KÄYTTÄJÄKERTOMUSMENETELMÄ

MOSS-projektin määrittely ja testaaminen

Emma Vento

Opinnäytetyö
Marraskuu 2008

Liiketalous

JYVÄSKYLÄN
AMMATTIKORKEAKOULU

JYVÄSKYLÄN
AMMATTIKORKEAKOULU

KUVAILULEHTI
Päivämäärä
7.11.2008

Tekijä(t)	Julkaisun laji	
VENTO, Emma	Opinnäytetyö	
	Sivumäärä 54	Julkaisun kieli suomi
	Luottamuksellisuus ☐ Salainen _____saakka	

Työn nimi
KÄYTTÄJÄKERTOMUSMENETELMÄ
MOSS-projektin määrittely ja testaaminen

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma

Työn ohjaaja(t)
KIVIAHO, Niko

Toimeksiantaja(t)
Digia Oyj

Tiivistelmä
Opinnäytetyön tarkoituksena oli selvittää, millainen on käyttäjäkertomustekniikka. Lisäksi tutkittiin, kuinka käyttäjäkertomukset soveltuvat MOSS-sovelluksen toimintojen kuvaamiseen testausta varten, sekä kuinka käyttäjäkertomukset soveltuvat MOSS-sovelluksen testaamiseen. Työn toimeksiantaja oli Digia Oyj, joka tekee paljon tietojärjestelmiä MOSS-alustalle.

Työn aikana tutkittiin käyttäjäkertomuksia ja kehitettiin Digialle käyttäjäkertomuskortteja MOSSin yleisimmistä toiminnallisuuksista. Digian konsultit voivat hyödyntää kortteja uusien MOSS-projektien myyntityössä, ja korttien pohjalta voidaan suunnitella asiakaskohtaisia konfiguraatioita.

Käyttäjäkertomusten käyttöä vaatimusmäärittelyn apuvälineenä ei päästy käytännössä testaamaan, mutta teoriassa käyttäjäkertomuksia voidaan suositella. Käyttäjäkertomuksissa asiakkaiden ja kehittäjien yhteistyöllä on suuri merkitys ja kommunikointi paranee. Tämä johtaa ohjelmistojen laadun paranemiseen ja asiakastyytyväisyyden kasvamiseen. Käyttäjäkertomuksilla pyritään kehittämään juuri sellainen ohjelmisto, minkä asiakas tarvitsee.

Käyttäjäkertomuskortit helpottavat myös testaustyötä, sillä kortit toimivat sellaisenaan valmiina testitapauksinaan, joten erillistä testaussuunnittelua ei tarvita. Lisäksi testaaminen voidaan aloittaa aikaisemmin iteratiivisen ohjelmistokehityksen vuoksi. Työn aikana testattiin MOSS-sovellusta käyttäjäkertomusten perusteella, ja uusi menetelmä osoittautui yhtä helpoksi kuin perinteinen testaus.

Tutkimuksen lopputuloksena voidaan suositella käyttäjäkertomusten käyttöönottoa. Tutkimuksessa ei oteta kantaa siihen, paraneeko ohjelmistoprojektien laatu ja asiakastyytyväisyys todella tulevaisuudessa. Tätä aihetta voidaan käsitellä mahdollisessa jatkotutkimuksessa, jossa verrataan käyttäjäkertomus-MOSS -projekteja perinteisiin MOSS-projekteihin pidemmällä aikavälillä.

Avainsanat (asiasanat)
asiakastyytyväisyys, käyttäjäkertomus, MOSS, testaus, vaatimusmäärittely

Muut tiedot

JYVÄSKYLÄ UNIVERSITY OF APPLIED SCIENCES

DESCRIPTION
Date
7.11.2008

Author(s)
VENTO, Emma

Type of Publication
Bachelor's Thesis

Pages
54

Language
Finnish

Confidential
☐ Until_____

Title
USER STORIES METHOD
The definition and testing of MOSS projects

Degree Programme
Business Information Systems

Tutor(s)
KIVIAHO, Niko

Assigned by
Digia Plc

Abstract
The purpose of the bachelor's thesis was finding out what kind of methods user stories are. Furthermore, it was investigated how user stories could be used to describe the functionalities of a MOSS application for testing, and how user stories apply to the testing of MOSS applications. The work was assigned by Digia Plc, which makes lots of information systems for the MOSS platform.

During this thesis, user stories were investigated and user story cards about the most common functionalities of the MOSS platform were developed for Digia. Digia's consultants can use the cards to sell new MOSS projects and configurations can be tailored to each customer separately.

Usage of user stories as an aid for requirements specification couldn't be tested in practice but in theory user stories can be recommended for this. In user stories, the co-operation between the customers and developers has a great meaning which leads to improved communication. Because of this the quality of software will be improved and customer satisfaction increased. The goal of using user stories is to develop exactly the kind of software that the customer needs.

User story cards also make testing easier because cards work as ready test cases so separate test planning will become unnecessary. In addition, testing can be started earlier because of iterative software development. During this work a MOSS application was tested on the basis of user stories and the new method was found out to be as easy as the traditional one.

As the result of the thesis the deployment of user stories can be recommended. The thesis will not comment on whether or not the quality of software products and the amount of customer satisfaction will really increase in the future. This subject can be handled in possible continued studies that will compare user story utilizing MOSS projects to traditional MOSS projects on a longer time span.

Keywords
customer satisfaction, user story, MOSS, testing, requirements specification

Miscellaneous

SISÄLTÖ

KUVIOT

1 JOHDANTO

Ostaisitko auton, joka ei sovellu käyttötarkoitukseensa? Entäpä jos juuri hankkimasi pesukone jättäisi pesuohjelman puolitiehen ja pyykit jäisivät likaisiksi? Luultavasti vaatisit hinnanalennusta tai peräti kaupan purkamista. Ohjelmistoteollisuudessa keskeneräiset ja tarkoitukseen soveltumattomat ohjelmistot ovat kuitenkin arkipäivää ja jopa hiljaisesti hyväksyttyjä, vaikka puutteellisista järjestelmistä aiheutuu pahimmillaan liiketoiminnallisia tappioita. Monesti nämä ongelmat saavat alkunsa aivan kehitysprojektien alkutaipaleella, jolloin ohjelmiston vaatimuksia vasta kerätään.

Vaatimusten keräämiseen on olemassa menetelmä, joka säästää aikaa ja rahaa, vähentää turhaa, toistuvaa työtä sekä parantaa ohjelmistojen laatua: käyttäjäkertomukset eli User Storyt. Jotta ohjelmisto saavuttaisi sille asetetut vaatimukset, on ohjelmisto testattava huolellisesti laadittuine testitapauksineen. Ohjelmistojen testaaminen on hyvin tärkeä, mutta usein aliarvostettu osa ohjelmistokehitystä. Käyttäjäkertomusmenetelmästä on apua myös testausvaiheessa.

Käyttäjäkertomusten avulla voidaan rakentaa ohjelmistoja, jotka todella kohtaavat käyttäjien tarpeet. Kuulostaa erittäin hyvälle, mutta onko kyseessä pelkkä sanahelinä vai voiko käyttäjäkertomuksista olla todellista hyötyä ohjelmistokehityksessä? Tämän aiheen tutkimiseen syvennytään tarkemmin tässä opinnäytetyössä.

1.1 Opinnäytetyön taustat, tavoitteet ja rajaukset

Opinnäytetyön toimeksiantajana on Digia Oyj, joka on suomalainen kansainvälistyvä ohjelmistoalan yritys. Digia koostuu kolmesta divisioonasta, jotka ovat Teollisuus ja kauppa, Finanssi ja palvelut sekä Telekommunikaatio.

Digian Jyväskylässäkin toimivassa Teollisuus ja kauppa -divisioonassa on ongelmana ollut jo pidempään vakiintuneiden testauskäytäntöjen puute. Tapana

on ollut, että ohjelmoijat testaavat sovelluksia muun työnsä ohella, mikä ei ole ollut paras mahdollinen menetelmä. (Honkaranta & Ylönen 2008.)

Käyttäjäkertomukset ovat varteenotettava tutkimuskohde Digialle, sillä Microsoft-partnerina Digialla tehdään paljon MOSS-projekteja (Microsoft Office SharePoint Server). MOSS on valmisalusta, jolla voidaan rakentaa esimerkiksi erilaisia portaaleja. MOSS sisältää paljon valmiskomponentteja, joita voidaan hyödyntää sellaisenaan ja lisäksi Digialla räätälöidään asiakkaan toiveiden mukaisia ohjelmoituja komponentteja kulloisenkin tarpeen mukaan. Projekteissa on ensiarvoisen tärkeää varmistaa, että asiakas ja toimittaja puhuvat samaa kieltä ja ovat yhtä mieltä toiminnallisuuksista. Tähän tarkoitukseen käyttäjäkertomukset ovat ennakkotietojen perusteella soveltuva menetelmä. (Ylönen 2008.)

Opinnäytetyössä raapaistaan käyttäjäkertomusten lisäksi MOSS-maailmaa, sillä Digia toivoo saavansa työn lopputuloksena käyttäjäkertomuskortteja, joita voi käyttää apuna tulevien MOSS-projektien myyntityöhön, määrittelyihin sekä testaamiseen säästäen samalla aikaa, turhaa toistuvaa työtä ja parantaen sovellusten laatua. (Honkaranta & Ylönen 2008.)

Digialle koituu työstä myös muita hyötyjä. Käyttäjäkertomusten avulla asiakas saa tilatun sovelluksen lopputuloksesta suurimman mahdollisen hyödyn itselleen ja pidemmällä aikavälillä asiakastyytyväisyys toivottavasti paranee. Asiakastyytyväisyyden toteutumista ei kuitenkaan ajankäytön rajallisuuden vuoksi ehditä mittaamaan, mutta tästä voi tehdä tarvittaessa myöhemmin oman tutkimuksensa. Myös ohjelmoijien työtaakkaa saadaan kevennettyä parantamalla testausmenetelmiä.

1.2 Opinnäytetyö käytännössä

Käyttäjäkertomuksiin ja testauksen yleisiin piirteisiin pääasiallisesti keskittyvän teoriaosuuden lisäksi opinnäytetyöhön liittyy empiirinen osio, jossa tehdään käyttäjäkertomuskortteja, joita Digia voi toivottavasti hyödyntää jatkossakin ohjelmistojen määrittelyssä ja testaamisessa. Tutkimukseen kuuluu osana MOSS-sovelluksen testaaminen laadittujen käyttäjäkertomuskorttien pohjalta.

Tästä käytännön osiosta koostetaan havainnot siitä, kuinka hyvin kyseinen menetelmä soveltuu Digian käyttöön.

1.3 Tutkimuskysymykset

Tutkimuksessa pyritään vastaamaan seuraaviin kysymyksiin:

- Millainen käyttäjäkertomustekniikka on?
- Kuinka käyttäjäkertomustekniikka soveltuu alustasovelluksen toimintojen kuvaamiseen testausta varten?
- Kuinka käyttäjäkertomustekniikka soveltuu valmisalustalle tehtävän sovelluksen testaamiseen?

2 OHJELMISTOPROJEKTI

Ohjelmistoprojektit ovat usein luonteeltaan monimutkaisia ja monimutkaiset projektit eivät onnistu ilman tarkkaa suunnittelua. Ohjelmistoprojekti on sitä paremmin suunniteltu, mitä aiemmin siinä olevat ongelmat paljastuvat. Panostamalla alkupään tehtäviin, kuten määrittelyvaiheeseen ja arkkitehtuurisuunnitteluun, voidaan myöhemmissä vaiheissa rakennettaessa ja testattaessa ohjelmistoa säästää selvää rahaa. Tutkimusten mukaan projektiin alkuvaiheessa syntyneen virheen korjaaminen tulee myöhemmin maksamaan 50–200 kertaa enemmän kuin sen välitön korjaaminen. Projektin alkuvaihe luokin perustaa koko projektin menestykselle. (McConnell 1998, 29.)

2.1 Miksi käyttäjä kannattaa sitouttaa projektiin?

Ohjelmistoprojektissa on tärkeää sitouttaa ohjelmiston tulevat käyttäjät projektiin jo alkuvaiheessa. Onnistuneen ohjelmiston edellytys on, että suunnitellut loppukäyttäjät käyttävät sitä ja pitävät siitä. Ohjelmistosuunnittelijat saattavat kiinnittää huomionsa loppukäyttäjän kannalta merkityksettömiin ongelmiin ja kehittää ratkaisuja, vaikkakin näppäriä, niihin. Markkinoijien vikana on ladata ohjelmistot niin täyteen erilaisia ominaisuuksia, että loppukäyttäjät eivät löydä omiaan joukosta. (McConnell 1998, 46.)

Projekti hyötyy käyttäjän varhaisesta sitouttamisesta monin tavoin. Säästyy aikaa ja säästytään monilta vaatimusten muutoksilta. Vaillinaisen määrittelyn vuoksi joutuu puuttuvia ominaisuuksia rakentelemaan jälkikäteen. Käyttäjät myös löytävät ohjelmistosta paljon puutteita, jos heidät tuodaan mukaan projektiin vasta sen loppuvaiheessa. (McConnell 1998, 46.)

Käyttäjät pitävät laadukkaampina ohjelmistoja, joiden kehitykseen heidät on otettu jo varhain mukaan. Näin voidaan varmistaa, että rakennettava ohjelmisto sopii käyttäjien tarpeisiin ja vastaa odotuksia. Tämä tarkoittaa ohjelmiston ulkoista laatua. Sen vastakohta on sisäinen laatu, joka tarkoittaa vakaammasta arkkitehtuurista, moduulisuunnittelusta ja toteutuksesta johtuvaa alhaista

virhemäärää. Sisäinen laatu laskee, jos käyttäjien tiedossa olevat vaatimukset muuttuvat jatkuvasti. Suunnittelijoidenkin työ vaikeutuu, jos he joutuvat pitkin projektia lisäilemään etukäteen suunnittelemattomia toimintoja jo olemassa olevaan ohjelmarunkoon, johon niitä ei alun perinkään ollut tarkoitus rakentaa. Standish Group -niminen tutkimuslaitos tutki vuonna 1994 yli 8000 ohjelmistoprojektia, ja tutkimuksessa selvisi, että loppukäyttäjän mukana pitäminen projektissa on tärkein projektin menestymiseen vaikuttava tekijä. Samalla käyttäjäpalautteen puuttuminen johti merkittävästi epäonnistumiseen. (McConnell 1998, 47 - 48.)

2.2 Loppukäyttäjien tunnistamisen tärkeys

Yksi tärkeimpiä asioita rakennettaessa uutta ohjelmistoa on oppia tuntemaan loppukäyttäjät. Käyttäjiä voi ohjelmistolla olla monenlaisia: asentajia, ylläpitäjiä, järjestelmän pääkäyttäjiä ja muita tukihenkilöitä unohtamatta peruskäyttäjää näppäimistönsä ääressä. (Nielsen 1993, 73.)

