UUDEN SaaS-OHJELMISTOTUOTTEEN SUUNNITTELU: CASE AnyCase ASIANHALLINTA

Mikko Valjakka

Opinnäytetyö
Joulukuu 2012
Tietojärjestelmäosaamisen
koulutusohjelma

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojärjestelmäosaamisen koulutusohjelma

VALJAKKA, MIKKO:
Uuden SaaS-ohjelmistotuotteen suunnittelu: case AnyCase Asianhallinta

Opinnäytetyö 78 sivua, joista liitteitä 10 sivua
Joulukuu 2012

Tämän opinnäytetyön tarkoituksena oli tuottaa tietoa, joka tukee uuden AnyCase Asianhallinta -ohjelmistotuotteen suunnittelua. Ohjelmiston tarkoituksena on toteuttaa asianhallintaan liittyvien asiakirjojen työnkulkuja organisaatiossa. Ohjelmiston prototyyppi on ollut opinnäytetyötä tehdessä työn alla. Ohjelmisto julkaistaan avoimen lähdekoodin lisenssillä.

Tarkastelun kohteena ovat olleet ohjelmistolle valittu SaaS-jakelumalli, sekä ohjelmiston arkkitehtuuri. Työssä on selvitetty, miten SaaS-jakelumallin käyttö vaikuttaa ohjelmiston ominaisuuksiin, rakenteeseen, suorituskykyvaatimuksiin, arkkitehtuuriin ja markkinointiin. Eräs päätelmä on, että SaaS-jakelumalli vaikuttaa näihin ohjelmiston piirteisiin niin suuresti, että ohjelmisto on varta vasten toteutettava SaaS-jakelumallia varten. Merkittävimmät SaaS-jakelumallin erityisvaatimukset ohjelmistolle liittyvät ohjelmiston instanssin luomiseen asiakkaalle. Toinen erityispiirre on varautuminen ohjelmiston käyttövolyymin nopeaan kasvuun etenkin, jos uudet asiakkaat voivat ottaa ohjelmiston käyttöön verkosta oma-aloitteisesti.

Toinen selvityskohde koski AnyCase Asianhallinta -ohjelmistotuotteen arkkitehtuuria. Arkkitehtuurivaihtoehtojen selvittämisessä sovellettiin erityistä CBSP-vaatimusanalyysiä, jonka avulla ohjelmiston arkkitehtuuri voidaan johtaa ohjelmiston vaatimuksista. Menetelmä soveltuu käytettäväksi tilanteissa, joissa ohjelmisto on innovatiivinen, eikä sen arkkitehtuurivalinta ole itsestään selvä. Menetelmän antaa tulokseksi pisteytyksiä eri arkkitehtuurityyleille. AnyCase Asianhallinta sai parhaat pisteet client-server- ja SOA-arkkitehtuureille. Ensiksimainittua arkkitehtuuria oli jo sovellettu ohjelmiston prototyypissä. Sanomapohjainen arkkitehtuuri sai myös korkeat pisteet. Vastaavasti kerrosarkkitehtuuri sai huonot pisteet. Tuloksia on tarkoitus soveltaa ohjelmistotuotteen tulevassa kehitystyössä.

Samassa yhteydessä arvioitiin itse CBSP-analyysimentelmän käyttökelpoisuutta. Menetelmä on helposti laajennettavissa kattamaan uusia arkkitehtuurityylejä. Nykyiset systeemityömenetelmät eivät sisällä välineitä ohjelmiston arkkitehtuurin johtamiseksi. CBSP-menetelmän tarkoituksena on korjata tämä puute. Menetelmän soveltaminen on kuitenkin työlästä.

Asiasanat: työnkulku, SaaS, ohjelmistopalvelu, ohjelmistoarkkitehtuuri, vaatimusten hallinta

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Information System Competence

VALJAKKA, MIKKO:
Designing a new SaaS software product: case AnyCase Document Workflow Generator

Master's thesis 78 pages, appendices 10 pages
December 2012

The objective of this thesis was to produce information, that would support the development of ”AnyCase Document Workflow Generator” software product. The purpose of the software is to implement document workflows. While writing this thesis, the prototype of the software was under progress. The software will be published under open source licence.

In this thesis, the distribution model and the architecture of the software are under observation. The software uses SaaS (Software as a Service) distribution model. We study how the use of SaaS distribution model affects the features, structure, performance, architecture, and marketing of a software product. One of the conclusions is that the impact of the SaaS distribution model requires that the software must be dedicately designed to implement it. One of the most remarkable features in SaaS-based software is the instanciation of the software for the customer. Another key feature is the usage volume of the software. SaaS distribution model means that the usage volume may grow rapidly and unpredictably, especially if new customers are able to deploy the software independently.

The other main area of observation is the architecture of the software. In this case, the choice of architecure is not a trivial one. In order to determine applicable architectural styles, a special requirement analysis method CBSP was applied. The CBSP method is used to derive software architecture from software requirements. This unique method applies in cases of innovative software without obvious candidate for architecture. The output of the method are scores for architectural styles. AnyCase software got the best scores for client-server- and SOA architectures. Client-server architecure was already applied in the prototype of software. Message-based architecure got a high score, too. Consequently, layerd-based architecture scored poorly. The results will be applied in futher development of the AnyCase software.

Also, we evaluated the usability of CBSP method itself. The method can be easily extended to cover new architectural styles. The current sofwtare engineering methods exclude instrumentation for deriving software architecture. CBSP method aims to fill this gap. However, applying CBSP may be a costly undertaking.

Keywords: workflow, SaaS, software architecture, requirement analysis

SISÄLLYS

1. Johdanto

Tämän opinnäytetyö liittyy uuden ohjelmistotuotteen ”AnyCase Asianhallinta” kehittämiseen. Työn tarkoituksena on sekä analysoida tuotteen kehittämisessä tehtyjä suunnittelupäätöksiä, että tuottaa uutta tietoa tuotteen jatkokehitystä varten.

1.1. Ohjelmistotuote ”AnyCase Asianhallinta”

Jo pitkään on kehitetty ratkaisuja dokumenttien käsittelyn helpottamiseksi ja nopeuttamiseksi automaation avulla. Tämän kehityksen ensimmäinen vaihe on ollut saada dokumentit sähköiseen muotoon, jolloin niiden jakelu ja varastointi tehostuu. Nykyisin dokumenttien luonti tapahtuu pääasiassa sähköisesti, lukuun ottamatta käsin tapahtuvaa lomakkeiden täyttämistä. Dokumentin allekirjoittaminen edellyttää vielä vuonna 2011 enimmäkseen dokumentin saattamista paperimuotoon.

Seuraavana vaiheena dokumenttien käsittelyn automatisoimisessa voidaan ajatella dokumenttien työnkulkujen automatisoimista. Yksittäinen dokumentti ajatellaan tässä yhteydessä välineeksi, joka liittyy jonkin asian käsittelyyn. Toisinaan dokumentti ja asia, johon se liittyy, rinnastuvat toisiinsa lähes yksi yhteen, kuten ”opinnäytetyö”. Toisinaan tietyn asian käsittelyyn liittyy joukko dokumentteja, kuten esimerkiksi ”sairauseläkehakemus” tai ”hankesuunnitelma”. Asian käsittely voidaan mallintaa liiketoimintaprosessina, mutta samaan asiaan liittyvien yksittäisten dokumenttien työnkulut voivat silti olla hyvin erilaisia. Esimerkiksi jos käsiteltävä asia on ”hankesuunnitelma rakennuksen rakentamiseksi”, on asiaan liittyvän teknisen suunnitelman työnkulku erilainen kuin sen rahoitussuunnitelman työnkulku. Dokumenttien työnkulun automatisoimisessa joudutaankin pohtimaan sitä, valitaanko automaation kohteeksi vain yksittäisen dokumentin käsittely, vai se liiketoimintaprosessi, jossa dokumentti on osana.

AnyCase Asianhallinta –ohjelmisto on tarkoitettu sähköisten dokumenttien työnkulkujen automatisoimiseen. Ohjelmisto ei ota kantaa asiakirjojen sisältöön eikä sisällä välineitä asiakirjojen muokkaamiseksi. Ohjelmiston toimintaperiaate on yleisesti ottaen seuraava:

- Tietyn asian työnkulkua varten luodaan työnkulkumalli. Työnkulkumalli koostuu 1) asiakirjakansiosta, johon kopioidaan työnkulussa tarvittavat dokumentti-

pohjat. Työnkulun aikana kansioon kerääntyvät kaikki asian käsittelyyn liittyvät dokumentit; sekä 2) prosessikuvauksesta, joka esitetään vuokaavion avulla. Työnkulku koostuu tehtävistä, joihin liittyy tekijä, sekä kehote tehdä dokumenteille tarvittavat toimenpiteet. Lisäksi työnkulkuun voi liittyä ehdollisuutta ja päätöksentekoa. Ohjelmiston tärkeä perusajatus on se, että asian käsittelyn työnkulku tarkoittaa *koko kansion* virtuaalista kuljettamista vaiheesta toiseen ja tekijältä toiselle, yksittäisten dokumenttien siirtelyn asemesta.

- Työnkulku käynnistetään luomalla työnkulkumallista asiakohtainen ilmentymä. Sovelluksessa käytetään tietysti tavalliselle käyttäjälle sopivaa terminologiaa. Työnkulkumalli voi olla nimeltään ”jäsenhakemus”, ja yksittäinen työnkulku nimeltään ”jäsenhakemus/Heikki Heikkinen”. Ohjelmisto lähettää ensimmäiseen tehtävään liittyvän kehotteen tekijälle sähköpostin ja www-liittymän kautta. Käyttäjä saa nähtäväkseen tehtäväkohtaisen toimenpidesivun, jossa hänen on helppo suorittaa tehtävä. Kun käyttäjä kuitaa tehtävän tehdyksi, työnkulun suoritus etenee seuraavaan vaiheeseen. Ohjelmiston tulevissa versioissa on myös mahdollista määritellä työnkulku käynnistymään ulkoisen herätteen, esim. suoraan asiakkaalta www-sivujen kautta tulevat toimenpidepyynnön kautta.

Ohjelmiston perusidea on se, että jokseenkin kaikki dokumenttipohjaisen asianhallinnan työnkulut ovat kuvattavissa käyttämällä suhteellisen suppeaa joukkoa primitiivejä, joita ovat asiakirjan tai aineiston tiedoksi saattaminen, hyväksyminen, luominen tai täydentäminen, allekirjoitus, muistiinpano, sekä mahdollisesti työnkulkua ohjaava päätös. Yksittäinen tehtävä asianhallinnan työkulussa sisältää yhden tai useamman tällaisen primitiivin.

AnyCase-ohjelmisto julkaistaan avoimen lähdekoodin sovelluksena. Valinnalle käytetään seuraavia perusteita. Asiakkaan luottamus järjestelmään paranee, kun asiakkaalla on periaatteellinen mahdollisuus tietää, miten hänen tietojaan käsitellään. Asiakas voi myös periaatteessa ottaa ohjelmiston omaan käyttöönsä omin neuvoin maksamatta mitään itse ohlemiston käytöstä. AnyCase-ohjelmiston edellyttämä ajoympäristö on kuitenkin sellainen, että vain kaikkein suurimpien asiakasorganisaatioiden on järkevää ajatella sen pystyttämistä itsenäisesti. Palvelun ensisijaiselle kohderyhmälle, pk-organisaatioille, ohjelmiston käyttömahdollisuuden hankinta SaaS-palveluna on ehdottomasti kannattavin vaihtoehto.

Tuotteen kilpailutilanne

Tällä hetkellä (2011) markkinoilla olevat järjestelmät, joilla voi toteuttaa sähköistä dokumenttien käsittelyä ja varastointia, jakautuvat karkeasti seuraaviin ryhmiin:

- Järjestelmät, joita käytetään lähinnä dokumenttien varastointiin, jakeluun, julkaisemiseen ja versionhallintaan, esim. Microsoft SharePoint
- Järjestelmät, joissa voi edellisen lisäksi määritellä ja toteuttaa dokumenttien työnkulkuja ohjelmoinnin ja makrojen avulla, esim. IBM Lotus Notes/Domino
- Järjestelmät, jotka ovat parametroitavissa tapauskohtaisesti, mutta ovat yleisesti tarkoitettu tiettyjen tarkasti säädeltyjen ja standardoitujen toimialojen asianhallintaan, esim. julkishallintoon.
- Yksin tietyn asian käsittelyyn tarkoitetut räärälöidyt järjestelmät.

On luonnollista, että kaikkien yllämainittujen ratkaisujen käyttö on sitä yleisempää, mitä suuremmasta käyttövolyymistä ja organisaatiosta on kysymys. Vastaavasti järjestelmien tarjonta on sitä kapeampi, mitä pienimuotoisemmasta toiminnasta on kysymys.

Erityiseen katvealueeseen dokumenttien sähköisen käsittelyn alueella kuuluvat kansalaisjärjestöt. Kansalaisjärjestöjen, lähinnä reksiteröityjen yhdistysten, mahdollisuudet investoida it-järjestelmiin ovat rajalliset, vaikka toiminnan volyymi ei muilla mittareilla mitattuna olisikaan vähäinen. Lisäksi järjestötoiminta on pitkälle verkostoitunutta. Tästä seuraa, että yksittäinen järjestö voi olla mukana toiminnassa, jonka kokonaisvolyymi on suuri, mutta kukaan yksittäisistä toimijoista ei voi ajatellakaan investoivansa nykyisiin markkinoilla oleviin sähköisen dokumentinhallinnan tai prosessinhallinnan kokonaisratkaisuihin. Esimerkki tällaisesta toiminnasta on kansalaisjärjestöjen kehitysyhteistyö, jossa osapuolina ovat yleensä Ulkoministeriö, kehitysyhteistyöjärjestö (monesti lähetysjärjestö), tämän yhteistyöorganisaatiot sekä avun vastaanottajamaan organisaatio, lisäksi vielä Kehitysyhteistyön Palvelukeskus KePa.

On nähtävissä, että pienet ja keskisuuret organisaatiot, jotka käsittelevät tietoa ja joiden toiminta on yhä verkostoituneempaa, on yhä vaikeampaa ajatella dokumenttien käsitte-

lyn automaation käyttöönottoa, sillä tarjolla olevat järjestelmät edellyttävät paitsi suuria investointeja ja vaativaa käyttöönottoprojektia, myös varsin suurta homogenisuutta käyttöympäristön suhteen. Viimeksimainittu vaatimus johtuu paitsi teknisistä syistä, myös siitä, että järjestelmät on suunniteltu organisaatioiden sisäiseen käyttöön. Niitä on siksi vaikea sovittaa prosesseihin jotka ovat organisaatioiden välisiä, tai joihin muutoin liittyy ulkoinen asiakas.

AnyCase Asianhallinta –ohjelmistotuote on alun perin suunniteltu tukemaan pk-organisaatioiden asianhallintaa edellämainitun kaltaisissa olosuhteissa. Se on uusi palveluinnovaatio eikä markkinoilla varsinaisesti ole toista tuotetta tai palvelua, joka suoraan kilpailisi sen kanssa.

1.2. Tutkimusongelma ja hypoteesi

Tämän opinnäytetyön tavoitteena on vastata seuraaviin kysymyksiin:

1. Mitkä ovat uuden SaaS-ohjelmistotuotteen kehittämisen erityispiirteet? Kirjallisuusanalyysin perusteella kartoitetaan SaaS-jakelumallia käyttäviin ohjelmistotuotteisiin kohdituvia erityisvaatimuksia ja verrataan niitä AnyCase –ohjelmistotuotteessa käytettyihin ratkaisuihin.

2. Millainen ohjelmistoarkkitehtuuri soveltuu parhaiten AnyCase-ohjelmistotuotteelle?

   Ohjelmistoarkkitehtuurin valinta on uuden ohjelmistotuotteen tai -tuoteperheen kohdalla tärkeä suunnittelupäätös, joka tehdään aikaisessa vaiheessa. Usein arkkitehtuurin valinta on triviaali, koska suurin osa uusista ohjelmistoista on muunnelmia aiemmista tuotteista tai ideoista. Usein myös ohjelmistoympäristö on annettu. Sen sijaan jos kyseessä on ohjelmistoinnovaatio, arkkitehtuurin valinta on avoin kysymys, ja punnittavaksi voi tulla useita vaihtoehtoja. Tällaisessa epätyypillisessä tilanteessa on mahdollista soveltaa erityistä CBSP-vaatimustenhallintamenetelmää, jonka avulla voidaan ohjelmiston vaatimuksista johtaa ohjelmistoarkkitehtuuri (tarkemmin sanottuna arkkitehtuurityyli, ks. luku 3). Toinen erikoispiirre AnyCase-ohjelmistossa on, että se on tyypiltään

sovelluskehitin. Silloin vaatimusmäärittelystä tulee kaksitasoinen: toinen taso liittyy ominaisuuksiin, joita tarvitaan sovellusten kehittämiseksi, toinen taso liittyy ominaisuuksiin, jotka liittyvät kehitettyjen sovellusten eli tässä tapauksessa ohjelmiston avulla luotujen asianhallinnan työnkulkujen käyttöön.

Hypoteesina on, että AnyCase-ohjelmistotuotteen keskeiset suunnitteluratkaisut on valittu niin, että ne tukevat hyvin sekä SaaS-jakelumallia että asiakaskohtaisia asianhallinnan työnkulkuja.

2. OHJELMISTOJEN JAKELUMALLIT

Kaksi pääasiaa liittyy tapaan, jolla tietokoneohjelmisto saadaan asiakkaan käyttöön. Toinen on ohjelmakoodin tai sen suoritusmahdollisuuden fyysinen siirtäminen asiakkaalle. Toinen on ohjelmiston käyttöoikeuden siirto. Kokonaisuuteen lasketaan lisäksi mukaan dokumentaation ja tukipalvelujen tarjoaminen asiakkaalle.

Yleisiä kaupallisia ohjelmistojen jakelutapoja ovat mm.

- käyttöoikeuden myöntäminen asiakkaalle ei lisenssointi. Käyttöoikeus voi olla määräaikainen tai jatkuva. Tähän jakelumalliin liittyy usein ohjelmiston ja mahdollisesti käyttöohjeiden luovuttaminen jonkin fyysisen median muodossa, nykyisin lähinnä DVD-levyllä tai USB-muistitikulla.
- OEM-lisenssoinnissa (original equipment manufacturer -) ohjelmistolisenssi on kytketty uutena myytävään laitteeseen, tunnetuin esimerkki on Windows-käyttöjärjestelmä.
- Verkon kautta tapahtuva ohjelmistojen jakelu, ks. jäljempänä
- Avoimen lähdekoodin ohjelmistot (open source). Vaikka ohjelmiston jakaminen asiakkaan käyttöön ilmaiseksi ei ensi alkuun kuulostakaan kovin kaupalliselta, avoimet ohjelmistot ovat yhä yleisemmin osa sellaisia palvelukokonaisuuksia, joihin liittyy merkittävää liiketoimintaa. Tyypillisesti esim. monet kaupalliset palvelut pyröivät avoimen lähdekoodin Linux-käyttöjärjestelmän päällä. Usein myös asiakas voi löytää itselleen sopivan avoimen lähdekoodin ohjelmiston, mutta hänelle on järkevämpää ostaa sille käyttöympäristö kuin investoida ja ylläpitää sellaista itse. Avoin lähdekoodi itsessään on enemmänkin julkaisu- kuin jakelutapa, mutta tyyppillisesti ohjelmiston alkuperäinen versio lähdekoodeineen on saatavissa verkon kautta, ja tuotteistettuja versioita (distribuutioita) voi ostaa esim. DVD-levyllä, jolloin asiakas maksaa ohjelmiston paketoinnista, ei itse ohjelmistosta.

2.1. Ohjelmistot palveluna: ASP, SaaS, pilvipalvelut

Ohjelmistojen käyttöpalvelujen historian voidaan katsoa alkaneen 1960-luvulla keskustietokoneiden osituskäyttönä (time-sharing) (Kivelä 2010, 5). Asiakas käyttää ohjelmistoa vuokraa tai palvelumaksua vastaan. Tyypillisesti ohjelmisto saatetaan asiakkaan käyttöön verkon kautta. Menetelmästä käytettäviä nimityksiä ovat sovellusvuokraus (ASP, application service provider) ja ohjelmistopalvelu (SaaS, software as a service). Termejä käytetään monesti samassa merkityksessä, eivätkä ole vakiintuneita. Englanninkielinen ASP-lyhenne viittaa selkeästi palvelua tuottavaan yritykseen tai organisaatioon, mutta suomenkielisissä lähteissä sovellusvuokraukseen on viitattu yleisesti termillä ASP (Tieke ry 2011; Kivelä 2011, 24). SaaS-lyhenteen käyttö on yleistymässä, ASP:n vähenemässä (Kivelä 2010, 28). Termiä "sovellusvuokraus" käytettiin aikaisemmin etenkin tilanteessa, jossa vuokrattava sovellus olisi ollut myös hankittavissa käyttäen perinteistä lisenssointia. Nykyisin palveluna tarjottavat ohjelmistot on käytännössä aina laadittu varta vasten tähän tarkoitukseen. Käytämme ohjelmiston tarjoamisesta palveluna jatkossa lyhennettä SaaS.

Pilvipalvelut (cloud computing) on yleisnimitys palveluntarjoajan IT-resurssien, kuten ohjelmistojen, laitteistojen tai palvelujen, dynaamiselle tarjoamiselle asiakkaiden käyttöön verkon välityksellä (Salo 2010, 16). SaaS-palvelut voidaan määritellä pilven kautta käytettäviksi ohjelmistoiksi (Reese 2009, 2). Voidaan puhua myös pilvisovelluksista (cloud application).

2.2. Esimerkkejä SaaS-palveluista

Tämän opinnäytetyön kannalta tärkeimpiä esimerkkejä SaaS-palveluista ovat kotimaiseen käyttöön tarkoitetut talouden ja hallinnon palvelut yrityksille sekä muut asianhallintaa sivuavat palvelut. Nämä ovat samalla oiva esimerkki SaaS-palveluntarjonnan meneillään olevasta nopeasta laajentumisesta.

Visma-taloushallinnon verkkopalut (Visma Oy 2011) tarjoaa norjalainen Visma AS. Palvelut ovat tarjolla pohjoismaissa, Hollannissa ja Iso-Britanniassa. Palveluvalikoima

kattaa Suomessa taloushallinnon peruspalvelujen (laskutus, kirjanpito, palkanmaksu) lisäksi toiminnanohjauksen (ERP), myynnin- ja asiakkuudenhallinnan (CRM) sekä taloushallinnon ulkoistuspalvelut.

Netvisor-taloushallinnon verkkopalvelut (Netvisor Oy 2011) tarjoaa kotimainen Netvisor Oy. Palveluvalikoimaan kuuluvat talouhallinnon peruspalvelujen lisäksi henkilöstö- ja asiakkuudenhallinta, sekä kehittyneet raportointityökalut yritysjohdon käyttöön.

Salesforce CRM (Salesforce 2011) on suuri, kansainvälinen asiakkuudenhallintapalvelujen tarjoaja. Palvelut on pitkälle tuotteistettu, etusivulla avautuvat suoraan päätuotteet hintoineen ja mahdollisuus ottaa ne heti käyttöön.

