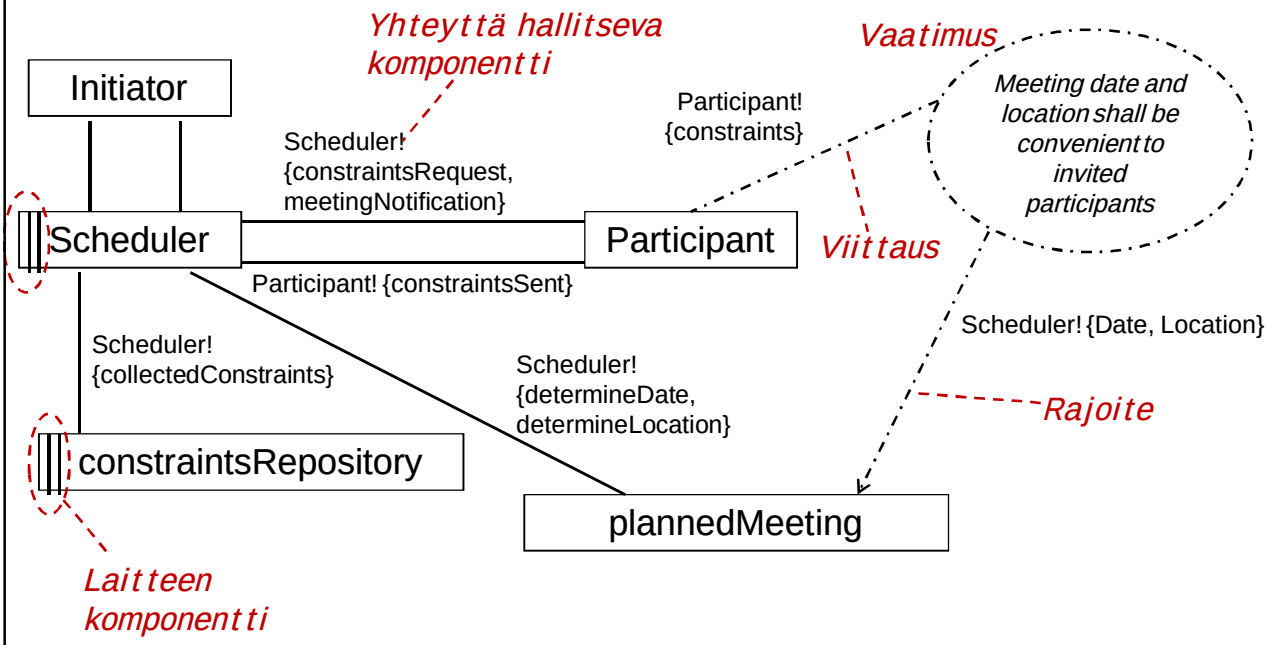
Esimerkki: Kokousten varausjärjestelmä

Kokouksista sopimiseen halutaan automaattinen järjestelmä (meeting scheduler). Järjestelmän avulla kokoonkutsuja (initiator) ehdottaa osallistujille (participant) kokouksen aikaväliä. Osallistujat lähettävät takarajaan (deadline) mennessä järjestelmälle ko. aikaväliä koskevat rajoituksensa (constraints): päivät ja kellonajat, jotka eivät sovi, sekä sopivat päivät ja kellonajat mahdollisesti priorisoituina. Tärkeiden (important) osallistujien rajoituksilla on suurempi painoarvo ja he voivat esittää rajoituksia myös kokouksen pitopaikalle. Järjestelmä ilmoittaa kokoonkutsujalle ja osallistujille kokouksen ajan (date, schedule) ja paikan (location), mikäli löytää sopivat. Vaihtoehdoista se valitsee parhaan mahdollisen. Ellei järjestelmä löydä sopivaa aikaa ja paikkaa, aloitetaan uusi kierros.

Kokousjärjestelmän sisältökaavio



Kokousjärjestelmän ongelmakaavio



5.2.7. ER-kaaviot

- *ER-kaavioilla* (Entity-Relationship diagrams) kuvataan ongelmakentän käsitteitä tai tulevan järjestelmän komponentteja sekä niiden välisiä yhteyksiä.
- Alkuperäiset ER-kaaviot esiteltiin jo 1976, mutta sittemmin niistä on tehty paljon erilaisia variaatioita.
  - Tutuin variaatio on (UML:n) *luokkakaavio* (class diagram). Myös se kuvaa elementtien (oliot) yhteyksiä (perintä, riippuvuus, erikoistuminen, ym.) ja rakenteita (attribuutit).