Usein paras keino saada selville, mitä loppukäyttäjät haluavat, on kysyä heiltä suoraan. Näin ei tarvitse tuhlata aikaa määrittelypalavereissa kiistelemällä siitä millaisia loppukäyttäjät olisivat ja mitä he ohjelmistolla tekisivät. Kehittäjien on toisinaan vaikea kommunikoida loppukäyttäjien kanssa. Tähän on syynä esimerkiksi toimittajan tarve ”suojella” kehittäjiään asiakkailta, sillä asiakkaiden pelätään ohittavan heitä varten perustetut tekniset tukipalvelut ja ottavan suoraan yhteyttä ongelmissa kehittäjiin, jotka puolestaan ajautuvat kauemmas varsinaisesta työnkuvastaan. Myyntihenkilöstö voi myös suhtautua nihkeästi kehittäjien ja asiakkaiden yhteistyöhön, sillä myyjät saattavat pelätä kehittäjien pilaavan vahingossa asiakassuhteet sanomalla asiakkaan kuullen vääriä asioita tai jopa loukkaavan asiakasta. (Nielsen 1993, 73 - 74.)

Joissakin tapauksissa loppukäyttäjät on hyvin helppo yksilöidä. Näin on esimerkiksi silloin, kun lopputuotetta on tarkoitus käyttää tietyn organisaation tietyssä yksikössä. Toisissa tapauksissa käyttäjät voivat olla hyvin laajalta alueelta, jolloin käyttäjiä profiloidaan valitsemalla oletetusta käyttäjäjoukosta muutamia edustajia vaatimusten keräämistä varten. (Nielsen 1993, 74.)

Kuinka sitten saadaan käyttäjät pitämään ohjelmistosta? Yksinkertaisesti projektiryhmän tulee kysyä, mitä käyttäjät haluavat, näyttää, mitä aiotaan rakentaa ja ottaa suunnitelmasta vastaan palautetta. Jatkon määrää käyttäjien ja kehittäjien välinen kommunikointi, johon on kyllä jaksettava nähdä vaivaakin. (McConnell 1998, 46.)

2.3 Uudet tuotantomenetelmät

Ohjelmistotuotantoon kuuluu olennaisena osana uusien tuotantomenetelmien ja tekniikoiden kehittäminen. Tämä ei aina kuitenkaan automaattisesti merkitse suoraa tietä menestykseen. Näennäisesti parantuneen tuottavuuden taustalla saattaa olla muutakin vaikutetta kuin tuotantoprosessiin sovellettu uusi menetelmä. Vaikka menetelmästä käytössä olevat tutkimusraportit olisivat kuinka luotettavan tuntuisia, kannattaa huomioida myös raportin kirjoittajan motiivit. Jos kirjoittajana toimii esimerkiksi laatupäällikkö, saattaa olla kirjoittajan edun mukaista osoittaa yrityksen saavuttaneen menetelmän ansiosta huomattavia säästöjä. Monestihan uuden menetelmän käyttöönotto näyttää nostavan tuottavuutta, mutta ilmiötä voi osaltaan selittää sekin, että kokeiluun osallistuvat ihmiset ovat usein erittäin motivoituneita ja innostuneita. Tämän vuoksi tuottavuus nousisi joka tapauksessa menetelmästä riippumatta. (Haikala & Märijärvi 2002**,** 26 - 27.)

Samalla voi tapahtua ns. Hawthorne-ilmiö, joka tarkoittaa sitä, että asiat, joihin kiinnitetään erityistä huomiota, tehostuvat, koska ihmisten työmotivaatio lisääntyy. Epäonnistuneista kokeiluista ei myöskään usein haluta pitää meteliä. Jotta voitaisiin sanoa jotakin varmaa uuden menetelmän paremmuudesta muihin verrattuna, pitäisi tuloksia tarkasteltaessa kiinnittää huomiota yksittäisen raportin absoluuttisten lukujen sijasta tulosten yleiseen linjaan. Samankaltaisia tuloksia olisi saatava pidemmällä aikavälillä monissa eri paikoissa. (Haikala & Märijärvi 2002**,** 26 - 27.)

3 VAATIMUSTEN MERKITYS OHJELMISTOPROJEKTISSA

Ohjelmistojen tuotantoprosessin tavoitteena on päätyä asiakasvaatimuksista asiakasvaatimukset täyttävään ohjelmistoon. Tämän varmistamista kutsutaan vaatimustenhallinnaksi, jonka keskeinen tehtävä on varmistaa, että lopputuote vastaa asiakkaan vaatimuksia ja siinä ovat kaikki halutut ominaisuudet. Käytännön merkityksestään huolimatta vaatimustenhallinta jää usein liian vähälle huomiolle, jolloin projektin lopputulos voi olla jotakin aivan muuta kuin alun perin oli määrä. Liioista ohjelmiston ominaisuuksista voi puolestaan olla haittaa, sillä niiden tekemiseen on kulutettu kallisarvoista aikaa, joka puolestaan maksaa. Turhista ominaisuuksista ei koskaan saada vastinetta rahoille, sillä niille ei ole käyttöä. (Haikala & Märijärvi 2002**,** 93 - 94.)

Jotta ohjelmistotuotanto olisi tehokasta, tulee asiakasvaatimusten olla oikein ymmärrettyjä ja mahdollisimman muuttumattomia. Mitä myöhäisemmissä vaiheissa vaatimuksiin tehdään muutoksia, sitä enemmän ne aiheuttavat lisätyötä. Ohjelmistoprojektien riskilistojen kärjessä ovatkin muuttuvat, virheelliset ja puuttuvat asiakasvaatimukset. Asiakasvaatimukset tulee analysoida, jotta voidaan selvittää kunkin vaatimuksen perimmäinen tarve, priorisoida vaatimukset ja sovittaa niihin liittyvät ristiriidat. (Haikala & Märijärvi 2002**,** 96.)

Vaatimuksia kerätään monella eri tavalla. Niitä saadaan esimerkiksi markkinoinnista, tutkimalla kilpailevia tuotteita ja omasta organisaatiosta. Lisää ideoita syntyy rakentamalla prototyyppejä, pitämällä aivoriihiä ja keräämällä asiakaspalautetta tuotteen aiemmista versioista. (Haikala & Märijärvi 2002, 96.)

Asiakasvaatimuksia ja niiden liittymistä ohjelmistovaatimuksiin voidaan kuvata ja kartoittaa esimerkiksi käyttötapausten eli use casejen avulla. Ne ovat sanallisia kuvauksia, jotka kuvaavat käyttäjien tapaa käyttää rakennettavaa järjestelmää. Näin järjestelmän määrittely liitetään sen käyttöympäristöön. (Haikala & Märijärvi 2002, 97 - 98.)

Vaatimusten priorisointi on tärkeää. Se toteutetaan peilaamalla vaatimuksia liiketoimintaan: saadaanko vaatimuksen toteuttamisesta tarpeeksi tuottoja vastapainoksi sen aiheuttamille kustannuksille ja liittyykö vaatimuksen toteuttamiseen mahdollisesti aikatauluriskejä. Sen avulla voidaan päättää mitkä ominaisuudet otetaan mukaan toteutukseen ja mitä jätetään mahdolliseen myöhempään versioon. Jos projektin aikataulu on tiukka, voidaan vähemmän tärkeitä ominaisuuksia karsia pois. Priorisointi ei yksin riitä, vaan vaatimukset on myös sovitettava toisiinsa sopiviksi, jottei ohjelmistosta tulisi sekavaa ominaisuuksiltaan. Vaatimuksissa saattaa olla keskenään ristiriitaisuuksia, jotka on karsittava pois. (Haikala & Märijärvi 2002, 98.)

Vaatimusten analysoinnin perusteella on mahdollista valita projektille malli, jonka mukaan edetään. Jos vaatimusten arvioidaan pysyvän suhteellisen muuttumattomina, on vesiputousmalli hyvä vaihtoehto. Jos taas vaatimukset tulevat suurella todennäköisyydellä muuttumaan myöhemmin, on suositeltavaa käyttää inkrementaalista mallia tai prototyyppitapaa. Suunnitteluvaiheessa vaatimusten muutosherkkyys vaikuttaa myös arkkitehtuurin valintaan. (Haikala & Märijärvi 2002, 98 - 99.)

Asiakasvaatimusten pohjalta tehdään ohjelmiston määrittely, joten vaatimusten keräämiseen kannattaa panostaa. Vaatimuksia tosin löytyy pitkin määrittelytyötä ja myöhemminkin ja niihin tehdään tarkennuksia pitkin projektia. Vaatimuksia todennetaan eli *verifioidaan* pitkin projektia eri vaiheiden päättyessä vertaamalla vaiheen vaatimuksia sen tulosdokumentteihin. Testausvaiheessa vaatimukset verifioidaan vertaamalla testien tuloksia vastaaviin määrittelyihin. *Validoinnilla* eli kelpoistamisella osoitetaan, että järjestelmä vastaa asiakkaan tarpeita. Tämä voidaan tehdä testaamalla järjestelmää sen oikeassa käyttöympäristössä tai vaikkapa prototyyppien avulla varhaisemmissa vaiheissa. (Haikala & Märijärvi 2002**,** 99.)

4 OHJELMISTOKEHITYSMALLIT

Tässä kappaleessa esitellään lyhyesti kaksi työssä mainittavaa ohjelmistokehitysmallia: vesiputousmalli ja inkrementaalinen ohjelmistokehitys. Ketterät menetelmät mainitaan myös, koska ne ovat tiiviissä yhteydessä inkrementaaliseen kehitykseen.

Vesiputousmalli

Vesiputousmalli on yksi tyypillisimpiä ohjelmistokehitysmalleja. Se on lineaarinen ja vaiheet päättyvät ennen uuden alkamista. Projektin vaiheille ovat tyypillisiä tietyt dokumentit, jotka luodaan tietyissä vaiheissa ja katselmoidaan ennen niiden hyväksymistä. Projektin vaiheiden päättymisestä ja seuraavaan vaiheeseen siirtymisestä päätetään myös erikseen. Suurin vesiputousmallin heikkous on sen hitaus: tuotteen markkinoille päästäminen viivästyy helposti, sillä vesiputousmallissa vaiheet seuraavat tiukasti toisiaan ja jopa testausvaiheen tulee edetä tietyssä järjestyksessä. Kun sitten yhdessä testausvaiheessa ilmenee ongelmia, viivästyvät muutkin vaiheet ja siten koko projekti. (Loveland, Miller, Prewitt & Shannon 2005, 43 - 44.)

Ketterät menetelmät

Ketterät ohjelmistonkehitysmenetelmät pyrkivät saamaan kehitettävän ohjelmiston nopeasti tuotantoon. Ohjelmistoja kehitetään iteratiivisesti. Ketterien menetelmien vahvuutena on se, että ne sopeutuvat muuttuviin vaatimuksiin läpi koko kehitysprosessin; vesiputousmallissa pyritään saamaan kaikki vaatimukset selville alkuvaiheessa ja muutoksia odotetaan tulevan vain muutamia. Ketterät menetelmät ovat riippuvaisia kehittäjien ja lopputuotteen käyttäjien yhteistyöstä. Näissä menetelmissä koodarit koodaavat niin nopeasti ja tehokkaasti kuin vain mahdollista, ja tässä piileekin yksi heikkous. Nopean etenemisen vuoksi kehittäjät eivät aina ole ajan tasalla kollegoidensa ratkaisuista ja niiden kierrättämismahdollisuuksista. Hitaammassa vesiputousmallissa tämä yleensä onnistuu. (Loveland ym. 2005, 49 - 50.)

Iteratiivinen ohjelmistokehitys

Iteratiivinen ohjelmistokehitys kuuluu ketteriin menetelmiin. Tämän mallin mukaisesti kaikkien ohjelmistovaatimusten ei tarvitse olla selvillä, jotta koodaaminen voitaisiin aloittaa. Ohjelmistoa rakennetaan asteittain eli iteraatioittain, ja jokainen iteraatio sisältää samat vaiheet: vaatimusmäärittely, suunnittelu, koodaaminen ja testaaminen. Tätä samaa toistetaan iteraatiosta toiseen sykleittäin. Aina iteraation päättyessä koodi testataan ennen uuteen iteraatioon siirtymistä. Vaiheen päätyttyä päätetään, mitä toiminnallisuuksia seuraavassa iteraatiossa kehitetään. Iteratiivisessa mallissa asiakasta tarvitaan läpi koko projektin ja tämä edesauttaa asiakasvaatimusten täyttymistä. Tarpeellisten muutosten tekeminen järjestelmään kesken projektin onnistuu hyvin. (Hambling, Morgan, Samaroo, Thompson & Williams 2007, 37 - 38.)

5 MÄÄRITTELY

Jotta projektin eri vaiheet onnistuisivat, on määrittelyjen oltava huolellisesti tehtyjä. Puutteelliset ja virheelliset määrittelyt voivat myöhemmässä vaiheessa käydä varsin kalliiksi. Käytännössä on kuitenkin erittäin vaikea saada selville ohjelmiston kaikkien tulevien käyttäjäryhmien todelliset tarpeet. Asiakas on tarpeen sitouttaa projektiin jo varhaisessa vaiheessa eli määrittelyissä. Hyvä määrittely määrittelee kaikki ohjelmistossa tarvittavat asiat ja vain ne, on testattavissa, tarkka, virheetön ja ymmärrettävä. Tämä kaikki ei ole kuitenkaan mahdollista ja määrittelyihin jää usein virheitä, puutteellisuuksia sekä moniselitteisyyksiä, joihin joudutaan palaamaan projektin myöhemmissä vaiheissa. Tässä korostuu tarve kommunikointiin asiakkaan kanssa, jossa voi myös ilmetä väärinkäsityksiä. Vaikka määrittely olisikin hyvä, ei projektin onnistumisesta ole takeita. Vaikka projektin aikataulu ja budjetti olisivatkin onnistuneet hyvin, voi lopputulos olla epäonnistunut, jos asiakkaan tarpeet on ymmärretty väärin. Myöskään tekniseltä kannalta hyvä järjestelmä ei ole onnistunut, jos sen käyttäjät eivät halua järjestelmää käyttää. (Haikala & Märijärvi 2002**,** 65 - 66.)

Määrittelyjen laatimiseen ja dokumentointiin käytetään kuvaustekniikoita eli notaatioita, jotka ilmaisevat eri kielillä erilaisia asioita. Tällaisia ovat esimerkiksi tietovuokaaviot sekä tila- ja luokkakaaviot. Notaatiota sovelletaan erilaisilla ohjelmistonkehitysmenetelmillä, kuten esimerkiksi SA-menetelmä (eli Structured Analysis, joka on yleisimpiä ohjelmistojen määrittely- ja suunnittelumenetelmiä) tai OMT++. Kuvaustekniikoilla voi yleensä kuvata kuitenkin vain osan määrittelyä, joten niillä laadituilla kuvauksilla lähinnä havainnollistetaan luonnollisella kielellä laadittua kuvausta. (Haikala & Märijärvi 2002**,** 67 - 69.)