2.3. SaaS-jakelumallin hyödyt

SaaS-liiketoiminta on voimakkassa kasvussa (Gartner: Gartner Newsroom 2011). Asiakas odottaa saavansa sovelluspalvelua käyttämällä seuraavia hyötyjä. Sääksjärvi, Lassila, Nordström 2005, s.183 listaa kursivoidut hyötyodotukset:

- *Ohjelmistopalvelu mahdollistaa sen, että asiakas voi paremmin keskittyä omiin ydintoimintoihinsa.* (Toisin sanoen, SaaS-palvelut ovat osa yleistä ulkoistamiseen tähtäävää liiketoimintalogiikkaa.)
- *Ohjelmistopalvelun avulla sovelluksen käyttämisessä tarvittava tekninen asiantuntemus on paremmin ja/tai helpommin saatavilla.*
- *Ohjelmistopalvelu lyhentää järjestelmän käyttöönottoon kuluvaa aikaa.*
- *Ohjelmistopalvelu mahdollistaa laajemman ja joustavamman valikoiman maksutapoja.* (SaaS-palvelujen hinnoittelumallina on yleensä maksaminen vain toteutuneesta käytöstä.)
- *Ohjelmistopalvelu helpottaa päivityksiä ja versionhallintaa*
- *Ohjelmistopalvelun tarjoaja luo kokonaisvaltaisemman sovellustarjonnan yhdistelemällä ohjelmistoja eri lähteistä*
- *Ohjelmistopalvelu tarjoaa asiakkaille mahdollisuuden käyttää sellaisia ohjelmistoja, jotka olisivat heille liian kalliita ostaa.*

- *Ohjelmistopalvelu mahdollistaa sovellusten käytön ajasta ja paikasta riippumatta* (Käytännössä valtaosa SaaS-palveluista näkyy asiakkaalle selainkäyttöisinä ohjelmina.)
- *Ohjelmistopalvelu järjestelmään kohdistuvat investoinnit / aloituskustannukset ovat pienemmät.*
- *Ohjelmistopalvelu tuo asiakkaalle huomattavasti paremman IT-infrastruktuurin arvioitaessa luotettavuutta, turvallisuutta ja skaalautuvuutta.* (Tärkeä osa SaaS-palvelun ideologiaa on, että sovellusta ajetaan pääasiassa palveluntuottajan alustalla, jolloin asiakkaan investointi sovelluksen ajoympäristöön on vähäinen tai olematon).
- *Ohjelmistopalvelu laajentaa asiakkaalle potentiaalisten sovellusten lukumäärä*
- *Ohjelmistopalvelu parantaa saatavilla olevia parametrointimahdollisuuksia asiakkaalle tarjolla olevissa ohjelmissa.* (Tämä hyötyodotus perustuu siihen, että SaaS-ohjelmistot on lähtökohtaisesti tarkoitettu laajalle asiakaskunnalle, jolloin niiden räätälöiominaisuuksien oletetaan olevan hyvät.)

SaaS-palvelujen tuottamiseen liittyy seuraavia osa-alueita, joiden toteuttaminen tyypillisesti jakautuu useamman yrityksen/organisaation kesken (Kivelä 2010, 39). SaaS-ohjelmistot tyypillisesti noudattavat nykyaikaista verkossa ajettavien ohjelmistojen monikerros- ja/tai komponenttiarkkitehtuuria. Ajoympäristöön useimmiten liittyy tietokantapalvelimia, sovelluspalvelimia ja www-palvelimia. SaaS-palveluntarjoaja voi toteuttaa nämä palvelut itse tai ostaa ne sopivalta palvelinoperaattorilta (tai "pilvestä").

2.4. SaaS-jakelumuodon erityisvaatimukset ohjelmistotuotteelle

2.4.1. Ohjelmointitekniset erityispiirteet

Ohjelmistotuotteen ohjelmistoteknisen suunnittelun kannalta ohjelmiston tarjoaminen SaaS-muotoisesti tarkoittaa, että

- ohjelmiston kaikkien tai lähes kaikkien ominaisuuksien tulee olla selainkäyttöisiä, tai muutoin niin että käytettävä asiakaspääteohjelmisto on helposti otettavissa käyttöön verkon kautta.

- asiakas ei itse asenna ohjelmistoa, mutta lähtökohtaisesti suorittaa itse sen käyttöönoton ja parametroinnin.
- ohjelmiston asiakkaat ovat yleensä yrityksiä tai yhteisöjä, jotka määrittelevät itse ohjelmiston varsinaiset loppukäyttäjät.
- samaa ohjelmistoa käyttää monta asiakasta/käyttäjää samanaikaisesti.

Ohjelmisto voidaan toteuttaa niin, että jokaiselle asiakkaalle luodaan ohjelmistosta oma kopio palveluntarjoajan palvelimelle. Tämä mahdollistaa esim. ohjelmakoodin räätälöinnin asiakaskohtaisesti, mutta vaikeuttaa ohjelmiston versionhallintaa merkittävästi. Järkevämpää on yleensä pyrkiä toteuttamaan ohjelmisto niin, että sen ominaisuuksiin kuuluu asiakaskohtaisten instanssien luominen ja riittävän laajat asiakaskohtaiset parametrointimahdollisuudet. ”Asiakaskohtainen instanssi” tarkoittaa tässä yhteydessä sitä, että ohjelmisto on rakennettu niin, että se mahdollistaa monen toisistaan riippumattoman asiakasorganisaation perustamisen ja tietojen käsittelyn. Samalla tulee kuitenkin säilyä rajoitettu mahdollisuus kaikkien instanssien keskitettyyn hallintaan, esimerkiksi ohjelmiston ja tietokannan versiopäivitykseen.

Edelleen ohjelmiston tietokantaratkaisun yhteydessä on päätettävä, ovatko eri asiakkaiden tiedot yhdessä tietokannassa vai asiakaskohtaisissa kannoissa. Tietoturva- ja tehokkuusnäkökohdat puoltavat jälkimmäistä vaihtoehtoa.

2.4.2. Skaalautuminen

Yksi syy siihen, miksi SaaS-ohjelmistopalvelujen tarjoajan kannattaa pikemminkin ostaa palvelinkapasiteetti palvelinoperaattorilta kuin ylläpitää tällaista kapasiteettia itse, on palvelun vaatiman kapasiteetin huono ennustettavuus ja suuri vaihteluväli. Palvelu voi saada yllättäen useita suuria asiakkaita, jolloin vaadittava palvelinkapasiteetti voi moninkertaistua. SaaS-palvelujen luonteeseen kuuluu, että asiakas olettaa saavansa tarvitsemansa palvelun lähes viivytyksettä. SaaS-palvelujen tarjoajan voi olla järkevää tukeutua palvelinoperaattoreiden tarjoamiin erilaisiin ns. on demand -ratkaisuihin tai "pilveen", joilla lisäkapasiteettia saa käyttöön liukuvasti ja lyhyellä varoitusajalla.

2.4.3. Turvallisuus

SaaS-palvelun kohdalla palvelujen turvallisuus on erityinen puheenaihe. Asiakkaan kokemat turvallisuusuhat SaaS-palveluihin liittyen ovat osittain samoja kuin ulkoistamisessa koetut riskit yleensä. Palvelujen turvallisuuden osa-alueita ovat mm.

- Tietoturvallisuus (Reese 2009, 99-118). Voiko organisaation kriittisiä tietoja siirtää ulkoiseen palveluun? Kysymys mutkistuu entisestään, jos SaaS-palvelun tarjoaja ostaa palvelinkapasiteettinsa pilvestä. Silloin tieto siitä, missä data fyysisesti sijaitsee, ei pääsääntöisesti ole saatavilla. Silloin ei ole myöskään mahdollista tietää, mitkä osapuolet kulloinkin vastaavat infrastruktuurin eri osista. Toisaalta pilvitektologia tarjoaa mahdollisuuden nopeaan virhetilanteesta toipumiseen. Asiakkaan SaaS-palveluinstanssin tulisi olla sellainen, että se ongelmatilanteissa saadaan nopeasti otettua käyttöön vaihtoehtoisella alustalla. Pilviteknologian ideologiaan kuuluu, että tällainen vaihto voi tapahtua ns. lennossa.
- Yksityisyys (Salo 2010, 104,105). SaaS-palveluntuottajan tarjotessa samaa palvelua useille asiakkaille, asiakkaiden sisäänkirjautuminen, instanssit ja tietovarastot sijaitsevat ja tapahtuvat yhteisessä teknisessä ympäristössä. Voi olla vaikeaa saada selville, keillä kaikilla on pääsy tämän ympäristön eri osiin. Lisäksi ohjelmiston puutteellinen tietoturva voi aiheuttaa, että asiakkaat pääsevät lukemaan tai jopa muokkaamaan toistensa tietoja. Reese (2010, 102) suositteleekin, että asiakas salakirjoittaa kaiken datansa, jonka hän siirtää pilvipalveluun.
- Jatkuvuus. SaaS-palvelun käyttöönotto sitoo organisaatiota enemmän kuin ohjelmiston hankinta omaan käyttöön esim. lisenssoinnilla. Omassa käytössä olevan ohjelmiston käyttöä voi jatkaa ainakin jonkin aikaa, vaikka ohjelmiston tuottaja katoaisi kartalta. SaaS-palveluntuottajaa valitsessaan organisaatiota kiinnostaa palveluntuottajan elinkelpoisuus ja tieto siitä, mitä organisaatiolle merkitsee, jos palvelun käytöstä joudutaan luopumaan, tai vaihtamaan palvelun tuottajaa.

AnyCase-ohjelmistotuotteen kohdalla avoimen lähdekoodin lisenssoinnilla ajatellaan olevan asiakkaan näkökulmasta seuraavia tietoturvallisuutta parantavia vaikutuksia:

- Tietojen käsittelytapa on julkinen. Tosin tarkkaan ottaen tästä ei vielä seuraa, ettei tietovarkaus voisi olla mahdollinen, ovathan tietokannat yleisesti ottaen luettavissa myös sovelluksista irrallaan.
- Täysin samalle palvelulle voi todennäköisemmin löytyä useita tarjoajia.
- Asiakkaalla on myös vaihtoehto asentaa ohjelmistoa itsenäisesti.

2.4.4. Järjestelmäintegraatio

Mikään tietojärjestelmä ei ole olemassa erillään muista. Vaatimukset eri tietojärjestelmien integraatiolle kasvavat koko ajan. Yleisiä integraation välineitä ovat tämän päivän XML-pohjaiset tekniikat. Niiden avulla kuvataan sekä liittymiä (rajapintoja) että datan tallennus- ja siirtomuotoja. SaaS-ohjelmistojen integraatiolle ollaan luomassa standardeja, jotka hyödyntävät mm. SOA-arkkitehtuurin parissa tehtyä kehitys- ja standardointityötä (Sun, Zhang, Chen, Zhang, Liang 2007, 563). Koska SaaS-ohjelmistot ovat uusia ja verkkokäyttöön rakennettuja, on niissä yleensä otettu hyvin huomioon erilaiset integraatiotarpeet ja -ratkaisut. Asiakasorganisaatio voi olettaa, että tietyt SaaS-ohjelmistot osaavat vaihtaa tietoja keskenään. Tämä on luonnollinen vaatimus samoja liiketoimintaprosesseja tukeville ohjelmistoille, kuten toiminnanohjaus- ja taloushallinto-ohjelmistoille. Tyypillinen kaikkiin ohjelmistoihin liittyvä integraatiotarve on myös se, että ohjelmistoon kirjautuminen voidaan toteuttaa käyttäen organisaation oman verkon käyttäjätunnuksia. Tämä on teknisesti yksinkertaista toteuttaa esim. LDAP-protokollan avulla, mutta organisaatiolle voi olla ongelmallista sallia ulkoiselle SaaS-palvelulle edes lukuoikeutta omaan käyttäjätunnusten hakemistopalveluunsa.

2.5. AnyCase-ohjelmiston jakelumalli

AnyCase-ohjelmiston tämänhetkinen jakelumalli ja -menetelmä on suunniteltu ylimalkaisesti tukemaan mahdollisimman helppoa käyttönottoa, mutta vertailua muihin tuotteisiin ei ole tehty, eikä niistä ole otettu varsinaisesti mallia. Tämän opinnäytetyön yksi tavoite on kerätä tietoa jakelumallin kehittämiseksi.

AnyCase-ohjelmiston käyttöönotto tapahtuu nykyisen suunnitelman mukaan seuraavasti: Palvelun julkisella verkkosivulla on toiminto ”aloita asianhallinta tästä”. Uusi asiakas (yritys tai muu organisaatio) voi rekisteröityä käyttäjäksi antamalla yhteystietonsa. Rekisteröinnin seurauksena asiakkaalle syntyy välittömästi palveluinstanssi, jossa rekisteröinnin suorittanut henkilö on oletusarvoisesti instanssin pääkäyttäjä. Rekisteröinti vahvistetaan sähköpostilinkin kautta. Pääkäyttäjä pääsee heti käsiksi valmiiden työnkulkumallien luetteloon, josta hän voi valita omaan organisaatioonsa sopivia malleja ja muokata niitä. Pääkäyttäjä voi luoda omalle organisaatiolleen muita käyttäjiä ja antaa näille oikeuksia. Asianhallinnan työnkulun käynnistäminen edellyttää, että työnkulkumalli on valittu, sille on osoitettu vastuuhenkilö ja mahdolliset organisaation omat asiakirjapohjat on liitetty osaksi työnkulkumallia.

Edellä kuvatun käyttöönottomenetelmän suunnitteluperiaatteita ovat nopeus, helppous ja ilmaisuus. Käyttöönoton suurin haaste on se, että toisin kuin esim. taloushallinto-ohjelmissa, AnyCase-sovelluksen konteksti on uusi: asiakas on todennäköisesti ensi kertaa automatisoimassa oman organisaationsa asianhallinnan työnkulkuja. Siksi kaikenlaisella tukimateriaalilla ja tukipalvelulla on käyttöönotossa suuri merkitys. Materiaalissa konteksti joudutaan selittämään perusteista lähtien ja asiakkaan on mm. ymmärrettävä, että hänellä tulisi olla selkeä ennakkokäsitys niistä työnkuluista, joita palvelun avulla on tarkoitus automatisoida.

Sopivan hinnoittelumallin löytäminen AnyCase-palvelulle on haastava tehtävä. Aidon SaaS-palvelun tunnuspiirteenä on maksaminen vain toteutuneesta käytöstä (Reese 2011, 3). Tämän periaatteen mukaisesti asiakas ottaa AnyCase-palvelun verkosta ensin koekäyttöön maksutta. Koekäytölle ei tässä tapauksessa ole suunniteltu muuta rajoitusta kuin aikaraja: ajan umpeutuessa asiakas joko siirtyy maksulliseen käyttöön tai asiakkaan palveluinstanssi poistetaan järjestelmästä. Koekäyttömahdollisuus toteutetaan automaattisesti. Koekäyttöajan kannattaa olla suhteellisen pitkä, koska palvelukonteksti on uusi ja asiakkaan tutustuminen palveluun vie aikaa. Parhaassa tapauksessa asiakas on koekäyttöajan umpeutuessa jo ehtinyt käynnistää useita työnkulkuja, jolloin hän on jo todennäköisesti sitoutunut järjestelmän käyttöön.

Toteutuneen käytön tekninen mittaaminen on yksinkertaista. Mittareina voi käyttää AnyCase-ohjelmistotuotteen kohdalla esim. asiakirjojen määrää, työnkulkujen määrää,

rekisteröityjen käyttäjien määrää tms. Mutta kaikki nämä mittausperusteet ovat asiakkaan ja sitä kautta myös palvelun myynnin kannalta vaikeita. Koska palvelukonteksti on uusi, asiakkaan voi olla hyvin vaikea hahmottaa etukäteen, millainen tekninen volyymi käytöllä tulee olemaan. Organisaatiokohtaiset työnkulut voivat olla lisäksi hyvin erilaisia, toisella asiakkaalla voi olla harvoja käyttäjiä ja työnkulkuja, jotka ovat kuitenkin pitkiä ja monimutkaisia ja organisaation toiminnan kannalta keskeisiä. Toisella asiakkaalla voi puolestaan olla paljon pieniä työnkulkuja, esim. pelkkää asiakirjojen tiedoksi saattamista ja kuittaamista.

Hyvän hinnoittelumallin tulee olla selkeä ja ennustettava ja parhaassa tapauksessa tuttu jostain muusta yhteydestä. Tällaisia voisivat olla esim. käyttäjien määrä tai organisaation henkilöstön määrä. Toisaalta nämä eivät ole täysin linjassa toteutunut käyttö -periaatteen kanssa.

Edelleen on huomioitava, että palvelun käyttöönotto merkitsee asiakkaalle muutosta ja sitoutumista, vaikka käyttöönotto olisi teknisesti tehty miten helpoksi tahansa. Teknisesti täysin virtaviivaisessa käyttöönotossa onkin enemmän kyse markkinonnista kuin pragmaattisuudesta. Vastaavasti hinnoittelun kannalta jonkinlainen käyttöönottomaksu, joka sisältää esim. tietyn määrän asiakasneuvontaa, olisi perusteltu, ja se olisi myös asiakkaan helppo ymmärtää. AnyCase-palvelun hinnoittelumallia ei ole vielä valittu.

Tekniseltä kannalta palvelun asiakaskohtainen instanssi AnyCase-järjestelmässä tarkoittaa, että palvelun kaikki asiakkaat kirjautuvat samaan verkkopalveluun ja käyttävät samaa kopiota ohjelmistosta, mutta kullakin asiakkaalla on oma tietokanta sekä tiedostorakenne. Ohjelmiston tietoturvaominaisuuksissa on kiinnitetty erityistä huomiota asiakasinstanssien eristämiseksi toisistaan. Lisäksi on mahdollista käyttää käyttöjärjestelmätason käytönvalvontaa tiedosto-operaatioiden yhteydessä.

AnyCase-palveluun on tarkoitus sisällyttää LDAP-tuki, mikä tarkoittaa, että käyttäjät voivat kirjautua palveluun käyttäen oman organisaationsa käyttäjätunnuksia. Rinnakkainen mahdollisuus on, että käyttäjätunnuksina käytetään nykyaikaisten verkkopalvelujen tapaan sähköpostiosoitteita.

3. VAATIMUSTEN HALLINTA JA ARKKITEHTUURI

3.1. Vaatimusten hallinta

Vaatimusten hallinnalla (software requirement analysis) tarkoitetaan asiakkaan tietojärjestelmään kohdistamien vaatimusten kirjaamista yhtenäisellä tavalla ainakin siten, että

- vaatimukset on ilmaistu tekniseltä kannalta riittävän yksilöivästi,
- vaatimukset on ilmaistu yksikäsitteisellä/mitattavalla tavalla niin, että on mahdollista objektiivisesti sanoa, täyttääkö toteutettu järjestelmä vaatimukset vai ei;
- vaatimukset on priorisoitu,
- vaatimuksien yhteys toisiinsa on todettu.

Vaatimusmäärittelyn tarkoituksena on kuvata kaikkien sidosryhmien ymmärtämällä tavalla se, mitä järjestelmällä on tarkoitus pystyä tekemään. Järjestelmän toimittaja ja tilaaja tukeutuvat vaatimusmäärittelyyn muodostaakseen yhteisen käsityksen siitä, mitä tilaaja on tilannut ja vastaako toimitettu järjestelmä tilaajan tarpeita. Kaikki vaatimusten hallintaa käsittelevät lähteet nostavat vaatimustenhallinnan lähtökohdaksi asiakaskeskeisyyden.

Vaatimusten luokittelusta ja kirjaamisesta

Vaatimuksia voi luokitella mm. seuraavasti (Viegers 2003, 8-13):

- Liiketoiminnalliset vaatimukset (business requirements) ilmaisevat ne tilaajaorganisaation asettamat tavoitteet, jotka tilattavan järjestelmän avulla on tarkoitus saavuttaa. Samalla ne toimivat järjestelmän rajauksena.
- Käyttäjän vaatimukset (user requirements) ilmaisevat, mitä tietyn käyttäjän tulee pystyä tekemään järjestelmällä. Vaatimus ilmaisee tavallisesti jotain liiketoiminnallista tavoitetta ja voi pohjautua käyttötapausanalyysiin (use case, ks. jäljempänä).
- Toiminnalliset vaatimukset (functional requirements) kertovat, mitä toimintoja ohjelmiston tulee sisältää, jotta käyttäjä pystyy suorittamaan hänelle kuuluvat tehtävät.

- Ei-toiminnalliset vaatimukset (non-functional requirements) voivat liittyä mm. liiketoimintasääntöihin (business rules), organisaation tietostrategiaan ja -malliin, järjestelmään kohdistuviin teknisiin rajoitteisiin (constraints), rajapintoihin, tietoturvatasoon sekä muihin laadullisiin odotuksiin (quality attributes).

Vaatimuksista kootaan vaatimusmäärittelydokumentti, joka käsittää toiminnalliset vaatimukset, sekä järjestelmän suunnitteluun ja toteutukseen liittyvät ei-toiminnalliset vaatimukset.

Yksittäinen toiminnallisen vaatimuksen tie vaatimusmäärittelydokumenttiin voi olla seuraava:

- Liiketoiminnallisten vaatimusten pohjalta muodostetaan näkemys uudesta tietojärjestelmästä. Näkemyksen pohjalta listataan käyttäjätason vaatimuksia.
- Käyttäjätason vaatimukset toimivat käyttötapausanalyysin (use case) lähtökohtana.
- Käyttötapauksista johdetaan joukko toiminnallisia vaatimuksia, jotka analysoidaan ja priorisoidaan.
- Riittävän merkityksellinen ja selkeästi ilmaistu tominnallinen vaatimus päätyy vaatimusmäärittelydokumenttiin.

Hyvän vaatimuksen tunnuspiirteitä ovat (Viegers 2003, 22-25):

- kattavuus: vaatimus kuvaa halutun toiminnallisuuden kokonaisuudessaan niin, että järjestelmän kehittäjän on mahdollista suunnitella ja toteuttaa se.
- oikeellisuus: vaatimuksen tulee kuvata haluttu toiminnallisuus täsmällisesti. Oikeellisuuden varmistaminen tapahtuu kommunikoimalla vaatimuksen lähteen eli pääsääntöisesti käyttäjien kanssa.
- sovellettavuus: vaatimuksen tulee olla toteutettavissa annetussa ympäristössä.
- tarpeellisuus: jokaisen vaatimuksen tulee perustua asiakkaan itse toteamaan tarpeeseen, tai muuhun dokumentoituun tarpeeseen.
- priorisointi: on suositeltavaa, että jokainen toiminnallinen vaatimus, ominaisuus ja käyttötapaus priorisoidaan. Tämä antaa projektille tarpeellista liikumavaraa mm. ajankäytön ja kustannusten osalta.