Kokousjärjestelmän ER-kaavio

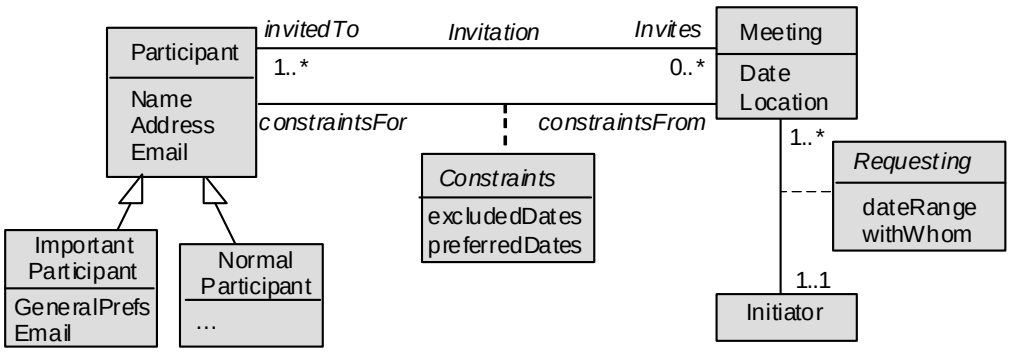


Figure 4.5 – Portion of an entity-relationship diagram for the meeting scheduler

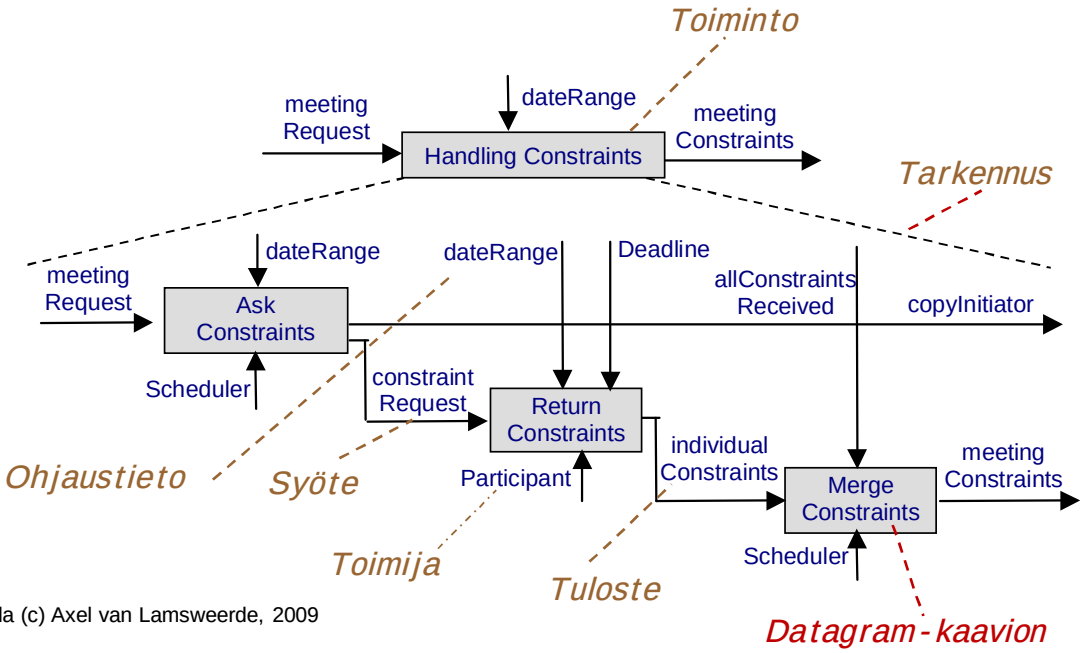
Kuvalla (c) Axel van Lamsweerde, 2009

- Kaaviossa on komponentteja, perintää, rakenteita, yhteyksiä ja yhteyksien astelukuja.

5.2.8. SADT-kaaviot

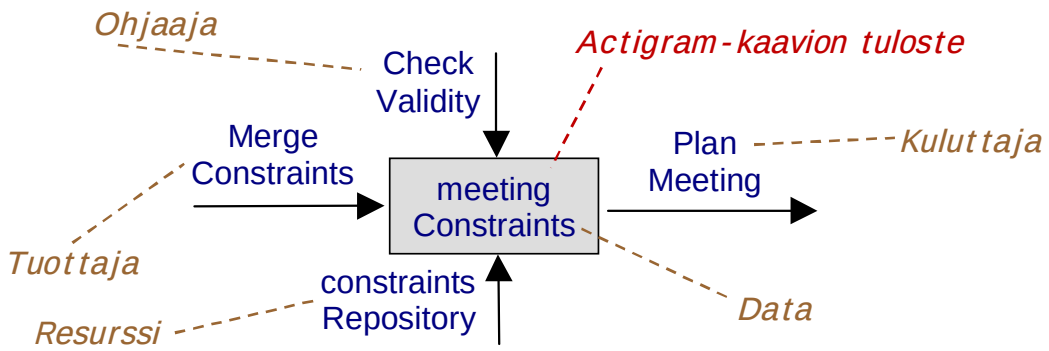
- *SADT-kaaviolla* (Structured Analysis and Design Technique) kuvataan tietojen käsittelyä eri toiminnoilla. Kaavioita on kahta toisiinsa sidoksissa olevaa tyyppiä:
  - *Actigram-kaaviot* (actigrams) liittävät *toiminnot* (activities) toisiinsa tietoriippuvuuksien (syötteet ja tulosteet) avulla.
  - *Datagram-kaaviot* (datagrams) liittävät *tiedot* toisiinsa ohjausriippuvuuksien (tiedon tuottajat ja kuluttajat) avulla.
- Kaavioilla on yhteys:
  - Actigram-kaavioiden syötteiden ja tulosteiden täytyy löytyä datagram-kaavioista.
  - Datagram-kaavioiden tietojen tuottajien ja kuluttajien täytyy löytyä actigram-kaavioista.