Menetelmät ovat usein kehitetty käytettäviksi tietyissä projektin vaiheissa, mutta niitä voidaan sovittaa myös muihin vaiheisiin. Esimerkiksi SA-menetelmää voidaan käyttää vaatimusten analysointiin ja määrittelyyn, johon se on ensisijaisesti kehitetty, mutta myös ohjelmiston suunnitteluun. Monia standardeiksi muodostuneita notaatioita käytetään eri menetelmissä. Usein menetelmien ja notaatioiden välinen ero onkin hiuksenhieno. Esimerkiksi tie-

tovuokaavion sisällyttämisestä määrittelydokumenttiin saatetaan puhua SA-menetelmän käyttämisenä. Menetelmiä voidaan harvoin soveltaa suoraan käytäntöön, ja ne on sovitettava kulloiseenkin sovellusalueeseen ja yrityksen toimintatapoihin. (Haikala & Märijärvi 2002**,** 67 - 69.)

6 LAADUN KÄSITE

Ohjelmiston laatu syntyy sitä tehtäessä, ei jälkikäteen lisäämällä. ISO-standardeissa laatu on määritelty siten, että laadukas tuote täyttää sille asetetut vaatimukset. Tämä sisältää niin toiminnalliset kuin ei-toiminnallisetkin vaatimukset. Toiminnalliset vaatimukset kuvaavat ohjelmiston toimintoja ja ei-toiminnalliset ovat esimerkiksi suorituskykyyn liittyviä vaatimuksia. (Haikala & Märijärvi 2002**,** 191.)

Laadukkaan ohjelmiston rakentaminen vaatii hyvää kommunikointia eri osapuolten välillä. Laadun käsite ei ole itsestäänselvyys projektin eri osapuolille ja loppukäyttäjille, sillä laatua voidaan arvioida eri tavoin. Objektiivisesti arvioitaessa tuotteesta pystytään laskemaan tai mittaamaan, kuinka hyvin se täyttää tuotteelle asetetut vaatimukset. Subjektiivisessa laadun arvioinnissa määrää se, kuinka hyvin tuote täyttää asiakkaan odotukset ja miltä se tuntuu käyttää. Kolmas tapa arvioida laatua kattaa seikat, joita ei voida arvioida, koska näitä asioita ei ole vielä tapahtunutkaan. Tuotteen laadukkuus on näin arvioitavissa vasta ajan myötä. Tällä asialla ei ole merkitystä silloin, kun asiakas tekee tuotteesta ostopäätöksen, mutta se on mukana asiakkaan neuvotellessa toimittajan kanssa jatkokauppoja. Käytännössä laatua kuitenkin arvioidaan useimmiten subjektiivisesti, joten laatukäsityksen voi kiteyttää liiketaloudellisesti niin, että tärkeintä on tyytyväinen asiakas. (Haikala & Märijärvi 2002**,** 192.)

Järjestelmätestaus on osa laadunvarmistusta. Mikä tahansa testaus ei kuitenkaan ole tae laadukkuudesta, vaan on huomioitava monia eri seikkoja. Järjestelmätestaus kannattaa teettää ulkopuolisilla testaajilla, sillä vaikka ohjelmistosuunnittelijat löytävät koodistaan tietyn määrän virheitä, vaatii maksimaalisen virhemäärän löytäminen ajattelutavan muuttamista. Ohjelmistosuunnittelijat ovat luonnostaan taipuvaisempia testaamaan siten, että ohjelma saadaan toimimaan, eikä niin, että se hajoaa. Testaussuunnittelu kannattaa aloittaa heti, kun vaatimukset ovat selvillä ja näin varmistetaan, ettei testaus viivytä ohjelmiston julkaisua. Vaihetoimitusmallissa järjestelmätestaus kannattaa aloittaa myös jo ensimmäisessä toteutusvaiheessa, sillä suorituskelpoista ohjel-

mistoa on tällöin usein valmiina. Testaukselle on osoitettava lisäksi riittävästi resursseja, mutta ne on suhteutettava ohjelmiston luonteeseen. Järjestelmä, joka vastaa ihmishengistä, tarvitsee lukuisia testaajia yhtä kehittäjää kohden, mutta esimerkiksi pienehkö yrityksen sisäinen liiketoimintajärjestelmä selviää paljon vähemmällä testaamisella, testausta ei tarvitse automatisoida ja testausvaatimuksistakin voidaan tinkiä. (McConnell 1998, 137 - 139.)

On kuitenkin muistettava, että testaus ei ole ohjelmiston laadunvarmistuskeino, vaan tapa selvittää ohjelmiston laadun taso. Testaus ei myöskään ole riittävän tehokasta, jos se sijoittuu vain projektin loppupuolelle; ohjelmiston laatu voi olla heikko, koska alkuvaiheessa on säästetty työtä ja näin ohjelmistosta löytyy valtavasti virheitä. Tämä johtaa siihen, että tarvitaan paljon testaajia ja seuraavissa projekteissa on vaikea kohdentaa henkilöstöresursseja projektin alkupäähän, jolloin loppupään tilanne pahenee ja huonon laadun kierre on valmis. (McConnell 1998, 137 - 139.)

7 TESTAUS

Testaus on välttämätön osa toimivien tietojärjestelmien rakentamista. Siitä huolimatta testausta pidetään monesti välttämättömänä pahana, joka syö paljon aikaa ja rahaa. Jos organisaation testauskäytännöt eivät ole vakiintuneet, saattaa testaus tarkoittaa hallitsematonta ja vaikeaa prosessia, jonka lopputuloksena ohjelmistonkehityksestä tulee ongelmallista. Harvoissa organisaatioissa testaukselle uhrataan tarpeeksi resursseja johtuen edellä mainituista seikoista. (Koomen & Pol 1999, 1.)

7.1 Järjestelmätestaus

Järjestelmätestaus tarkoittaa järjestelmän kokonaisvaltaisen toiminnan varmistamista. Sen tehtävä on todistaa kaikki vaatimukset täytetyiksi riittävän laadukkaasti. (McConnell 1998, 220.)

Testaus tarkoittaa suunnitelmallista virheiden etsimistä suorittamalla ohjelmaa tai sen osaa. Testaus-termi voi joskus tarkoittaa myös tarkastuksia ja ohjelmakoodin staattista analysointia. Käytännössä testaaminen on kuitenkin usein umpimähkäistä ohjelman kokeilemista jollakin syöttöaineistolla ja testaajana toimii ohjelman tekijä. Tästä seuraa se, että pyritään mieluummin osoittamaan järjestelmän toimivuus kuin todistamaan, että siinä on virheitä. Testaukseen käytetty aika ja lukuisat testitapaukset eivät myöskään automaattisesti merkitse sen tehokkuutta. Huolellisesti suunniteltu parin tunnin testaaminen tarkkaan valituilla testitapauksilla voi olla paljon tuloksellisempaa kuin päiväkausien hakuammunta. (Haikala & Märijärvi 2002**,** 282.)

Mitä aikaisemmin testaaminen voidaan aloittaa, sitä enemmän testaamiselle jää aikaa. Testaukselle asetettu aikaraja voi tulla suunniteltua aikaisemmin vastaan ja vaihe voi olla vaarassa jäädä kokonaan pois, jos kaikki testaaminen on jätetty tehtäväksi koodin valmistuttua. Koska vaatimusmäärittelyt toimivat perustana hyväksymistestaukselle, voidaan testien suunnittelu aloittaa heti, kun vaatimusdokumentaatio on valmis. Samalla dokumentaatiota käy-

dään läpi kriittisemmin ja saadaan selville, ovatko vaatimukset testattavissa. Myös vaillinaisia tai puuttuvia vaatimuksia voidaan löytää. (Hambling ym. 2007, 17.)

Testaus koostuu neljästä eri vaiheesta. Aluksi suunnitellaan testaus laatimalla testitapaukset ja testaussuunnitelma. Sitä seuraa testiympäristön luonti, itse testien suorittaminen ja lopuksi tulosten tarkastelu. Näihin työvaiheisiin liittyy kiinteästi virheiden jäljittäminen ja korjaaminen eli *debuggaus*. Testausprosessi kannattaa huomioida hyvin, sillä siihen kuluu tyypillisesti yli puolet ohjelmistoprojektin resursseista. Kompromisseja täytyy tehdä käytettävissä olevan ajan, rahan, välineiden ym. välillä. (Haikala & Märijärvi 2002**,** 281.)

Testausprosessiin liittyy useita henkilöitä. On koodareita, testaajia, joista osa voi olla tiettyyn alueeseen erikoistuneita ammattilaisia, ja loppukäyttäjiä, joita käytetään hyväksymistesteissä. Ohjelmisto olisi hyvä testauttaa henkilöillä, jotka eivät ole osallistuneet sen kehittämiseen, parhaimpia olisivat ulkoistetut testaajat. Tässä tulee usein kuluraja vastaan, sillä ulkopuolisten asiantuntijoiden käyttäminen on huomattavasti kalliimpaa kuin testauttaa koodi sen kehittäjillä. Koodareiden käyttämisessä on hyvätkin puolensa: virheet saadaan korjattua sitä mukaa, kun niitä löydetään. Haittapuolena on se, että omia virheitään on vaikea löytää. (Hambling ym. 2007, 26.)

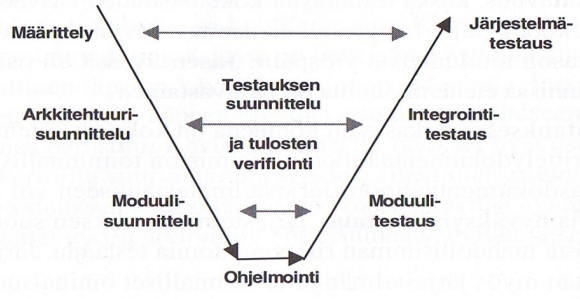
7.2 Mikä on virhe?

Virhe (error, mistake, bug) on poikkeama spesifikaatiosta eli määrittelystä, ja täten testaaminen ilman spesifikaatiota on mahdotonta, sillä lopputuloksen oikeellisuutta ei voida todeta. Useimmin testauksen pohjana toimivat toiminnallinen määrittely ja tekninen määrittely. Kyseisiä dokumentteja voidaan tulkita eri tavoin: asiakas näkee selviä virheitä siinä, mitä toimittaja väittää ohjelmiston ominaisuuksiksi. Kyseessä voi olla vakava virhe tai käyttäjää ärsyttävä pieni yksityiskohta, mutta sanoma on se, että paraskin spesifikaatio voi olla puutteellinen. Kaikkia ristiriitatilanteita ei voida ratkoa tutkimalla spesifikaatioita; siksi sopimuksiin ja suunnitelmiin tulee kirjata, miten näissä tilanteissa menetellään. (Haikala & Märijärvi 2002**,** 285.)

Yleinen arvio on, että ohjelmassa on yksi virhe muutamaa kymmentä ohjelmariviä kohden. Pitkään käytössä olleissa ohjelmistoissa saattaa olla yksi virhe tuhatta ohjelmariviä kohden. Noin 5 % virheistä on lisäksi sellaisia, ettei niitä koskaan havaita. Testauksessa käytettävä mahdollinen syöteavaruus voi olla valtava, tai vaikka viallinen kohta suoritettaisiinkin, se ei välttämättä aiheuta virhetoimintoa. Tämä voi kuitenkin aiheuttaa ohjelmistoon vian (fault), joka voi korjautua itsestään jonkin toisen toiminnon vuoksi tai jonka vaikutus voi kumoutua toisen virheen vaikutuksesta. Pahimmillaan vika aiheuttaa häiriön (failure), joka näkyy järjestelmästä ulospäin. (Haikala & Märijärvi 2002**,** 285.)

7.3 Testauksen V-malli

V-mallin mukaan erilaisia testaustasoja ovat moduuli- eli yksikkötestaus, integrointitestaus ja järjestelmätestaus, kuten Haikalan ja Märijärven (2002, 287) v-mallista kuviosta 1 käy ilmi. Joskus järjestelmätestausta seuraa erillinen kenttätestaus tai hyväksymistestaus. V-mallissa testauksen suunnittelu tapahtuu testaustasoa vastaavalla suunnittelutasolla: järjestelmätestaus suunnitellaan määrittelyvaiheessa, integrointitestaus suunnitteluvaiheessa ja moduulitestaus moduulisuunnittelussa. Testaustuloksia verrataan vastaaviin dokumentteihin. (Haikala & Märijärvi 2002**,** 286.)



KUVIO 1. V-malli

7.3.1 Moduulitestaus

Moduulitestaus tarkoittaa yksittäisen moduulin (noin 100–1000 ohjelmariviä) testaamista. Tämän testauksen suorittaa yleensä moduulin toteuttaja ja tuloksia verrataan tavallisimmin tekniseen määrittelydokumenttiin. Jotta testaus onnistuisi, joudutaan usein tekemään ns. testipetejä, joihin voi kuulua ohjelman ympäristöä simuloivia osia ja tynkämoduuleita. Simuloivat osat ovat useimmiten testiajureita, jotka mahdollistavat moduulin toteuttamien palveluiden kutsumisen ja tulosten tarkastelun. Tynkämoduuleita tarvitaan korvaamaan mahdollisesti vielä puuttuvat testattavan moduulin tarvitsemat muut moduulit. (Haikala & Märijärvi 2002**,** 287.)

7.3.2 Integrointitestaus

Integrointitestaus on moduulien tai moduuliryhmien yhdistelemistä ja painottaa moduulien välisten rajapintojen tutkimista. Tuloksia verrataan yleensä tekniseen määrittelyyn. Integrointitestaus liittyy kiinteästi moduulitestaukseen ja niitä ei useinkaan erotella toisistaan. Moduulitestaus voi tosin olla tehokkaampaa, sillä kun testattava kokonaisuus kasvaa integroitaessa moduuleja toisiinsa, vaikeutuu koodin tarkka läpikäynti. Integrointi voidaan tehdä joko kokoavasti edeten alimman tason moduuleista ylöspäin tai vastaavasti jäsentävästi eli päinvastoin etenemällä. (Haikala & Märijärvi 2002**,** 288.)

7.3.3 Järjestelmätestaus

Järjestelmätestaus testaa koko järjestelmän toimivuutta peilaten tuloksia toiminnalliseen määrittelyyn ja asiakasdokumentaatioon. Tähän vaiheeseen voi liittyä myös kenttä- sekä hyväksymistestaus. Testaajien tulee olla mielellään erillään kehitystyöstä. Järjestelmätestaukseen kuuluu lisäksi ei-toiminnallisten ominaisuuksien testaaminen, kuten kuormitus-, luotettavuus-, asennus- ja käytettävyystestit. (Haikala & Märijärvi 2002**,** 288.)