- yksikäsitteisyys: vaatimuksen ilmaiseminen niin, että kaikki osapuolet ymmärtävät samalla tavalla ja oikein. Tähän päästään käyttämällä selkeää, tarkoitukseen sopivaa kielellistä ilmaisutapaa, sekä varmistamalla, että osapuolet ymmärtävät käytettyä terminologiaa.
- tarkistettavuus: vaatimuksen pohjalta totetutettu toiminnallisuus tulisi olla testattavissa niin, että sen yhteys vaatimukseen käy selkeästi ilmi, mielellään vielä niin, että testaus voisi perustua sopivaan demonstraatioon jo ennen järjestelmän toteuttamista.

Edelleen hyvän vaatimusmäärittelyn tunnuspiirteitä kokonaisuudessan ovat:

- kattavuus: kaikkien vaatimusten ja muun vaadittavan informaation kuuluu olla mukana. Tämä varmistetaan parhaiten käymällä kertaalleen läpi käyttäjän haluama toiminnallisuus.
- ristiriidattomuus: vaatimukset eivät ole ristiriidassa sen enempää muiden saman ryhmän vaatimusten kuin esimerkiksi liiketoiminnallisten vaatimusten kanssa.
- muokattavuus: vaatimusten kirjaamisen tulee olla systemaattista niin, että viittaminen yksittäiseen vaatimukseen on mahdollista ja vaatimuksen muutoshistoria tunnetaan.
- jäljitettävyys: mikäli mahdollista, vaatimuksen tulisi olla jäljitettävä siten, että valmiista järjestelmästä voidaan osoittaa ne osat, jotka perustuvat vaatimukseen.

Vaatimusten jäljitettävyys (tracking) korostuu jatkossa esitettävässä ja sovellettavassa CBSP-vaatimustenhallintamenetelmässä, jonka tarkoituksena on johtaa ohjelmistoarkkitehtuuri vaatimusmäärittelystä. Arkkitehtuuriin liittyvät valinnat ovat järjestelmän kannalta kriittisiä, joten on tärkeää voida osoittaa tehtyjen valintojen yhteys asiakkaan esittämiin vaatimuksiin.

Vaatimusten jäljitettävyyden merkitys korostuu etenkin järjestelmän elinkaaren myöhemmissä vaiheissa (Viegers 2003, s. 357). Jäljitettävyyden tarjoamia etuja ovat mm.

- Katselmointi: etenkin turvallisuuden kannalta kriittisten järjestelmien kohdalla dokumentoitu jäljitettävyys auttaa osoittamaan tilaajalle, että kaikki vaaditut ominaisuudet on toteutettu.

- Muutosten sivuvaikutusten huomiointi: ellei vaatimusten jäljitettävyydestä ole pidetty kirjaa, on vaikea ennakoida järjestelmässä ilmeneviä sivuvaikutuksia, jotka aiheutuvat vaatimusmäärittelyssä tapahtuvista muutoksista.
- Ylläpito: liiketoiminnallisten vaatimusten muuttuessa järjestelmän ylläpito helpottuu, jos jäljitettävyysdokumentaation avulla voidaan suoraan tietää, mihin järjestelmän osiin muutos vaikuttaa.
- Järjestelmän uudistaminen: kun järjestelmästä tehdään seuraava versio, tai järjestelmä uudistetaan kokonaan, voidaan jäljitettävyysdokumentaation avulla vanhaa ja uutta toteutusta verrata vaatimusten teknisen toteuttamisen osalta. Edelleen koodin uudelleenkäyttö helpottuu, kun on nähtävissä, mihin liiketoiminnalliseen funktioon mikin ohjelmamoduli liittyy.
- Riskienhallinta: jos esimerkiksi avainhenkilö jättää projektin, jäljitettävyysdokumentaatio auttaa näkemään, mihin osaan kokonaisuudesta esim. tietty ohjelmamoduli liittyy.
- testaus: odottamattomien testaustulosten kohdalla sen tunteminen, mikä on testin, koodin ja vaatimusten yhteys, auttaa paikallistamaan ongelman, sekä myös välttämään päällekkäistä testaamista.

Jäljitettävyyden dokumentointivälineenä voidaan käyttää erilaisia jäljitettävyysmatriiseja (Sommerville, Sawyer 1997).

3.2. Ohjelmistoarkkitehtuurit

Termi "ohjelmistoarkkitehtuuri" on varsin laaja-alainen, ja sen merkitys eri asiayhteyksissä on erilainen (Koskimies, Mikkonen 2005, 18). Kuten rakennusarkkitehtuurissa, ohjelmistoarkkitehtuuritkin voivat liittyä mittakaavaltaan aivan eri kokoisiin asioihin. Siksi on tarpeen rajata termin merkitys tavalla, joka on tämän tutkimuksen kannalta tarkoituksenmukainen. Myöhemmin esiteltävässä CBMP-vaatimusanalyysimenetelmässä käytetään termiä "achitectural styles" viittaamaan joukkoon tapoja, joilla 1) ohjelmisto kokonaisuudessaan voidaan jakaa komponentteihin, 2) millainen on näiden komponenttien ontologia, ja 3) miten komponentit kommunikoivat keskenään. Tällöin voi-

daan puhua esimerkiksi asiakas-palvelin (client-server) -arkkitehtuurista tai erilaisista kerrosarkkitehtuureista (layered-).

Komponenttien välinen kommunikointi on perinteisessä systeemityössä jäänyt melko vähälle huomiolle, kun otetaan huomioon ne tavat, joilla laajasti hajautettuja tietojärjestelmiä nykyisin toteutetaan (Taylor, Medvidovic, Dashovy 2010, 158). Ohjelmallisten komponenttien liittymistä toisiinsa on mielekästä mallintaa erillisten connector-mallinnuselementtien avulla. Tästä elementtien kerroksesta käytetään myös nimitystä middleware. Näihin elementteihin voi liittyä arkkitehtuurin kannalta merkittäviä ominaisuuksia ja toiminnallisuuksia, kuten mm. sanomanvälitystä, olioiden pysyvyyttä (persistence) ja transaktioita.

3.2.1. Arkkitehtuurin merkitys ohjelmistotuotteelle

Arkkitehtuurin tärkeys tulee esiin ainakin seuraavin tavoin (Bass, Clements, Kazman 2003, 26):

1. Sidosryhmien kommunikointi: arkkitehtuuri kuvaa järjestelmän yleisesti tunnettujen abstraktioiden avulla;
2. Alkuvaiheen suunnitteluvalinta: arkkitehtuuriin liittyvät keskeiset valinnat joudutaan tekemään suunniteluvaiheen alussa, ja ne suuressa määrin sitovat ohjelmiston muuta kehitystyötä, käyttöönottoa ja jatkokehitystä. Arkkitehtuurin valinta antaa myös mahdollisuuden tehtyjen suunnittelupäätösten yhteiseen arviointiin.
3. Hyödynnettävissä oleva abstraktio: ohjelmiston arkkitehtuurin kuvaus toimii mallina muille tuleville ohjelmistoille.

Edelleen ohjelmistotuotannon ns. ketterissä (agile) menetelmissä korostetaan aikaista arkkitehtuurivalintaa. Esimerkiksi XP-menetelmässä (eXtreme Programming) koko ohjelmiston tuotantoprosessin lähtöpisteinä ovat rinnakkain "User stories" ja "Architectural spike" ("arkkitehtuurihahmotelma") (Extreme Programming 2012).

Arkkitehtuurin valintaperusteista

Tärkeimmäksi arkkitehtuurin valintaperusteeksi nostetaan yleensä ohjelmiston vaatimuksista johdetut laadulliset ominaisuudet (mm. Bass, Clements, Katzman 2003). Myös jäljempänä esiteltävässä ja sovellettavassa CBSP-vaatimustenhallintamenetelmässä järjestelmän vaatimuksista johdetut laadulliset attribuutit ovat keskeisessä asemassa johdettaessa järjestelmän vaatimusten ja arkkitehtuurin välistä yhteyttä (Grünbacher, Egyed, Medvidovic 2004) .

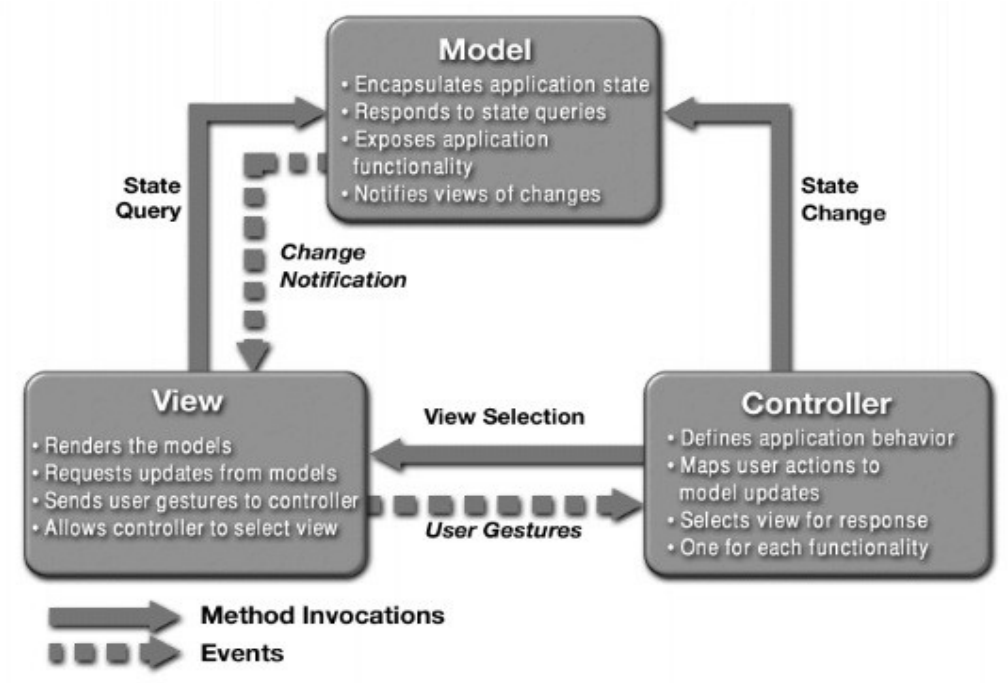
Ihannetapaus olisi se, että koko ohjelmisto voisi nojautua yhteen arkkitehtoniseen ratkaisuun. Tämä ei käytännössä ole useinkaan mahdollista (Koskimies, Mikkonen 2005, 39). Arkkitehtuuri pilkkoutuu osiin (architecture/design fragment). Tähän on ainakin seuraavia syitä. Ohjelmistotuotannossa sovelletaan yleisesti ns. suunnittelumalleja (design pattern), jotka tarjoavat vakiomuotoisen tavan suunnitella yksittäisen komponentin tai toiminnon rakenne (mm. Larman 1998, 189). Edelleen ohjelmistotuotannossa käytetään paljon ns. sovelluskehyksiä (application framework), joiden avulla ohjelmistojen tuotantoa ja ylläpitoa tehostetaan ja yhdenmukaistetaan. Sovelluskehykset ovat usein varsin laajoja ja monimutkaisia kokonaisuuksia, jotka samalla tarjoavat kehitettäville sovelluksille valmiin arkkitehtuurin (Koskimies, Mikkonen 2005, 187).

3.2.2. MVC-arkkitehtuuri

Yleinen esimerkki arkkitehtuurista, joka soveltuu laajasti käytettäväksi mm. kaupallis-hallinnollisessa ohjelmistotuotannossa, on MVC (Model-View-Controller) (kuvio 1). Siinä sovelluksen rakenne pohjautuu sovelluslogiikan (Controller), käyttoliittymän (View) ja tietomallin (Model) erottamiseen toisistaan (Wikipedia: Model-View-Controller 2012). Käyttöliittymä (V) vastaanottaa käyttäjän syötteen, controller (C) muuttaa tapahtuman mallin (M) ymmärtämään muotoon ja mahdollisesti muuttaa mallin tilaa. MVC-arkkitehtuuri on itsessään varsin yleisluontoinen sovelluksen kokonaisarkkitehtuuri. Esimerkiksi Controller-tehtäviä suorittavat ohjelmiston osat voivat muodostaa oman palveluperustaiseen arkkitehtuuriin (SOA) pohjautuvan kokonaisuuden. Toisaalta

johonkin suureen tietojärjestelmään liittyvä yksittäinen, käyttöliittymän sisältävä tietopalvelu voi perustua paikalliseen MVC-pohjaiseen toteutukseen.

Ns. kolmikerrosarkkitehtuurilla (3-tier architecture) on vanhastaan viitattu rakenteeseen, jossa järjestelmä jaetaan käyttöliittymään, sovelluslogiikaan ja tietomalliin. Kolmikerrosarkkitehtuuri viittaa MVC-mallia korostetummin näiden kolmen komponentin fyysiseen erillisyyteen, eikä se mm. salli suoria viittauksia käyttöliittymä- ja tietomallikomponentin välillä toisin kuin MVC (Wikipedia: Multitier architecture 2012).



KUVIO 1: MVC-malli (Oracle Corp. 2012)

Monet sovelluskehykset (application framework) pohjautuvat MVC-arkkitehtuuriin. Nämä sovelluskehykset tarjoavat valmiita pohjia toiminnoille ja rakenteille, joita esiintyy sovelluksissa toistuvasti. Sovelluskehykset ovat ohjelmointikieli ja -ympäristökohtaisia. Niitä ovat esim. Zend PHP-ohjelmointikielelle, Django Python-kielelle sekä Java Server Faces, joka hyödyntää Java EE-teknologiaa.

3.2.3. Asiakas-palvelin -arkkitehtuuri

Asiakas-palvelin (client-server) -arkkitehtuurin perusajatus on paketoida palvelin eli tietyn resurssin hallinta siten, että resurssin käyttäjien eli asiakkaiden ei tarvitse huolehtia reurssin käyttöön liittyvistä teknisistä yksityiskohdista, kuten yhteiskäyttöön liittyvistä ongelmista (Koskimies, Mikkonen 2005, 136). Yleinen tapa järjestää toiminta on sellainen, jossa palvelin odottaa passiivisena asiakkaan yhteydenottoa. Yhteydenotto avaa istunnon (session), jonka kuluessa palvelin tuottaa asiakkaalle jonkin palvelukokonaisuuden. Tämän jälkeen istunto yleensä sulkeutuu.

Asiakas-palvelin -arkkitehtuurilla haetaan usein mahdollisuutta järjestelmän tehokkaaseen hajauttamiseen. Asiakas- ja palvelinohjelmistot voivat yhteistä kutsurajapintaa lukuunottamatta olla täysin erillisiä ajoympäristön (prosessorin ja käyttöjärjestelmän) suhteen. Nykyisin asiakas- ja palvelinkomponenttien välinen yhteys järjestetään pääsääntöisesti jotain IP-pohjaista tietoliikenneprotokollaa käyttäen.

3.2.4. Kerrosarkkitehtuurit

Kerrosarkkitehtuurissa ohjelmalliset komponentit on ryhmitelty palvelukerroksiksi siten, että suoraan liiketoimintalogiikkaa palveleva (esim. käyttöliittymä-) ohjelmakomponentti edustaa ylintä ja abstraktiotasoltaan korkeinta kerrosta. Muut kerrokset tarjoavat järjestelmän käyttöön primitiivejä, jotka mahdollistavat edustasovellusten toteuttamisen ilmaisuvoimaisesti ja alustariippumattomasti (Taylor, Medvidovic, Dashovy 2010, 106). Yleensä edellytetään, että kukin kerros voi pyytää palveluja vain välittömästi alapuolellaan olvasta kerroksesta.

Yleinen kerrosarkkitehtuurin ilmentymä on virtuaalikone (virtual machine) (Taylor, Medvidovic, Dashovy 2010, 105). Virtuaalikoneen tehtävänä on toteuttaa jokin palveluprimitiivien joukko, jonka avulla ylemmän kerroksen ohjelmat voivat toteuttaa käyttäjän tai vielä ylemmän palvelukerroksen antamia tehtäviä. Esimerkiksi Java-virtuaalikone (JVM) suorittaa Java-kielisiä ohjelmia piilottaen näiltä laitteistoon ja käyttöjärjestelmään liittyvät yksityiskohdat.

3.2.5. Sanomapohjaiset arkkitehtuurit

Sanomapohjainen arkkitehtuuri soveltuu järjestelmään, johon tiedetään tulevan mukaan joukko keskenään kommunikoivia komponentteja, mutta niiden määrästä ja laadusta ei ole tarkkaa tietoa (Koskimies, Mikkonen 2005, 136). Toiminta järjestetään niin, että komponentit kommunikoivat jonkin yhteisen välitysmekanismin, väylän (bus) kautta. Sanomanvälitysjärjestelmä voi pitää kirjaa väylään kytkeytyvistä komponenteista, niiden tarjoamista palveluista ja sanomaprotokollista. Komponentteja ei ole tarpeellista jakaa asiakas/palveluntuottajarooleihin, kuten client-server -arkkitehtuurissa. Palvelupyynnöt välittyvät väylän kautta joko suoraan ohjautuen tietylle komponentille, tai ns. broadcast-periaatteella, jolloin kaikki komponentit saavat tiedot sanomasta, mutta vain asiaankuuluva komponentti reagoi siihen) (Taylor, Medvidovic, Dashovy 2010, 120). Jälkimmäisestä menettelyä kutsutaan usein myös tapahtumapohjaiseksi (event-based) arkkitehtuuriksi. Kirjallisuudessa käytetään kuitenkin ristiin nimityksiä sanomapohjainen tai tapahtumapohjainen.

3.2.6. Pipe-and-filter

Pipe-and-filter -arkkitehtuuri soveltuu tilanteisiin, joissa ohjelmiston tehtävänä on käsitellä syötetietoja ja reagoida niihin pääasiassa ilman tarvetta takaisinkytkentään tai käyttäjän ohjaukseen käsittelyn aikana (Taylor, Medvidovic, Dashovy 2010, 111). Ohjelmalliset komponentit muodostavat ketjun, jossa syötevirta jalostetaan asteittain halutuksi tulostiedoksi tai toiminnaksi. Komponenttien välistä kommunikointikanavaa kutsutaan putkeksi (pipe). Komponentit voivat olla toiminnassa jatkuvasti ja käsitellä dataa sitä mukaa, kun edellinen komponentti saa sitä luovutettua eteenpäin.

3.2.7. C2-arkkitehtuuri

C2 -arkkitehtuuri on edellä mainittuja arkkitehtuurityylejä monimutkaisempi ja täsmällisemmin kuvattu. Päällisin puolin se yhdistää sanomapohjaisen- ja kerrosarkkitehtuu-

rin. C2-arkkitehtuuri soveltuu järjestelmiin, joissa ajoympäristö on heterogeeninen ja hajautettu. Uusien komponenttien liittäminen järjestelmään on helppoa. Arkkitehtuuri mahdollistaa samalla myös MVC-logiikan kurinalaisen toteuttamisen (Taylor, Medvidovic, Dashovy 2010, 126).

C2-arkkitehtuurissa komponentit ryhmitellään kerroksiin. Kunkin kerroksen komponentit ovat tietoisia vain välittömästi ylä- tai alapuolellaan olevasta kerroksesta. Komponenttien välinen suora kommunikointi ei ole sallittua, vaan kaikki yhteydet tapahtuvat sanomapohjaisesti connector-elementtien kautta. Komponenttien rajapinnat määritellään rajapintoina ylä- ja alapuoliseen connector-elementtiin. Sanomat ovat joko palvelupyyntöjä (request) tai ilmoituksia (notifications).

3.2.8. SOA-arkkitehtuuri

Palveluorientoitunut- eli SOA-arkkitehtuuri (Service-Oriented Architecture) on tekemässä voimakkaasti tuloaan (2012). Siinä järjestelmät muodostetaan hajautetusti verkossa olevista, toisilleen palveluja tarjoavista komponenteista. Tärkeässä roolissa ovat ns. middleware-komponentit, jotka toteuttavat palvelujen yhdistämisen kokonaisuudeksi sekä palvelupyyntöjen välityksen (Gorton 2011, 65).

SOA-toteutuksiin liittyviä asioita ovat mm. palvelujen "orkestraation" järjestäminen, millä tarkoitetaan palvelujen yhteenkytkennän topologiaa. Vaihtoehtoja ovat keskitetyn palveluhakemiston ja palvelupyyntöjen välitysohjelmiston (broker) käyttö, tai palvelun-tarjoajien kytkeytyminen renkaanksi ilman varsinaista keskusta. Edelleen yleinen toteutusväline SOA-arkkitehtuurissa on SOAP-protokolla (Simple Object Access Protocol), joka mm. määrittelee palvelupyynnöille XML-pohjaisen sanomarakenteen, sekä tarjolle asetettavien palvelujen kuvauskielen.

3.2.9. Arkkitehtuurin liittyviä käsitteitä

Termiä "ohjelmistoarkkitehtuuri" ei ole täsmällisesti määritelty, mutta edellä luetellut arkkitehtuurit on sellaisiksi tunnistettu useissa lähteissä. Ohjelmistoarkkitehtuureihin

liittyy joitakin käsitteitä ja teknologioita, jotka on syytä mainita tässä yhteydessä:

- Oliopohjaisuus, oliomallinnus. Oliopohjaisuutta ei itsessään yleensä luokitella ohhjelmistoarkkitehtuuriksi. Oliopohjaisuus on ohjelmallisten komponenttien toteutustapa, jota sovelletaan hyvin erikokoisissa mittakaavoissa. Siinä ajatellaan, että tiettyyn kohteeseen liittyvät tiedot ja toiminnot tulee paketoida yhdeksi kokonaisuudeksi, jossa kohteen tietoja voi käsitellä vain siihen liittyvien toimintojen kautta. Kun olioajattelua sovelletaan suuressa mittakaavassa kohdealueen liiketoimintaan, voidaan puhua hajautettujen olioiden ohjelmistoarkkitehtuurista (distributed objects) (Taylor, Medvidovic, Dashovy 2010, 132). CORBA (Common Object Request Broker Architecture) on OMG:n standardi, joka kuvaa miten verkkoon hajautettujen olioipohjaisten komponenttien yhteistoiminta järjestetään. Standardin ensimmäinen versio julkaisitiin jo vuonna 1991. Java Enterprise Edition (Java EE) on alun perin Sun Microsystems:n kehittämä, sittemmin Oracle:n hallintaan siirtynyt oliopohjainen teknologia, joka on nykyisin (2012) merkittävä asema yritystason hajautettujen tietojärjestelmien toteutuksessa. Teknologia kattaa paitsi itse Java-ohjelmointikieleen kuuluvat laajennukset ja luokkakirjastot, myös määritykset ohjelmistojen ajoympäristölle. Ajoympäristöä toteuttavat palvelimet mahdollistavat palvelinpohjaisten web-sovellusten suorittamisen (Java servelet/JSP -tekniikat) sekä toimivat EJB (Enterprise Java Bean) -komponenttien ajoalustoina. Java EE sisältää välineet mm. web services -tyyppisten palvelujen toteuttamiseen, mitä voi käyttää SOA (palvelukeskeisen) -arkkitehtuurin toteutuksessa; toisaalta EJB-komponenttityyppeihin sisältyvät sanomapohjaiset komponentit (message-driven beans) tukevat sanomapohjaista arkkitehtuuria. Microsoftin .NET -arkkitehtuuri ja -teknologia on suora vastine Java EE:lle ja kilpailee suosiosta sen kanssa.
- Middleware. Kun tietojärjestelmistä tulee yhä hajautetumpia, komponenttien yhteistoiminnasta vastaavat ohjelmistot nousevat yhä merkittävämpään asemaan. Middleware-toiminnallisuutta voivat toteuttaa erilliset ohjelmat, tai esim. sovelluspalvelimet muun toiminnan ohessa. Middleware-tyyppejä ovat mm. (Gorton 2011, 40) erilaiset sanomanvälitysjärjestelmät (MOM = message-oriented middleware; message brokers = ohjelmistot jotka mahdollistavat hajautettujen oliopohjaisten komponenttien kommunikoinnin), web service/SOAP -palvelin-

ohjelmistot, sekä ylimmällä tasolla liiketoimintaprosessien integraatio-ohjelmistot (ochestrators).

