Teksti: Tarja Raussi
Pentti Virtanen

Vaatimusmäärittelyä ketterästi vai perinteisesti?

Tarja Raussi on pitkän linjan systeemityöammattilainen, joka kouluttaa ja konsultoi Tieturi Oy:ssä mm. prosessien mallintamista ja vaatimusten määrittelyä ja hallintaa. Vaikka UML käyttötapauksineen onkin Tarjan toinen kotimainen kieli, hänen mielestään vaatimusmäärittelyn tavan valintaan vaikuttaa kulloinenkin tilanne.

Pentti Virtanen on toiminut ohjelmistokehittäjänä, projektipäällikkönä ja ohjelmistotuotannon prosessien kehittäjänä vuodesta 1981. Hän on tietojenkäsittelytieteen filosofian tohtori ja sertifioitu Scrum Practitioner, joka kouluttaa ja konsultoi Tieturi Oy:ssä mm. ketteriä menetelmiä, ohjelmistoarkkitehtuureja ja tietoturvaa.

Onko ketterän ja perinteisen vaatimusmäärittelyn välillä oikeasti eroja? Eivätkö molemmat tähtää laadukkaaseen lopputulokseen? Vai voiko kummallakin tavalla saada sutta aikaiseksi?

Vaatimusmäärittelyn merkitys projektin onnistumiselle on tunnustettu Standish Groupin tutkimuksissa jo yli 10 vuoden ajalta. Koko ajan tärkeimpänä kriteerinä on ollut käyttäjien osallistuminen - se on tuoreimmassakin tutkimuksessa numero 1. Samoin vaatimusten selkeyttä ja muutosten hallintaa on painotettu. Näitä ei kukaan kiistäne.

Vaan miten vaatimusmäärittelyä pitäisi tehdä? Pitäisikö kaikista toiminnallista vaatimuksista tehdä käyttötapaukset? Riittäisikö Scrumin tyyppinen tuotoslista? Onko siinä riittävästi informaatiota? Keskustelua käyvät perinteistä vaatimusmäärittelyä kannattava Hannele ja ketteryyttä suosiva Pena.

Dokumentoinnista

Hannele: Kunnon kuvaukset olla pitää!

Vasta jos vaatimukset on kunnolla kuvattu proosana tai käyttötapausten avulla, voi toteutuksesta tulla jotakin järkevää. Eihän koodari voi mitenkään tajuta jostain yksittäisistä lauseista tai parin rivin kuvauksista mitä hänen pitäisi tehdä. Ja mahtaisiko silloin projektipäälliköllekään olla kokonaiskuvaa järjestelmän laajuudesta ja monimutkaisuudesta?

Pena: Tuotoslistaan kirjoitetaan tosiaankin vain käyttäjätarina yhdellä virkkeellä mutta se ole kaikki. Ketterä ohjelmistotuotanto perustuu jatkuvaan vuorovaikutukseen asiakkaan kanssa. Määrittelyä tekevä asiakas on mukana työssä koko sen ajan tarkentaen ja ohjaten sitä ohjelmiston kehittyessä. Suullinen, vuorovaikutteinen kommunikaatio vähentää väärinymmärryksiä verrattuna perinteiseen kirjalliseen vaatimusmäärittelyyn.

Määrittelyä tarkennetaan myös luomalla testitapaukset hetikohta työn alkaessa. Asiakkaalla on tässä usein apuna ammattitestaaja, joka osaa täsmentää vaatimukset yksiselitteisillä testitapauksilla.

Ketterässä kehityksessä projekti ei viivästy odottaessaan kokonaisvaltaisen vaatimusmäärittelydokumentin valmistumista, johon voi pahimmassa tapauksessa mennä useita vuosia. Dilbert ei saanut alkua pitemmälle kolmessa vuodessakaan

Hannele: ”Asiakas on mukana ohjaten...” Hmm... Silloin se vaatisi aika monta henkilöä, kun käytännössä isojen järjestelmien parissa yksi käyttäjä tekee vain tiettyjä hommia. Eikä välttämättä kellään ole selkeää kokonaiskuvaa kaikista osa-alueista. Tämä viimeksi mainittu tuntuu tulevan jatkossa yhä enemmän esille, sillä kun jatkuvasti palkataan vain määräaikaisia, niin eihän kenellekään ennätä kehittyä mitään kokonaiskuvaa. Ja varsinkaan silloin, jos projektia tehdään ketterästi eli ei dokumentoida ollenkaan. Dokumentista voisi katsoa, miten asiat liittyvät toisiinsa.