7.3.4 Regressiotestaus

Virheiden korjaaminen tulee sitä kalliimmaksi, mitä korkeammalla V-mallissa ollaan. Virheiden korjaamisesta voi aiheutua myös uusia virheitä ja tällaisia virheitä voidaan testaustasosta riippumatta eliminoida uudelleentestauksella eli regressiotestauksella. Tämä on tärkeää siksi, että järjestelmätestauksessa havaittu virhe voi aiheuttaa muutoksia useisiin moduuleihin ja jotta muutostar-

peet eivät jäisi huomaamatta, ovat muutkin moduulit testattava uudelleen ja lopuksi suoritettava uusi järjestelmätestaus. Tästä johtuen testauksen automatisointi säästää kustannuksia huomattavasti. (Haikala & Märijärvi 2002**,** 288.)

7.3.5 Muut testaukset

Alfa-testaus tarkoittaa toimittajan tekemää testausta ennen kuin järjestelmä luovutetaan asiakkaalle. Tämä tehdään toimittajan testiympäristössä oman henkilöstön voimin. (Hambling ym. 2007, 45.)

Beta-testaus on asiakkaan suorittamaa testausta, josta voi saada irti paljon hyötyä. Vaikka toimittaja kuinka testaisi kehitettävää tuotetta omassa ympäristössään, on testiympäristö silti keinotekoinen; beta-testaus suoritetaan todellisessa loppukäyttöä vastaavassa ympäristössä ja tulokset voivat siten olla hyvinkin valaisevia. Beta-testauksessa tärkeitä aihealueita ovat kaikki, jotka vaikuttavat asiakkaan liiketoimintaan. Tärkeää tietoa voidaan saada esimerkiksi kuormittamalla järjestelmää maksimaalisella käyttäjämäärällä, ja lisäksi beta-testaus paljastaa lopputuotteen yhteensopivuuden asiakkaan muiden järjestelmien kanssa. Jokainen beta-testiympäristö on erilainen, sillä jokainen asiakas luo omia tarpeitansa vastaavan ympäristön. (Loveland ym. 2005, 41 - 42.)

Staattinen testaus on testaamista, jossa koodia ei suoriteta. Suurin osa virheistä saa alkunsa siitä, että ihminen tekee virheen, joka voi olla vaikkapa virhe määrittelydokumentaatiossa. Tällaiset virheet ovat halvempia ja kivuttomampia testata jo ennen kuin ne päätyvät koodiin asti, joten dokumentaation katselmoinneista on suuri apu. *Dynaaminen testaaminen* tarkoittaa testitapausten suorittamista testidatan avulla, joten sitä voidaan kutsua varsinaiseksi järjestelmän testaamiseksi. Sekä staattinen että dynaaminen testaaminen ovat kuitenkin kumpikin tärkeä osa testauskokonaisuutta. (Hambling ym. 2007, 15.)

Käytettävyystestauksella testataan pääasiallisesti järjestelmän käyttöliittymää. Tavoitteena on varmistaa, että käyttäjä selviää tehtävistä, joita hänen tulisikin suorittaa valmiissa järjestelmässä. Määrittelyvaiheessa käytettävyyttä testataan prototyypin avulla ja myöhemmin valitaan otos ohjelmiston tulevista käyttäjistä seuraten heidän suoriutumistaan erilaisista tehtävistä. Testaus voidaan

tehdä laboratoriossa, jossa on erilliset laitteet käyttäjän toimien rekisteröintiä varten. Toinen tapa on pyytää testikäyttäjiä ajattelemaan ääneen ja videoida sitten ohjelman käyttötilanteita. Kolmas vaihtoehto on käyttää käytettävyyteen erikoistuneita asiantuntijoita arvioinnissa. (Haikala & Märijärvi 2002**,** 289.)

7.4 Testausdokumentaatio

Testauksessa syntyy paljon dokumentaatiota. ISO 9000-3 -ohjeen mukaisesti suositellaan tehtäväksi testaussuunnitelma ja -raportti sekä järjestelmä- että integrointitestauksesta. Moduulitestaustasolla laatukäsikirjan ohjeistus korvaa testisuunnitelman. Pienemmissä projekteissa integrointi- ja järjestelmätesti-suunnitelmat voidaan yhdistää yhdeksi dokumentiksi. (Haikala & Märijärvi 2002**,** 296 - 297.)

Testaussuunnitelman pohjana voivat toimia esimerkiksi:

- määrittelydokumentaatio
- kokemus, tarkastuslistat
- suunnitteludokumentaatio
- laatujärjestelmän ohjeistus ja
- vanha testaussuunnitelma. (Haikala & Märijärvi 2002**,** 296.)

Testaussuunnitelmassa kerrotaan, mitä testejä tehdään ja milloin, miten ne järjestetään ja mitä lopputulosten tulisi olla. Testauksen lopettamiskriteerit tulee myös määritellä. Tämä on erityisen tärkeää, sillä kriteerit voivat poiketa paljonkin toisistaan. Esimerkiksi lopettamisen perusteena voi olla testaukselle varatun ajan loppuminen tai että 100 % löydetyistä virheistä on korjattu tai että 100 % testitapauksista on ajettu. Joissakin tapauksissa testaus lopetetaan vasta, kun 100 %:n regressiotestaus on suoritettu kaikkien korjausten jälkeen. (Haikala & Märijärvi 2002**,** 297.)

Kun testaussuunnitelma on tehty, sitä analysoidaan ja arvioidaan, kuinka siitä saataisiin suurin mahdollinen hyöty irti. Tässä vaiheessa asetetaan ehdot, kuinka ohjelmiston tulisi toimia missäkin testaustilanteessa. Jos odotettuja tuloksia ei ole kirjattu ylös, on suuri vaara, että tärkeitä yksityiskohtia jää huomaamatta. Jotta oikeanlaiset testitapaukset saataisiin poimittua, tulee pereh-

työ vaatimuksiin, arkkitehtuuriin, suunnitteluun ja rajapintoihin. Samalla selvitetään, ovatko vaatimukset sekä järjestelmä todella testattavissa. (Hambling ym. 2007, 22 - 23.)

Riippumatta testaustasosta löydetyt virheet tulee raportoida ja analysoida. Virheestä on kirjattava sen kuvaus, vakavuusaste, löytöajankohta, miten se olisi voitu löytää aiemmin, milloin se oli tehty ja miten se olisi voitu estää. Virheet voidaan kirjata järjestelmätestauksessa esimerkiksi virhepäiväkirjaan tai erilliselle virheilmoituslomakkeelle. Raporttien avulla saadaan selville testaukseen käytetty aika ja yrityksen laatujärjestelmän kehityskohteet. (Haikala & Märijärvi 2002**,** 297 - 298.)

Testauksen loppupuolella tarkastetaan ovatko testaukselle suunnittelussa asetetut lopetuskriteerit saavutettu. Jos testejä on suoritettu vähemmän kuin lopetuskriteereissä on määritelty, on vaihtoehtona joko muuttaa lopetuskriteeriä pienemmäksi tai suorittaa enemmän testejä. On myös mahdollista, että tarvitaan enemmän testitapauksia, kuin alun perin suunniteltiin. Testauksesta kirjoitetaan raportti asiakkaalle, josta käy ilmi suoritetut testit tuloksineen, poikkeukset sekä tapaukset, joita ei testattu. (Hambling ym. 2007, 24.)

Testaamisvaiheen päättyessä tarkistetaan dokumentaatiosta, että kaikki määritelty on toimitettu asiakkaalle. Sovitut raportit ovat kirjoitettuna. Testausympäristö dokumentoidaan ja arkistoidaan, samoin kuin testi-infrastruktuuri ja käytetyt työkalut. Seuraavaan vaiheeseen siirtyvät järjestelmän kehitystehtävät kirjataan selkeästi ylös. On myös hyvä raportoida, mitä projektista opittiin ja mitä voisi kehittää tulevia projekteja ajatellen. (Hambling ym. 2007, 25.)

7.5 Testitapausten valinta

Testitapaukset voidaan valita kahdella eri tavalla: lasilaatikkotestauksella (white box testing) tai mustalaatikkotestauksella (black box testing). Lasilaatikkotestauksessa hyödynnetään tietoa ohjelman toteutuksesta valittaessa testitapauksia. Mustalaatikkotestaus tarkoittaa testitapausten valitsemista spesifikaatioiden perusteella tietämättä ohjelman toteutuksesta. Harmaalaatikkotestaus (gray box) hyödyntää testattaessa tietoa ohjelman toteutusperiaatteista.

V-mallissa siirryttäessä alatasoilta ylemmäs muuttuu testauksen luonne yleensä lasilaatikkotestauksesta yhä enemmän mustalaatikkotestaukseksi. (Haikala & Märijärvi 2002**,** 289.)

Mustalaatikkotestauksessa testitapausten valinta tapahtuu jakamalla syöteavaruus ekvivalenssiluokkiin. Tässä oletetaan, että jos ohjelma toimii yhdellä ekvivalenssiluokan edustajalla, toimii se myös muilla kyseisen luokan edustajilla. Tätä kutsutaan ekvivalenssiositukseksi ja sen mukaan syötteet voidaan jakaa seuraavasti: epäkelvot syötteet, liian pienet tai suuret syötteet sekä kelvolliset syötteet. Testitapauksiksi voidaan valita lisäksi luokkien rajatapauksia eli tällöin käytetään *raja-arvoanalyysiä*. (Haikala & Märijärvi 2002**,** 290.)

7.6 Riittävä testaus?

Testauksen määrän arviointi on vaikeaa, ellei jopa mahdotonta. Testausta jatketaan usein järjestelmätasolla, kunnes joko aika tai rahat tai molemmat loppuvat. Testauksen lopettamisen hyväksymiskriteerit tulisi määrittää testaussuunnitelmassa. Esimerkiksi testauksen voidaan sopia olevan riittävää, kun löydettyjen virheiden määrä tasaantuu. Tällöin on kuitenkin vaikea arvioida testaukseen tarvittavaa aikaa, sillä on mahdotonta sanoa, koska tasaantuminen tapahtuu. Projektilla voi olla myös kiinteä, lukkoon lyöty aikataulu ja kiinteät resurssit. (Haikala & Märijärvi 2002**,** 291.)

Kaikkea mahdollista ei voida testata, joten jossakin vaiheessa tulee päättää, milloin on testattu tarpeeksi. Päätöstä helpottaa, kun on tiedossa, millaisia riskejä järjestelmässä voi olla ja kuinka laadukas sen tulee olla. Tärkein tapa huolehtia siitä, että järjestelmä on laadukas, on priorisoida testitapaukset ja suorittaa tärkeimmät testit ensin. Tällöin voidaan varmistua siitä, että milloin tahansa testaus sitten päättyykin, on aina tärkeimmät testit tehty. Tärkeimmät testitapaukset tunnistaa siitä, että asiakkaat arvostavat tiettyjä toiminnallisuuksia enemmän ja riskien toteutuessa niistä aiheutuu suurinta vahinkoa. (Hambling ym. 2007, 13.)

Toiseksi tärkeintä on asettaa kriteerit testauksen lopettamiselle ilman, että aikaraja tai muut paineet vaikuttavat asiaan. Nämä kriteerit määrittävät mm.

sen, kuinka suuri osa ohjelmistosta testataan ja kuinka paljon valmiissa tuotteessa saa olla virheitä. (Hambling ym. 2007, 13 - 14.)

Yksi keino tutkia testauksen kattavuutta on ns. virheiden kylväminen. Tällöin ohjelmakoodiin lisätään tahallisesti virheitä, ja löydettyjen virheiden määrästä voidaan yrittää päätellä ohjelmaan jääneiden virheiden lukumäärää. Jos löydettyjen ”kylvettyjen” virheiden määrä vastaa suunnilleen ”oikeiden” virheiden määrää, voidaan virhemäärää arvioida. Ohjelmasta voidaan myös tehdä eri versioita, joista jokaiseen kylvetään eri virhe. Tätä kutsutaan mutaatiotestaukseksi. Virheiden kylvämiseen liittyy kuitenkin seuraavia ongelmia:

- Koska resurssit ovat usein rajatut, lisätyöt eivät ole toivottavia.
- Kaikki kylvetyt virheet on muistettava poistaa.
- Testaus ei ole ehkä validia, koska koodia on muutettu testauksen jälkeen.
- Oikeanlaisen kylvettävän virheen löytäminen voi olla vaikeaa.

Tapaa voidaan käyttää tosin menestyksekkäästikin siten, että dokumentteja tarkastettaessa tekijä voi jättää niihin selvästi itse huomaamiaan virheitä testatakseen tarkastajien perehtymisen huolellisuutta. (Haikala & Märijärvi 2002**,** 294.)

Edes huolellisesti suunniteltujen ja toteutettujen testitapausten avulla ei ole mahdollista todistaa järjestelmää virheettömäksi. Testauksella voidaan osoittaa ohjelmistossa olevat virheet. Siltikään kaikkia virheitä ei saada selville, sillä testaukseen käytettävissä oleva aika on rajallinen ja kaikkien mahdollisten testitapausten määrä olisi liian suuri testattavaksi. Tämä ei tarkoita sitä, ettei testaukseen kannattaisi panostaa, vaan sitä, ettei ohjelmiston toimivuuteen kannata luottaa liikaa, vaikka testaustulokset olisivatkin hyviä. Tämä korostaa myös hyvää ohjelmiston etukäteissuunnittelua sekä ohjelmointia. (Haikala & Märijärvi 2002**,** 284.)

7.7 Testauksen työkalut

Jotta testaus olisi mahdollisimman tehokasta, olisi etenkin regressiotestauksen hyvä olla automatisoitu. Automatisoinnin työkaluja ovat esimerkiksi testipetigeneraattorit, testitapausgeneraattorit, vertailijat ja testikattavuustyökalut.

Testityökalujen käyttö voi vaikuttaa kuitenkin järjestelmän toimintaan ja näitä ongelmia ilmenee etenkin testattaessa sulautettuja järjestelmiä. (Haikala & Märijärvi 2002**,** 294.)

Testipetigeneraattorilla generoidaan testipeti testattavalle ohjelmamoduulille. Testipedille kuvataan testikuvauskielellä ajettava testi. Samalla kielellä on mahdollista kuvata halutut testitulokset, joten testin tulosten tarkastelu voidaan automatisoida. Ns. play back -työkaluilla voidaan järjestelmätestauksessa nauhoittaa testitilanteen syötteet ja käyttää uudelleen myöhemmin. (Haikala & Märijärvi 2002**,** 294.)

Testitapausgeneraattorin käyttö voi monesti olla mahdotonta, sillä sen käyttö edellyttää testattavan ohjelman toiminnan formaalia määrittelyä. Generaattorin käyttö onnistuu kuitenkin esimerkiksi silloin, jos ohjelman käyttöliittymä on määritelty tilakaaviona. Tällöin testiaineisto generoidaan tilakaavion perusteella ja testitulosten tarkastelukin on mahdollista automatisoida. (Haikala & Märijärvi 2002**,** 294 - 295.)

Testauksen tuloksia voidaan verrata aikaisempiin tuloksiin erityisten vertailijaohjelmien avulla. Niiden käyttöä vaikeuttavat kuitenkin muuttuvat tiedot, kuten kellonajat tuloksissa. Testikattavuusanalysaattoreilla mitataan puolestaan testin kattavuutta kattavuusmittojen avulla. Järjestelmätestauksessa kattavuusanalysaattorin käyttö voi olla kuitenkin hankalaa, sillä se voi kasvattaa koodin kokoa ja hidastaa suoritusta. (Haikala & Märijärvi 2002**,** 295.)