- Design patterns eli suunnittelumallit tarjoavat valmiin ratkaisutavan johonkin usein toistuvaan, selkeästi tunnistettavaan ja rajattavaan ongelmaan. Suunnittelumalleja esiintyy erilaisissa mittavakaavoissa ohjelmistojen kokonaisarkkitehtuurista yksittäisen ohjelmakomponentin rakenteeseen. Suunnittelumalli ei ota kantaa välineisiin, joilla ratkaisu toteutetaan (Koskimies, Mikkonen 2005, 136). Esimerkiksi edellä kuvatusta MVC-arkkitehtuurista saatetaan käyttää nimitystä suunnittelumalli (Bass, Clements, Kazman 2003, 127). Yksittäisten ohjelmallisten komponenttien suunnitteluun on saatavilla valmiita UML-kuvauksin esitettyjä suunnittelumalleja (mm. Larman 1998, 189) .

3.3. AnyCase-järjestelmän vaatimukset

AnyCase-palvelu sisältää kaksi päätoimintoa: asianhallinnan työnkulkujen sovelluskehittimen, sekä ympäristön näiden työnkulkujen suorittamiseksi. Järjestelmän innovatiivisin osa on sisäinen malli, joka mahdollistaa asianhallinnan työnkulkujen kuvaamisen yksinkertaisella, mutta samalla riittävän tehtävänläheisellä tavalla. Järjestelmään kohdistuvat vaatimukset voi jakaa neljään ryhmään:

1. asiakkaan vaatimukset, jotka kohdistuvat suoritettaviin työnkulkuihin
2. asiakkaan vaatimukset, jotka kohdistuvat työnkulkumallien kuvaamiseen
3. järjestelmään kohdistuvat sisäiset vaatimukset työnkulkujen kuvaus- ja suoritusmallille; nämä johdetaan asiakkaan vaatimuksista 1 ja 2
4. SaaS-jakelumallista seuraavat liiketoiminnalliset vaatimukset

Kohdan 3 vaatimukset ovat asiakkaan vaatimusten yleistyksiä. Näiden vaatimusten perusteella luodaan järjestelmän sisäinen rakenne- ja toimintamalli.

Kohdan 4 vaatimusten esittäjänä on yritys/organisaatio, joka hallinnoi AnyCase-järjestelmää. Perusvaatimus on, että järjestelmä mahdollistaa usean asiakasorganisaation palvelemisen yhtäaikaa ja toisistaan riippumatta niin, että yksittäinen asiakasorganisaatio ei ole tietoinen muista käyttäjistä. Tämän vaatimuksen toteuttamiseen liittyy paitsi

rakenteellisia ja toiminnallisia , myös kapasiteettiin ja tietoturvallisuuteen liittyviä kysymyksiä.

AnyCase-palvelun kehittämistä varten on pilottiasiakkaalta kerätty lähinnä 1-kohdan vaatimuksia. Valistunut pilottiasiakas on osannut kuitenkin lähestyä asiaa osittain myös 2-kohdan kannalta tekemällä omiin liiketoimintaprosesseihinsa liittyviä yleistyksiä liittyen mahdollisiin tarpeisiin tulevaisuudessa, tai erilaisissa poikkeustilanteissa. Pilottiasiakkaan kanssa asetetut vaatimukset on listattu liitteessä 1.

2-kohdan vaatimukset on pitänyt enimmäkseen johtaa niistä yleisistä tarpeista, joita 1-kohdan vaatimukset ilmentävät; toisin sanoen 2-kohdan vaatimukset (työnkulkumallien kuvaamiseen liittyvät vaatimukset) ovat aidoista asiakasvaatimuksista johdettuja yleistyksiä. Niitä päästään arvioimaan kunnolla vasta, kun ohjelmiston prototyyppiä testataan useilla pilottiasiakkailla. Vasta tässä vaiheessa asiakkaan on mahdollista kunnolla ymmärtää, mistä työnkulkumallien kuvaamisessa on käytännössä kysymys, ja tehdä siihen liittyviä parannusehdotuksia. Vaikka asiakas onkin vain päällisin puolin tietoinen siitä sisäisestä mallista, johon työnkulkujen kuvaukset perustuvat, asiakaspalautetta tarvitaan työnkulkujen kuvaamiseessa käytettävän käyttöliittymän kehittämiseksi.

3.4. Vaatimusten ja arkkitehtuurin välinen yhteys / AnyCase

Yhteyttä ohjelmiston arkkitehtuurin ja vaatimustenhallinnan välillä ei ole laajasti tutkittu (Grünbacher ym. 2004). Tämä johtunee siitä, että valtaosa ohjelmistoista suunitellaan ja toteutetaan olemassaolevien mallien ja alustojen avulla palvelemaan suoraan asiakkaan määrittelemää tarkoitusta, jolloin vaatimusten ja arkkitehtuurin välinen yhteys on pääasiassa triviaali. Tilanne on kuitenkin toinen, kun kehitteillä oleva ohjelmisto on aidosti innovatiivinen, tai jos se on tyypiltään sovelluskehitin, kuten AnyCase:n tapauksessa. Sovelluskehittimellä tarkoitetaan ohjelmistoa, jonka avulla voi tuottaa periaatteessa rajattoman määrän erilaisia sovelluksia tietylle sovellusalueelle. Sovelluskehitin perustuu siten näkemykseen siitä, mitkä kaikki asiakasvaatimukset ovat mahdollisia. Tietyn sovellusalueen sisällä tiedetään, että mahdollisten yksittäisten asiakasvaatimusten joukko on äärellinen, mutta niistä on voitava tuottaa rajattomasti kombinaatioita. Silloin asiakkaan välittömät vaatimukset kohdistuvat vain kehittimellä tuotet-

taviin sovelluksiin. Asiakasta varmasti kiinnostaa vastaisuuden varalta myös, mitä kaikkea muuta toiminnallisuutta sovelluskehittimen avulla on mahdollista toteuttaa.

AnyCase-ohjelmiston perusajatus on se, että suurin osa asianhallinnan työnkuluista on kuvattavissa seuraavanlaisten yleisten vaatimusten kautta:

- asiaan voi liittyä useita asiakirjoja
- asiaan voi liittyä useita toimijoita omasta organisaatiosta tai sen ulkopuolelta
- asian työnkulku voidaan kuvata joukkona tehtäviä, joiden suoritusjärjestyksen voi kuvata esim. vuokaaviona
- yksittäinen tehtävä työnkulussa voi tarkoittaa seuraavia asioita: asiakirjan tai asiakirjojen luominen, täydentäminen, hyväksyminen, kommentoiminen, tiedoksi saattaminen jne. ja/tai käsittelyä ohjaava päätös.

Ajatuksen käyttökelpoisuutta on mahdollista testata vertaamalla todellisia, sovelluskohtaisia asiakasvaatimuksia mallin tarjoamiin mahdollisuuksiin. Ratkaiseva kysymys on, voidaanko asiakkaan vaatimuksia yleistää sopivasti niin, että ratkaisu on yksittäiselle asiakkaalle edelleen riittävä, samalla kun valitut yleistykset auttavat tekemään järjestelmästä yleiskäyttöisen.

3.5. CBSP-menetelmä

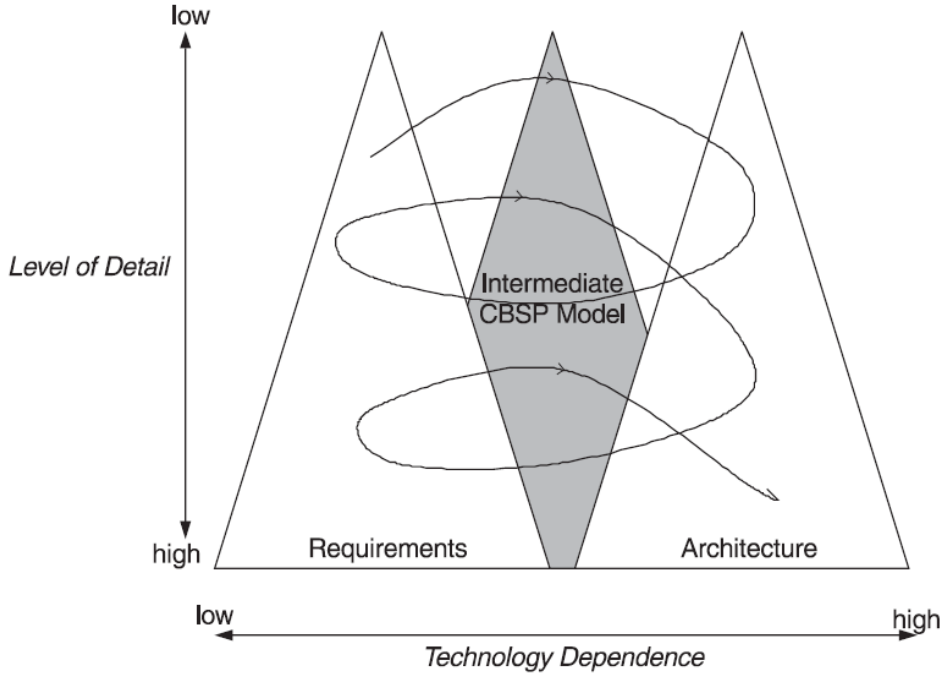
Grünbacher ym. (2004) esittelevät artikkelissa "Reconciling software requirements and architectures with intermediate models" periaatteen ja menetelmän, jolla voidaan johtaa vaatimusten ja arkkitehtuurin välinen yhteys. Menetelmään on viitattu mm. erilaisissa vaatimusten hallintaa käsittelevissä konferensseissa, kuten IEEE Requirements Engineering 2002-2006, ICSE '03, lisätietoa ks. http://www.icse-conferences.org/2003/ (luettu 15.11.2012).

CBSP-menetelmän (Component-Bus-System-Property) tarkoituksena on auttaa vaatimusten pohjalta valitsemaan ohjelmiston arkkitehtuuri. Tämä tapahtuu mallintamalla iteratiivisesti vaatimusten ja arkkitehtuurin välisiä yhteyksiä. Kuvio 2 esittää, miten arkkitehtuuria ja vaatimuksia välittävä CBSP-malli sijoittuu suhteessa yksityiskohtien mää-

rään ja tarkastelukulman teknologisuuteen. Mallin avulla päästään johtamaan vaatimusten ja arkkitehtuurin välistä yhteyttä suunnitteluprosessin alusta asti.

CBSP-Menetelmän perusajatus on se, että mikä tahansa ohjelmistovaatimus voi suorasti tai epäsuorasti antaa tietoa ohjelmiston arkkitehtuurista. Mallin laatijat toteavat mm. seuraavat asiat ongelmallisiksi haettaessa vaatimusten ja arkkitehtuurin välistä yhteyttä:

- Vaatimusten ilmaisutapa on usein vapaamuotoinen, arkkitehtuurin taas muodollinen, mistä syystä merkitystason yhteyksiä näiden välillä voi olla vaikea tunnistaa.
- Ei-toiminnallisten vaatimusten yhteys arkkitehtuuriin ei monesti ole ilmeinen.
- Vaatimusten ja arkkitehtuurin yhtäaikainen iteratiivinen työstäminen merkitsee, että arkkitehtuurivalintoja joudutaan aluksi tekemään epätäydellisten vaatimusten pohjalta.
- Yksilöityjen vaatimusten yhteys tiettyihin arkkitehtuurivalintoihin ei ole yksinkertainen asia, sillä tietyllä vaatimuksella voi olla yhteys useisiin arkkitehtuurin osiin ja päinvastoin. Siirtyminen komponenttipohjaiseen ohjelmistotuotantoon on korostanut tarvetta hallita joustavasti tätä asiaa.



KUVIO 2: CBSP-konteksti (Grünbacher ym. 2004, 236)

Menetelmän keskeiset osat ovat

- taksonomia, jonka mukaan vaatimukset luokitellaan ja josta on yhteys arkkitehtuuriin, sekä
- prosessi, jonka avulla johdetaan vaatimuksia ja arkkitehtuuria yhdistävä CBSP-malli ("Intermediate CBSP Model")

Menetelmä tarjoaa mm.

- kevyen tavan vaatimusmäärittelyjen muokkaamiseksi ja tarkentamiseksi yksinkertaisessa arkkitehtuurilähtöisessä viitekehyksessä;
- tavan valikoida arkkitehtuurin kannalta merkitykselliset vaatimukset;
- tavan osoittaa tehtyjen suunnittelupäätösten perusteet sidosryhmille;
- käytännön sovittaa yhteen ristiriitaisia tulkintoja painotuskertoimien avulla

3.5.1. CBSP-taksonomia

Lyhenne "CBSP" tulee sanoista component, bus, system, properties. Tärkeä osa CBSP-menetelmää on vaatimusten luokittelu arkkitehtuurin kannalta merkittävien piirteiden mukaan. Luokittelun yhteydessä vaatimus muotoillaan uudelleen valitun luokittelukriteerin näkökulmasta. Uudelleen muotoileminen voi merkitä vaatimuksen tarkentumista (refinement) tai yleistystä (generalization). Alkuperäisestä vaatimuksesta voi seurata useita CBSP-luokiteltuja vaatimuksia. Toisaalta etenkin P)roperties -luokan uudelleenmuotoillut vaatimukset voivat olla yleispäteviä niin, että ne kuvaavat kerralla jotain järjestelmän ominaisuutta, joka ilmenee eri tavoin useissa yksittäisissä lähtövaatimuksissa.

TAULU 1. CBSP-taksonomia (esimerkit kursivoitu) (Grünbacher ym. 2004, 238):

C	Component: malliin sisältyvä komponentti, joka voi olla edelleen joko ohjelmallinen komponentti (C_P) tai datakomponentti (C_D) *Vaatimus: käyttäjän on voitava suoraan muokata laskentataulukon dataa* *C_P: laskentataulukon UI-komponentti* *C_D: laskentataulukon data*
B	Bus: malliin sisältyvä asioita yhdistävä/välittävä elementti (connector) *Vaatimus: Muokatun laskentataulukon data on voitava tallettaa tiedostoksi.* *B: Liitännäinen joka yhdistää käyttöliittymäkomponentin ja tietovaraston*

S	System: malliin sisältyvä ominaisuus, joka liittyy tai vaikuttaa koko malliin tai merkittävään osajoukkoon mallin C- ja B-elementtejä. *Vaatimus: Käyttäjän on voitava valita sopivat datasuotimet ja visualisointitavat.* *S: Järjestelmän tietovaraston, datan prosessoinnin ja datan visualisoinnin tulee olla selkeästi eroteltu toisistaan.*
CP	Component Properties: komponentin ominaisuudet, tässä yhteydessä tarkoitetaan yleisiä ominaisuuksia kuten luotettavuus, siirrettävyys, laajennettavuus ym. *Vaatimus: käyttäjän tulee voida esittää visuaalista dataa etäkäyttöisesti ilman tarpeetonta viivettä* *CP: Visualisointikomponentin tulee olla tehokas ja tukea inkerementoivaa päivitystä*
BP	Bus Properties: yhteyselementin ominaisuudet *Vaatimus: Toiminnon lisääminen järjestelmään tulee tapahtua niin, että alhaallaoloaika minimoidaan* *BS: Järjestelmän tulee toteuttaa komponenttien liittäminen ja poistaminen ajon aikana.*
SP	System Properties: koko järjestelmään liittyvät yleiset ominaisuudet *Vaatimus: Laskentatulukon data tulee kryptata verkossa siirtoa varten* *SP: Järjestelmän tulee olla tietoturvallinen.*

3.5.2. CBSP-prosessi

Prosessi on iteratiivinen ja se käsittää viisi vaihetta (Grünbacher ym. 2004, 238-244). Iterointikierrosten määrää ei kiinnitetä, sitä jatketaan kunnes informaatio on saatu jalostettua niin pitkälle, että tuloksia voi pitää merkityksellisinä. Kukin vaihe jäsennetään vielä osatehtäviin aloitus, tehtävä, verifiointi, päätös (ETVX = Entry, Task,Verification, eXit).

Vaihe 1: Vaatimusten valinta iterointikierrosta varten. Tekijät edustavat kaikkia sidosryhmiä. Arkkitehtuurin kannalta vähemmän tärkeät tai muutoin analyysiin soveltumattomat vaatimukset suodatetaan pois. Jäljelle jäävistä vaatimuksista todetaan niiden 1)

merkittävyys projektin onnistumisen kannalta ja 2) soveltuvuus, mikä tässä yhteydessä tarkoittaa vaatimuksen toteuttamiseen liittyviä reunaehtoja (tekniset, taloudelliset ym.)

Vaiheen 1 osatehtävät ovat:

- aloitus: saatavilla tulee olla vaatimusmäärittelyt ja sovittuna priorisointi- ja arvostusmenetelmät
- tehtävä: sidosryhmien edustajat karsivat vähemmän tärkeät tai huonosti soveltuvat vaatimukset jatkosta
- verifiointi: tarkistetaan valinnat sekä todetaan valintojen yksimielisyys tai erimielisyys sidosryhmien kesken; mielipiteiden hajotessa valmistaudutaan keskustelemaan aiheesta.
- päätös: valittujen vaatimusten joukko seuraavaa vaihetta varten

Vaihe 2: Vaatimusten arkkitehtuuriperustainen luokittelu. Tekijät: ohjelmistosuunnittelijat (arkkitehdit). Vaatimukset luokitellaan CBSP-taksonomian mukaan. Kullkein vaatimukselle luodaan profiili, joka kertoo sen yhteydestä arkkitehtuuriin käyttäen yhtä tai useampaa luokitteluperustetta C,B,S,CP,BP,SP sekä näihin liitettyä merkitsevyyspisteytystä relevanssin mukaan 0..3 (0-ei lainkaan, 3-täysin). Jos ohjelmistosuunnittelijoita on useita, profiiliin merkitään pisteytysten keskiarvo.

Vaiheen 2 osatehtävät ovat:

- aloitus: saatavilla tulee olla 1-vaiheessa valitut vaatimukset, sekä arvostusmenetelmä sovittuna
- tehtävä: ohjelmistosuunnittelijat toteuttavat vaatimusten CBSP-luokittelun
- verifiointi: tarkistetaan luokittelun kattavuus
- päätös: arkkitehtoninen luokitteluprofiili kaikille vaatimuksille, sekä mahdolliset äänestystulokset

Vaihe 3: Luokitteluristiriitojen tunnistaminen ja ratkaisu. Eri ohjelmistosuunnittelijat voivat tehdä toisitaan poikkeavia luokituspäätöksiä vaiheessa 2. Ristiriitaisten valintojen perusteiden tiedostaminen on vaatimusanalyysin ja jatkokehityksen kannalta tär-

keää. Vaiheen sisältönä on näiden ristiriitojen ja moniselitteisyyksien sovittelu ja konseuksen saavuttaminen.

Vaiheen 3 osatehtävät ovat:

- aloitus: saatavilla tulee olla vaiheessa 2 tuotetut vaatimusten luokitteluprofiilit, sekä mahdolliset äänestystulokset
- tehtävä: suunnittelijat keskustelevat mielipide-erojen syistä; suunnittelijat päivittävät vaatimuksia; suunnittelijat karsivat arkkitehtuurin kannalta epäoleelliset vaatimukset
- verifionti: varmistetaan ettei keskeisiä vaatimuksia hylyytetty
- päätös: todetut ristiriidat ja moniselitteisyydet; arkkitehtuurin kannalta relevantit vaatimukset

Vaihe 4: Vaatimusten muokkaaminen arkkitehtuurin näkökulmasta. Suunnittelijat pilkkovat ja muotoilevat uudelleen vaatimukset, joiden profiiliin sisältyy useampia CBSP--kategorioita. Tästä voi helposti seurata, että jokin järjestelmään sisältyvä asia tai kohdetyyppi toistuu useissa vaatimuksissa. Tästä pyritään pääsemään eroon eri tavoin, kuten tunnistamalla ensin järjestelmästä vain rakenteelliset piirteet, jotka kuuluvat kategorioihin C,B,S. Sen jälkeen niiden ominaisuudet todetaan erikseen alakohtina. Toinen lähestymistapa on tunnistaa samanaikaisesti järjestelmän rakenneosa ja sen ei-toiminnalliset ominaisuudet. Tapoja voi käyttää rinnakkain. Vaiheen lopputuloksena vaatimuksista on kehitetty joukko pelkistettyjä CBSP-elementtejä ja näiden välisiä riippuvuuksia.

Vaihe 4:n osatehtävät ovat:

- aloitus: käytettävissä vaiheen 3 aikana dokumentoidut ristiriidat ja moniselitteisyydet; sekä arkkitehtuurin kannalta relevantit vaatimukset
- tehtävä: suunnittelijat osittavat ja muotoilevat uudelleen vaatimukset, jotka ulottuvat useaan CBSP-kategoriaan. Redundantit vaatimukset eliminoidaan.
- verifionti: onko redundanssi minimoitu?
- päätös: CBSP-elementit ja näiden keskinäiset riippuvuudet

Vaihe 5: arkkitehtonisten elementtien ja tyylien painottaminen CBSP:n mukaan

Tässä vaiheessa vaatimuksien pohjalta johdettujen CBSP-mallin elementtien tulisi olla sellaisia, että niistä on eliminoitu sidosryhmien väliset näkemyserot, ja että jokainen elementti on voitu selkeästi sijoittaa johonkin kuudesta CBSP-taksonomian luokasta. Tältä pohjalta päästään käsiksi arkkitehtuurihahmotelmaan.

CBSP-menetelmään sisältyvää viittä arkkitehtuurityyliä analysoidaan tarkemmin jäljempänä. Vaiheen 5 tavoitteena on CBSP-mallin elementtien ja näiden keskinäisten riippuvuuksien perusteella johtaa yhteys sopivimpaan arkkitehtuurityyliin. Tyyliehdokkaita voi löytyä useita. Arkkitehtuurityylin valinta perustuu paitsi CBSP-elementteihin sinänsä, näiden elementtien ominaisuuksiin (properties). Arkkitehtuurityylin valintaa ohjaa taulukko 2, josta ilmenevät arkkitehtuurityylit ja niihin keskeisesti liityvät ominaisuudet painokertoimineen. Painokerroin kertoo tietystä ominaisuudesta, miten relevantti se on kullekin arkkitehtuurityylille. Arkkitehtuurityylin valinta tapahtuu kuitenkin pääasiassa laadullisin, sanallisin perustein.