Osaako käyttäjä kertoa toteuttajalle tarvittavat säännöt riittävän yksiselitteisesti? Kehittäjä kun usein tulee jostain it-talosta eikä ole koskaan edes kuullut moisesta sovellusalueesta.

Itse olen törmännyt juuri tässä kohdin ongelmaan: vaikka kuinka yritetään laittaa sääntöjä selkeästi kuvauksiin, niin kuitenkin aukkoja jää aina.

Pena: Ketteryyden rinnastaminen dokumentoimattomuuteen on varsin yleinen väärinkäsitys. Ketterät projektit dokumentoivat, mutta tekevät sen eri tavalla. Lähdekoodi on ohjelmoijalle edelleenkin tärkein tiedon lähde. Sen laatuun kiinnitetään paljon huomiota. Käyttäjille pitää edelleen tehdä käyttöohjeet. Lisäksi ketterässä työtavassa syntyvät testitapaukset, jotka esimerkkien avulla kertovat kuinka ohjelmiston pitäisi toimia. Testitapaukset säilytetään samassa paikassa lähdekoodien versionhallinnassa kanssa, jolloin ne myös löytyvät - mitä ei voi aina sanoa perinteisistä dokumenteista.

Työn pilkkominen pieniin merkityksettömiin sirpaleisiin on eräs perinteisen työtavan ongelmista, joihin ketteryyden serkku, laiha ohjelmistotuotanto (Lean Software Development) antaa lääkkeitä. Suuri joukko ihmisiä sähläämässä kymmenien projektien kanssa yhtä aikaa ei voi olla menestyksellinen tapa tehdä töitä. Ketteryyden käyttöönoton tarkoituksenakin on tuoda organisaation ongelmat näkyville. Kertomasi on yksi korjausta kaipaavista ongelmista.

Mutta palatakseni vaatimusmäärittelyyn, mikä onkaan ison vaatimusspeksin arvo dokumenttina olettaen ensin, että löydät oikean version siitä. Todennäköisesti se on vanhentunut ja ohjelmiston toteutus on aivan toisenlainen.

Hannele: Niin, viittasin kyllä omassa kommentissani siihen oikeaan liiketoimintatyöhön, en projektiin. Harvempi ihminen tekee työssään hienosti prosessia päästä päähän. Pahimmillaan se oma työ voi olla yksittäisiä työtehtäviä, jotka sijoittuvat ihan eri prosesseihin. Ja silloin on kyllä vaikea nähdä omaa osuuttaan prosessissa.

Joo, tuo mitä sanoit dokumenttien versioista, pitää kyllä paikkansa - valitettavasti. Liian usein käy niin, että vaatimusmäärittelyn dokumentti tehdään ja sitten se unohdetaan pölyttymään hyllyyn. Vaatimusmäärittelykuvaustenhan pitäisi olla aina ajan tasalla, sillä ne kertovat, mitä liiketoiminta haluaa järjestelmän tekevän.

Silloinhan on helppoa ottaa aina viimeisin versio. Sillä ihan samalla tavalla kuin kerroit testitapausten menevän ketterässä koodin mukana versionhallintaan, niin vaatimusdokumentaatiokin voidaan laittaa versionhallintaan. Eli samassa paketissa on määrittelyt ja niitä vastaava koodi + muut tarvittava aineisto.

Vaatimusmäärittelykuvauksia voidaan käyttää myös uusien ihmisten perehdyttämiseen ko. järjestelmään. Näkyyhän esim. käyttötapauskaaviosta suoraan, kuka saa käyttää mitäkin toiminnallisuutta.

Pena: Vaatimusmäärittelydokumenttien käyttö perehdyttämisessä järjestelmään perustuu olettamukseen, että vaatimusdokumentit ja lopullinen järjestelmä ovat yhdenmukaiset. Tämä on harvoin totta, sillä kehitettäessä uutta tuotetta tai ylipäätään mitä tahansa ohjelmistoa, on luonnollista ja hyödyllistä, että hyvät ideat otetaan käyttöön ja ristiriitaisuudet korjataan. Tällöin vaatimusdokumenttien ylläpidosta tulisi kokopäivätyö. Reaalimaailmassa tämä jätetään tekemättä.