Kokousjärjestelmän actigram-kaavio



Kuvalla (c) Axel van Lamsweerde, 2009

Kokousjärjestelmän datagram-kaavio



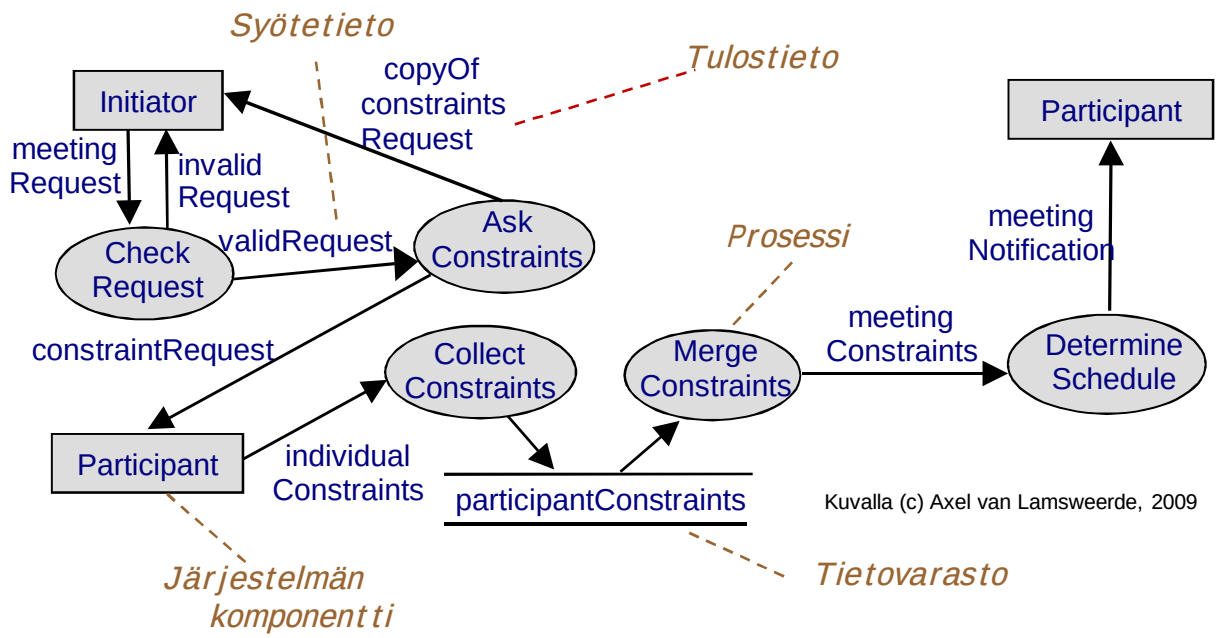
Kuvalla (c) Axel van Lamsweerde, 2009

- SADT-kaaviotekniikalla tietojen sisältö, käsittely ja ohjaus saadaan kuvattua näppärästi yhdessä.
- Kaaviot voidaan verifioida automaattisilla työkaluilla.

5.2.9. Tietovuokaaviot

- *Tietovuokaaviot* (dataflow diagrams) ovat yksinkertaistettu muoto actigram-kaavioista. Niillä kuvataan tietojen kulkua ja käsittelyä järjestelmässä.
- Järjestelmään tulee tietoa, jota käsitellään sen prosesseissa sopivien sääntöjen avulla. Tieto siirtyy eteenpäin järjestelmässä prosessilta toiselle.
  - Käsittelysäännöt kuvataan prosesseittain kaavion yhteydessä sopivalla spesifiointikielellä tai tarkentamalla prosessia aliprosesseiksi.
- Prosessien ja niiden välillä kulkevien tietojen lisäksi tietovuokaaviossa on tietovarastoja, jotka toimivat tiedon pysyvinä talletuspaikkoina.

Kokousjärjestelmän tietovuokaavio



5.2.10. Käyttötapauskaaviot

- *Käyttötapauskaaviot* (use case diagrams) yksinkertaistavat SADT-kaavioita ja tietovuokaavioita ryhmittelemällä toisiinsa liittyvät tiedot ja tietojen käsittelyoperaatiot *käyttötapauksiksi* (use cases).
  - Käyttötapaus kuvaa järjestelmän tarjoamaa palvelua ja siihen vuorovaikutuksessa olevia järjestelmän komponentteja.
- Käyttötapauskaaviot tarjoavat selkeän tekniikan kuvata tulevan järjestelmän toiminnallisuutta. Toisaalta kaaviot ovat kovin yleisellä tasolla, joten yksityiskohdat täytyy kuvata muilla spesifiointikielillä.