Vaihe 5:n osatehtävät ovat:

- aloitus: käytettävissä vaiheen 4 aikana tuotetut CBSP-elementit ja näiden keskinäiset riippuvuudet
- tehtävä: suunnittelijat tunnistavat alaustavasti arkkitehtuurityylin/tyylit; suunnittelijat tunnistavat alustavasti arkkitehtoniset elementit; analysoidaan CBSP-elementtien ominaisuudet (P=properties)
- verifionti: sovittamalla CBSP-elementit ja arkkitehtoniset elementit yhteen varmistetaan että ratkaisu on käyttökelpoinen.
- päätös: ehdokkaat arkkitehtuurityyleiksi; ehdokkaat komponenteiksi; ehdokkaat liitoksiksi (connectors)

3.5.3. CBSP-menetelmään sisältyvät arkkitehtuurityylit

Menetelmään valmiiksi sisältyvät arkkitehtuurityylit on lueteltu sarakkeissa taulussa 2. Dimensions- ja properties -sarakkeissa oleva informaatio on keseisessä asemassa: se toimii linkkinä vaatimusten ja arkkitehtuurin välillä luettelemalla ominaisuuksia, jotka voidaan toisaalta yhdistää vaatimuksista johdettuun CBSP-elementtiin (vaiheet 1-4) ja toisaalta tietyyn arkkitehtuurityyliin. Painokeroimet ++,+,0,- ilmaisevat, missä määrin kukin arkkitehtuuri tukee kutakin ominaisuutta.

TAULU 2. CBSP-menetelmään sisältyvät arkkitehtuuritylit (Grünbacher ym. 2004, 244)

CBSP Dimensions	*Properties*	*Client-Server*	*C2*	*Event-Based*	*Layered*	*Pipe-and-Filter*
Data Component	aggregated	++	++	++	+	−
	persistent	++	o	o	o	o
	streamed	−	−	−	−	++
	cached	++	+	−	−	−
Processing Component	service provide/consume only	++	o	o	o	o
	has N interfaces	++	+	++	−	−
	stateful	+	++	++	+	−
	Loose coupling	+	+	++	−	++
	can be migrated	+	++	++	−	−
Connector/bus	synchronous	++	−	+	++	−
	asynchronous	−	++	++	−	++
	local	−	++	o	++	+
	distributed	++	++	++	−	+
	secure	+	o	o	+	o
(sub)System	efficient	o	+	+	o	−
	scalable	+	o	−	−	+
	evolvable	++	++	++	−	++
	portable	o	+	o	++	o
	reliable	o	o	−	o	o
	dynamically reconfigurable	+	++	++	−	++

Legend: ++ extensive support + some support o neutral − no support

Taulussa 2 listatut arkkitehtuurityylit ovat:

Client-Server (asiakas-palvelin) -arkkitehtuureissa palvelua pyytävä ohjelmiston osa eli asiakasohjelmisto (client) on yhteydessä palveluntuottajaan (server) synkronoiduilla palvelupyynnöillä ja välittömillä tai lähes välittömillä vastauksilla. Client-ohjelmiston instanssit ovat tietoisia palvelujen sijainnista, mutta eivät tavasta jolla palvelu toteutetaan. Client-instanssit eivät myöskään ole tietoisia toisistaan.

C2 -arkkitehtuuri tarkoittaa sitä, että ohjelmiston tehtävät jaetaan hierarkkisesti eri komponenteille. Komponentit kommunikoivat sanomapohjaisesti. Kukin komponentti on tietoinen vain niistä palveluista, joita välittömästi ylempi hierarkkiataso tarjoaa.
Huom. CBSP-dokumentaatiossa luonnehditaan C2-arkkitehtuuria ilmaisulla "event-based" (tapahtumapohjainen), kun muut lähteet esim. Taylor ym. (2010, 126) käyttävät pääasiassa ilmaisua "message-based" (sanomapohjainen). Tässä kontekstissa ilmaisujen voi kuitenkin katsoa tarkoittavan samaa.

Event-based -arkkitehtuurit tarkoittavat ohjelmiston jakoa komponentteihin, jotka viestivät keskenään tapahtuma(sanoma-)pohjaisesti. Viestintä voi olla synkronista tai asynkronista. Komponentit voivat olla sekä palvelujen pyytäjiä että tuottajia. Sanomanvälitys tapahtuu erikseen tätä tarkoitusta toteuttavalla osajärjestelmällä.

Layered (kerros) -arkkitehtuureilla tarkoitetaan järjestelyä, jossa ohjelmiston komponentit on jaettu palvelukerroksiksi. Komponentit kommunikoivat ajantasaisilla palvelupyynnöillä. Komponentti voi pyytää palveluja vain välittömästi allaan olevasta palvelukerroksesta.

Pipe-and-filter -arkkitehtuurit liittyvät tilanteisiin, joissa tietoa jalostetaan, analysoidaan tai konvertoidaan. Toiminta voidaan jakaa useamman suotimen(filter) kesken, joista kukin suorittaa jonkin osatehtävän. Pipe (putki) tarkoittaa yleistä välitysmekanismia, jolla suodattimista koostetaan kokonainen prosessi.

On huomattava, että luetellut arkkitehtuurit eivät ole varsinaisesti toistensa vaihtoehtoja, eivätkä kaikki luetellut arkkitehtuurityylit edusta samaa mittakaavaa.

3.5.4. Ehdotuksia mallin laajentamiseksi

CBSP-malli on tarkoitettu laajennettavaksi. CBSP-menetelmässä arkkitehtoniset kategoriat kuvataan ominaisuuksien (properties) joukkona (taulu 2), mikä tekee kuvaustavasta abstraktimman sekä myös laajemman ja pitkäikäisemmän, kuin jos arkkitehtuurit kuvattaisiin triviaalisti luettelemalla niihin liittyviä yksittäisiä valintoja ja tekniikoita.

CBSP-malli on esitelty 2003. Jos se olisi julkaistu esim. 2011, yleisten arkkitehtuurityylien joukota sopisi olettaa löytyvän esimerkiksi SOA:n, palvelukeskeisen arkkitehtuurin. Tästä herää kysymys, onko CBSP-menetelmä jäänyt ajasta jälkeen. Laatusanoihin perustuvat luokittelutapa kuitenkin mahdollistaa taulukon 2 laajentamisen helposti, lisäämällä uusi arkkitehtuuri uudeksi sarakkeeksi (taulu 3).

Varsin luontevaa on ajatella, että arkkitehtuurityylien ohella taulukkoa 2 täydennettäisiin esimerkiksi ajantasaisilla suunnittelumalleilla (design pattern, ks. luku 3.2.3, arkkitehtuuriin liittyviä käsitteitä). Suunnittelumalleissa kiteytyy se, mitä ohjelmistotuotannon puolella voi pitää ns. parhaina käytäntöinä, joten pelkästään CBSP-mallissa lueteltujen ominaisuuksien (properties) yhdistäminen suunnittelumalliin relevanssikertoimien kera antaisi ohjaavaa tietoa suunnittelumallin käyttäjälle. Erilaisia suunnittelumalleja on paljon, joten kysymykseen tulisi lähinnä se, että CBSP-mallia laajentamalla testattaisiin jonkin edeltä valitun mallin soveltuvuutta tiettyyn tapaukseen.

TAULU 3: ehdotus SOA-arkkitehtuurin lisäämiseksi CBSP-mallin arkkitehtuurityyleihin

CBSP dimensions	Properties	Service-oriented (SOA)
data component	aggregated	++
	persistent	++
	streamed	-
	cached	+
processing component	service provide/consume only	++
	has N interfaces	++
	stateful	+
	loose coupling	+
	can be migrated	++
connector/bus	synchronous	++
	asynchronous	+
	local	-
	distributed	++
	secure	+
(sub)system	efficent	o
	scalable	++
	evolvable	++
	portable	o
	reliable	o
	dynamically reconfigurable	++

3.5.5. CBSP-menetelmän soveltaminen AnyCase-ohjelmistoon

AnyCase-ohjelmiston keskeiset osat arkkitehtuurin näkökulmasta ovat:

- henkilö- ja dokumenttirekisteri, joiden sisältö ja käyttötapa on lähinnä tavanomainen, eli tietueiden ja liitetiedostojen hakua ja ylläpitoa monen käyttäjän verkkoympäristössä;

- sovelluskehitin, jolla määritellään asianhallintasovelluksia; SaaS-jakelumallin käytöstä johtuen tämäkin ominaisuus on toteutettava viime kädessä monen käyttäjän verkkoympäristössä.
- sovellusten ajoympäristö, jossa käyttäjät ympäri tietoverkkoa pääsevät suorittamaan heille kuuluvia tehtäviä omalla vuorollaan.

Ohjelmiston erityinen piirre on se, että kaikkien yllämainittujen osien toiminnallisuus ja toteutustapa on varsin erilainen. Jos kutakin osaa lähdettäisiin toteuttamaan erikseen, tuloksena olisi luultavasti vaikeasti hallittava arkkitehtuurien sekoitus.

CBSP-mallin soveltamisen tarkoituksena on löytää vastauksia mm. seuraaviin kysymyksiin:

- Mikä tai mitkä arkkitehtuurit soveltuisivat järjestelmän perustaksi parhaiten?
- Ovatko ohjelmiston prototyyppiä varten jo tehdyt valinnat linjassa menetelmän antamien tulosten kanssa?
- Voisiko menetelmän antamia tuloksia välittömästi hyödyntää järjestelmän prototyypin suunnittelussa ja toteutuksessa?

Tärkein kysymys on kuitenkin seuraava: Onko mahdollista pitäytyä yhdessä arkkitehtuurissa kaikkien ominaisuuksien osalta, vai onko työnkulkusovellusten ajoympäristö toteutettava kokonaan eri pohjalta kuin ohjelmisto muuten? Sovelluksen karkea jakaminen toisaalta työnkulkusovellusten ajoympäristöön ja toisaalta muihin osiin on perusteltavissa seuraavasti: ensiksimainittu toteuttaa yhtenä kokonaisuutena prosessin, jonka tehtävät jakautuvat eri henkilöille sykronoidusti. Muut ominaisuudet, kuten työnkulkumallien määrittely, henkilö- ja dokumenttirekisteri sekä työnkulkujen monitorointi, ovat järjestelmän toiminnan kannalta triviaaleja yksittäisen käyttäjän erillisiä toimenpiteitä, joiden volyymi on alhainen. Siksi CBSP-vaatimusanalyysi jaetaan vaiheen 1 yhteydessä kahtia: ryhmän yksi muodostavat vaatimukset, jotka liittyvät suoraan työnkulkusovelluksiin. Ryhmän kaksi muodostavat vaatimukset, jotka kohdistuvat kaikkiin muihin eli työnkulkusovelluksia tukeviin ominaisuuksiin.

Koska allekirjoittanut soveltaa menetelmää omaan ohjelmistoprojektiinsa, eikä arkkitehtuuri- ja vaatimusanalyysiä varten ole käytettävissä muita henkilöitä, on asetelma

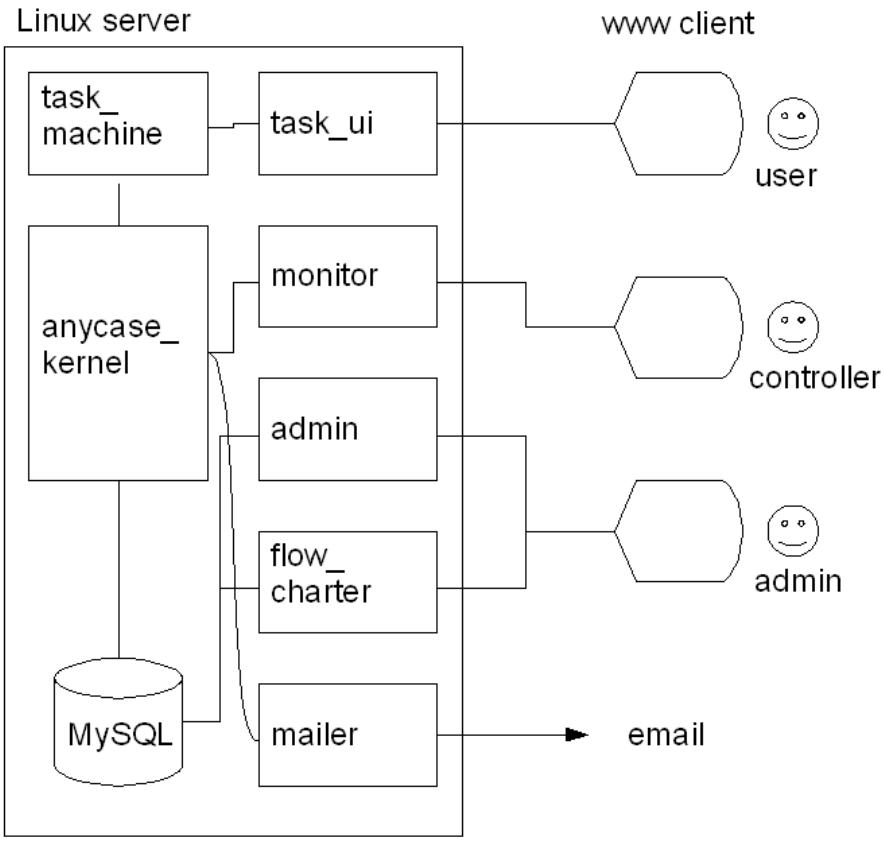
lähtökohtaisesti subjektiivinen. Siksi seuraavassa luvussa, jossa raportoidaan CBSP--mentelmän soveltaminen AnyCase-ohjelmistoon, pyritään seikkaperäisesti perustelemaan tehdyt valinnat ja arvostusperusteet. Näiden tarkkaan dokumentointiin CBSP-menetelmäohjeistus kehottaa joka tapauksessa.

3.5.6. AnyCase-ohjelmiston prototyypissä käytetty arkkitehtuuri

Ennen varsinaisen CBSP-analyysin aloittamista tutustumme AnyCase-ohjelmiston prototyypissä käytettyyn arkkitehtuuriin, joka perustuu tekijän (allekirjoittaneen) intuitioon. Tätä jo tehtyä arkkitehtuurivalintaa voidaan pitää vaatimusanalyysin hypoteesina, ja CBSP-analyysin päätteeksi arvioidaan sitä, antaako analyysi aihetta muokata ohjelmiston arkkitehtuuria prototyyppivaiheen jälkeen.

AnyCase-prototyypin tärkeimmät ohjelmalliset komponentit ilmenevät kuviosta 3. Komponenttien selitykset ovat taulussa 4. Rakenne toteuttaa aiemmin lueteltuja arkkitehtuurityylejä seuraavasti:

- Client-server. Selainpohjaisena ohjelmisto toteuttaa lähtökohtaisesti asiakas-palvelin -arkkitehtuuria. Asiakkaan työasemaan tai selaimeen ei liity mitään erityisesti AnyCase-ohjelmistoa varten vaadittavia komponentteja tai asetuksia. Asiakkaan liittymä on ns. thin client -tyyppiä, jossa toteutetaan pelkkiä käyttöliittymäelementtejä HTML- ja JavaScript-pohaisesti.
- MVC. Työnkulkujen päivittäisten tehtävien (roolit user ja controller) suorittamiseen käytetttävät ohjelman osat toteuttavat model-view-controller -rakennetta, jossa käyttöliittymää ja sovelluslogiikkaa varten ovat omat erilliset komponentit.
- Event-based. Loogiselta kannalta asianhallinann työnkulkujen surituksen eteneminen on luonnollista ajatella tapahtuma- tai sanomapohjaisesti. Järjestelmä tiedottaa asynkronisesti käyttäjiä työnkulun etenemisestä, käyttäjä tiedottaa järjestelmää työnkulkuun liittyvän tehtävän suorittamisesta. Järjestelmään liittyy myös sisäsyntyisiä, lähinnä aikaan sidottuja tapahtumia. Prototyypissä tapahtumien (sanomien) välitystä ei ole toteutettu erillisenä toimintona, mutta sama asia toteutuu loogisesti tietokantaan talletettavien tapahtumatietojen kautta. Järjestelmän voisi muuttaa käyttämään sopivaa ohjelmallista sanomajonokomponenttia yleisrakenteen muuttumatta.



KUVIO 3. AnyCase-prototyypin ohjelmalliset komponentit

TAULU 4. Selitykset kuvioon 3

user	rooli: tavallinen työnkulkuun osallistuva käyttäjä
conrtroller	rooli: työnkulun valvoja
admin(role)	rooli: organisaation pääkäyttäjä, mm. määrittelee työnkulut
anycase_kernel	työnkulkuprosessori, muuttaa työnkulkujen tilaa
task_machine	yksittäisen tehtävän ohjaus ja tilakone
task_ui, monitor, admin, flow_charter	käyttöliittymäkomponentit yksittäiselle tehtävälle, työnkulkujen seuraamiselle ja hallinnalle, sekä työnkulkujen määrittelylle
mailer	sähköpostiviestien lähetys
MySQL	tietovarasto

4. CBSP-ANALYYSIN SOVELTAMINEN AnyCase-OHJELMISTOON

4.1. CBSP- vaihe 1: Merkityksellisimpien vaatimusten valinta jatkoanalyysiä varten

AnyCase-pilottiasiakkaan vaatimukset on listattu liiitteessä 1.
Vaatimusten valintaperusteet ovat CBSP-menetelmäohjeistusta mukaillen

- vaatimukset, joiden toteuttaminen on projektin onnistumisen kannalta keskeistä
- vaatimukset, joilla ilmeisesti on arkkitehtoninen ulottuvuus

Vastaavasti sellaiset vaatimukset karsitaan, jotka

- ovat vähemmän tärkeitä; ks. kuitenkin lisähuomio alla
- luettelonomaisesti täydentävät jotain toista vaatimusta lisäämättä asiaan mitään tekniseltä kannalta uutta
- ovat tekniseltä kannalta triviaaleja

Vaatimusten prioriteetin osalta on huomioitava seuraavaa. Alemman prioriteetin vaatimusta ei voi automaattisesti karsia pois, vaan sillä voi olla 1) arkkitehtuurin kannalta kriittinen vaikutus etenkin silloin, jos vaatimuksen yleistäminen tai edelleen jalostaminen lisää sen merkittävyyttä; 2) tuotaessa SaaS-sovellus verkkoon jakelumallille tyypillisellä tavalla niin, että sen voi kuka tahansa potentiaalinen asiakas ottaa käyttöön automaattisesti, ei vaatimusten prioriteetteja voi kiinnittää yhden tai edes muutaman asiakkaan näkemyksen pohjalta. Innovatiivisen teknologian käyttöönotolle on tyypillistä, että käyttäjät löytävät uusia, odottamattomia käyttötapoja. Siksi tässä tapauksessa kiinnetään huomiota erityisesti vaatimuksen mahdollista vaikuttavuutta arkkitehtuuriin, vaatimuksen prioriteetista riippumatta.

Joitakin esimerkkejä:

- Vaatimus nf1.1: "Työnkulkuun osallistuminen ei saa edellyttää työasemakohtaisia ohjelmistoasennuksia", prioriteetti 2 (= toivottava mutta ei pakollinen). Vaikka pilottiasiakas voisikin tarvittaessa hyväksyä työasemakohtaiset ohjelmistoasennukset, on vaatmus SaaS-jakelumallin puitteissa kuitenkin katsottava pakolliseksi. Vaatmus on myös ohjelmistoarkkitehtuurin kannalta varsin merkittävä, joten se valikoituu mukaan meneillään olevassa CBSP-vaiheessa 1.

- Vaatimus nf4.2 henkiörekisteri / henkilöllä voi olla useita osoitteita, joista yksi on aktiivinen (prioriteetti 1, pakollinen). Tämä on pilottiasiakkaan erikoisvaatimus, koska asiakas lähettää työntekijöitään ulkomaisiin projekteihin. Vaatimus tarkoittaa käytännössä yhden lisärelaation luomista tietokantaan, joten se ei tuo mitään teknisesti uutta. Vaatimus ei myöskään viittaa mihinkään yleisempään ominaisuuteen tai tarpeeseen, joten vaatimus karsitaan pois jatkoanalyysistä.

Valittujen vaatimusten jako ryhmiin 1 ja 2

Vaatimusten jaossa ryhmiin 1=työnkulkusovellukset ja 2=tukevat ominaisuudet on huomioitava, että ryhmiin valikoituu osittain yhteisiä vaatimuksia. Alkuperäisistä vaatimuksista enemmistö on ei-toiminnallisia vaatimuksia. Tähän vaikuttaa se, että sovelluksen keskeinen erityispiirre on sen tietomalli. Tietomalli on puolestaan yhteinen sovelluksen kaikille toiminnallisille ominaisuuksille.Taulussa 5 luetellaan CBSP-analyysiin valitut vaatimukset ja niiden jako ryhmiin 1 ja 2.

TAULU 5: CBSP-analyysiin valitut vaatimukset

id	vaatimus	ryhmä 1	ryhmä 2
1 nf1.1	Työnkulkuun osallistuminen ei saa edellyttää työasemakohtaisia ohjelmistoasennuksia	x	
2 nf1.3	Työnkulun osapuolet voivat olla myös oman organisaation ulkopuolella	x	x
3 nf1.5	Asiakirja voi olla myös viittaus sähköiseen tai fyysiseen dokumenttiin	x	x
4 nf2.1	Työnkulkuun voi liittyä yksi tai useampia asiakirjoja	x	x
5 nf2.13	Järjestelmän tulee mahdollistaa työnkulut, jotka ovat kuvattavissa prosessi- tai vuokaavion avulla: yksittäiset tehtävät tulee voida suorittaa peräkkäin, ehdollisesti tai toistuvasti.	x	x
6 nf2.14	Asiakkaan tulee itse voida määritellä haluamansa työnkulut ja muuttaa niitä.		x
7 nf3.1	Toimijan on saatava tieto työnkulkuun liittyvästä tehtävästä sekä sähköpostin että www-liittymän kautta	x	
8 nf5.1	Käyttäjärekisterin käyttäjätiedot on voitava hakea asiakkaan omasta hakemistopalvelusta esim. LDAP-protokollalla		x
9 f1.8	Tehtäväsivulla on oltava mahdollisuus tehdä vapaamuotoinen merkintä asianhallintamuistioon	x	
10 f2.2	Ohjaajan tulee voida käynnistää uusi työnkulku valitsemansa työnkulkumallin pohjalta.		x
11 f2.5	Työnkulun tulee edetä automaattisesti seuraavissa tilanteissa: - työnkulkuun liittyvä tehtävä on suoritettu - työnkulkuun liittyvä ajoituksen perusteella	x	
12 f4.13	Pääkäyttäjän tulee voida simuloida työnkulkumallia reaaliaikaisesti niin, että työnkulun eteneminen suunnitellulla tavalla voidaan todeta.		x

4.2. CBSP-vaihe 2: vaatimusten arkkitehtoninen luokittelu

Todetaan kustakin valitusta vaatimuksesta (taulu 6), kuinka relevantteja ne ovat suhteessa kuhunkin CBSP-kategoriaan.