Ohjelmistoon perehdyttämisessä kannattaa käyttää käyttäjien osalta käyttö-ohjeita, jotka tietysti tehdään myös ketterissä projekteissa ja kehittäjien osalta lähdekoodia ja niitä tukevia malleja. Etenkin Extreme Programming kiinnittää valtavasti huomiota koodin luettavuuteen ja ymmärrettävyyteen. Mallitkaan eivät ole pannassa ketterässä ohjelmistotuotannossa. Ne eivät kuitenkaan ole itsetarkoitus vaan kiinteä osa kehitystyötä.

Liiketoimintaprosessien kuvauksia käytetään ihmisten työn ohjaamiseen myös ketterässä ohjelmistotuotannossa. Erona on se, että ne syntyvät projektin aikana ei kauan sitä ennen. Koska myös liiketoimintaprosessit muuttuvat sitä mukaan kun opitaan uutta, kannattaa niiden sisältö jäädyttää niin myöhään kuin mahdollista.

Hannele: Käyttäjien perehdyttämiseen tietysti käytetään käyttöohjetta. Ja koulutusmateriaaleja.

Mutta ajattelinkin lähinnä sovellusohjelman ylläpitäjiä. Vie tolkuttomasti aikaa, jos koodaaja lähtee etsimään koodista, mihin kohtaan muutokset tulevat. Ja siks toiseks, kun muutostyötä tilataan ylläpidossa, vaatimukset pitää kuitenkin kuvata. Onhan silloin helpompi, jos voi viitata olemassa olevaan kuvaukseen ja kertoa, mikä toiminnallisuudessa muuttuu. Mutta kuten sanoit, valitettavasti reaalimaailmassa tuo dokumenttien päivitys kyllä tahtoo unohtua, jos sitä ei ole otettu normaaliksi työtavaksi.

Eivät kai liiketoimintaprosessit sen takia muutu, että projektissa opitaan uutta???!!! Prosessi on prosessi ja se muuttuu vasta, kun prosessia kehitetään tai liiketoiminta muuttuu.

Artikkelin alussa viitataan Standish Groupin CHAOS 2004 –tutkimukseen, josta osia on julkaistu netissä Standish Groupin Jim Johnsonin haastattelussa 25.8.2006 http://www.infoq.com/articles/Interview-Johnson-Standish-CHAOS

Ketterästi vai perinteisesti?

Tarkan tason työnkulku voi kyllä hioutua projektin aikana - varsinkin jos on kyse valmisohjelmiston käyttöönotosta. Liian usein on käynyt niin, että kun kuvittelimme saavamme säästöjä ja hyötyä valmisohjelmistosta, niin sen toimintatapa onkin ihan erilainen ja jopa hankala. Ja sitten käyttäjät manailevat, kun ohjelman käyttö on hankalaa.

Luettavuus ja ymmärrettävyys

Pena: Koska ohjelmakoodi on todellakin tarkoitettu ohjelmoijien luettavaksi, kiinnitetään sen luettavuuteen ja laatuun erityistä huomiota. Olen olemassa koodausstandardit, jota tiimin jäsenet valvovat tekemällä töitä pareina ja antamalla kaikkien tiimin jäsenten tehdä työtä koko koodin kanssa. Pariohjelmointi on erinomainen tapa tehdä koodikatselmointia.

Isossa ohjelmistossa tarvitaan tietenkin myös malleja ja pakkausrakenne, joka ohjaa seuraavaa tehtävää tekevän ohjelmoijan suoraan muutosta tarvitsevaan osaan ohjelmistossa. Malleja ja ohjelmiston rakennetta ei kuitenkaan jäädytetä liian aikaisin vaan sitä mukaa kun ohjelmisto kehittyy. Mallien tekemisen pitää palvella ohjelmiston rakentamista, eikä olla oma tarpeettomia paperidokumentteja tuottava norsunluutorni.

Hannele: Joo, olen minäkin urani aikana nähnyt sellaisia tarpeettomia dokumentteja, mutta myös tarpeellisia.

Olen kokenut hyväksi toiminnallisuuden kuvauksessa käyttää käyttötapauksia - ne todella toimivat. Saadaan selkeästi kuvattua, mitä käyttäjä tekee ja mitä järjestelmä tekee. Ja lisäksi tulee kuvattua, mitä järjestelmän sisällä pitää tapahtua, eli käsittelysäännöt ja virhetarkistukset.