8 KÄYTTÄJÄKERTOMUS

Koska ohjelmistovaatimukset ovat usein kommunikaatio-ongelma, täytyy niiden, jotka haluavat uuden ohjelman joko myydäkseen tai käyttääkseen sitä, kommunikoida niiden kanssa, jotka ohjelman rakentavat. Kommunikoinnin tulee olla tasapuolista eri osapuolien välillä, jotta projektilla olisi edes mahdollisuus menestyä. Käyttäjäkertomukset eli user storyt toimivat tämän kommunikoinnin apuvälineenä.

8.1 Mikä on käyttäjäkertomus?

Käyttäjäkertomus kuvaa ohjelmiston tai järjestelmän toiminnallisuutta, joka on arvokas asiakkaalle. Se koostuu kolmesta osasta:

1. Kirjoitettu kuvaus, jota käytetään suunnittelun apuna ja muistilappuna.
2. Keskustelu kertomuksesta, jotta sen yksityiskohdat saadaan selville.
3. Testit ja dokumentin yksityiskohdat, joiden perusteella voidaan päätellä kertomuksen olevan valmis.

Käyttäjäkertomukset kirjoitetaan yleensä paperille eli kortille. Vaikka kortti on käyttäjäkertomuksen näkyvin osa, se ei ole tärkein. Kun kortti sisältää itse kuvauksen vaatimuksesta, sen yksityiskohdat selviävät siitä käytävässä keskustelussa ja lopulta se vahvistetaan. Esimerkiksi työnhakuportaalin käyttäjäkertomuksia voivat olla: Käyttäjä voi lähettää sivustolle hakemuksen tai käyttäjä voi rajoittaa, ketkä näkevät hänen hakemuksensa. (Cohn 2007, 4.)

Käyttäjäkertomukset eivät kuvaa ei-toiminnallisia vaatimuksia, kuten tietokantayhteyksiä tai sitä, millä ohjelmointikielellä ohjelma kirjoitetaan. Esimerkiksi vaatimus, että ohjelman tulee olla kirjoitettu C++ -kielellä, on huono, koska ohjelman käyttäjät eivät yleensä välitä, millä kielellä ohjelma on kirjoitettu. Myös vaatimus tietynlaisesta tietokantayhteydestä on huono, koska loppukäyttäjiä eivät kiinnosta tekniset yksityiskohdat. Jos tietty tietokantayhteys kuitenkin on käyttäjien kannalta oleellinen asia, on asia ilmaistava siten, että käyttäjä arvostaa sitä. (Cohn 2007, 5.)

Vaikka käyttäjäkertomuksessa on yksinkertaista sanoa, mitä asiakas ohjelmalta haluaa, ongelmana on, kuinka se toteutetaan näiden suppeilta vaikuttavien kertomusten perusteella. Yksityiskohtia tarvitaan, jotta koodaaminen ja testaaminen olisi mahdollista. Monet yksityiskohdat voidaan ilmaista lisäkertomuksilla ja usein on parempi olla paljon kertomuksia kuin liian pitkiä sellaisia. Liikoja yksityiskohtiakaan ei kertomuksiin kannata kirjoittaa, vaan niihin on paras palata sitten, kun ne ovat ajankohtaisia keskustelemalla niistä asiakkaan kanssa. Muutamat huomiot kortissa ovat vielä paikallaan, mutta pääpaino on keskustelussa. (Cohn 2007, 5.)

Loppukäyttäjien odotusten ja korttien vastaavuus saadaan parhaiten selville testauksen avulla. Nämä odotukset voidaan kirjoittaa korttien takapuolelle, jos kyseessä ovat paperilappuset. Odotukset toimivat eräänlaisena muistilappuna testaustilanteessa siitä, kuinka ohjelman tulisi toimia. Jos kortit kirjoitetaan sähköiseen muotoon, on kyseisessä sovelluksessa usein mahdollisuus kirjoittaa lisähuomioista. Kertomusten keräämiseen varta vasten suunniteltuja ohjelmia ovat esimerkiksi VersionOne, Xplanner ja Select Scope Manager. Testikuvausten tulee olla lyhyitä ja valmiita niiden ei tarvitse olla, sillä niitä voidaan lisäillä tai poistaa milloin vain. Tarkoitus on antaa testikuvauksella kertomuksesta lisätietoa, jotta kehittäjät tietävät päätellä, milloin tarvittava vaatimus on toteutettu. (Cohn 2007, 8.)

8.2 Prosessin eteneminen

Perinteisesti projekteissa (etenkin vesiputousmallin mukaisesti etenevissä) asiakkaat ja käyttäjät ovat projektin kanssa tekemisissä alun vaatimusmäärittelyjen yhteydessä ja seuraavan kerran vasta testauksen ollessa ajankohtaista. Tässä välissä kosketus asiakkaaseen saattaa puuttua täysin. Käyttäjäkertomukset kuitenkin sitouttavat asiakkaan projektiin koko sen keston ajaksi. Asiakkaan tulee näytellä todella aktiivista roolia kirjoitettaessa kortteja; kirjoittaminen kannattaa aloittaa yleensä luokittelemalla ohjelman loppukäyttäjät. (Cohn 2007, 8.)

Käyttäjäkertomuksille on tärkeää, että nimenomaan asiakas kirjoittaa ne. Kortit pitää olla kirjoitettuna kaupallisella kielellä, ei teknisellä, jotta asiakas voi prio-

risoida ne. Lisäksi asiakas on ohjelman päävisionäärinä paras asiantuntija kuvaamaan sen toimintaa. (Cohn 2007, 9.)

Olennaisimmat käyttäjäkertomukset kirjoitetaan projektin alussa, mutta niitä voidaan lisäillä pitkin matkaa. Erityisissä kertomustenkirjoitus-workshopeissa jokainen osallistuja kirjoittaa niin monta kertomusta kuin mahdollista. Kehittäjät arvioivat koottujen kertomusten perusteella, paljonko niiden toteuttamiseen kuluu aikaa. Yhteistyössä asiakas ja kehittäjät valitsevat iteraation pituuden, tavallisimmin yhdestä neljään viikkoa. Jokaisen iteraation loppuun mennessä kehittäjät ovat velvollisia toimittamaan täysin käytettävän koodin jostakin sovelluksen osasta. Asiakas määrittelee iteraation aikana testit ja toimii yhteistyössä kehittäjien kanssa testauksen suorittamiseksi. Asiakas varmistaa, että kehitystiimi on matkalla kohti haluttua päämäärää. (Cohn 2007, 9.)

Kehittäjät arvioivat valitun iteraation keston perusteella, kuinka paljon työtä per iteraatio he pystyvät tekemään. Tätä kutsutaan *vauhdiksi (velocity)*. Arvioinnin avustuksella voidaan tehdä raaka arvio tai julkaisusuunnitelma siitä, mitä työtä tehdään missäkin iteraatiossa ja montako iteraatiota tarvitaan. (Cohn 2007, 9.)

Kertomukset lajitellaan ja pinotaan siten, että jokainen pino edustaa yhtä iteraatiota. Korkeimmalle priorisoidut kertomukset sijoitetaan ensimmäiseen pinoon ja siten ensimmäiseen iteraatioon, toiseksi priorisoidut toiseen ja niin edelleen. Tätä jatketaan niin kauan, että projektin aikaraja tulee vastaan tai pinot edustavat haluttua tuotetta. Toteutuneiden iteraatioiden avulla voidaan laskea projektille sen todellinen kesto, sillä ennen kuin yhtään iteraatiota on toteutettu, on mahdotonta sanoa sen todellista kestoa. Parantuneen aika-arvion myötä pinoja voidaan joutua muokkaamaan: joko lisäämään tai poistamaan kertomuksia riippuen niiden toteutuksen vaikeusasteesta. (Cohn 2007, 10.)

Hyväksymistestauksen avulla todennetaan, että jokainen kertomus toimii juuri kuten asiakas halusikin sen toimivan. Iteraation alussa, kun toteuttajat aloittavat työnsä, alkaa myös testien määrittely. Tämä voi tarkoittaa esimerkiksi testien kirjoittamista korttien taakse tai vaikkapa automaattisten testaustyökalujen

valmistelua. Antamalla testitapaukset toteuttajille varhaisessa vaiheessa varmistutaan siitä, ettei olennaisia asioita unohdu toteutuksesta. (Cohn 2007, 13.)

8.3 Hyväksymistestit

Hyväksymistestien kirjoittamisen yksi syy on ilmaista kertomusten yksityiskohtia. Tämä on parempi tapa kerätä vaatimuksia kuin kirjoittaa pitkiä listoja perinteisen vaatimusmäärittelyn hengessä. Testitapauksia kirjoitetaan korttien taakse sitä mukaa, kun niitä syntyy ja ne toimivat myöhemmin apuna varmistettaessa, että kertomus on koodattu kokonaan ja oikein. Yksityiskohdat toimivat toteuttajien muistiapuna koodatessa, jolloin riski unohtaa jokin tärkeä ominaisuus pienenee. Testitapaukset on vain kirjoitettava ennen uuden iteraation aloittamista. (Cohn 2007, 70.)

Jotta ohjelma vastaisi asiakkaan vaatimuksia, on asiakkaan itsensä kirjoitettava testitapaukset. Toteuttajat ja testaajat voivat olla apuna kirjoittamisessa, mutta asiakkaan on osallistuttava vähintään niiden testien määrittelyyn, joiden perusteella testaus aikanaan katsotaan päättyneeksi. Testejä pitää kirjoittaa niin kauan, kuin ne tuovat lisäarvoa kertomukselle. Koska hyväksymistestien tehtävä on osoittaa, että ohjelma on kelvollinen asiakkaalle, joka on osallistunut ohjelman kehittämiseen, tulee asiakkaan ottaa osaa testaukseenkin. (Cohn 2007, 70.)

Hyväksymistestit tulisi suorittaa vähintään jokaisen iteraation päätteeksi. Koska seuraavassa iteraatiossa voi jokin osa jo toimivaksi todetusta koodista rikkoutua, kannattaa kaikista pääiteraatioista tehdä hyväksymistestit. Mitä useampi iteraatio on takana, sen aikaa vievemmäksi testaus muuttuu, joten tässä vaiheessa on hyvä harkita testauksen automatisointia, mikäli mahdollista. (Cohn 2007, 70.)

8.4 Käyttäjäkertomusten kerääminen

Kerättäessä käyttäjäkertomuksia kannattaa hyväksyä se, ettei kaikkia vaatimuksia saada selville yhdellä kertaa. Ne myös muuttuvat ajan myötä ja niiden tärkeysjärjestys voi muuttua. Käyttäjäkertomusten yksi etu on se, että niitä

voidaan helposti kirjoittaa eri tarkkuustasoilla. Kertomus voidaan kirjoittaa sille tasolle, mikä juuri sillä hetkellä on tiedossa ja täydentää sitä yksityiskohdilla myöhemmin. Kertomus voidaan myös kirjoittaa hyvin yleisellä tasolla ja pilkkoa pienempiin osiin tarvittaessa. Tämä tarkoittaa sitä, että samoja kertomuksia voidaan käyttää monille sovelluksille. Näin voidaan vähentää työmäärää projektin alkuvaiheessa. (Cohn 2007, 44.)

Käyttäjäkertomuksia voidaan kerätä eri tavoin: käyttäjähaastatteluilla, kyselyillä, havainnoimalla ja workshopien avulla. Käyttäjähaastatteluissa on tärkeää haastatella ohjelman todellisia käyttäjiä. Haastateltavien tulee myös edustaa eri käyttäjärooleja. Jos käyttäjäkunta on hyvin suuri ja halutaan vastauksia juuri tiettyihin kysymyksiin, kannattaa käyttää kyselylomakkeita. Havainnointi on menetelmänä tehokas, mutta siihen on harvoin mahdollisuutta, jos kehitystyö ei tapahdu asiakkaan ympäristössä. Aina käyttäjät eivät ehkä osaa ilmaista tarpeitaan suoraan, vaan kehittäjät näkevät heidän todelliset tarpeensa vasta havainnoimalla heitä. Workshopit kokoavat yhteen kaikki kehitettävän ohjelmiston osapuolet ja ideana on kirjoittaa mahdollisimman monta kertomusta. Kertomusten priorisointia ei tässä vaiheessa vielä suoriteta. Myöskään ongelmien ratkaisuja ja ulkoasusuunnitteluja ei näihin workshopeihin tule sisällyttää. (Cohn 2007, 52.)

8.5 Millainen on hyvä käyttäjäkertomus?

Käyttäjäkertomuksen kirjoittaminen kannattaa aloittaa tunnistamalla ohjelman käyttäjät ja se, kuinka käyttäjät ovat vuorovaikutuksessa tulevan ohjelman kanssa. Kertomukset on myös hyvä kirjoittaa aluksi hyvin yleisellä tasolla ja tarkentaa niitä myöhemmin. Suuret kertomukset on hyvä pilkkoa pienemmiksi tietyin perustein, kuten esimerkiksi teknisten rajojen mukaan. Tämä pilkkominen voi auttaa löytämään virheitä ohjelman eri kerroksista jo varhain. (Cohn 2007, 76.)

Kortteihin kannattaa sijoittaa toisinaan rajoitteita. Nämä toimivat hyvinä muistilappuina kehityksessä ja auttavat luomaan täsmällisiä testitapauksia. Tällainen rajoite voi olla vaikkapa se, että ohjelman on tuettava 50 yhtäaikaista käyttäjää. (Cohn 2007, 78.)

Monesti vaatimukset ja ohjelman määrittely yhdistetään toisiinsa vaatimuksia kerättäessä. Käyttäjäkertomusten kohdalla itse ohjelman määrittely kannattaa jättää taka-alalle, sillä tässä eksytään helposti käyttöliittymäsuunnittelun puolelle, jota korttien kirjoittaminen ei ole. Käyttöliittymäsuunnitteluun liittyy monesti runsaasti näyttökaappauskuvia, joten se ei sovellu käyttäjäkertomuskortteihin. Tällaisten vaatimusten kohdalla kannattaa käyttää muuta formaattia kuin kortteja. (Cohn 2007, 80.)

Kertomukset on hyvä osoittaa tietylle roolille: loppukäyttäjä saa näin kasvot kehittäjän mielessä ja loppukäyttäjän tarpeet tulee tyydyttää. Kertomuksista tulee myös luettavampia, kun ne kohdistetaan yhdelle ihmiselle usean sijasta. Esimerkiksi lauseesta ”käyttäjät voivat muokata tietojaan sivustolla” voi saada käsityksen, että käyttäjät voivat muokata toistensa tietoja. Yksittäisellä käyttäjällä merkitys on selkeämpi. (Cohn 2007, 81.)