Kokousjärjestelmän käyttötapauskaavio

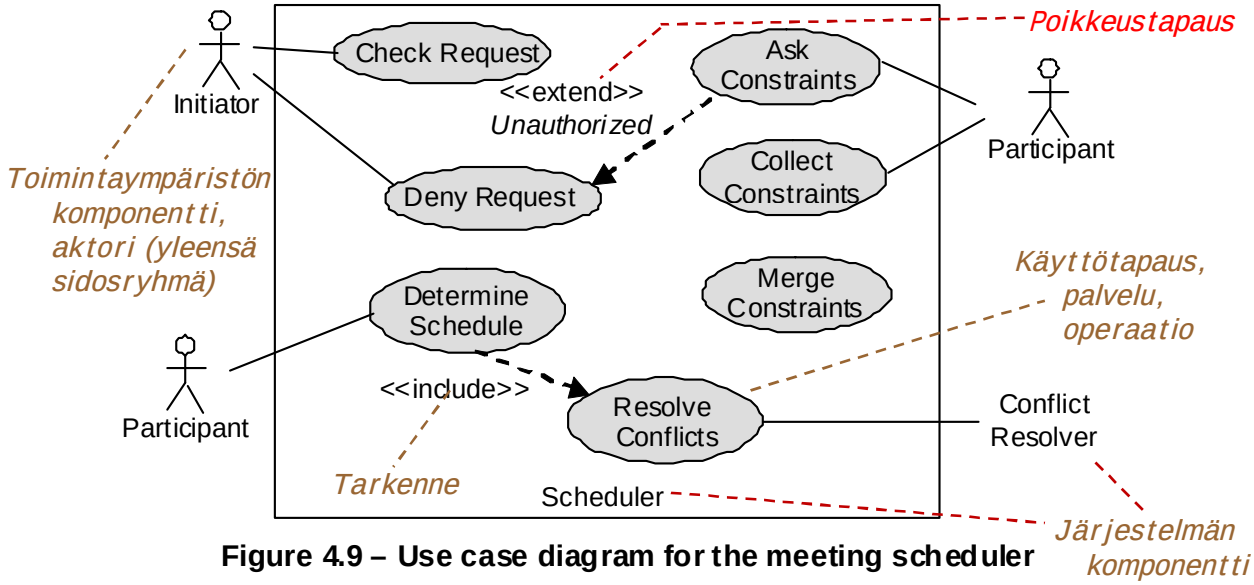


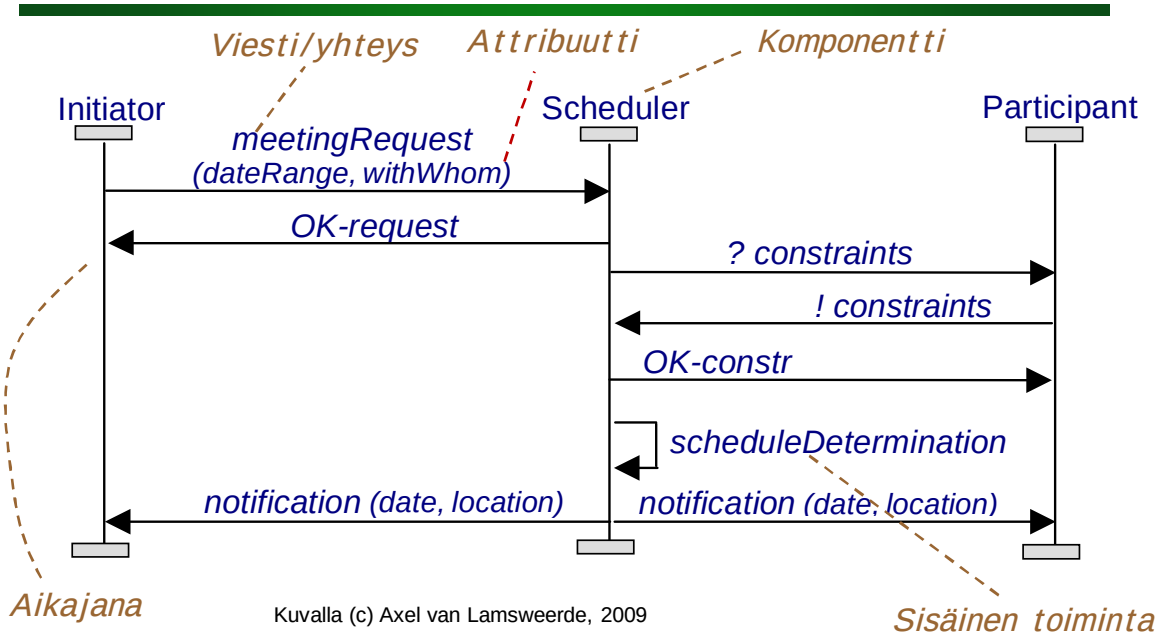
Figure 4.9 – Use case diagram for the meeting scheduler

Kuvalla (c) Axel van Lamsweerde, 2009

5.2.11. Viestiyhteyskaaviot

- *Viestiyhteyskaavio* (event trace diagram) kuvaa skenaariona järjestelmän komponenttien välisen vuorovaikutuksen (viestinvälityksen, kommunikaation).
- Viestiyhteyskaavio kuvaa skenaarioon osallistuvien komponenttien tehtävät toisiinsa liittyvinä aikajanoina. Aika juoksee kaaviossa ylhäältä alas.
- Kaaviossa kuvataan komponentit, komponenttien toisilleen lähettämät pyynnöt ja vastaukset sekä toisinaan arviot pyyntöjen käsittelyyn kuluvasta ajasta.
- Kukin viestiyhteyskaavio kuvaa vain yhtä toimintaskenaariota, joten niitä tarvitaan useita järjestelmän koko toiminnan kuvaamiseksi.

Kokousjärjestelmän viestiyhteyskaavio

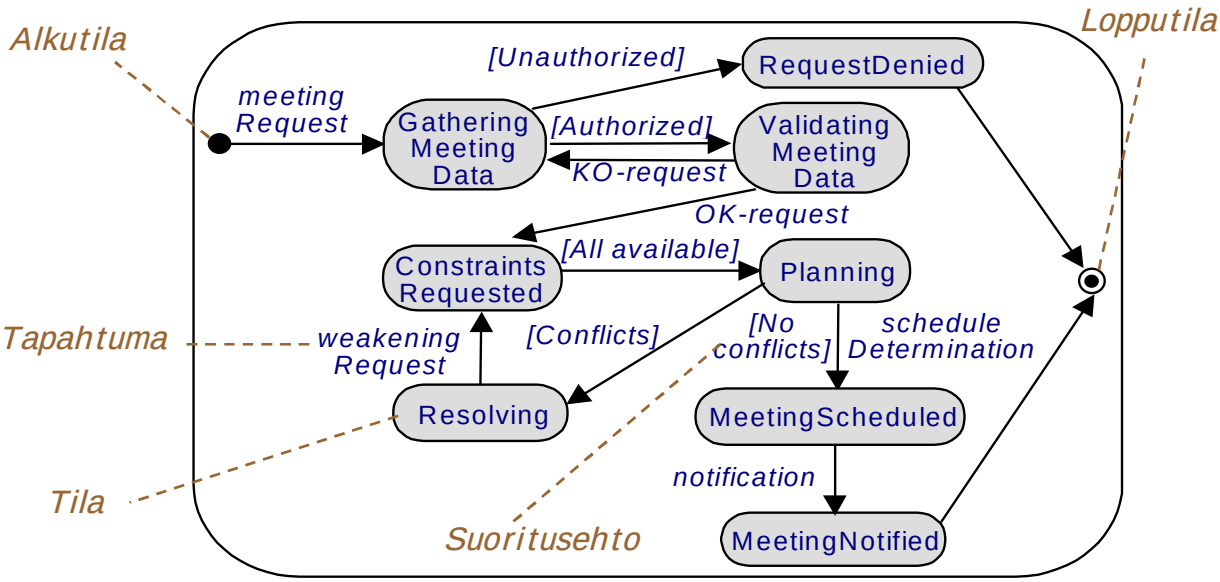


Kuvalla (c) Axel van Lamsweerde, 2009

5.2.12. Tilasiirtymäkaaviot

- *Tilasiirtymäkaavio* (state machine diagram) on kompakti tapa kuvata komponentin koko elinkaari. Vaatimuksena on, että komponentilla on äärellinen joukko *tiloja* (states) ja *tapahtumia* (events).
  - Tila kuvaa tiettyä komponentin tilannetta. Tapahtumat siirtävät komponentin tilasta toiseen.
- Komponentti siirtyy tilasta toiseen, kun se vastaanottaa tapahtuman ja kun määrätty *suoritusehto* (guard) toteutuu.
  - Suoritusehto on tilasiirtymäkaavion välttämätön reunaehto. Siirtymä tilasta toiseen tapahtuu vain, jos tilojen välinen suoritusehto on voimassa.
- Tavallisten tilojen lisäksi tilasiirtymäkaaviossa on alkutila (initial state), joka kuvaa komponentin elinkaaren alkua, ja lopputiloja (final state), jotka kuvaavat komponentin elinkaaren loppua.