TAULU 6. AnyCase-vaatimusten CBSP-luokitus

	vaatimus	**C**	**B**	**S**	**CP**	**BP**	**SP**
1	Työnkulkuun osallistuminen ei saa edellyttää työasemakohtaisia ohjelmistoasennuksia	3	3	3	2	3	3
2	Työnkulun osapuolet voivat olla myös oman organisaation ulkopuolella	1	2	1	1	1	2
3	Asiakirja voi olla myös viittaus sähköiseen tai fyysiseen dokumenttiin	2	2	1	2	1	1
4	Työnkulkuun voi liittyä yksi tai useampia asiakirjoja	3	0	2	1	0	1
5	Järjestelmän tulee mahdollistaa työnkulut, jotka ovat kuvattavissa prosessi- tai vuokaavion avulla: yksittäiset tehtävät tulee voida suorittaa peräkkäin, ehdollisesti tai toistuvasti.	3	1	3	3	0	2
6	Asiakkaan tulee itse voida määritellä haluamansa työnkulut ja muuttaa niitä.	2	0	3	2	0	2
7	Toimijan on saatava tieto työnkulkuun liittyvästä tehtävästä sekä sähköpostin että www-liittymän kautta	3	2	2	3	1	2
8	Käyttäjärekisterin käyttäjätiedot on voitava hakea asiakkaan omasta hakemistopalvelusta esim. LDAP-protokollalla	1	2	1	1	1	1
9	Tehtäväsivulla on oltava mahdollisuus tehdä vapaamuotoinen merkintä asianhallintamuistioon	3	0	1	2	0	0
10	Ohjaajan tulee voida käynnistää uusi työnkulku valitsemansa työnkulkumallin pohjalta.	3	0	2	2	0	1
11	Työnkulun tulee edetä automaattisesti seuraavissa tilanteissa: - työnkulkuun liittyvä tehtävä on suoritettu - työnkulkuun liittyvä ajoituksen perusteella	3	1	3	3	1	3
12	Pääkäyttäjän tulee voida simuloida työnkulkumallia reaaliaikaisesti niin, että työnkulun eteneminen suunnitellulla tavalla voidaan todeta.	3	0	1	2	0	1

Asteikko: 0-ei lainkaan, 1-osittain, 2-suuressa määrin, 3-täysin
Symbolit C, B, S, CP, BP, SP ks.s.36 .

Vaiheen 1 verifiointina menetelmäohje mainitsee "check level of consensus among stakeholders". Tätä selvitystä varten ei ole käytettävissä muiden sidosryhmien edustajia, mutta huomiota tulisi kiinnittää kohtiin, joissa eri henkilöt antavat kohteille selvästi poikkeavia painokertoimia.

4.3. CBSP-vaihe 3: Luokitteluristiriitojen tunnistaminen ja ratkaisu

Tässä tarkastelussa sivuutamme vaiheen maininnalla, koska kyse on työryhmän eriävien näkemysten yhteensovittamisesta. Vaiheessa korostetaan ristiriitojen taustalla olevien syiden julkituomista. Vaiheeseen sisältyy myös mahdollisuus karsia vaatimusluettelosta tarpeettomia vaatimuksia. Perusteena voisi olla esim. se, että vaatimuksen profiilissa olisi vain kertomia 0..1, ts. vaatimus ei olisi merkityksellinen yhdessäkään CBSP-kategoriassa. Todetaan, että esim. vaatimus 8 (käyttäjätietojen hakeminen asiakkaan omasta LDAP-rekisteristä) on valittujen vaatimusten joukossa vähiten painotettu, mutta toisaalta se on ainoa täsmällisesti yksilöity vaatimus koskien järjestelmän rajapintoja muihin järjestelmiin, joten on varsin kiistatonta pitää sitä tärkeänä B (bus)-kategoriassa.

4.4. CBSP-vaihe 4: Vaatimusten muokkaaminen arkkitehtuurin näkökulmasta

Vaiheen sisältönä on muokata vaatimuksia niin, että kukin vaatimus täsmää ensisijaisesti vain yhteen CBSP-kategoriaan. Käytännössä tämä tarkoittaa sellaisten vaatimusten pilkkomista osiin, jotka on todettu relevanteiksi useissa kategorioissa (taulut 7 ja 8). Tämän jälkeen pyritään vielä minimoimaan vaatimusten mahdollinen redundanssi.

TAULU 7: Vaatimukset, jotka vaativat osittamista.

	vaatimus	**C**	**B**	**S**	**CP**	**BP**	**SP**
1	Työnkulkuun osallistuminen ei saa edellyttää työasemakohtaisia ohjelmistoasennuksia	3	3	3	2	3	3
4	Työnkulkuun voi liittyä yksi tai useampia asiakirjoja	3	0	2	1	0	1
5	Järjestelmän tulee mahdollistaa työnkulut, jotka ovat kuvattavissa prosessi- tai vuokaavion avulla: yksittäiset tehtävät tulee voida suorittaa peräkkäin, ehdollisesti tai toistuvasti.	3	1	3	3	0	2
6	Asiakkaan tulee itse voida määritellä haluamansa työnkulut ja muuttaa niitä.	2	0	3	2	0	2
7	Toimijan on saatava tieto työnkulkuun liittyvästä tehtävästä sekä sähköpostin että www-liittymän kautta	3	2	2	3	1	2
10	Ohjaajan tulee voida käynnistää uusi työnkulku valitsemansa työnkulkumallin pohjalta.	3	0	2	2	0	1
11	Työnkulun tulee edetä automaattisesti seuraavissa tilanteissa: - työnkulkuun liittyvä tehtävä on suoritettu - työnkulkuun liittyvä ajoituksen perusteella	3	1	3	3	1	3

TAULU 8: taulun 7 vaatimukset ositettuna

	vaatimus	C	C_P	C_D	B	S	CP	BP	SP	1	2
1	Työnkulkuun osallistuminen ei saa edellyttää työasemakohtaisia ohjelmistoasennuksia	3			3	3	2	3	3		
1.1	Ohjelman, jolla käyttäjä suorittaa työnkulkuun liittyvän tehtävän, tulee olla selainpohjainen.		3		2	2	2	1	1	x	
1.2	Työnkulkuihin liittyvä tietojen hallinta ja automaattinen päätöksenteko suoritetaan keskitetysti palvelimella/palvelimilla		2		1	3	2	1	3	x	
4	Työnkulkuun voi liittyä yksi tai useampia asiakirjoja	3			0	2	1	0	1		
4.1.	Työnkulkuun liittyy sähköinen asiakirjakansio, jossa voi olla 0..n asiakirjaa tai viittausta			3	0	1	1	1	1	x	x
5	Järjestelmän tulee mahdollistaa työnkulut, jotka ovat kuvattavissa prosessi- tai vuokaavion avulla: yksittäiset tehtävät tulee voida suorittaa peräkkäin, ehdollisesti tai toistuvasti.	3			1	3	3	0	2		
5.1	Järjestelmän tulee sisältää suunnittelutyökalu sekä suoritin, joilla voidaan kuvata ja suorittaa prosessikaavioon perustuvia työnkulkuja.		3		0	1	1	1	1		x
5.2.	Järjestelmän tulee voida toteuttaa prosessikaavion muodossa kuvattuja työnkulkuja.		1		0	3	0	0	1	x	
6	Asiakkaan tulee itse voida määritellä haluamansa työnkulut ja muuttaa niitä.	2			0	3	2	0	2		
6.1.	Asiakkaalle tulee tarjota käyttöliittymä työnkulkujen määrittelyyn ja muokkaamiseen.		3		0	2	2	0	1		x
6.2.	Järjestelmän tulee sallia joustava työnkulkukaavojen lisääminen ja muuttaminen		1		0	3	1	0	2		x
7	Toimijan on saatava tieto työnkulkuun liittyvästä tehtävästä sekä sähköpostin että www-liittymän kautta	3			2	2	3	1	2		
7.1	Järjestelmässä tulee olla sähköpostia lähettävä toiminto, joka tiedottaa käyttäjiä työnkulkuun		3		2	1	2	1	1	x	
7.2	liittyvistä tehtävistä Järjestelmän tulee tarjota selainpohjainen käyttöliittymä työnkulkuun liittyvien tehtävien seuraamista varten		3		0	1	1	0	1	x	
10	Ohjaajan tulee voida käynnistää uusi työnkulku valitsemansa työnkulkumallin pohjalta.	3			0	2	2	0	1		
10.1	Järjestelmässä tulee olla toiminto, jossa käyttäjä voi riittävin oikeuksin valita työnkulkumallin ja käynnistää työnkulun	3	3		0	1	1	0	1		x
10.2	Järjestelmään tulee voida tallettaa työnkulkumalleja			3	0	1	1	0	1		x
11	Työnkulun tulee edetä automaattisesti	3			1	3	3	1	3		

	seuraavissa tilanteissa: - työnkulkuun liittyvä tehtävä on suoritettu - työnkulkuun liittyvä ajoituksen perusteella										
11.1	Järjestelmään sisältyvän työnkulkusuorittimen on vietävä työnkulkua eteenpäin automaattisesti		2		0	3	1	0	2	x	

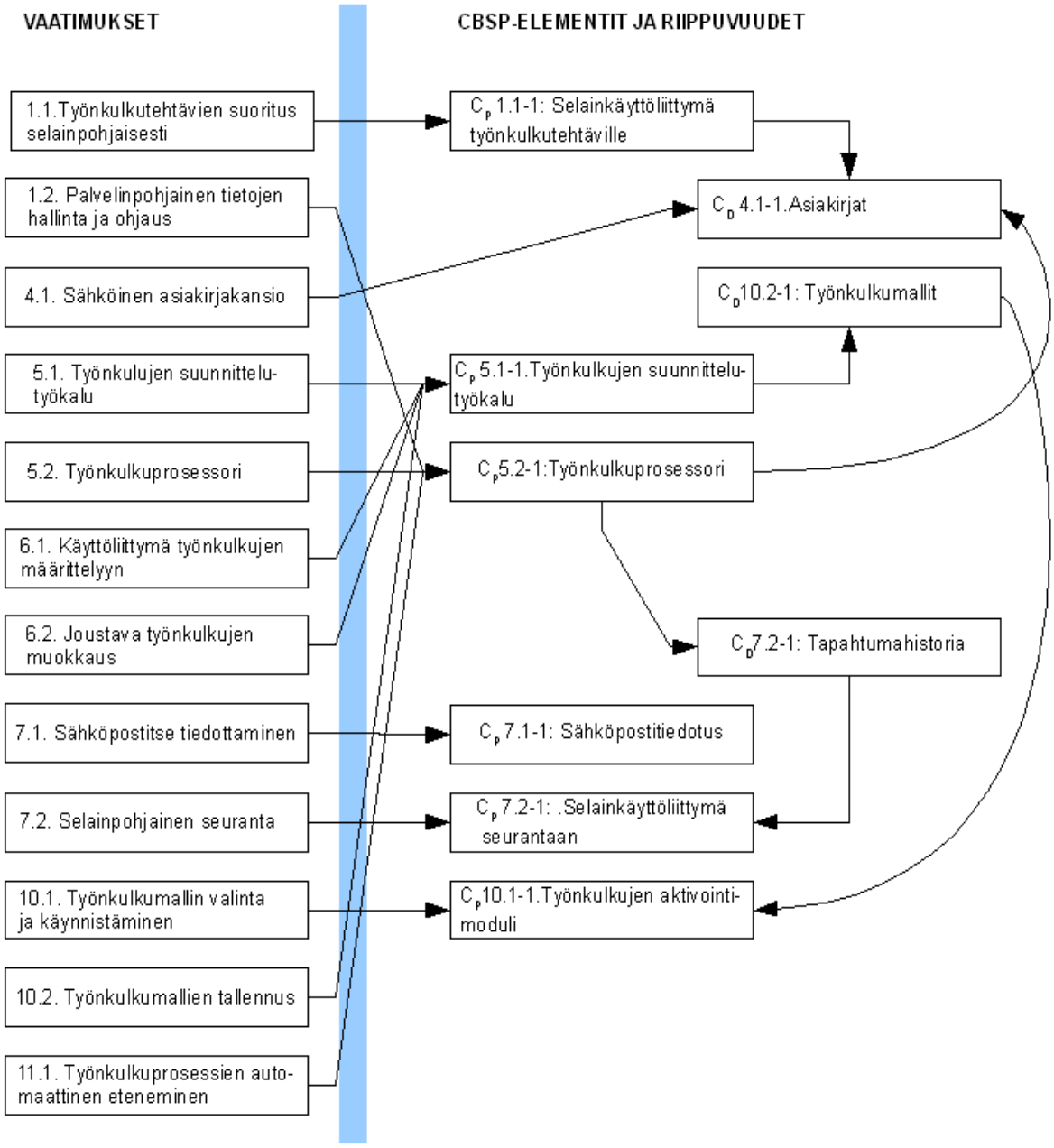
taulun 8 symbolit:

C	komponentti
C_P	ohjelmallinen komponentti
C_D	datakomponentti
B	(bus) asioita yhdistävä/välittävä elementti
S	(system) yleinen ominaisuus
CP	komponentin ominaisuudet
BP	yhteyselementin ominaisuudet
SP	järjestelmän yleiset ominaisuudet
1	vaatimukset jotka liittyvät suoraan toteutettaviin työnkulkuihin
2	työnkulkusovelluksia tukevat ominaisuudet, kuten työnkulkumallien määrittely tms.

Ositus ja painokertoimien valinta ovat tietysti vain approksimaatioita, jotka ovat sitä vakuuttavampia mitä laajempi työryhmä on niitä tekemässä.

Seuraavaksi pyritään minimoimaan redundanssi. Redundanssin mahdollisuus liittyy tässä etenkin "työnkulkumallin" käsitteeseen: se mainitaan tavalla tai toisella vaatimuksissa 5,6,10 ja 11. Voidaan kuitenkin todeta, että esim. vaatimukset 5 ja 6 painottuvat eri asioihin, ensimmäisen ilmentäessä, mistä työnkulkumalleissa on kyse ja jälkimmäisen ilmentäessä, miten mallintamismahdollisuuden on toteuduttava järjestelmässä.

Kuviosta 4 ilmenevät muokatut vaatimukset, niistä johdetut CBSP-elementit ja näiden väliset suhteet. CBSP-elementit jaetaan pääasiassa C_P (processing component)- ja C_D (data component) -kategorioihin. Elementtien välisten yhteyksien voi tulkita edustavan välittäviä "Bus"-ominaisuuksia. Alustava yhteys vaatimusten ja arkkitehtonisten piirteiden välillä saa näkyvän muodon. Kuvasta on helppo todeta mm. ne valmiin ohjelmiston osat, joita ainakin tarvitaan.



KUVIO 4: Vaatimukset, CBSP-mallielementit ja riippuvuudet

4.5. Vaihe 5: arkkitehtonisten elementtien ja tyylien painottaminen CBSP:n mukaan

Tässä vaiheessa luetteloidaan edellisessä vaiheessa tunnistetut CBSP-elementit riippuvuuksineen, ja kuvaillaan näiltä vaadittavia ominaisuuksia käyttäen samoja laatusanoja, kuin millä taulussa 2 (s. 41) kuvataan arkkitehtuurityylejä. Näin saadaan ominaisuuksia

kuvailevien ilmaisujen kautta luotua yhteys CBSP-elementtien ja arkkitehtuurityylien välille (taulu 9).

TAULU 9. Arkkitehtuurityylit ja AnyCase CBSP-elementit

CBSP-luokka	**ominaisuus**	**client-server**	**C2**	**Event-based**	**Lay-ered**	**Pipe& filter**	**SOA**
C_D asiakirjat (1,2)	persistent aggregated	++ ++	o ++	o ++	o +	o -	o ++
C_D työnkulkumallit (2)	persistent	++	o	o	o	o	o
C_D tapahtumahistoria (1)	persistent	++	o	o	o	o	o
C_P selainkäyttöliittymä työnkulkutehtäville (1)	service provide/con-sume has N interfaces	++ ++	o +	o ++	o -	o -	++ ++
C_P työnkulkujen suunnittelutyökalu (2)	service provide/con-sume stateful	++ +	o ++	o ++	o +	o -	++ +
C_P työnkulkuprosessori (1)	has N interfaces stateful loose coupling	++ + +	+ ++ +	++ ++ ++	- + -	- - ++	++ + ++
C_P sähköpostitiedotus (1)	stateful loose coupling	+ +	++ +	++ ++	+ -	- ++	+ ++
C_P selainkäyttöliittymä (1) seurantaan	service provide/con-sume has N interfaces	++ ++	o +	o ++	o -	o -	++ ++
C_P työnkulkujen aktivointimoduli (2)	service provide/con-sume stateful	++ +	o ++	o ++	o +	o -	++ +
B tk.prosessori - tk.tehtävät - asiakirjat - historia (1)	asynchronous distributed secure	- ++ +	++ ++ o	++ ++ o	- - +	++ + o	- ++ +
B työnkulkumallit - työnkulkukontrolli (1,2)	asynchronous secure	- +	++ o	++ o	- +	++ o	- +
S system	scalable (1) evolvable (2) reliable (1) dynamically reconfi-gur. (1)	+ ++ o +	o ++ o ++	- ++ - ++	- - o -	+ ++ o ++	++ ++ o ++
plussia yht (miinukset yht)		*36 (2)*	*25 (0)*	*30 (2)*	*7 (11)*	*13 (8)*	*34(2)*

Joitakin selityksiä tauluun 9 liittyen:

- merkinnät (1,2) viittaavat kyseisen elementin taustalla olevan vaatimuksen luokitukseen koskien sitä, liittyikö kyseinen vaatimus 1) varsinaisiin työnkulkusovelluksiin vai 2) työnkulkujen määrittely- ja muihin tukiominaisuuksiin
- taulukon B (Bus) -elementit edustavat kuvassa 2 todettuja CBSP-elementtien yhteyksiä, jotka ovat yhtä tärkeitä suunnitteluelementtejä kuin samassa kuvassa todetut C (componenet) -elementit. Bus-elementit on niputettu yhteen teknisin

perustein seuraavasti. Ensimmäisessä ryhmässä "työnkulkuprosessori - työnkulkutehtävät - asiakirjat - historia" olevat elementit kuuluvat yhteen sillä perusteella, että ne aktivoituvat samanaikaisesti työnkulkuprosessorin määräämällä tavalla. Tällöin näiden välisiä yhteyksiä voidaan luonnehtia yhteisillä ilmauksilla, vaikka yhteyksiä toteuttaisikin lopulta useampi kuin yksi ohjelmallinen komponentti. Toisessa ryhmässä "työnkulkumallit - työnkulkukontrolli" on vastaavasti kyse laadullisesti omasta, homogeenisesta ryhmästä.

- taulukkoon on lisätty SOA-arkkitehtuuri (ks. 3.2.2.7), joka ei ole mukana alkuperäisessä CBSP-mallissa. Jatkossa on otettava huomioon, että alkuperäisessä CBSP-mallissa listatut arkkitehtuurien ominaisuudet perustuvat aiempiin selvityksiin, kun taas SOA-sarakkeen attrbuutit ovat pelkästään tekijän arvioita.

Tauluista 9 ja 10 ilmenee taulun 8 ositus vaatimusryhmien 1 (varsinaiset työnkulkusovellukset) ja 2 (työnkulkujen määrittely- ja tukiominaisuudet) mukaan.

TAULU 10: Arkkitehtuurityylit ja AnyCase CBSP-elementit / vaatimusryhmä 1

CBSP-luokka	**ominaisuus**	**client-server**	**C2**	**Event-based**	**Layered**	**Pipe& filter**	**SOA**
C_D asiakirjat (1,2)	persistent aggregated	++ ++	o ++	o ++	o +	o -	o ++
C_D tapahtumahistoria (1)	persistent	++	o	o	o	o	o
C_P selainkäyttöliittymä työnkulkutehtäville (1)	service provide/consume has N interfaces	++ ++	o +	o ++	o -	o -	++ ++
C_P työnkulkuprosessori (1)	has N interfaces stateful loose coupling	++ + +	+ ++ +	++ ++ ++	- + -	- - ++	++ + ++
C_P sähköpostitiedotus (1)	stateful loose coupling	+ +	++ +	++ ++	+ -	- ++	+ ++
C_P selainkäyttöliittymä (1) seurantaan	service provide/consume has N interfaces	++ ++	o +	o ++	o -	o -	++ ++
B tk.prosessori - tk.tehtävät - asiakirjat - historia (1)	asynchronous distributed secure	- ++ +	++ ++ o	++ ++ o	- - +	++ + o	- ++ +
B työnkulkumallit - työnkulkukontrolli (1,2)	asynchronous secure	- +	++ o	++ o	- +	++ o	- +
S system	scalable (1) reliable (1) dynamically reconfigur. (1)	+ o +	o o ++	- - ++	- o -	+ o ++	++ o ++
plussia yht (miinukset yht)		*25(2)*	*19 (0)*	*24(2)*	*5(10)*	*11(6)*	*26(2)*

TAULU 11: Arkkitehtuurityylit ja AnyCase CBSP-elementit / vaatimusryhmä 2

CBSP-luokka	**ominaisuus**	**client-server**	**C2**	**Event - based**	**Layered**	**Pipe& filter**	**SOA**
C_D asiakirjat (1,2)	persistent aggregated	++ ++	o ++	o ++	o +	o -	o ++
C_D työnkulkumallit (2)	persistent	++	o	o	o	o	o
C_P työnkulkujen suunnittelutyökalu (2)	service provide/consume stateful	++ +	o ++	o ++	o +	o -	++ +
C_P työnkulkujen aktivointimoduli (2)	service provide/consume stateful	++ +	o ++	o ++	o +	o -	++ +
B työnkulkumallit - työnkulkukontrolli (1,2)	asynchronous secure	- +	++ o	++ o	- +	++ o	- +
S system	evolvable (2)	++	++	++	-	++	++
plussia yht (miinukset yht)		*15(1)*	*10 (0)*	*10(0)*	*4(2)*	*4(3)*	*11(1)*

Tauluja 9, 10 ja 11 havainnoimalla voidaan todeta seuraavaa.

- Client-server: vaikuttaa ensi näkemältä mahdolliselta valinnalta koko sovellusta ajatelleen. Soveltuvuus on suurinta komponenttien kohdalla (CD, CP). Client-server saa parhaat pisteet etenkin vaatimusryhmän 2 (työnkulkumallien määrittely- ja hallintatoiminnot) kohdalla. Edelleen client-server saa miinuksia (=ei tukea) varsin rajoitetusti.
- C2 eli hierarkkinen sanomapohjainen arkkitehtuurityyli saa myös melko korkeat pisteet, eikä itse asiassa lainkaan miinuksia. C2:n soveltuvuus näyttää lisäksi jakautuvan tasaisesti komponentteihin (CD, CP), väylään (Bus) ja järjestelmään (System) liittyvien attribuuttien kesken. Erityisesti tämä tekee myös C2:ta varteenotettavan arkkitehtuurikandidaatin, vaikka kokonaispisteet eivät olekaan tilaston kärjessä.
- Event-based: sijoittuu pisteiden perusteella erittäin hyvin etenkin varsinaisten työnkulkuominaisuuksien (vaatimusryhmä 1) kohdalla, ja hieman heikommin vaatimusryhmän 2 kohdalla. Tulosta voi tältä osin pitää odotettuna, onhan työnkulussa juuri kyse siitä, että järjestelmä reagoi työnkulun osapuolten aiheuttamiin tapahtumiin.