Joskus olen kyllä huomannut, että käyttäjät kuvaavat tuonne helposti myös manuaalisia juttuja eli kuvaavat myös sellaista mitä ei ole tarkoitus toteuttaa järjestelmään.

Onkos tähän ketterällä puolella mitään ratkaisua? Vai mennäänkö vain suusanallisen pohjalta tekemään mikä-se-olikaan tuotoslistaa?

Pena: Tarkoitat varmaan käyttäjätarinoita (user story)? Ne ovat yksinkertaisia virkkeitä, joiden avulla kerrotaan, mitä ohjelmiston pitää sisältää. Ne kirjoitetaan tuotoslistalle muistilapuksi ja niitä täydennetään suusanallisesti, mutta muistiinpanojen tekemistä ei tietenkään ole missään kielletty. Mm. digikuvat valkotauluille piirretyistä käyttöliittymäluonnoksista ovat suosittuja.

Scrum on hyvin yksinkertainen kehys, joka ei sisällä ohjeita siitä, miten insinöörikäytännöt pitää hoitaa. Käyttäjien manuaalisen toiminnan kuvaaminen on asia, jonka tekotavan projekti päättää. Joissain organisaatioissa käytetään kirjallisia toimenkuvia, joissain taas ei.

Ei-toiminnalliset vaatimukset ja arkkitehtuuri

Hannele: Hankalampi juttu on nuo suorituskyky- yms. teknisemmät vaatimukset. Niitä on kyllä aika vaikea kuvata. Ja olen huomannut, etteivät toimittajat tahdo sitoutua niihin mitenkään.

Pena: Arkkitehtuuri on tärkeässä osassa ketterässä ohjelmistotuotannossa. Se ei kuitenkaan ole norsunluutorneissa tehty PowerPoint-arkkitehtuuri vaan konkreettisesti yhteydessä ohjelmiston tekemiseen. Ennen kuin tuotoslistaa (product backlog) pääsee kunnolla tekemään, tarvitaan arkkitehtuurivisio: näkemys siitä mitä kaiken kaikkiaan ollaan tekemässä. Se voi olla yksi A3, joka kertoo mistä kokonaisuudesta on kysymys ja miten se liittyy ympäröiviin järjestelmiin.

PowerPoint-arkkitehtuurin vastakohta on suoritettava arkkitehtuuri, joka yksinkertaisimmillaan sisältää ohjelmointikielellä tehdyt rajapintamääritykset, integraatiotestit ja stubit, jotka toteuttavat integraatiotestit. Jatkuvasti ajettavat integraatiotestit varmistavat kehitystyön yhden- ja arkkitehtuurin mukaisuuden työn edetessä.

Ei-toiminnallisten vaatimusten testaaminen tapahtuu luonnollisena osana ensimmäisiä työjaksoja (Sprint). Scrumissa pääosa ensimmäisten työjaksojen työstä menee arkkitehtuuriin ja infrastruktuuriin. Koska mukana on kuitenkin aina myös liiketoimintaa hyödyntävää toiminnallisuutta, tulee arkkitehtuuri testattua jo ensimmäisten työjaksojen aikana.

Hannele: Arkkitehtuuri on sitten arkkitehtien asia.

Vaatimusmäärittelyssä me keskitytään siihen, millaista toiminnallisuutta ja tietosisältöä haluamme järjestelmään. Tosin tuleehan siellä esille myös niitä nopeita vasteaikoja ja ”pitää noudattaa java-arkkitehtuuria” yms. asioita. Minusta siinä onkin ristiriita, että yleensä opetetaan, että arkkitehtuurin voi luoda vasta kun vaatimukset on kuvattu, mutta eihän uusi järjestelmä tule tyhjiöön. Siellähän on jo muut järjestelmät ja ehkä jopa sovittu välineistö mitä pitää käyttää. Lisäksi useimmat projektit ovat ylläpitoprojekteja - olemassa olevaa sovellusta kehitetään, jolloin arkkitehtuuri on jo olemassa. MItenkäs tuo mainitsemasi Scrum tähän suhtautuu?

PowerPoint on ihan hyvä väline, sillä se löytyy useimmilta! Jos arkkitehtuuria ei ole kuvattu mitenkään, niin pitääkös taas etsiskellä koodia, ennen kuin tietää mitä pitää korjata. Meillä ainakin arkkitehdit antavat myös toteutusohjeita mm. miten koodi jaetaan moduleihin jne. Usein arkkitehti kulkeekin toteutuksen projektipäällikkö -nimikkeellä. Koodarit tarvitsevat ohjausta, muutenhan sovellus on yksi tilkkutäkki, jos jokainen tekee tyylillään.