Kokousjärjestelmän Scheduler-komponentin tilasiirtymäkaavio



Kuvalla (c) Axel van Lamsweerde, 2009

5.2.13.1. Semi-formaalien kielten etuja

- Kielet ovat hyviä yleiskuvauksen työkaluja.
- Tärkeiden ominaisuuksien korostaminen on helppoa.
- Graafinen kuvauskieli tarjoaa yhteisen semantiikan, jota eri sidosryhmien on (toivottavasti) helppo ymmärtää.
- Työkalutukea on olemassa kaavioiden laatimiseen ja analysointiin samoin kuin niitä vastaavien koodirunkojen automaattiseen generointiin.
- Kaaviot on helppo arkistoida ja liittää vaatimusmäärittelyn dokumentaatioon.
- Kaavioissa on tilaa vapaalle dokumentaatiolle, joten tekniikat eivät rajoita ilmaisuvoimaa.

5.2.13.2. Semi-formaalien kielten haittoja

- Kaavioiden tulkinnoissa voi olla moniselitteisyyttä. Sama asia voidaan kuvata monin eri tavoin. Sama korkean abstraktiotason kaavio voidaan tulkita monin eri tavoin.
- Formalismi ei ulotu kuvausten kommentinomaisiin selityksiin, joten niitä ei voi verifioida automaattisin työkaluin.
- Analyysi rajoittuu kaavioiden syntaksiin ja termistön yhteneväisyyteen.
- Kielillä saadaan kuvattua toiminnallisia ja rakenteisia vaatimuksia, mutta ei kovinkaan hyvin tulevan järjestelmän tavoitteita, ei-toiminnallisia vaatimuksia tai ongelmakentän oletuksia.

- Semi-formaalit kielet riittävät normaalien järjestelmien tekoon, mutta kriittisiin järjestelmiin tarvitaan vahvempia työkaluja.

5.2.14. Formaali määrittely

- *Formaali määrittely* (formal specification) korjaa semi-formaalien kielten ongelmat lisäämällä tarkan syntaksin ja semantiikan myös kuvausten selityksiin.
- Formalismia käyttämällä vaatimusmäärittelyssä löydetyt ominaisuudet määritellään täsmällisesti ja yksiselitteisesti.
- Formaalilla kielellä kuvatut ominaisuudet voidaan verifioida automaattisesti.
- Toisaalta formaali määrittely vaatii syvällistä erikoisosaamista, jota ei ole kaikilla sidosryhmillä eikä edes keskiverto-ohjelmistokehittäjillä.

Formaalit spesifiointikielet 1

- Formaalit spesifiointikielet perustuvat matemaattiseen formalismiin ja jakautuvat seuraavasti:
  - *Propositiologiikkaan* (propositional logic) perustuvat kielet kuvaavat boolean-arvoisia sääntöjä, joita yhdistetään loogisilla operaattoreilla ($\wedge$, $\vee$, $\neg$, $\Rightarrow$, $\Leftrightarrow$).
  - *Predikaattilogiikkaan* (predicate logic) perustuvat kielet käyttävät propositiologiikan lisäksi muuttujia, vakioita, lausekkeita ja kvanttoreita ($\forall$, $\exists$).
  - *Korkeamman kertaluvun funktioihin* (higher-order functions) perustuvat kielet laajentavat propositio- ja predikaattilogiikkaa siten, että funktioiden argumentit voivat olla muuttujien ja vakioiden lisäksi funktioita.
  - *Temporaalilogiikkaan* (temporal logic) perustuvat kielet lisäävät formalismiin ajan hallinnan. Kielet perustuvat *historian* (history) käsitteeseen. Historia kuvaa elementtien toimintaa ajassa (”joskus tulevaisuudessa”, ”aina tulevaisuudessa”, ”joskus menneisyydessä”, ”aina siitä lähtien kun” jne.).

Formaalit spesifiointikielet 2

- *Tiloihin* perustuvissa (state-based) kielissä lauseet kuvaavat mallinnettavan järjestelmän tiloja tietyllä hetkellä. Tunnetuin formaali spesifiointikieli Z on tilapohjainen kieli.
- *Tapahtumiin* perustuvissa (event-based) kielissä lauseet kuvaavat mallinnettavan järjestelmän siirtymistä tilasta toiseen. Siirtymät kuvataan siirtymäfunktioina, joiden lähtö- ja maalitilat, laukaisevat tapahtumat ja suoritusehdot määritellään formaalisti.
- *Algebraan* perustuvissa kielissä (algebraic specification) järjestelmän operaatiot kuvataan yhtälöinä ja järjestelmässä käsiteltävä tieto abstrakteina tietotyyppeinä.

Kokousjärjestelmä temporaalilogiikalla

$\forall$ p: Participant, m: Meeting
Intended(p,m) $\wedge$ Notified(p,m) $\wedge$ Convenient(m,p) $\Rightarrow$
$\Diamond$ Participates(p,m)

Joskus tulevaisuudessa

$\forall$ p: Participant, m: Meeting
Notified(p,m) $\Rightarrow$ $\Box$ Notified(p,m)

Aina tulevaisuudessa

6. Vaatimusten laadunvarmistus

- Vaatimusmäärittelyn spiraalin päättää *vaatimusten laadunvarmistus* (requirements quality assurance).
- Työvaiheessa varmennetaan, että löydetyt, arvioidut ja spesifioidut vaatimukset
  - täyttävät sidosryhmien tarpeet,
  - käsittelevät koko ongelmakentän ja
  - eivät ole hallitsemattomalla tavalla ristiriidassa keskenään tai ongelmakentän reunaehtojen kanssa.