- Layered: analyysin perusteella vaikuttaa soveltuvan huonosti tähän tarkoitukseen. Etenkin varsinaisten työnkulkuominaisuuksien kohdalla (vaatimusryhmä 1) tuen puute (miinukset) näyttää merkittävältä.
- Pipe and filter: ei myöskään vaikuta ilmeiseltä arkkitehtuurivalinnalta. Tuen puute tarvittaville ominaisuuksille korostuu. Pipe and filter saa pisteitä lähinnä yleistä joustavuutta ilmentävistä attribuuteista, kuten loose coupling, asynchronous, evolvable.
- SOA: saa varsin korkeat pisteet etenkin varsinaisten työnkulkuominaisuuksien kohdalla (vaatimusryhmä 1). Muistutetaan jälleen, että allekirjoittanut on lisännyt SOA:n CBSP-malliin käyttäen omaa arviotaan erilaisista painokertoimista. Muille CBSP-mallissa esiintyville arkkitehtuurityyleille annetut painokertoimet perustuvat CBSP-mallin julkaisijoiden aiempiin tukimuksiin (Grbch lähde]. Valitut painokertoimet ilmentävät ajatusta, että SOA-arkkitehtuurilla tavoitellaan paljolti samoja päämääriä kuin client-server -arkkitehtuurissa. Nämä varaukset SOA:n suhteen on pidettävä mielessä käsillä olevassa analyysissä. Näyttää kuitenkin siltä, että SOA voi olla varteenotettava vaihtoehto AnyCase-järjestelmän arkkitehtuuriksi. SOA ei saa miinuksia sen enempää kuin Client-server, ja lisäksi sen tuki järjestelmän (System) yleisille laadullisille vaatimuksille on jonkin verran edellämainittua vaihtoehtoa parempi.

4.6. CBSP-analyysimenetelmän arviointia

Ennen lopullisten johtopäätösten vetämistä CBSP-menetelmän anatamista tuloksista on paikallaan arvioida itse menetelmää, sekä niitä tapoja, joilla menetelmän antamia tuloksia kannattaa soveltaa.

On ilmeistä, että menetelmän liiketoiminnallinen päätarkoitus on antaa selkänojaa arkkitehtuurin valinnalle tilanteessa, jossa valinnalle on olemassa aitoja vaihtoehtoja. Menetelmä on myös varsin työvoimavaltainen, joten se on käyttökelpoinen vain suuren kokoluokan projekteissa. Menetelmän kehittäjiät ovat tarjonneet kaksi esimerkkiä, jotka ilmentävät hyvin sopivia käyttökohteita: 1) Cargo router alkuperäisessä CBSP-artikkelissa kuvaa mallin soveltamista järjestelmään, jonka tarkoituksena on kriisitilanteissa palvella julkishallintoa katastrofiavun jakelussa, sekä 2) iPhone Taskmind, joka on tarkoi-

tettu ajantasaiseen projektiryhmän sisäiseen viestintään. Kummassakin esimerkissä on kyse suuren mittakaavan innovatiivisesta ohjelmistoprojektista. Menetelmän työvoimavaltaisuus johtuu siitä, että menetelmän eri vaiheissa edellytetään kaikkien sidosryhmien edustajien osallistumista, vieläpä samanaikaisesti. Kaikki osallistujat tulee lisäksi ainakin omalta osaltaan perehdyttää menetelmän tarkoitukseen ja tavoitteisiin.

On myös huomoitava, että menetelmä on kvalitatiivinen huolimatta siitä, että se näyttää antavan antavan tulokseksi pisteitä ja suhdelukuja. Numeerisuutta käytetään menetelmässä vain kommunikoinnin välineenä informaation pelkistämiseksi. Kaikkien numeroiden alkuperä on erilaisissa painokertoimissa, jotka ovat yksinomaan kvalitatiivisia.

Edelleen on huomattava, että menetelmässä työstetään alkuperäisistä vaatimuksista johdettuja tulkintoja ja arviointeja useassa vaiheessa ja jopa iteratiivisesti. Tämä voi johtaa degeneroitumiseen ja jonkin ennakkoasenteen sulautumiseen osaksi aineistoa. Epäilemättä siksi menetelmän ohjeistuksessa korostetaankin eri sidosryhmien osallistumisen tärkeyttä.

On epätodennäköistä, että menetelmä antaisi tulokseksi jonkin tietyn arkkitehtuurin erityisen suurella painokertoimella, ja muut arkkitehtuurit saisivat vain vähäisiä pisteitä. Tällaisessa tilanteessa arkkitehtuurin valinta olisi luultavasti ilmeistä ilman CBSP-menetelmääkin. Menetelmän osoittaessa kahden tai useamman arkkitehtuurin relevantiksi, on syytä tarkastella, miten pisteet jakautuvat eri CBSP-elementtien kesken. Näin voidaan saada ideoita siitä, millä tavoin eri arkkitehtuurityylejä kannattaisi soveltaa koko järjestelmässä. Tässä kohtaa astuu kuvaan myös mallin tarjoama jäljitettävyys: on mahdollista sanoa, mitkä tietyt asiakasvaatimukset puhuvat tietyn sunnitteluratkaisun puolesta, tai sitä vastaan.

Menetelmän erityinen lisäarvo projektille tulee siitä, että sidosryhmät voivat todeta arkkitehtuurin valinnan, kustannusten kannalta kriittisen suunnittelupäätöksen, tapahtuneen objektiivisesti asiakasvaatimusten pohjalta.

5. TULOKSET JA YHTEENVETO

5.1. CBSP-analyysin tulosten tulkinta ja merkitys AnyCase-ohjelmistoprojektissa arkkitehtuurityyleittäin

Kun havainnot luvussa 4.5. sekä edellisessä luvussa 4.6. esitetyt näkökohdat otetaan huomioon, voidaan tuloksista tehdä seuraavia tulkintoja ja johtopäätöksiä.

Layered-arkkitehtuurityyli saa CBSP-analyysin perusteella heikoimman tuen. Tätä ei voi pitää yllättävänä, sillä ratkaistava ongelma ei millään oleellisella tavalla pidä sisällään ajatusta erilaisista palvelukerroksista. Tällä perusteella layered-arkkitehtuurityyliä ei valittaisi koko järjestelmän arkkitehtuurityyliksi, mikä ei kuitenkaan sulje pois tapaa toteuttaa jokin yksittäinen komponentti tai toiminta.

Pipe-and-Filter -arkkitehtuurityyli menestyy edellistä hieman paremmin. Periaatteessa voisi ajatella, että asiakirjojen työnkulut ilmentävät "liukuhihnaa", jossa tietoa jalostetaan vaihe vaiheelta. Pipe-and-Filter -arkkitehtuurityylissä kyse on kuitenkin käsitteellisesti alemman tason toiminnasta, eikä tyyli ole siksi kunnolla sovellettavissa koko järjestelmään. Pipe-and-filter saa kuitenkin jonkin verran pisteitä attribuuteista "loose coupling" ja "asynchronous", mitä yleisiä ominaisuuksia AnyCase-järjestelmä toki ilmentää.

C2- ja Event-based -arkkitehtuurityylejä voidaan tässä käsitellään yhdessä, sillä ne ovat rakenteellisesti samankaltaisia ja saavat tehdyssä CBSP-analyysissä pisteitä samoista asioista. C2 on yhdistelmä sanomapohjaisesta- ja kerrosarkkitehtuurista. Se kehitettiin alun perin, jotta MVC-mallia (malli-näkymä-logiikka) voitaisiin soveltaa myös laajoissa, hajautetuissa järjestelmissä (Taylor, Medvidovic, Dashovy (2010), s.124]. Puhuttaessa asiakirjojen työnkulusta sanomapohjaisuus vaikuttaa päällisin puolin luontevalta mallinnustavalta. CBSP-attribuutit "stateful", "asynchronous","loose coupling" nousevat esiin sekä C2:n että yleisen tapahtumapohjaisen (Event-based) tyylin kohdalla. Event-based -arkkitehtuurityyli nousee jonkin verran korkeammaksi pisteytyksessä kuin C2 kautta linjan ilman, että C2-tyyli saisi minkään yksittäisen CBSP-elementin kohdalla edukseen erottuvaa painokerrointa. Tällä perusteella voidaan ajatella, että C2:n erikois-

tuneet, tiukat piirteet, jotka liittyvät kerrosten väliseen kommunikointiin, eivät toisi lisäarvoa AnyCase-järjestelmälle, ja ne voitanee sivuuttaa. Sen sijaan etenkin AnyCase-järjestelmän varsinaiset työnkulkuominaisuudet (taulu 10) nostavat sanomapohjaisuuden selkeästi esiin.

Client-Server -arkkitehtuurityyliä noudattava ratkaisu on AnyCase-järjestelmän kohdalla tietyiltä osin vääjäämätön jo pelkästään johtuen SaaS-jakelumallista. Selainkäyttöisyys johtaa itsessään ainakin yksinkertaisella tasolla välttämättä client-server -tyyppiseen perusratkaisuun. Asiakassovellus voi olla ns. thin client, joka toteuttaa lähinnä käyttöliittymätoimintoja, tai thick client, joka toteuttaa käyttöliittymän ohella niitä osia sovelluslogiikasta, jotka voi toteuttaa online-vuorovaikutuksessa yksittäisen käyttäjän kanssa. Client-server ei kuitenkaan sulje pois tai ole vaihtoehto sanomapohjaisuudelle (Event-based) tai muille arkkitehtuurityyleille, sillä kyse on lisäksi siitä, miten palvelinpuolen (Server) toiminnallisuus toteutetaan. Päädytään siihen, että selainkäyttöisissä, keskitetysti tuotetuissa palveluratkaisuissa client-server -arkkitehtuuri toteutuu aina ainakin client:n näkökulmasta. Client-komponentti on yhteydessä aina yhteen Server-komponenttiin, joka piilottaa clientilta tavan, jolla palvelu tuotetaan. Tämä tapa voi puolestaan perustua laajaan kirjoon erilaisia arkkitehtonisia ja teknisiä ratkaisuja, ja mm. jakautua fyysisesti usean palvelimen kesken.

SOA-arkkitehtuurin soveltamisessa painottuu muita lueteltuja arkkitehtuureja enemmän se, että kyse on ohjelmistoympäristön kokonaisarkkitehtuurista. Uuden ohjelmistotuotteen rakentaminen kokonaan SOA-pohjaisena edellyttäisi, että asiakkaat ovat valinneet tai valitsemassa SOA-kokonaisratkaisun. AnyCase-ohjelmiston tapauksessa SOA-arkkitehtuurin saamat korkeat CBSP-pisteet kertovat, että SOA olisi täysin varteenotettava pohjaratkaisu etenkin silloin, jos ohjelmisto asennettaisiin suoraan asiakasorganisaation omaan tietojenkäsittely-ympäristöön. SaaS-jakelumallin käyttö kuitenkin muuttaa tilannetta: asiakkaalle tarjotaan sovelluksen käyttöä varten pelkästään selainpohjainen käyttöliittymä, ja kaikenlainen muu integraatio asiakkaan omaan tietojärjestelmään ei ainakaan suoralta kädeltä ole mahdollista (= teknisesti mahdollista, mutta kaupalliselta kannalta vaikeaa). Tässä tilanteessa SOA:n käytöstä ei olisi juuri hyötyä. Mutta SOA:n soveltuvuus sinänsä on tärkeä strateginen tieto: tarvittaessa ohjelmistosta voidaan tulevaisuudessa tarjota versioita, jotka integroituvat osittain tai kokonaan SOA-pohjaiseen ympäristöön.

5.2. AnyCase-järjestelmän arkkitehtuurin valinta CBSP-analyysin tuella

AnyCase-järjestelmän kohdalla on perusteltua päätyä siihen, että järjestelmän päätehtävä, työnkulkujen toteuttaminen, toteutetaan selainpohjaisen thin client -liittymän kautta. Myös työnkulkujen hallintatoiminnot on käytännössä järkevää toteuttaa samankaltaisen liittymän avulla, vaikka työnkulkumallien suunnittelutyökalu yksinään sopisi ehkä paremmin työasemapohjaiseksi ohjelmaksi. Näin ohjelmisto näkyy asiakkaan näkökulmasta helposti ymmärrettävänä ja käyttöönotettavana client-server -ratkaisuna. Työnkulkujen toteutus palvelinpäässä on puolestaan perusteltua toteuttaa tapahtuma- tai sanomapohjaisen arkkitehtuurin avulla. Jokainen työnkulun vaihe tuottaa sanoman, jonka perusteella logiikkaa toteuttava palvelukomponentti ohjaa työnkulkua eteenpäin, ja email-palvelu tarvittaessa tiedottaa käyttäjää seuraavasta tehtävästä. Tätä kautta sanomapohjaisuus konkretisoituu myös asiakkaalle. Ajastusta toteuttava palvelukomponentti tuottaa puolestaan ajan kulumiseen liittyviä sanomia, kuten muistutuksia määräaikojen umpeutumisesta. Tulevaisuuden tarpeita ajatelleen voidaan ajatella, että järjestelmän käyttämä sanomarajapinta rakennetaan alusta asti yhteensopivaksi SOA-arkkitehtuurissa yleisesti sovellettavien tekniikoiden kanssa. Käytännössä tämä tarkoittaa web service/SOAP (Simple Object Access Protocol) - palvelutekniikan ja sanomarakenteiden käyttöä. SOAP:n käyttö ei yksinään tee järjestelmästä SOA-pohjaista, mutta on itsessään helppo toteuttaa ja helpottaa SOA-arkkitehtuuriin siirtymistä myöhemmin.

Edelleen CBSP-analyysin pohjalta voidaan todeta, että kerrosarkkitehtuuri (layered-) soveltuu AnyCase-järjestelmään huonosti, vaikka kerrosajattelu yleisenä abstraktiona tuleekin helposti mieleen uuden järjestelmän mallinnusta aloitettaessa.

5.3. SaaS-jakelumallin erityispiirteiden huomioiminen arkkitehtuurissa

Kun opinnäytetyö on otsikoitu "Uuden SaaS-ohjelmistotuotteen suunnittelu..", niin voi tulla mieleen, miksi CBSP-vaatimusanalyysiä ei käytetty juuri SaaS-jakelumallin tuomien erityispiirteiden analysoimiseksi. Vastaus on, että SaaS-jakelumallin vaikutuksia ohjelmiston arkkitehtuuriin voi pitää siinä määrin kaavamaisia ohjelmiston tyypistä riip-

pumatta, että pelkästään tähän tarkoitukseen CBSP-analyysimenetelmän soveltaminen olisi ylimitoitettua. Näitä kaavamaisia vaikutuksia ovat mm.:

- Topologia: SaaS-jakelumallissa ohjelmiston instanssi ei ole olemassa yksinään, vaan ohjelmistosta on olemassa 0..n instanssia yhden juuri-instanssin alla. Tämän juuri-instanssin tehtävänä on mahdollistaa ohjelmistoinstanssien elinkaaren hallinta sekä niiden piilottaminen toisiltaan. Sama koskee tietomallia ja tietokantaa: tietomalliin tulee sisältyä se, että asiakaskohtaisia, toisistaan riippumattomia tietokantoja on useita, tai että asia ainakin näyttäytyy sillä tavoin asiakkaalle.
- Vaikutus tietoverkkoratkaisuun: SaaS-jakelumallin menestyksekäs kaupallinen käyttö yleensä edellyttää, että asiakas voi alkaa käyttää palvelua ilman merkittäviä muutoksia oman organisaation tietoverkkoratkaisussa. Perusoletus on, että asiakas voi käyttää palvelua selaimelta samalla tavoin kuin muitakin internet-palveluja. Muunlainen integroituminen asiakkaan omaan tai asiakkaan käytössä oleviin tietojärjestelmiin on mahdollista, mutta tällöin on puhe pitkälle menevästä yhteistyötä yksittäisen asiakkaan kanssa, jolloin SaaS-jakelumallia käytetään jonkin suuremman palvelukokonaisuuden osana.
- Vaikutus tietoturvaan: SaaS- ja pilvipalvelut, vaikka ovatkin käsitteellisesti eri asioita, herättävät asiakkaissa samankaltaisia tietoturvaan liittyviä kysymyksiä. Voiko yrityksen tai organisaation kriittisiä tietoja luovuttaa ulkopuolisen palvelun käsiteltäväksi? Tämä pakottaa viemään SaaS-palvelun teknisen turvallisuuden korkealle tasolle, mutta asiakkaan luottamuksen saamiseksi voidaan tarvita erilaisia rakenteellisia, kuten palvelun arkkitehtuuriin liittyviä toimia. Nämä voivat liittyä tietovaraston toteutustapaan ja varmenteiden käyttöön. Voidaan esimerkiksi järjestää, että asiakkaan kaikki aineisto salakirjoitetaan ennen fyysistä siirtoa SaaS-palveluun.
- Vaikutus skaalautuvuuteen. SaaS-palvelujen kohdalla skaalautuvuus tulee tarkasteluun kahdesta näkökulmasta: kuinka suuri kasvu tai vaihtelu on yksittäisen asiakkaan käyttövolyymissä, sekä kuinka nopeasti uusia asiakkaita saadaan. Jos palvelua myydään verkossa itsepalveluperiaatteella, saattaa palvelu saada asiakkaita odotetun kohderyhmän ulkopuolelta, esimerkiksi odotettua paljon suuremman asiakkaan. Jotta esim. pienen SaaS-palvelun tarjoajan olisi mahdollista varautua nopeisiin vaihteluihin käyttövolyymin kertaluokissa, hänen kannattaa itse turvautua pilvipalveluihin. Tämä puolestaan johtaa tietoturvakysymysten mut-

kistumiseen, mistä käytävää keskustelua voi seurata miltei päivittäin alan verkkolehdissä.

5.4. Yhteenveto

Tämän opinnäytetyön tavoitteena oli tuottaa tietoa, joka auttaisi AnyCase-ohjelmiston teknisessä suunnittelussa ja markkinoinnissa. Tiedonkeruun kohteet valittiin sen mukaan, mitkä ovat uuden ohjelmistotuotteen kannalta kriittisiä kohtia: ohjelmiston aiottu jakelumalli sekä ohjelmiston arkkitehtuuri. Suunnitelmissa oli sisällyttää työhön myös käyttöönotto-ominaisuuksien analyysi, mutta se oli käytännön syistä rajattava ulkopuolelle.

SaaS (Software-as-a-Service) -eli ohjelmistopalvelu-jakelumallista puhuttaessa markkinoiden mielikuvat ovat tällä hetkellä vähintään yhtä tärkeitä, elleivät jopa tärkeämpiä kuin tekniseen toteutustapaan liittyvät asiat. Verkosta käytettävien palvelujen tietoturvallisuuteen liittyvät riskit ja uhkakuvat ovat saaneet pontta alan jättiläisten, kuten Googlen, tavasta käsitellä asiakkaiden tietoja, tai oikeammin yhtiön tavasta jättää tällaisia asioita pelkkien arvailujen varaan. Yksityishenkilöiden kohdalla sama koskee sosiaalista mediaa ja siellä tapahtuvia kytköksiä kaupalliseen toimintaan, mistä tavallisille käyttäjille ei kerrota.

Oli näissä mielikuvissa perää tai ei, ne on otettava huomioon SaaS-tuotteen suunnittelussa. Asiakkaan luottamuksen voittamiseksi joudutaan tekemään enemmän. Palvelun tuottajan avoimuus on tärkeintä, asiakkaan on voitava tietää missä hänen tietojaan käsitellään ja miten. AnyCase-ohjelmiston kohdalla tähän tarpeeseen vastataan mm. tekemällä lähdekoodi avoimeksi.

Avoimuutta hankaloittaa kuitenkin se, että SaaS-palveluilta vaadittava skaalautuvuus houkuttelee palveluntuottajaa turvautumaan muihin SaaS- tai varsinaisiin pilvipalveluihin. Asiakkaan kannalta tämä johtaa juuri siihen ei-toivottuun tilanteeseen, jossa kukaan ei varsinaisesti tiedä, minne kaikkialle verkossa käsiteltävät tiedot saattavat päätyä. Myös vastuu palvelun jatkuvuudesta voi hämärtyä. AnyCase-pilottiasiakkaan ensimmäinen huoli oli, että järjestelmä ehkä "nielee" asiakirjat eikä anna niitä takaisin. Tämä

kuvastanee laajemminkin yritysten suhtautumista siihen ajatukseen, että organisaation tärkeät tiedot siirrettäisiin "pilveen".

SaaS-jakelumallilla on vaikutusta myös ohjelmiston arkkitehtuuriin. Jakelumalli edellyttää ohjelmistolta useimmissa tapauksissa niin syvällisiä rakenteellisia erityispiirteitä, että ei-SaaS -ohjelmiston muuttaminen SaaS-muotoon on iso työ, mikäli ylipäätään mahdollista. Ohjelmiston tulee tarjota useita rinnakkaisia, näennäisesti toisistaan riippumattomia instansseja. Ohjelmiston käyttöönoton tulee useimmissa tapauksissa voida tapahtua suoraan verkosta. Ohjelmiston ajoympäristön tarvitsema kapasiteetti voi vaihdella rajusti.

Edellä luetellut SaaS-jakelumallin erityispiirteet ovat olleet pintapuolisesti tiedossa AnyCase-ohjelmiston kehitystyön alusta asti, mutta tehty selvitys on antanut lisätietoa monista tekijöistä, jotka on otettava huomioon kun tuotetta aletaan tuoda markkinoille.

Toisena tarkastelukohteena oli AnyCase-ohjelmiston arkkitehtuuri. Arkkitehtuurin valinta ei tässä tapauksessa ole triviaali, koska ohjelmisto on uudentyyppinen. Vaikka työnkulkusovelluksia on markkinoilla, ovat ne suljettuja ja räätälöityjä järjestelmiä, joiden toteutuksesta ei anneta tietoja. Myös SaaS-jakelumallin käyttö vaikuttaa arkkitehtuuriin. Tästä syystä opinnäytteessä käytettiin koemielessä erikoistunutta CBSP-menetelmää, jolla ohjelmiston arkkitehtuuri voidaan johtaa ohjelmiston vaatimuksista tavalla, joka on myöhemmin jäljitettävissä. Menetelmä on tarpeellinen etenkin tilanteissa, joissa tietyn arkkitehtuurin valinta ei ole ilmeistä, ja kehityspanokset ovat suuria. Muunlaisissa, tavanomaisemmissa ohjelmistoprojekteissa menetelmä olisi turhan suuritöinen. Tämä opinnäytetyö tarjosi sopivat puitteet menetelmän käytölle, jolloin oli mahdollista tarkastella myös itse CBSP-menetelmää, sen käyttökelpoisuutta ja soveltuvuutta. Menetelmä perustuu siihen, että toisaalta järjestelmän vaatimuksiin ja toisaalta arkkitehtuureihin liittyy yhteisiä laadullisia attribuutteja, joiden esiintyminen ja painotus osoittavat yhteyden vaatimusten ja arkkitehtuurin välillä.