Pena: Vaatimusten kirjoittaminen kokonaan irrallaan arkkitehtuurista on haasteellinen tehtävä, sillä nämä riippuvat voimakkaasti toisistaan. Esimerkiksi asiakas saattaa vaatia miljoonaa yhtäaikaista käyttäjää alle sekunnin vasteajalla kunnes kuulee kuinka kalliiksi tällaisen toteuttaminen tulee. Helposti tulee myös vaadittua asioita, joiden toteuttaminen on kohtuuttoman hankalaa. Pienellä muutoksella vaatimuksiin voidaan saada suuria parannuksia arkkitehtuuriin ja siten myös toteutukseen.

Scrum soveltuu luonnostaan ylläpitoprojekteihin. Muutostarpeet kirjataan tuotoslistaan, priorisoidaan ja toimitetaan kuukausittaisissa toimituserissä. Olemassa olevaa arkkitehtuuria voidaan tietysti käyttää myös Scrumia käyttävässä ylläpitoprojektissa.

PowerPoint on tarkoitettu diaesitysten tekoon. Ohjelmistoarkkitehtuurien kuvaamiseen käytetään yleensä UML-mallinnusnotaatiota, jolle on olemassa hyvä, muihin ohjelmistotuotannon työkaluihin integroitu työvälinetuki. Loogisen arkkitehtuurin määritteleminen kokonaisvaltaisesti etukäteen on käytännössä mahdotonta aivan samalla tavalla kuin toimistokäyttäjälläkin kansiorakenteen ennakointi. Niinpä liikkeelle lähdetään arkkitehtuurin visiolla, jota sitten täydennetään työn edetessä. Kuten huomasitkin, arkkitehtina toimiva tekninen projektipäällikkö on mukana koko projektin ajan ja pystyy ohjaamaan työtä sekä suullisesti, että ohjelmiston rakenteita kehittäen.

Sopiiko ketteryys kaikkeen?

Hannele: Eihän tuo Scrum ihan mahdottomalta kuulostakaan, jos sitä kerran tuolla tavoin hallitusti tehdään...

Itse kun olen törmännyt vain sellaisiin ketteryyden nimiin vannoviin, jotka sanovat ettei mitään tarvitse dokumentoida tai arkkitehtuuriakaan luoda, vaan tehdään vain. Tuli eittämättä mieleen nuo 80-luvun RAD-jutut, jotka olivat enemmän ”hutaisten tehty” -juttuja.

Mutta taitaa olla niin, ettei tuo ketteryyskään joka paikkaan sovi?

Pena: Tilanneriippuvaistahan tämä on. Esimerkiksi pienissä ylläpitotöissä työn ennustettavuus on aika hyvä, sillä tekninen ympäristö, tekijät ja tilaajat ovat kaikki samoja kuin ennenkin. Silloin on mahdollista kirjoittaa vaatimukset dokumenttina ja saada aivan kelvollinen muutos ohjelmistoon. Ketteryys taas on parhaimmillaan silloin, kun ei tiedetä mitä tarvitsee tehdä puhumattakaan siitä miten se pitäisi tehdä.

Jälkikirjoitus:

Tämä artikkelin pohja luotiin ketterästi Tieturin web-sivuston keskustelupalstalla. Keskustelua isännöi Tarja omalla nimellään. Mutta sitten hän toi keskusteluun perinteisen näkökulman Hannelen roolissa, kun taas Pentti otti ketteryyden puolestapuhujan roolin nimettömänä keskustelussa. Roolinimiä käytettiin siksi, jotta keskustelu ei leimautuisi sisäpiirijutuksi. Paras työskentelytapa vaatimusmäärittelyyn on riippuvainen projektista ja olosuhteista. Ei siis ole olemassa vain yhtä oikeaa tapaa tehdä vaatimusmäärittelyä.

Lue koko keskustelu http://www.tieturi.fi/web/guest/keskustelut ja sieltä keskustelualue Teknologiat ja menetelmät. Kommentit, vastaväitteet ja täydennykset ovat luonnollisesti tervetulleita.

Molempi parempi! Eli valitaan tapa tarpeen ja tilanteen mukaan.