Laadunvarmistuksen tekniikat

- Vaatimusten laadunvarmistukseen on neljä päätekniikkaa:
  - *Tarkastukset* (inspections) ja *katselmoinnit* (reviews) ovat ryhmässä tehtäviä laadunvarmistusmenetelmiä. Niissä joukko sidosryhmien edustajia etsii vaatimuksista puutteita ja virheitä.
  - *Kyselyt* (queries) ovat tekniikka, jota voidaan käyttää, kun vaatimusmäärittelyn termistö ja vaatimukset on talletettu tietokantaan. Tällöin voidaan sopivilla tietokantakyselyillä löytää vaatimusmäärittelystä ristiriitaisuuksia ja puutteita.
  - *Animaatiot* (animations) ovat prototyyppien yleistys. Niissä vaatimukset kuvataan sellaisella tavalla, että niitä vastaava järjestelmä on testattavissa sopivalla simulaattorilla.
  - *Formaali verifiointi* (formal verification) kattaa tekniikat, joita voidaan käyttää formaalien spesifikaatioiden laadunvarmistukseen ja oikeaksi todistamiseen.

Tarkastukset ja katselmoinnit

- Tarkastuksissa ja katselmoinneissa joukko asiantuntijoita kokoontuu yhteen etsimään dokumentista puutteita ja virheitä. Tekniikat eivät ole sidoksissa vaatimusmäärittelyyn.
  - Tarkastus on muodollinen tapahtuma, jolla on puheenjohtaja ja esityslista. Kokouksessa etsitään dokumentista virheitä, mutta ei yritetä korjata niitä.
  - Katselmoinnit ovat epämuodollisempia kuin tarkastukset. Niissä useampi henkilö etsii yhdessä dokumentista virheitä. Puheenjohtajaa ei yleensä ole ja keskustelu on vapaampaa kuin tarkastuksissa.
- Sekä tarkastuksissa että katselmoinneissa kannattaa käyttää *tarkistuslistoja* (checklists). Tarkistuslista kertoo lyhyesti, mitkä ovat katsottavan tyyppisen dokumentin yleisimmät virheet.

Kyselyt

- Semi-formaaleja kieliä käytettäessä spesifikaatio kannattaa tallentaa tietokantaan. Oikein toteutettuna tietokannan avulla on mahdollista rekonstruoida koko spesifikaatio.
- Tietokantakyselyillä on tällaisessa tilanteessa mahdollista varmistaa, että spesifikaatio on yhtenäinen ja noudattaa ko. spesifiointikielen semantiikkaa.
- Tietokantakaavion luonti, tietojen tallennus ja verifioivat kysymykset on mahdollista tehdä automaattisesti semi-formaaleilla kielillä tehdyistä kaavioista.

Animaatiot

- Semi-formaaleista spesifikaatioista ei näe suoraan, miten hyvin niillä kuvatut vaatimukset täyttävät sidosryhmien tarpeet. Animaatiot auttavat tähän.
- Animaatiossa spesifikaation mukaista toimintaa simuloidaan sidosryhmille visuaalisesti. Animaatiossa nähdään, miten spesifikaatiot käyttäytyisivät todellisessa tulevassa järjestelmässä.
- Animaatiot soveltuvat parhaiten toiminnallisiin spesifikaatioihin (tietovuo-, viestiyhteys- ja tilasiirtymäkaaviot).
- Animaatiot ovat sukua prototyypeille, mutta eivät ole sama asia.
  - Prototyypeissä kirjoitetaan oikeaa koodia, jota käytetään mahdollisimman oikeassa käyttöympäristössä.
  - Animaatioissa spesifikaatiosta tehdään automaattisesti tai puoliautomaattisesti kuvaus, joka suoritetaan simulaattorissa.

Formaali verifiointi

- Jos spesifikaatio on tehty formaalilla kielellä, sitä voidaan tarkistaa tai todistaa formaalisti:
  - Kuvauksille voidaan tehdä syntaksitarkistukset ohjelmointikielten tapaan.
  - Kuvausten yhdenmukaisuus ja täydellisyys on mahdollista tarkistaa.
  - Kuvausten eli ”mallin” yhteensopivuus ongelmakentän tiettyjen ominaisuuksien kanssa on (rajoitetusti) mahdollista todistaa.
- Jos spesifikaatio tehdään formaalisti, sen laadunvarmistus yleensä onnistuu melko hyvin.

7. Vaatimusten evoluutio

- Olipa vaatimusmäärittelyn eri vaiheet läpikäyty miten huolellisesti tahansa, vaatimusdokumenttia täytyy yleensä päivittää:
  - Havaitaan virheitä.
  - Sidosryhmien tavoitteet ja toiveet muuttuvat.
  - Sidosryhmien priorisoinnit muuttuvat.
  - Toimintaympäristö (liiketoiminta, muut järjestelmät) muuttuu.
  - Ongelmakentän lait tai oletukset muuttuvat.
- Muutoksia voi ilmaantua vaatimusmäärittelyprosessin aikana, ohjelmistokehityksen myöhemmissä vaiheissa ja järjestelmän käyttöönoton jälkeen.
- Muutokset aiheuttavat vaatimuksille *evoluutiota*: syntymistä, eriytymistä, valintaa, kuolemista ja muutoshistoriaa.

Vaatimusten versiointi 1

- Jokainen vaatimusmäärittelyn spiraalikierros tuottaa yhden *version* vaatimusdokumentista.
- Jokainen versio on joko *revisio* tai *variantti*:
  - Kyseessä on revisio, jos muutokset (korjaukset, parannukset) kohdistuvat yhden tietyn tuotteen vaatimuksiin.
  - Kyseessä on variantti, jos muutosten tuloksena saadaan vaatimusmäärittely samaan perusversioon perustuvalle mutta siitä selvästi eroavalle uudelle tuotteelle, jolla on omia ominaisuuksia ja joka on yleensä suunnattu jollekin erityiselle käyttäjäryhmälle.
- Jos tuotteella on useita variantteja, on kyseessä *tuoteperhe* (product family, product line).
- Kustakin revisiosta voi syntyä useita variantteja ja kustakin variantista useita revisioita.