Ihannetapauksessa asiakas kirjoittaa käyttäjäkertomuksen; usein toteuttajat kuitenkin auttavat joko kirjoittamalla itse kertomuksen kortille workshopissa tai ehdottamalla uusia. Vastuuta kirjoittamisesta ei kuitenkaan saa sysätä toteuttajille, eikä kannatakaan, sillä kirjoittamalla itse asiakas oppii ymmärtämään kertomuksia paremmin, ja tämä puolestaan helpottaa niiden sijoittamista iteraatioihin. Kertomukset priorisoidaan sen mukaan, kuinka tärkeäksi asiakas ne kokee. (Cohn 2007, 82.)

Korttien päätarkoitus on toimia ominaisuuskeskustelujen muistilappuina ja ne tulee pitää lyhyinä. Niihin lisätään olennaiset yksityiskohdat, mutta niiden ei tule korvata keskustelua liioilla sellaisilla. (Cohn 2007, 82.)

8.6 Mitä käyttäjäkertomukset eivät ole?

Käyttäjäkertomusten lisäksi on kolme muuta yleistä tapaa lähestyä vaatimuksia: käyttötapaukset, IEEE 830 -ohjelmistovaatimusmäärittely sekä vuorovaikutuskäsikirjoitukset. (Cohn 2007, 133.)

8.6.1 Käyttötapaukset

Käyttötapaukset eli use caset liitetään osaksi useita ohjelmistonkehitysmenetelmiä, kuten OMT++. Käyttötapauksia pidetään usein hyvänä vaihtoehtona käyttäjävaatimusten kartoittamiseen. Ideana on kuvata järjestelmän toiminnallisuuksia ja järjestelmän käyttäjien sillä suorittamia tapahtumaketjuja. Apuna kuvaamisessa voidaan käyttää esimerkiksi tapahtumasekvenssikaavioita ja kuhunkin käyttötapaukseen liittyy käyttäjärooleja, jotka suorittavat kyseisen käyttötapauksen. Käyttötapaukset ja roolit kuvataan käyttötapauskaaviossa. Jos käytetään UML-mallin mukaista notaatiota, voidaan käyttötapauskaavioon merkitä myös käyttötapauksien välisiä suhteita. Tarkoitus on pitää kuitenkin käyttötapaukset yksinkertaisina ja kaikkien asianosaisten ymmärrettävissä. Käyttötapaus alkaa aina käyttäjäroolin aloitteesta ja päättyy järjestelmän käyttäjälleen tuottamaan lisäarvoon, jolloin käyttäjä on saanut halutun tehtävän suoritetuksi. (Haikala & Märijärvi 2002**,** 155.)

Hyvä käyttötapaus on ymmärrettävä, joten kuvaustavan on oltava mahdollisimman konkreettinen. Käyttötapauksen tulee kuvata asiakasvaatimuksia ja siinä ei oteta kantaa järjestelmän toteutukseen. Käyttötapauksen testattavuus on tärkeää, sillä se luo pohjan järjestelmätestaukselle joko yhtenä tai useampana peräkkäin ajettavana testitapauksena. Käyttötapaus ei saa olla liian suuri, joten yksi A4 on sopiva koko. Kaikkia yksityiskohtia ei myöskään kannata ottaa mukaan käyttötapaukseen. Käyttötapauksia voidaan hyödyntää määrittelyn lisäksi muissakin vaiheissa. Ne luovat hyvän pohjan käyttöohjeille ja oliokeskeisissä menetelmissä niitä tutkimalla voidaan löytää oliomallin keskeisiä käsitteitä. (Haikala & Märijärvi 2002**,** 157.)

Käyttötapausten merkitys on toimia kommunikointivälineenä asiakasvaatimuksia kartoitettaessa ja kuvattaessa ohjelmistovaatimuksia. Niillä ei kuitenkaan voi korvata esimerkiksi toiminnallisen määrittelyn kuvauksia järjestelmän toiminnasta ja siksi kaikista järjestelmän ominaisuuksista ei kannata tehdä käyttötapauksia. Jotkut asiakasvaatimukset voivat näkyä useissa käyttötapauksissa ja jotkut eivät missään niistä. Käyttötapausta ei voi käyttää pohjana arkkitehtuurisuunnittelulle ja toteutustyötä on vaikea jakaa ohjelmoijille käyttötapauskohtaisesti. (Haikala & Märijärvi 2002**,** 158.)

Yksi selkeimpiä eroja käyttötapausten ja käyttäjäkertomusten välillä on laajuus. Käyttäjäkertomukset ovat pienempiä, koska niiden kokoa rajoitetaan tarkoituksella. Käyttäjäkertomukselle voidaan asettaa rajoitteeksi, ettei sen toteutukseen saa kulua esimerkiksi viittä päivää enempää aikaa, jotta sitä voidaan käyttää apuna projektin aikatauluttamisessa. Toinen eroavaisuus on kestoaika. Käyttötapaukset ovat olemassa niin kauan kuin projekti kestää tai on ylläpidon alaisena. Käyttäjäkertomukset lakkaavat olemasta iteraation päätyttyä, kun ne on lisätty ohjelmaan. (Cohn 2007, 140.)

Käyttäjäkertomuksia ei juurikaan arkistoida, vaan ne hävitetään vaiheen päätyttyä. Käyttötapauksiin sisällytetään usein kuvauksia käyttöliittymästä, koska ei tiedetä muutakaan sopivaa paikkaa niille ja koska käyttötapausten kirjoittajat keskittyvät liian aikaisin itse ohjelman kehittämiseen liiketaloudellisen osan sijasta. Käyttäjäkertomusten ollessa kyseessä käyttöliittymä tulee luonnostaan puheeksi asiakkaan kanssa käytävissä keskusteluissa. Käyttötapaukset ja käyttäjäkertomukset on tarkoitettu eri tilanteisiin. Käyttötapauksen ideana on toimia dokumenttina siitä, mitä ohjelmasta on sovittu asiakkaan ja kehittäjien välillä, kun taas kertomukset ovat muistin tukena asiakkaan tarpeista käytävissä keskusteluissa. Käyttötapaus syntyy tarkan ja pitkän analyysin lopputuloksena; käyttäjäkertomuksia käytetään pohjana tarkemmalle keskustelulle ja suunnittelulle. (Cohn 2007, 140.)

8.6.2 IEEE 830 -ohjelmistovaatimusmäärittely

IEEE 830 on IEEE:n (The Computer Society of the Institute of Electrical and Electronics Engineers) ohjeistus vaatimusmäärittelyn kirjoittamisesta. Se tunnetaan IEEE 830 -standardin nimellä ja se kattaa kuvauksen itse vaatimusmäärittelydokumentista, prototyyppien roolista ja hyvistä vaatimuksista. IEEE:n mukaiset määrittelyt alkavat yleensä sanalla ”järjestelmä”, esimerkiksi ”järjestelmällä voi maksaa ostokset verkkopankin välityksellä”. Järjestelmän vaatimukset listataan peräkkäin kaikki samaan listaan. Tämä tapa on aikaa vievä ja virhealtis. Mitä isompi järjestelmä on kyseessä, sitä enemmän listassa on vaatimuksia. Pitkää listaa voi olla tylsä lukea, jolloin kasvaa riski olla huomaamatta joitakin vaatimuksia. Toteuttajien on myös vaikea saada tällaisesta dokumentista kokonaiskuvaa järjestelmästä. Vaatimusmäärittely on työläs toteuttaa ja asiakas on usein lopputuloksen nähtyään tyytymätön, jolloin kallista

työaikaa on kulunut paljon hukkaan. IEEE:n tapa määritellä voi myös vahvistaa käsitystä siitä, että järjestelmä on valmis, kun se täyttää kaikki listatut vaatimukset; käyttäjän vaatimusten toteutumisesta se ei sano mitään. Projektin lopputulos ei siten läheskään aina tyydytä asiakasta. Lyhyesti sanottuna IEEE:n vaatimusmäärittely listaa vaatimukset, käyttäjäkertomus kertoo käyttäjän tavoitetilan. (Cohn 2007, 135.)

Usein projekti etenee siten, että vaatimusten dokumentointiin IEEE:n mukaisesti tuhlataan runsaasti aikaa projektin alkuvaiheessa ja kun dokumentti sitten ojennetaan toteuttajille, he arvioivat siihen kuluvan ajan olevan vähintään tuplasti sen, mitä asiakas alun perin ajatteli. Aikaa on näin jo hukattu kirjoittamalla liian kauan vaatimuksia, joita toteutustiimillä ei ole sitten enää aikaa toteuttaa. Lisää aikaa tuhlataan arvioimalla, mitkä ominaisuudet voidaan aikapulan vuoksi jättää pois toteutuksesta. Projektin kokonaiskustannusten arvioiminen menee näin myös poikkeuksetta pieleen. (Cohn 2007, 136.)

8.6.3 Vuorovaikutuskäsikirjoitukset

Käyttäjäkertomukset eivät ole käsikirjoituksia (scenarios) siitä, kuinka käyttäjä on vuorovaikutuksessa tietokoneen kanssa. Käsikirjoituksia ei pidä sotkea käyttötapauksiinkaan. Vuorovaikutuskäsikirjoitukset ovat pidempiä ja käsittävät suurempia kokonaisuuksia kuin käyttötapaukset. Käsikirjoituksille tai skenaarioille, kuten niitä voidaan myös kutsua, on tyypillistä, että niissä mainitaan paikka, jossa toimitaan, toimijat, maalit tai tavoitteet sekä toiminnot ja tapahtumat. Käyttäjäkertomuksista käsikirjoitukset eroavat laajuudellaan ja yksityiskohtaisuudellaan. Käsikirjoitukset koostuvat monista käyttäjäkertomuksista. Käyttötapauksista käsikirjoitukset eroavat siten, että ne jättävät tilaa keskustelulle valmistuttuaan, ja usein niissä onkin avoimia kohtia, jotka sitä vaativat. (Cohn 2007, 142.)

8.7 Mikä aiheuttaa käyttäjäkertomuksissa ongelmia?

Käyttäjäkertomus voi olla liian lyhyt, jolloin se aiheuttaa ongelmia kertomusten arvioinnissa ja aikatauluttamisessa. Lyhyille kertomuksille asetettujen aikaarvioiden toteutuminen voi vaihdella huomattavasti riippuen siitä, missä järjestyksessä kertomukset toteutetaan ja menetetty aika on aina pois muiden ker-

tomusten kehittämiseltä. Kertomuksia kannattaa pilkkoa vasta, kun ne on sijoitettu iteraatioon. (Cohn 2007, 157.)

Iteraatioiden suunnittelu vaikeutuu, jos kertomusten välillä on riippuvuuksia. Tällöin kertomukset joudutaan sijoittamaan iteraatioihin vain, koska toinenkin tietty kertomus on lisätty siihen, eikä kahta ilman kolmatta. Tätä ilmenee, jos kertomukset ovat liian lyhyitä tai pilkottu epäsopivasti. Ratkaisu on yhdistellä riippuvaisia kertomuksia toisiinsa tai tutkia, onko pilkotussa kertomuksessa toiminnallisuuksia kaikista sovelluksen kerroksista. (Cohn 2007, 158.)

Käyttäjäkertomuksia käytettäessä luullaan monesti, että kaikki asiat tulisi ilmaista kertomuksina. Kaikkia vaatimuksia ei kuitenkaan onnistuta ilmaisemaan kertomuksina ja näitä ovat yleensä ei-toiminnalliset vaatimukset. Ei-toiminnalliset vaatimukset kuvaavat itse järjestelmää, kuten sen ulkoasua, ylläpidettävyyttä, oikeellisuutta, käytettävyyttä, turvallisuutta ja suorituskykyä. Monet tällaiset vaatimukset toimivat järjestelmän toiminnan rajoitteina ja ne voidaan ilmaista kirjoittamalla ne käyttäjäkertomuskortteihin rajoitteiksi. Esimerkiksi vaatimus ”järjestelmä kirjoitetaan C# -kielellä” on rajoite ja vaikuttaa muun järjestelmän suunnitteluun, joten se voidaan lisätä korttiin kohtaan ”rajoitteet”. (Cohn 2007, 178.)

Käyttöliittymäkuvauksia ei kannata liittää kertomuksiin liian aikaisin; ne pahimmillaan rajoittavat projektia. Esimerkiksi jos käyttäjän kerrotaan tekevän tiettyjä toimintoja tietyissä näkymissä, ei kovin varhaisessa vaiheessa voida vielä tietää, millaisia näkymiä käyttöliittymään lopulta tulee. (Cohn 2007, 160.)

Liiat yksityiskohdat tekevät huonon käyttäjäkertomuksen ja vaarana on, että aikaa käytetään enemmän kertomuksen kirjoittamiseen kuin siitä keskusteluun. Tätä voidaan ehkäistä kirjoittamalla kertomukset yksinkertaisesti korteille, jolloin kullekin kertomukselle on vain rajallinen tila. Jos tila silti loppuu, korttia on pienennettävä. Liiat yksityiskohdat karsiutuvat pakosta. (Cohn 2007, 159.)

8.8 Miksi tulisi käyttää käyttäjäkertomuksia?

Miksi kaikista vaatimustenkeruumenetelmistä pitäisi käyttää juuri käyttäjäkertomuksia? Käyttäjäkertomukset edistävät suullista kommunikointia. Vaikka vaatimukset olisi kirjoitettu paperille ja niistä olisi yksimielisesti sovittu, ei se silti tarkoita automaattisesti sitä, että ne ovat juuri sitä, mitä asiakas haluaa. Kirjoitettu teksti voi saada myös monia eri merkityksiä ja väärinkäsitysten uhka on suuri. Runsas suullinen kommunikointi edistää tietoutta rakennettavasta järjestelmästä koko projektiryhmässä. (Cohn 2007, 146.)

Käyttäjäkertomukset ovat ymmärrettäviä sekä asiakkaan että kehittäjän näkökulmasta; ennen kaikkea ne ovat vapaita liioista teknisistä yksityiskohdista, jotka tekevät vaatimukset vaikeasti luettaviksi. Tämä rohkaisee asiakkaita osallistumaan suunnitteluun. Vaikka esimerkiksi käyttötapaukset ovat vapaita teknisestä jargonista, täytyy asiakkaan opetella erikseen lukemaan niiden formaattia. Käyttötapauksissa käytetään kenttiä, kuten laajennos (extension), ennakkoehdot (preconditions) ja takuut (guarantees), jotka ovat etenkin ensikertalaiselle lukijalle vaikeita ymmärtää. IEEE:n dokumentit sisältävät puolestaan myös teknistä jargonia ja ovat pitkiä sekä hierarkkisia. Käyttäjäkertomukset ovat helppo tapa sitoa asiakkaat ohjelman kehitykseen; monet projektit epäonnistuvat, koska yhteys lopputuotteen käyttäjiin puuttuu. (Cohn 2007, 148.)