AnyCase-ohjelmistolle tehty CBSP-analyysi antoi selkänojaa jo tehdyille suunnittelupäätöksille (client-server -arkkitehtuuri), mutta lisäksi osoitti muidenkin arkkitehtuurityylien soveltuvuuden (sanomapohjaisuus, SOA). Edelleen menetelmä osoitti, mitkä arkkitehtuurityylit soveltuisivat käyttöön huonosti (mm. kerrosarkkitehtuuri). Analyysin

antama tieto oli hyödyllistä ja uskottavaa, ja se tulee varmasti vaikuttamaan jatkossa AnyCase-sovelluksen kehittämiseen.

Menetelmän käyttäminen täydessä laajuudessaan edellyttäisi, että kaikkien sidosryhmien edustajat yhdessä antavat asioille erilaisia painokertoimia. Tämä tarkoittaisi suurta työtuntien määrää, mistä syystä menetelmää ei käytetä kuin erikoistapauksissa. Menetelmään sisältyy kuitenkin mielekäs tapa muokata asiakkaan vaatimuksista abstraktioita, joiden väliset yhteydet todetaan, ja joihin liitetään huolellisesti valittuja ja painotettuja attribuutteja. Analyysimenetelmän tämä vaihe jo yksinään auttoi muodostamaan järjestelmästä kokonaiskuvan. Menetelmä on myös itse laajennettavissa ja siten pidettävissä ajan tasalla.

Tämä selvitys tulee vaikuttamaan AnyCase-ohjelmiston tekniseen toteutukseen monin sellaisin tavoin, joiden tulisi ilmetä käyttäjälle luotettavuutena, tasaisena suorituskykynä, yhteensopivuutena ja helppokäyttöisyytenä - toisin sanoen juuri niillä tavoilla, joita käyttäjä ei huomaa.

LÄHTEET

ASP-palvelut. Tieke ry 2011. Luettu 19.8.2011. http://www.tieke.fi/liiketoimintapalvelut/ict_klusteri/ict_klusterin_teemoja/asp-palvelut/

Bass,Len; Clements, Paul; Kazman, Rick 2003. Software Architecure in Practice, 2nd ed. USA: Addison-Wesley.

Extreme Programming 2012. Luettu 26.7.2012. http://www.extremeprogramming.org/map/project.html

Gartner: Gartner Newsroom 2011. Gartner Says SaaS Revenue Within the Enterprise Application Software Market to Total $9.2 Billion in 2010 (2010). Iso-Britannia. Luettu 19.8.2011. http://www.gartner.com/it/page.jsp?id=1492814

Goldsmith, Robin F 2004. Discovering real business requirements for software project success. Boston: Artech House.

Gorton, Ian 2011: Essential Software Architecture, 2nd ed. Berlin: Springer-Verlag.

Paul Grünbacher, Alexander Egyed, Nenad Medvidovic 2004. Reconciling software requirements and architectures with intermediate models. Software and System Modeling (ISSN 1619-1366) 2004, 235-253.

Eero Hyvönen (toim.) 2003. Ohjelmistoliiketoiminta. Helsinki: WSOY.

Kivelä, Ville 2010. Ohjelmistopalvelun evoluutio ASPista SaaSiin. Tampereen yliopisto. Pro gradu –tutkielma.

Wikipedia: Model-View-Controller 2012. Luettu 15.11.2012. http://en.wikipedia.org/wiki/Model-view-controller

Wikipedia: Multi-tier architecture 2012. Luettu 15.11.2012. http://en.wikipedia.org/wiki/Multitier_architecture

Netvisor Oy talushallinto 2011. Luettu 4.9.2011. http://www.netvisor.fi

Oracle Copr.: MVC model 2012. Luettu 27.7.2012. http://java.sun.com/blueprints/patterns/MVC-detailed.html

George Reese 2009. Cloud application architectures. Sebastopol (CA): O'Reilly.

Salesforce CRM 2011. Luettu 4.9.2011. http://www.salesforce.com

Salo, Immo 2010. Cloud computing - palvelut verkossa. Helsinki: WSOYPro Oy.

Sommerville, Ian; Sawyer, Pete 1997. Requirements Engineering: A Good Practice Guide. Chichester, England: John Wiley & Sons.

Markku Sääksjärvi, Aki Lassila, Henry Nordström 2005.
Evaluating the Software as a Service business model: From
CPU time-sharing to online innovation sharing. Proceedings of
the IADIS International Conference e-Society. Iadis assoc.: Qawra, Malta. Luettu 11.11.2012. http://www.iadis.net/dl/final_uploads/200505L024.pdf

Taylor, Richard N.; Medvidovic, nenad; Dashovy, Eric M. 2010. Software Architecture: Foundations, Theory, and Practice. Hoboken, USA: John Wiley & Sons.

Wiegers, Karl E 2003. Software requirements: practical techniques for gathering and managing requirements throughout the product development cycle / Karl E. Wiegers. 2nd ed. Redmond (Wash.): Microsoft Press.

Visma Oy taloushallinto 2011. Luettu 4.9.2011. http://www.visma.fi

LIITTEET

Liite 1: AnyCase-asiakasvatimukset

AnyCase-asianhallintajärjestelmä
Asiakasvaatimukset v.1.1.2012

Vaatimukset on kerätty haastattelujen, havainnoinnin ja asiakkaan laatimien prosessikaavioiden perusteella. Liitteenä kuvaus eräästä asiakkaan prosessista, jota järjestelmän avulla täytyy pystyä toteuttamaan. Erillistä käyttötapausanalyysiä ei tehty.

Vaatimuslähteet: tj) toiminnanjohtaja, hp) henkilöstöpäällikkö, mv) Mikko Valjakka

Prioriteetit: 1-pakollinen 2-toivottava [3-valinnainen]

Pilottiasiakkaan tarpeet yleisesti / tj

Järjestelmän käytön tarkoituksena on tehostaa asianhallinnan työnkulkuja automaation avulla. Työnkulut toteutetaan tällä hetkellä manuaalisesti. Järjestelmää pilotoidaan yhdistyksemme jäsenhakuprosessin avulla (kaavio). Prosessissa on n. 30 vaihetta. Pilotointi on voitava toteuttaa niin, että se koskee aluksi vain yhtä tapausta. Järjestelmään tulee sisältyä myös henkilörekisteri, joka mahdollistaa kansainvälisten työntekijöiden osoitetietojen ylläpidon sekä tarvitsemamme ryhmä- ja luokittelutiedot. Järjestelmän tulee olla sellainen, että asiakirjamme saadaan kopioitua sieltä pois milloin tahansa.

Ei-toiminnalliset (nf) vaatimukset

Käyttöympäristö (nf1)

ID	vaatimus	kuka	pri
nf1.1	Työnkulkuun osallistuminen ei saa edellyttää	hp	2

	työasemakohtaisia ohjelmistoasennuksia		
nf1.2	Työnkulkuun osallistuvalta työasemalta voidaan edellyttää verkkoyhteyttä, www-selainta, sähköpostia sekä kulloistenkin asiakirjojen tyypin mukaisia ohjelmistoja.	hp	1
nf1.3	Työnkulun osapuolet voivat olla myös oman organisaation ulkopuolella	hp	1
nf1.4	Asiakirjat voivat olla mitä tahansa tyyppiä	hp	1
nf1.5	Asiakirja voi olla myös viittaus sähköiseen tai fyysiseen dokumenttiin	mv	2
nf1.6	Järjestelmän on oltava monikielinen	hp	2
nf1.7	Kaikkien asiakirjojen tulee olla milloin tahansa siirrettävissä järjestelmästä ulos.	tj	1

Työnkulut ja asiakirjat (nf2)

ID	**vaatimus**	**kuka**	**pri**
nf2.1	Työnkulkuun voi liittyä yksi tai useampia asiakirjoja	hp	1
nf2.2	Asiakirja voi olla yhteinen usealle työnkululle	hp	2
nf2.3	Asiakirja tulee voida luoda - tyhjästä - pohjasta	hp	1
nf2.4	Asiakirja tulee voida luoda - usesta pohjaelementistä - muuntaen aiemmasta asiakirjasta - liitteenä aiempaan asiakirjaan	hp	2
nf2.5	Asiakirjoihin tulee voida liittyä tieto säilytysajasta	hp	2
nf2.6	Asiakirjoihin tulee voida liittyä tieto hyväksymisestä (yksi tai useampi henkilö) sekä allekirjoituksesta (yksi tai useampi henkilö)	hp	1
nf2.7	Asiakirjoihin tulee voida liittyä kommentteja/lausuntoja	hp	1
nf2.8	Järjestelmän tulee voida säilyttää kaikki asiakirjasta työnkulun aikana syntyneet versiot	hp	1
nf2.9	Työnkulun päättymisen yhteydessä asiakirjat tulee voida merkitä loppuunkäsitellyiksi ja/tai arkistoiduiksi sopivalla tavalla	hp	2
nf.2.10	Työnkulun päättymisen yhteydessä asiakirjat tulee voida siirtää osaksi toista työnkulkua	hp	2
nf2.1	Asiakirjojen näkyyvyyttä tulee voida säädellä työnkulkuun	hp	1

1	liittyvien roolien perusteella.		
nf2.12	Asiakirjaan tulee voida liittää tieto luottamuksellisuudesta	hp	2
nf2.13	Järjestelmän tulee mahdollistaa työnkulut, jotka ovat kuvattavissa prosessi- tai vuokaavion avulla: yksittäiset tehtävät tulee voida suorittaa peräkkäin, ehdollisesti tai toistuvasti.	hp	1
nf2.14	Asiakkaan tulee itse voida määritellä haluamansa työnkulut ja muuttaa niitä.	hp	1
nf2.15	Työnkulkukaavan tulee olla muokattavissa myös työnkulun ollessa käynnissä.	mv	2
nf2.16	Työnkulkukaavojen tulee olla kopioitavissa	hp	1
nf2.17	Työnkulkuun tulee aina liittää vastuuhenkilö	hp	1
nf2.18	Työnkulku tulee voida jäädyttää toistaiseksi tai keskeyttää kokonaan	hp	1

Työnkulun yksittäinen tehtävä (nf3)

ID	**vaatimus**	**kuka**	**pri**
nf3.1	Toimijan on saatava tieto työnkulkuun liittyvästä tehtävästä sekä sähköpostin että www-liittymän kautta	hp	1
nf3.2	Tehtävään tulee voida liittää tieto toimijasta, vapaamuotoinen kehote, viittaukset tarpeellisiin asiakirjoihin sekä määräaika.	hp	1
nf3.3	Tehtävään tulee voida liittää seuraavat toimenpidepyynnöt asiakirjoille: - tiedoksi antaminen - täytettäväksi/täydennettäväksi - hyväksyttäväksi - kommentoitavaksi/lausunnon anto	hp	1
nf3.4	Tehtävän toimenpidepyyntönä voi olla asiakirjan allekirjoittaminen	hp	2
nf3.5	Toimijan on voitava saada automaattinen muistutus tehtävän laiminlyönnistä määräajan puitteissa	hp	1
nf3.6	Tehtävän sisältönä voi olla vapaamuotoinen kysymys, johon toimija vastaa. Vastaus voi ohjata työnkulkua.	mv	1

nf3.7	Tehtävän suorittajana (toimijana) voi olla henkilö, roolin haltija tai ryhmä	mv	1

Käyttäjärekisteri (nf4)

ID	vaatimus	kuka	pri
nf4.1	Käyttäjärekisteriin on voitava tallettaa yksittäisestä henkilöstä - nimi, id, hetu, puhelin, email, osoitetiedot, organisaatio, tehtävä organisaatiossa, ryhmiin kuuluminen, tilatietoja, lisätietoja	hp	1
nf4.2	Henkilöllä voi olla useita osoitteita, joista yksi on aktiivinen	hp	1
nf4.3	Organisaatioista on voitava tallettaa nimi, email, osoite, yhteyshenkilö, puhelin	hp	1
nf4.4	Ryhmille on voitava antaa nimi, organisaatio- ja statustieto	hp	1

Liittymät (nf5)

ID	vaatimus	kuka	pri
nf5.1	Käyttäjärekisterin käyttäjätiedot on voitava hakea asiakkaan omasta hakemistopalvelusta esim. LDAP-protokollalla	mv	2
nf5.2	Asiakirjat on voitava kopioida kattojärjestön sisäiseen tietopalveluun "Insite"	hp	2

Tietoturva (nf6)

ID	vaatimus	kuka	pri
nf6.1	Palvelun tietoliikenteen tulee olla salattu.	mv	1
nf6.2	Palvelun käyttäjät on tunnistettava.	hp	1
nf6.3	Tunnistustasoja on oltava: - palvelun oma käyttäjätunnus ja salasana - sähköpostitunnistus	mv	1
nf6.4	Tunnistustasoja on oltava: - tunnistus organisaation omien lähiverkkotunnusten avulla	mv	2
nf6.5	Järjestelmän tulee pitää lokia käyttäjän toiminnasta	mv	2
nf6.6	Järjestelmään talletetetut asiakirjat ja muut tiedot on varmistuskopioitava säännöllisesti	mv	1
nf6.7	Kaikkien järjestelmään tallennettujen asiakirjojen tulee olla saatavilla milloin tahansa.	mv	1

Toiminnalliset vaatimukset

Työnkulkuun liittyvän yksittäisen tehtävän suorittaminen

ID	vaatimus	kuka	pri
f1.1	Tehtävän suorittajan tulee saada ilmoitus tehtävästä sähköpostiinsa.	hp	1
f1.2	Tehtävän suorittajan tulee saada toistuvia muistutuksia sähköpostiinsa, jos hän on laiminlyönyt tehtävän suorittamisen määräajassa.	hp	1
f1.3	Sähköpostiin tulevassa ilmoituksessa on oltava linkki, josta tehtävän suorittajalle avautuu www-lomake tehtävän suorittamista varten (=tehtävälomake)	hp	1
f1.4	Ennen pääsyä tehtävälomakkeelle käyttäjä on autentikoitava, ellei autentikointitasoksi ole määritelty pelkkää sähköpostia.	hp	1
f1.5	Käyttäjällä on oltava pääsy tehtävälomakkeelle myös suoraan www-palvelun kautta	hp	1
f1.6	Tehtäväsivulla on oltava seuraavat asiat: - ohjeet tehtävän suorittamiseksi - linkki/linkit, joista aukeavat tehtävään liittyvät asiakirjat sekä niiden mahdolliset aiemmat versiot - tehtävän asetuksista riippuen näkymä koko asianhallintakansioon - kohta, johon muokatut asiakirjat voi ladata	hp	1
f1.7	Tehtäväsivulla on oltava seuraavat asiat: - ohjeet käyttäjän äidinkielen mukaan	hp	2
f1.8	Tehtäväsivulla on oltava mahdollisuus tehdä vapaamuotoinen merkintä asianhallintamuistioon	mv	1
f1.9	Kysymys-tyyppisellä tehtäväsivulla on oltava: - kysymys ja vastausvaihtoehdot - linkit tarvittaviin asikirjoihin tai koko	mv	1

	asianhallintakansioon		
f1.10			

Työnkulun ohjaaminen

ID	vaatimus	kuka	pri
f2.1	Järjestelmän pääkäyttäjän tulee voida osoittaa työnkululle ohjaaja	mv	1
f2.2	Ohjaajan tulee voida käynnistää uusi työnkulku valitsemansa työnkulkumallin pohjalta.	mv	1
f2.3	Työnkulun käynnistyksen yhteydessä ohjaajan tulee voida syöttää - uuden työnkulun ohjaaja (ellei itse) - asiakirjapohjat - muut työnkulkumallissa esitiedoiksi määritellyt tiedot		
f2.4	Palvelun tulee tarjota työnkulun ohjaajalle näkymä, jossa hän voi - nähdä työnkulkujen vaiheen ja tilan - nähdä korostetusti ne työnkulut, joissa on puuttumisen tarvetta, kuten viivästyksiä tai virheitä - keskeyttää/jäädyttää työnkulun - uudelleenaktivoida jäädytetty työnkulku - pakottaa tehtävän suoritusta odottava työnkulku eteenpäin	mv	1
f2.5	Työnkulun tulee edetä automaattisesti seuraavissa tilanteissa: - työnkulkuun liittyvä tehtävä on suoritettu - työnkulkuun liittyvä ajoituksen perusteella	mv	1
f2.6	Työnkulun tulee pysähtyä automaattisesti seuraavissa tilanteissa - järjestelmän havaitessa tietoihin liittyvän virheen, esim. virheellisen sähköpostiosoitteen -> ilmoitus työnkulun ohjaajalle - järjestelmän havaitessa virheen työnkulkumallissa -> ilmoitus pääkäyttäjälle	mv	1
f2.7	Työnkulun päättyessä työnkulun ohjaajan tulee voida	hp	1

	kopioida tai siirtää asiakirjat ja asianhallintamuistio - pois järjestelmästä - tai osaksi toista työnkulkua		

Johdon näkymä

ID	vaatimus	kuka	pri
f3.1	Järjestelmän tulee tarjota organisaation johdolle näkymä, josta työnkulkujen etenemistä voi seurata.	mv	1
f3.2	Mainitusta (f3.1) näkymästä tulee käydä ilmi - työnkulun nimi- ja tunnistetiedot sekä ohjaaja - työnkulun vaihe, vastuuhenkilö ja tila - selvästi erottuen mahdolliset erikoistilanteet, kuten viivästyminen tai virhe	mv	1
f3.3	Mainitusta (f3.1) näkymästä tulee lisäksi käydä ilmi - päättyneet työnkulut ja niihin liittyvät tilastotiedot, kuten kokonaiskesto ja yksittäisten tehtävien kesto	mv	2

Työnkulkujen määrittely

ID	vaatimus	kuka	pri
f4.1	Järjestelmän pääkäyttäjän tulee voida luoda järjestelmään uusi työnkulkumalli.	mv	1
f4.2	Järjestelmän pääkäyttäjän tai hänen valtuuttamansa työnkulun ohjaajan tulee voida käynnistää uusi työnkulku työnkulkumallin pohjalta.	hp	1
f4.3	Järjestelmän tulee käyttöönottovaiheessa tarjota valmiita työnkulkumalleja, joita voi käyttää sellaisenaan tai joiden pohjalta voi muokata omia malleja	mv	1
f4.4	Järjestelmään tulee voida ladata ulkopuolisia työnkulkumalleja	mv	2
f4.5	Työnkulkumallille on voitava antaa vapaamuotoinen nimi ja kuvaus	mv	1
f4.6	Työnkulkumallit on voitava laatia graafisesti prosessi- tai	mv	1

	vuokaaviona		
f4.7	Työnkulkumallin osaksi tulee voida ladata 0..n kpl asiakirjapohjia tai niiden viitteitä	hp	1
f4.8	Työnkulkumallin prosessikuvauksen tulee voida liittää seuraavat toiminnot: - työnkulun alku - työnkulun loppu - tehtävä (tehtävän attribuutit on lueteltu kohdassa nf3) - kysymys-tyyppinen tehtävä (attribuutit ks. nf3) - valinnainen haarautuminen kysymykseen annetun vastauksen perusteella - valinnainen haarautuminen muun tilatiedon perusteella, kuten aika tai toistokertojen määrä - tapa kuvata tehtävien suoritus rinnakkain sekä rinnakkaisten haarojen yhtyminen - ajastus: työnkulku odottaa tiettyä päivämäärää - mahdolliset muut toiminnot, joita tarvitaan tavanomaisten prosessikuvausten mukaisten työnkulkujen kuvaamiseksi	mv	1
f4.9	Kohdassa f4.8 lueteltuja työnkulkumallin toimintoja tulee voida muokata, lisätä ja poistaa	mv	1
f4.10	Työnkulkumallin tulee olla muokattavissa jo käynnistetyille työnkuluille	mv	1
f4.11	Jos aktivoidun työnkulun mallia muutetaan, muutoksen on oletusarvoisesti koskettava vain kyseistä työnkulkua.	mv	1
f4.12	Työnkulkumalliin liittyville muuttujille (attribuuteille) on voitava asettaa arvo soveltuvilta osin seuraavilla tavoilla: - vakiona, jolloin arvo sisältyy itse työnkulkumalliin - työnkulkukohtaisesti, jolloin järjestelmä kysyy arvon työnkulun ohjaajalta työnkulun käynnistyessä (esim. määräaika, kohdehenkilö, asiakirja) - tapauskohtaisesti, jolloin järjestelmä kysyy arvoa	mv	1

	työnkulun ohjaajalta aina kun siihen viitataan, esim.		
f4.13	Pääkäyttäjän tulee voida simuloida työnkulkumallia reaaliaikaisesti niin, että työnkulun eteneminen suunnitellulla tavalla voidaan todeta.	mv	1

Pääkäyttäjän toiminnot

ID	**vaatimus**	**kuka**	**pri**
f5.1	Järjestelmällä tulee olla ainakin yksi pääkäyttäjä	hp	1
f5.2	Pääkäyttäjän tulee voida luoda uusia käyttäjiä	hp	1
f5.3	Pääkäyttäjän tulee voida määritellä uusia työnkulkumalleja	hp	1
f5.4	Pääkäyttäjän tulee voida muokata ja poistaa työnkulkumalleja	hp	1
f5.5	Työnkulkumallin poistaminen ei saa vaikuttaa meneillään oleviin työnkulkuihin	mv	1
f5.5	Pääkäyttäjän tulee voida käynnistää uusia työnkulkuja ja osoittaa niille ohjaaja	mv	1
f5.6	Pääkäyttäjällä tulee olla näkymä, jossa - näkyvät aktiiviset ja päättyneet työnkulut - aktiivisista työnkuluista näkyy nimi ja tunnistetiedot, ohjaaja, aktiivinen vaihe ja vastuuhenkilö, tila - pääkäyttäjän tulee voida pysäyttää tai keskeyttää työnkulku	mv	1
f5.7	Pääkäyttäjän tulee voida muokata ja poistaa käyttäjiä sekä vaihtaa näiden salasana ja tunnistustaso	mv	1
f5.8	Pääkäyttäjän tulee voida poistaa työnkulku asiakirjoineen. Poistotoimintoon tulee liittää asianmukaiset varotoimet.	mv	1
f5.9	Pääkäyttäjän tulee voida lisätä, päivittää ja poistaa työnkulmuallien asiakirjapohjia sekä työnkulkujen asiakirjoja.	mv	1

Peruskäyttäjän toiminnot

ID	**vaatimus**	**kuka**	**pri**
f6.1	Peruskäyttäjällä tulee olla käyttöliittymä, johin hän pääsee kirjautumisen kautta	mv	1

f6.2	Peruskäyttäjän käyttöliittymässä ovat seuraavat toiminnot: - pääsy tapahtumiin, jotka odottavat käyttäjän toimenpiteitä (f1) - luettelo suoritetuista tehtävistä - mahdollisuus päivittää omat yhteystiedot	mv	1