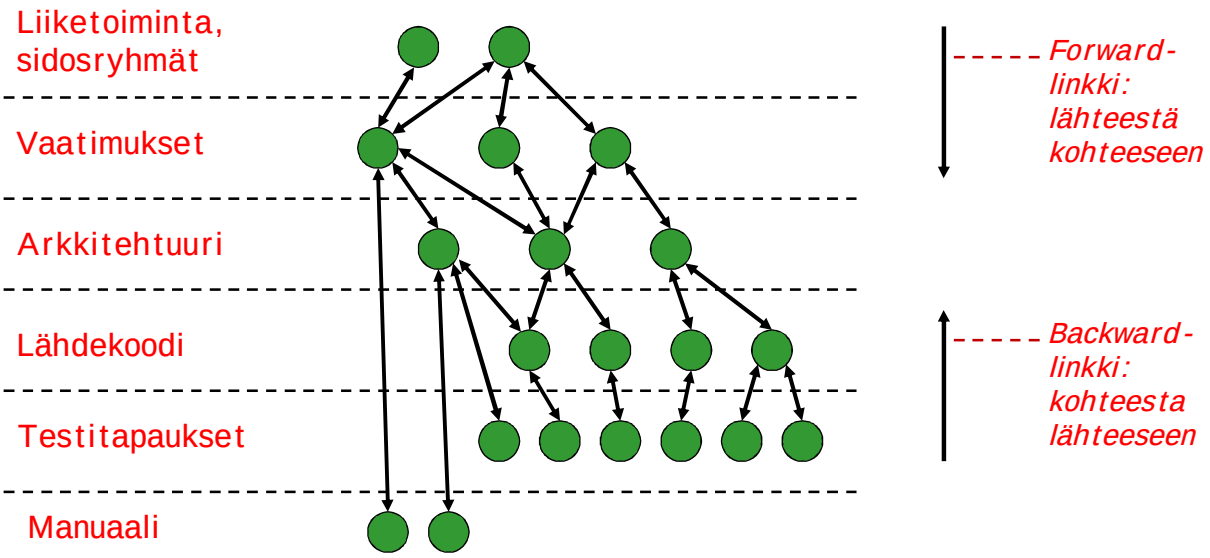
Vaatimusten versiointi 2

- Esimerkkejä kokousjärjestelmän revisioista:
  1. Korjataan vaatimusmäärittelyn sisäisiä ristiriitoja.
  2. Otetaan osallistujien pyynnöstä mukaan uusi vaatimus, jolla kokouksen aika ja paikka saadaan tekstiviestinä kännykkään.
- Esimerkkejä kokousjärjestelmän varianteista:
  1. Kokouksesta ilmoitetaan sekä aika että paikka.
  2. Kokouksesta ilmoitetaan ainoastaan ajankohta (kokoukset pidetään aina samassa paikassa).
  3. Tärkeät osallistujat saavat kokouksen ajan ja paikan suoraan sähköiseen kalenteriinsa.

Vaatimusten jäljitettävyys 1

- Jotta vaatimusten evoluutiota voidaan hallita, on vaatimusten oltava *jäljitettäviä* (traceable).
  - Jäljitettävyys (traceability): *mistä* vaatimus on peräisin, *miksi* se on tuotettu ja *minne* se johtaa.
  - Jäljitettävyys voidaan toteuttaa linkkiketjuna, josta löydetään (1) ne sidosryhmät ja liiketoiminnan tavoitteet, joista vaatimus on peräisin, (2) ne ohjelmiston osat, joihin tietty vaatimus vaikuttaa ja (3) ne vaatimukset, joiden takia tietty ohjelmiston osa on tuotettu.
  - Lisäksi linkitetään vaatimusmäärittelyjen revisiot ja variantit.
  - Jäljitettävyyslinkitystä voidaan hyödyntää mm. testauksessa (”mitä vaatimuksia vasten on tämä komponentti testattava”) ja ylläpidossa (”mihin muihin ohjelmiston osiin tämän vaatimuksen muuttaminen vaikuttaa”).

Vaatimusten jäljitettävyys 2



8. Yhteenveto

- Vaatimusmäärittely on ongelmakentän tutkimista, rajausta ja tarkkaa määrittelyä. Siihen kuuluu nykyjärjestelmän ja tulevan järjestelmän ominaisuuksien tulkinta ongelmakentässä.
- Tulevasta järjestelmästä täytyy vastata seuraaviin kysymyksiin:
  - MIKSI tulevaa järjestelmää tarvitaan?
  - MITEN tulevan järjestelmän tulee toimia?
  - MITEN HYVIN tulevan järjestelmän tulee toimia?
  - KUKA on vastuussa tulevan järjestelmän osista eli miten vastuut jaetaan?

Yhteenveto 2

- Vaatimusmäärittelyn kysymyksiin voidaan vastata eri tavoin, mikä johtaa erilaisiin ratkaisuvaihtoehtoihin. Vaihtoehdoilla on omat vahvuutensa ja heikkoutensa.
- Vaatimusmäärittelyä tehdään eri abstraktiotasoilla. Korkeimmalla tasolla ovat liiketoiminnan strategiset tavoitteet. Teknisten yksityiskohtien vaatimukset ovat alimmalla tasolla.
- Vaatimusmäärittelyssä on näytettävä, että matalamman tason vaatimukset toteuttavat korkeamman tason vaatimukset.