Käyttäjäkertomukset ovat oikean kokoisia suunnittelulle, mikä helpottaa niiden priorisoimista. Perinteisen IEEE:n standardin mukaisen vaatimusmäärittelyn vaatimusten priorisointi voi olla huomattavasti vaikeampaa asiakkaan näkökulmasta, sillä asiakkaalle koituvat hyödyt eivät käy niin selkeästi ilmi niistä. (Cohn 2007, 149.)

Käyttäjäkertomukset sopivat erityisen hyvin iteratiiviseen ohjelmistokehitykseen, sillä kaikkia vaatimuksia ei tarvitse kirjoittaa valmiiksi, ennen kuin koodaamisen voi aloittaa. Aluksi voidaan kirjoittaa vain osa kertomuksista, toteuttaa ne sekä testata ja aloittaa alusta uusilla kertomuksilla. (Cohn 2007, 149.)

Yksi kertomusten etu on, että yksityiskohtien suunnittelua voidaan lykätä myöhemmäksi. Tämä on hyvä etenkin tiukan aikataulun projekteissa: projektiryhmä voi hyvin nopeasti kirjoittaa joitakin kymmeniä vaatimuksia saadakseen kokonaiskäsityksen rakennettavasta järjestelmästä. Sitten valittuihin kortteihin lisätään yksityiskohtia ja koodaamaan päästään perinteistä IEEE:n mukaisesti etenevää projektia aikaisemmin. (Cohn 2007, 150.)

9 MIKÄ ON MOSS?

MOSS on Microsoftin palvelinpohjainen tuote ja sen lyhenne tulee sanoista Microsoft Office SharePoint Server. Se on julkaistu ensimmäistä kertaa vuonna 2001 ja uusin versio on vuodelta 2007. SharePoint on teknologia-alusta, joka sitoo yhteen toimistotyöntekijöiden asiakasohjelmien syötteet. Näitä tuotoksia voidaan hallita ja etsiä hakutoiminnolla SharePointin avustuksella. Myös tietoihin pääsyä voidaan rajata vain tietyille käyttäjäryhmille. (Williams 2007, 1 - 2.)

SharePoint mahdollistaa toimistotyöläisille eräänlaisen itsepalveluympäristön, jossa nämä voivat itse luoda sekä hallinnoida tarvitsemiaan dokumentteja ja jakaa niitä muille dokumenttikirjastojen välityksellä. SharePointin käyttäjät voivat esimerkiksi luoda projektikohtaisia tehtävälistoja, käyttää kalenteria, luoda asiakaskohtaisia räätälöityjä web-portaaleja kuin myös käyttää MOSSin valmiskomponentteja web-sivustojen rakentamiseen. (Williams 2007, 1 - 2.)

SharePoint tarjoaa palvelininfrastruktuurin tukemaan tietotyöläisten ja heidän työnantajiensa tarpeita. Tämä käsittää yhteistyön eri osapuolten välillä, dokumenttien varastoimisen ja mahdollisuuden tiedottaa sekä tulla tiedotetuksi tärkeistä asioista. Työntekijöitä uhkaava tietotulva voidaan välttää kohdistamalla oikea tieto vain tietyille ryhmille. (Williams 2007, 11 - 12.)

Tietotyöläisiä palkkaavien yritysten tulee puolestaan pystyä tarkastamaan, valvomaan, järjestämään, säilyttämään ja suojaamaan tietoa. SharePoint tarjoaa yrityksille mahdollisuuden sitoa tietotyöläisensä yhteen työkalujen avulla, joita he jo käyttävät: esimerkkinä tästä ovat Officen toimisto-ohjelmat (Word, Excel, PowerPoint), internetselaimet (Internet Explorer) ja e-mail (Outlook). Koska työkalut ovat työntekijöille ennestään tuttuja, vähentää se koulutuksen tarvetta ja tukikustannuksia. SharePointin avulla työnantajat tavoittavat työntekijänsä sieltä, missä nämä tekevät työtänsä eli työasemiensa äärestä. (Williams 2007, 11 - 12.)

10 MOSS-KÄYTTÄJÄKERTOMUKSET

Tässä kappaleessa esitellään käyttäjäkertomukset yleisimmistä MOSSin portaalipalveluista. Kertomukset on ryhmitelty toiminnallisuuksittain.

10.1 Hakutoiminto (Search)

- Käyttäjä voi hakea yhteystietoja puhelinluettelosta.

- Käyttäjä voi hakea portaalin sisältöä kuten dokumentteja ja erilaisia listauksia: esimerkiksi tapahtumia, kalenteritietoja, tehtäviä ja niin edelleen. Enterprise MOSS-palvelimen haku voidaan ulottaa myös verkkolevyille.

- Käyttäjä voi hakea tietoa erikseen: esimerkiksi puhelinluettelosta henkilöitä, portaalista sivuja tai dokumentteja ja muuta sisältöä, kuten kuvia.

- Käyttäjä voi hakea tietoja joko yksinkertaisella pikahaku- tai yksityiskohtaisemmalla laajennettu haku -toiminnallisuudella.

10.1.1 Pikahaku (The Quick Search)

- Käyttäjä voi käyttää pikahakua miltä tahansa sivulta portaalissa.

- Käyttäjä voi hakea pikahaulla tietoa sekä sivustoilta että sivuilta käyttämällä hakuehtona sanoja tai lauseita.

- Käyttäjä voi hakea pikahaun avulla sisältöä, joka voi olla esimerkiksi dokumentteja, PDF-tiedostoja ja muita sisältötyyppejä, kuten esimerkiksi multimediatiedostoja, miltä tahansa sivulta portaalissa.

10.1.2 Laajennettu haku (The Advanced Search)

- Käyttäjä voi hakea tietoja Boolen arvojen avulla.

- Käyttäjä voi hakea sivuja portaalista sivujen metatietojen avulla. Samoin käyttäjä voi hakea dokumentteja metatietojen perusteella.

- Käyttäjä voi hakea henkilöitä puhelinluettelosta nimen, esimiehen, sijainnin tai liiketoimintayksikön perusteella. Tämä ominaisuus on konfiguroitavissa.

- Käyttäjä voi hakea laajennetulla haulla tietoa erilaisten hakuehtojen avulla, kuten sanahaku Boolen operaattoreilla (AND, OR), dokumenttityyppi (esimerkiksi pöytäkirja, agenda), päivämäärä, tiedostotyyppi (esimerkiksi .doc, .xls tai .PDF), tai avainsanat. Tämä pitää konfiguroida erikseen.

- Käyttäjä voi määritellä miltä aikaväliltä hakutulokset haetaan aloitus- ja lopetuspäivämäärien perusteella. Tämä pitää konfiguroida erikseen.

10.2 Tuoreimmat päivitykset (Latest Updates)

- Käyttäjä näkee listauksen portaalin viimeisimmistä päivityksistä.

- Käyttäjä näkee vain profiilinsa mukaiset päivitykset.

- Julkaisija voi määrittää mikä sisältö (esimerkiksi uutiset) näkyy Tuoreimmat päivitykset -listalla.

10.3 Käyttötilastointi (Statistics)

Käyttötilastointi on MOSSin valmis ominaisuus.

- Käyttäjät, joilla on oikeus sivuston asetuksiin, kuten ylläpitäjät, suunnittelijat ja sisällön omistajat, voivat tarkastella portaalin käyttötilastoja.

- Portaalin ylläpitäjät ja sisällön omistajat voivat tarkastella käyttötilastoja käyttäjien aktiivisuudesta eri sisältöalueilla.

- Portaalin ylläpitäjät ja sisällön omistajat voivat tarkastella käyttäjien eniten käyttämiä sivuja.

- Ylläpitäjä voi tarkastella tilastoja tilastointiajanjaksoittain, joka on joko menneet 30 päivää tai menneet 12 kuukautta.

- Ylläpitäjä näkee hakutilastosta muun muassa millä hakusanoilla on haettu tietoa ja mikä on ollut suosituin tulossivu. Sivuston käytöstä listataan esimerkiksi eniten vieraillut sivut.

- Ylläpitäjän tarkastelema käyttötilasto sisältää raportteja sivuvierailuista, eniten vierailluimmista sivuista, aktiivisimmista käyttäjistä ja sivustoista, jotka ovat useimmiten viitanneet haettavalla sivulle.

- Ylläpitäjät voivat käyttötilastoista tulostettujen raporttien avulla tarkastella tietoja tehdyistä hauista ja sivuston käytöstä. Tärkeimmät raportit hyötyineen ovat:
    1. Hakujen määrä (viimeiset 30 päivää tai 12 kuukautta):
       Raportin käyttäjä voi päätellä tämän perusteella, kuinka hyvin sivusto toimii: jos hakujen määrä kasvaa tasaisesti, on sivuston käyttö kasvussa ja hakua pidetään hyvänä toimintona. Jos hakujen määrä laskee, on tilanne päinvastainen.

    2. Hakujen määrä per kiikari (scope):
       Raportin käyttäjä näkee, kuinka paljon eri scopeja käytetään suhteessa hakumääriin.

    3. Eniten haetut hakutermit (viimeiset 30 päivää):
       Raportin käyttäjä näkee, mitä palvelun käyttäjät odottavat löytävänsä ja tätä voidaan hyödyntää portaalin sisältösuunnittelussa ja toiminnallisessa kehityksessä. Sivustoa voidaan muokata siten, että hakutuloksista löytyvät parhaiten hakijoiden tarpeita vastaavat sivut.

4. Sivut, joita klikataan eniten hakutuloksista (viimeiset 30 päivää).

5. Hakutermit, jotka palauttavat nolla (0) tulosta (viimeiset 30 päivää):
   Raportin käyttäjä näkee tämän perusteella, mitkä hakutermit eivät tuota tuloksia. Näin näihin hakutermeihin voidaan reagoida ja määritellä niille tulokset, jotka ohjaavat hakijat oikeisiin paikkoihin.

6. Eniten klikatut best bets -tulokset (viimeiset 30 päivää):
   Raportin käyttäjä näkee listan niistä best bets -tuloksista, joita käyttäjät klikkaavat eniten.

7. Hakutermit, joille ei ole määriteltynä yhtään best bets -tulosta (viimeiset 30 päivää):
   Raportin käyttäjät voivat tämän raportin avulla toimittaa listan hakutuloksista, joita ei ole lainkaan muokattu, hakua kehittävälle työryhmälle paranneltavaksi.

8. Hakutermit, joiden tuloksissa on matalat klikkausprosentit (viimeiset 30 päivää):
   Raportin käyttäjä näkee millä hakutermeillä ei löydy hakijan tarpeita vastaavia sivuja, ja näin voidaan parantaa hakua.

10.4 Sivukartta (Sitemap)

Sivukartta on MOSSin vakiotoiminnallisuus.

- Käyttäjä voi tarkastella koko portaalin sisältöä ja rakennetta sivukartasta. Käyttäjä näkee sivukartasta pää- ja alasivut.

- Käyttäjä voi siirtyä sivukartasta suoraan linkin avulla haluamalleen sivulle.

- Käyttäjä näkee sivukartasta vain ne sivut, joihin hänellä on lukuoikeus.

10.5 Palaute (Feedback)

Palaute on konfiguroitava palvelu.

- Käyttäjä voi lähettää palautetta portaalista.
- Käyttäjä voi antaa palautetta omalla nimellään tai anonyymisti.
- Käyttäjä voi kohdentaa palautteen tietylle yksikölle tai koko organisaatiolle.
- Käyttäjä saa kuittauksen lähetetystä palautteesta.
- Käyttäjän lähettämä palaute tallennetaan. Myös vastaus palautteeseen tallennetaan.
- Palautetta voidaan tilastoida seuraavasti:
    1. keskimääräinen vastausaika
    2. vastausten määrä / kk / vuosi
    3. vastaanotetut palautteet / kk / vuosi / yksikkö/ organisaatio jne.
- Palautteen vastaanottajat (voi olla monia) voivat kirjautua palautearkistoon ja tutkia vastaamattomia palautteita riippumatta siitä, kenelle ne on kohdistettu.
- Palautteen vastaanottaja saa ilmoituksen sähköpostilla uudesta palautteesta.
- Vastaanottaja voi välittää palautteen edelleen toiselle vastaanottajalle.
- Vastaanottaja näkee palautteen saapumisajankohdan ja vastauksen ajankohdan.
- Vastaanottaja näkee palautteen antajan nimen ja sähköpostiosoitteen, jos palautteen antaja on niin määritellyt.

10.6 OmaSivu (MySite)

OmaSivua on saatavilla ainakin kahdeksan erilaista versiota, esimerkiksi sihteerin tai myyntihenkilöstön sivupohjat.

- Käyttäjällä on ns. oma sivu, jossa voi päivittää omia tietojaan ja tallentaa tärkeitä linkkejä.

- Käyttäjä näkee omalta sivultaan mm. eri sivujen ja työtilojen jäsenyytensä.

- Käyttäjä voi jakaa dokumentteja dokumenttikansioista muiden käyttäjien kanssa OmaSivun avulla.

- Käyttäjä voi käyttää OmaSivun kautta itselleen tärkeitä linkkejä ja dokumentteja ollessaan poissa toimistolta tai vaikkapa ulkomailla.

10.7 Dokumenttien hallinta (Document Management)

Versionhallinta sekä sisään- ja uloskuittauksenhallinta on konfiguroitava erikseen päälle tai pois päältä.

10.7.1 Versionhallinta (Versioning, Version Management)

- Käyttäjällä on käytettävissään kolmi-tasoinen versionhallinta:
  1. ei versiointia
  2. pääversiointi (esimerkiksi 1.0, 2.0, 3.0 jne.)
  3. pää- ja aliversiointi (esimerkiksi 0.1, 0.2 jne.).

- Käyttäjällä on saatavilla N (määrä konfiguroitavissa) viimeisintä pää- ja aliversiota dokumenteista.

- Käyttäjä voi määrittää kaikki versionhallintakombinaatiot kansiokohtaisesti.

- Käyttäjä voi määrittää näkevätkö muut käyttäjät aliversioita kuin laatija; jos eivät näe, aliversio ei näy myöskään hakutuloksissa. Jos aliversioita ei näytetä, on dokumentin haltijan julkaistava dokumentista pääversio, jotta muut näkevät sen

- Käyttäjä voi määrittää tuleeko aina uusi versio/aliversio, kun dokumentti tallennetaan samalla nimellä järjestelmään.

- Käyttäjä voi määrittää tuleeko uusi pääversio ”julkaista”.

- Käyttäjä voi kytkeä hyväksyntä- tai kommentointityönkulun esimerkiksi pääversion julkaisuun.

- Käyttäjä voi kommentoida versioita kunkin version omassa kommenttilaatikossa; muut käyttäjät näkevät kommentista, mitä dokumentissa on muutettu ilman, että heidän tarvitsee avata koko dokumenttia.

10.7.2 Sisään- ja uloskuittaus (Check-in and Check-out)

- Käyttäjä näkee kaikista dokumenttikirjastoista sisään- ja uloskuittaustiedon avulla, onko dokumentti saatavilla vai onko se jollakin muokattavana.

- Käyttäjä voi lukea toisen käyttäjän uloskuittaamaa dokumenttia, mutta ei muokata sitä samanaikaisesti.