Yhteenveto 3

- Vaatimusmäärittely on iteratiivinen prosessi toisiinsa sitoutuneita tehtäviä:
  - vaatimusten kartutusta
  - vaatimusten arviointia
  - oletusten tekoa
  - laadunvarmistusta
  - rajoitusten laadintaa
  - muuttuvien tavoitteiden hallintaa
  - dokumentointia
- Vaatimusmäärittelyn iteraatioihin osallistuu joukko sidosryhmiä, joilla voi olla ristiriitaisia tavoitteita ja vaatimuksia.

Yhteenveto 4

- Vaatimusmäärittelyssä kuvataan ongelmakentän ominaisuuksia:
  - Vaatimuksia: neuvoteltavissa olevia tavoitteita tulevalle järjestelmälle
  - Reunaehtoja: ongelmakentässä päteviä sääntöjä, joista ei voi neuvotella.
  - Oletuksia: ongelmakentässä päteviä sääntöjä, joista voi neuvotella.
- Ilman toimivia reunaehtoja ja oletuksia ongelmakentän asettamia vaatimuksia ei ole mahdollista kuvata oikealla tarkkuudella.

Yhteenveto 5

- Ongelmakentän ominaisuudet kuvataan spesifikaatiokielellä ja tallennetaan vaatimusdokumenttiin.
- Vaatimusdokumentti on vaatimusmäärittelyn lopputulos. Se sisältää ainakin löydetyt vaatimukset, reunaehdot, oletukset, tavoitteet ja vastuut.
- Iteratiivisuuden johdosta vaatimusdokumentti voi muuttua vaatimusmäärittelyn elinkaaren ajan, ja myös sen jälkeen.

Yhteenveto 6

- Huomattavaa:
  - Vaatimusmäärittelyn kohteena on koko tuleva järjestelmä, ei ainoastaan sen ohjelmistokomponentit.
  - Vaatimusmäärittelyä ja (arkkitehtuuri)suunnittelua on käytännössä pakko tehdä yhtä aikaa.
  - Vaatimuksista voidaan neuvotella, reunaehdoista ei.
  - Täsmällisen määrittelyn ei tarvitse olla formaali. Oikein käytettynä luonnollinen kieli riittää vaatimusten määrittelykieleksi.
  - Vaatimusmäärittely ei onnistu ilman yhtenäistä ja yksikäsitteistä ongelmakentän terminologiaa.
  - Vaatimusmäärittely on kaikkien ohjelmistoprosessien tärkeimpiä vaiheita, myös ketterien (agile) ja laihojen (lean).

Vaatimusmäärittelyn nykytila 1

- S. Elliott Sim, T.A. Alspaugh, B. Al-Ani: Marginal Notes on Amethodical Requirements Engineering: What Experts Learned from Experience. In: *Proc. 16th IEEE International Requirements Engineering Conference*, Barcelona, Spain, 2008, 105-114.
- L. Liu, T. Li, F. Peng: Why Requirements Engineering Fails: A Survey Report from China. In: *Proc. 18th IEEE International Requirements Engineering Conference*, Sydney, Australia, 2010, 317-322.

Vaatimusmäärittelyn nykytila 2

- Vaatimusmäärittely vaatii ohjelmistoteknistä kokemusta, sosiaalisia kykyjä ja kommunikointitaitoja.
- Vaatimusmäärittelijät toimivat siltana bisnesväen (ei teknistä tietämystä) ja ohjelmistokehittäjien (ei bisnesnäkemystä) välillä.
- Luonnollinen kieli on tärkein vaatimusmäärittelyn dokumentointi- ja spesifiointikieli.
- Toimiviksi todetut prosessit ja parhaat käytännöt vähentävät turhaa työtä ja virheitä, mutta niitä on käytettävä valikoiden.
- On tärkeämpää kirjoittaa vaatimukset sopivalla tarkkuus- ja abstraktiotasolla kuin tehdä vaatimusmäärittelydokumentista pitkä, yksityiskohtainen ja täydellinen.
- Vaatimusten tulee perustua liiketoimintaan eikä tekniikkaan.

Vaatimusmäärittelyn nykytila 3

- Eniten käytettyjä vaatimusmäärittelyn osavaiheita ovat vaatimusten kartutus, spesifiointi ja arviointi, vähiten käytettyjä puolestaan riskienhallinta, laadunvarmistus ja evoluution hallinta.
- Yleisimpiä vaatimusten kartutusmenetelmiä ovat haastattelut, prototyypit ja vastaavien nykyjärjestelmien taustatutkimus.
- Keskimäärin 10-20 % projekteihin käytettävästä työajasta kuluu vaatimusmäärittelyyn.
- Vaatimusten muuttuminen projektin aikana kuuluu asiaan, ja tapana on sopia muutoksista asiakkaiden kanssa.
- Vaatimusmäärittelyyn ei ole yleistä ”standardia”, vaan sen prosessi, menetelmät, tekniikat ja työkalut vaihtelevat suuresti.

Vaatimusmäärittelyn nykytila 4

- Suurimmat vaatimusmäärittelyn ongelmat:
  - Asiakkailla ei ole selvää ymmärrystä järjestelmästä eikä sen vaatimuksista.
  - Käyttäjien tarpeet ja toiveet muuttuvat jatkuvasti.
  - Ohjelmistokehittäjät eivät tunne ongelmakenttää riittävästi.
  - Projektien tiukka aikataulu rajoittaa liiaksi sidosryhmien ja kehitysryhmän välistä vuorovaikutusta.
  - Ongelmakentän ja järjestelmän välistä rajapintaa ei ole standardoitu, vaan se pitää joka projektissa määritellä erikseen.
- Vaatimusmäärittelyn kehityskohteita:
  - Vaatimusten kartutuksen, dokumentoinnin ja evoluution sisällyttäminen ohjelmistoprosesseihin ja ohjelmistoprojektien hallintaan.
  - Vaatimuksiin tehtävien muutosten ja lisäysten ennustaminen.
  - Vaatimusten linkittäminen testaukseen.
  - Erityisten vaatimusmäärittelyn työkalujen kehittäminen.