- Käyttäjän tulee sisäänkuitata muokkaamansa dokumentti ennen kuin muut käyttäjät voivat muokata sitä.

- Sivuston ylläpitäjä voi ”vapauttaa” muokatun dokumentin muiden käyttäjien saataville sisäänkuittauksella, jos käyttäjä unohtaa sisäänkuitata dokumentin.

10.7.3 Työnkulku (Workflows)

Työnkulku on konfiguroitava toiminnallisuus. Dokumenteille on konfiguroitava ja määritettävä hyväksyjät ja kommentoijat.

- Käyttäjä voi kommentoida ja/tai hyväksyä dokumentteja julkaistavaksi työnkulun kautta.

- Käyttäjä voi määrittää työnkulun käynnistettäväksi manuaalisesti tai esimerkiksi julkaistaessa uuden (pää/ali)version.

10.7.4 Sähköpostien hallinta (E-mail Management)

MOSSin dokumenttikirjasto voidaan konfiguroida vastaanottamaan sähköposteja, niiden liitteitä tai molempia tietyistä sähköpostiosoitteista.

- Käyttäjä voi tallentaa sähköposteja suoraan Outlookista MOSSin dokumenttikirjastoon tai verkkokansioihin.

- Käyttäjä voi lähettää dokumentin tai linkin siihen suoraan dokumenttikirjastosta sähköpostilla.

10.8 Sivujen julkaisu portaaliin

- Käyttäjä voi tehdä sivuista eri versioita portaaliin.

- Käyttäjä voi asettaa sivuille julkaisuajan ja poistoajan.

10.9 Sisällön personointi

- Käyttäjä voi kuulua erilaisiin käyttäjäryhmiin (esimerkiksi markkinointiryhmä, tuotekehitysryhmä) ja käyttäjä näkee vain sellaisen sisällön (esimerkiksi tietylle ryhmälle kohdistetut uutiset), johon hänellä on käyttäjäryhmänsä mukaiset oikeudet.

- Käyttäjä voi tehdä web-osien avulla sivuille sisältöä, joka voidaan kohdistaa tietylle käyttäjäryhmälle.

10.10 Murupolku (The Bread Crumb Path)

Tämä on MOSS 2007 -vakio-ominaisuus.

- Käyttäjä näkee murupolun jokaisella sisältösivulla.

10.11 Asiakirjanhallinta (Records Management)

- Käyttäjä voi asiakirjanhallinnan avulla varmistaa, että tärkeistä dokumenteista otetaan kopiot tai dokumentit lähetetään arkistoitavaksi niin pitkäksi aikaa kuin on tarpeellista (säilytysaika).

- Käyttäjä voi määrittää dokumenteille säilytysajan, jonka umpeuduttua järjestelmä poistaa arkistoidun dokumentin automaattisesti.

- Käyttäjä voi lähettää kopion dokumentista toiseen paikkaan sivustolla: tämä paikka voidaan määrittää erikseen esimerkiksi arkistoksi.

- Käyttäjän kopioidessa dokumenttia kyseisen dokumentin metatiedot eivät kopioidu uuteen paikkaan dokumentin mukana.

10.12 Sivustohakemisto (Site Directory)

- Käyttäjät ja portaalin ylläpitäjät voivat katsella sivustohakemistosta, joka on eräänlainen sivugalleria, listauksia heitä kiinnostavista sivuista. Nämä voivat olla esimerkiksi listoja uusimmista sivustoista ja sivuista, käyttäjän omista sivuista tai liiketoimintaan liittyvistä sivustoista.

- Esimerkki sivustohakemiston rakenteesta:
  1. käyttäjän lisäämät sivustot
  2. viimeisimmät dokumentit ja yhteistyösivustot

3. viimeisimmät projektisivustot ja
4. liiketoimintaprosesseihin liittyvät sivustot.

11 KÄYTTÄJÄKERTOMUSTEN HYÖDYNTÄMINEN KÄYTÄNNÖSSÄ

Koska kehitetyt käyttäjäkertomukset kuvaavat MOSSin yleisimpiä ominaisuuksia, voi Digia hyödyntää kortteja jo MOSS-tietojärjestelmien myyntityössä. Kun konsultti menee nykyisen tai potentiaalisen uuden asiakkaan luo hieromaan kauppoja uudesta portaalista, voi konsultti tai myyntimies esitellä aluksi MOSSin yleisellä tasolla. Seuraavaksi myyjä voi havainnollistaa käyttäjäkertomuskorttien avulla, mitä hyötyjä MOSSin käytöllä voisi saavuttaa. Myyjä esittelee eri vaihtoehtoja, joita asiakkaan sovellukseen voitaisiin konfiguroida perustoiminnallisuuksien lisäksi ja keskustellaan, mitä asiakassovellukseen lisättäisiin.

Kortteja jatkojalostetaan yleiseltä tasolta kunkin asiakkaan tarpeiden mukaisesti. Kortit toimivat keskustelunavaajana kehitettävälle portaalille ja edistävät asiakkaan osallistumista määrittelytyöhön. Oikeaoppisesti käyttäjäkertomuksia käytettäessä asiakkaan tulisi kehittää kertomukset oma-aloitteisesti, mutta tässä tapauksessa kortit on laadittu valmisalustasovellukselle helpottamaan Digian myyntityötä. Asiakas astuu kuvioon mukaan suunniteltaessa kyseisen asiakkaan konfiguraatioita. Jokaiselle asiakkaalle tulee siis lisätä käyttäjäkertomuskortteihin asiakaskohtaiset konfiguraatiot.

12 POHDINTA

Tämän opinnäytetyön yksi tarkoitus oli selvittää, millainen on käyttäjäkertomustekniikka. Tutkimuksen tuloksena voidaan sanoa, että asiakasvaatimukset saadaan käyttäjäkertomusten avulla paremmin selville. Tästä on hyötyä koko projektille, sillä keskittymällä projektin alkuvaiheessa ohjelmiston määrittelyyn huolellisesti säästetään myöhemmissä vaiheissa selvää rahaa. Mitä myöhemmin projektista löydetään puutteita, sitä kalliimmaksi puutteiden ja mahdollisten virheiden korjaaminen tulee. Tarkka määrittely- ja suunnitteluvaihe on avain projektin onnistumiselle. Käyttäjäkertomusten avulla voidaan myös hajauttaa päätöksenteko pitkin projektia, jolloin projektin alussa ei käytetä liikaa aikaa ”täydellisen” vaatimusmäärittelyn tekemiseen vaan listaa voidaan täydentää pitkin projektia iteratiivisen ohjelmistokehityksen vuoksi.

Käyttäjäkertomuksilla asiakas saadaan sitoutettua projektiin jo sen alkuvaiheessa. Vuorovaikutus asiakkaan ja kehittäjien välillä jatkuu läpi projektin toisin kuin perinteisissä projekteissa, joissa yhteistyö käsittää määrittelyvaiheen ja seuraava kosketus asiakkaaseen on hyväksymistestauksen yhteydessä. Asiakastyytyväisyys kasvaa jatkuvan vuorovaikutuksen myötä, sillä ohjelmiston laatu paranee. Yksi laadun mittarihan on se, että tuote soveltuu alkuperäiseen käyttötarkoitukseensa ja asiakas on tuotokseen tyytyväinen. Asiakas on koko ajan selvillä projektin etenemisestä ja voi iteratiivisen projektimallin vuoksi puuttua havaitsemiinsa epäkohtiin heti.

Työssä on haettu myös vastausta kysymykseen kuinka käyttäjäkertomustekniikka soveltuu alustasovelluksen toimintojen kuvaamiseen testausta varten. Käyttäjäkertomukset säästävät aikaa testausvaiheessa. Koska kortit ovat itsessään testitapauksia, säästetään testaussuunnittelussa myöhemmin ja voidaan varmistua siitä, että kaikki järjestelmän kannalta olennaiset toiminnallisuudet tulevat testattua. Testauksen kattavuus paranee ja sitä myötä ohjelmiston laadustakin saadaan parempi selvyys. Koska testausta on valmisteltu jo vaatimusten keräämisen yhteydessä, ei testaukselle tarvitse varata kohtuuttomasti aikaa testitapausten suunnittelun merkeissä. Testaamiselle on usein

varattu liian vähän aikaa muutenkin, joten käyttäjäkertomuksilla saadaan ohjelmisto testattua tehokkaammin lyhyessä ajassa kuin aiemmin, kun suunnittelu voidaan jättää pois suurelta osin. Testauksen varhaisella suunnittelulla voidaan estää virheitä jo ennen koodaamisvaihetta, sillä määrittelyjä käydään läpi tavallista kriittisemmin ja havaittuihin epäkohtiin voidaan puuttua heti. Samalla selviää, ovatko järjestelmän vaatimukset todella testattavissa ja puuttuuko jotain olennaista.

Käyttäjäkertomukset toimivat oikein käytettynä hyvänä kommunikointivälineenä asiakkaan ja toimittajan välillä. Ne voivat myös korvata menestyksekkäästi muita vaatimustenkuvaamismenetelmiä, kuten käyttötapaukset. Käyttötapauksilla ei saada välttämättä kuvattua kaikkia asiakasvaatimuksia, ja osa vaatimuksista voi näkyä useissa käyttötapauksissa. Käyttäjäkertomuksilla saadaan joka tapauksessa selville tärkeimmät vaatimukset ja toteutustyö on helppo jakaa toteuttajille toiminnallisuuksittain. Käyttötapauksissa tässä voi tulla ongelmia.
Tämän perusteella käyttäjäkertomukset toimivat hyvänä lähtökohtana testaukselle.

Jos ohjelmistoa kehitetään iteratiivisesti, ovat käyttäjäkertomusten hyödyt vielä suuremmat: ohjelmistoa voidaan testata monesti jo ensimmäisen iteraation päättyessä, jolloin testaus voidaan hajauttaa pitkin projektia ja projektin loppupään kiireiseltä testaamisen typistämiseltä voidaan välttyä. Jos projekti viivästyy suunnitellusta aikataulusta, on testausvaihe ensimmäinen, josta karsitaan etenkin perinteisemmissä projektimalleissa, kuten vesiputousmallissa. Projektin viivästyminen on ketterissä menetelmissä epätodennäköisempää juuri edellä mainitun testauksen hajauttamisen vuoksi.

Kuinka käyttäjäkertomukset sitten lopulta soveltuvat valmisalustasovelluksen testaamiseen? Käyttäjäkertomusten hyödyntämistä testauksessa on käytännössä kokeiltu kesällä 2008 Digian uuden intranetin testaamisessa. Intranetin testitapaukset suunniteltiin käyttäjäkertomusten muotoon ja kyseessä oli nimenomaan MOSS-alusta. Testaustilanne ei täysin vastannut todellista käyttäjäkertomusmenetelmää, sillä intranetin määrittelyä ei ollut tehty käyttäjäkertomuksilla, joista testitapaukset olisi saatu suoraan. Testitapaukset kirjoitettiin

tavallisen määrittelyn pohjalta käyttäjäkertomusten muotoon. Työmäärän vähenemisestä testausvaiheessa on siis vaikea sanoa mitään, sillä testaussuunnitelma jouduttiin tekemään alusta.

Käyttäjäkertomukset eivät osoittautuneet testauksessa perinteisiä testitapauksia vaikeammiksi testata; ongelmia ilmeni lähinnä useamman ihmisen muokatessa testaussuunnitelmaa. Koska käyttäjäkertomukset eivät olleet kaikille suunnitelmaa muokanneille henkilöille tuttuja, eksyi joukkoon perinteisiäkin testitapauksia testilistaa täydennettäessä. Jos käyttäjäkertomukset otetaan käyttöön, on hyvä perehdyttää kaikki niiden kanssa tekemisissä olevat henkilöt uuteen menetelmään. Kokonaisuutena intranetin testausprojektista voi kuitenkin sanoa käyttäjäkertomusten soveltuvan testaamiseen. Etenkin, jos testitapaukset on suunniteltu jo määrittelyvaiheessa, ei ongelmia pitäisi olla testauksessakaan.

Käyttäjäkertomukset ovat varteenotettava menetelmä otettavaksi käyttöön etenkin MOSS-projekteissa, joissa Digialla on ollut erityisesti testaamisessa ongelmia. Syynä ongelmiin on ollut mm. se, että MOSS-projektien tarkka määrittely ei ole ollut kovin mielekästä, jolloin tarkka määrittely testaustakin varten on puuttunut. Tämä on johtanut niukkaan testaamiseen. Aikaakin testaamiselle on usein varattu liian vähän. Käyttäjäkertomuksilla tähän voidaan löytää ratkaisu, sillä määrittely muuttuu hieman kevyemmäksi jo kehitettyjen käyttäjäkertomuskorttien avulla, joihin tarvitsee vain lisätä asiakaskohtaiset konfiguraatiot. Testaamisen aikapulaan käyttäjäkertomukset ovat myös osittainen ratkaisu testaussuunnittelun vähentyessä.

Käyttäjäkertomusten todellista hyötyä Digialle on vaikea ennustaa. Jotta hyödyt saataisiin selville, tulisi seurata käyttäjäkertomusten käyttöä MOSS-projekteissa pidemmällä aikavälillä ja verrata projekteja aikaan ennen käyttäjäkertomuksia. Tämä on tulevaisuuden hyvä tutkimuskohde, johon tässä opinnäytetyössä ei rajallisen aikataulun vuoksi voida ottaa kantaa. Käyttäjäkertomukset vaikuttavat teoriassa kuitenkin käyttökelpoisilta, joten Digialla ei ole estettä ottaa kyseistä menetelmää käyttöön.

LÄHTEET

Cohn, M. 2007. User Stories Applied For Agile Software Development. 8 p. Harrisonburg: Pearson Education.

Haikala, I. & Märijärvi, J. 2002. Ohjelmistotuotanto. 8 p., uud. p. Pieksämäki: Talentum Media.

Hambling, B., Morgan, P., Samaroo, A., Thompson, G. & Williams, P. 2007. Software Testing. 2 p., korj. p. Chippenham: The British Computer Society.

Honkaranta, A. & Ylönen, A. 2008. Vanhempi konsultti & projektipäällikkö, Digia Oyj, Teollisuus- ja kauppa -divisioona. Keskustelu 21.4.2008.

Koomen, T. & Pol, M. 1999. Test Process Improvement. Guildford: Pearson Education.

Loveland, S., Miller, G., Prewitt, R. & Shannon, M. 2005. Software Testing Techniques: Finding the Defects that Matter. Hingham: Charles River Media.

McConnell, S. 1998. Ohjelmistoprojektit - selviytymisopas. Jyväskylä: Suomen Atk-kustannus.

Nielsen, J. 1993. Usability Engineering. San Diego: Academic Press.

Williams, V.L. 2007. Microsoft SharePoint 2007 For Dummies. Indianapolis: Wiley Publishing.

Ylönen, A. 2008. Projektipäällikkö, Digia Oyj, Teollisuus- ja kauppa -divisioona. Keskustelu 9.9.2008.