Simo Tauriainen

OHJELMISTOTESTAUKSEN KEHITTÄMINEN

Opinnäyte
Kajaanin ammattikorkeakoulu
Ammattikorkeakoulun tekniikan jatkotutkinto
Osaamisen johtaminen –jatkokoulutusohjelma
Kevät 2005

Kajaanin ammattikorkeakoulu

OPINNÄYTETYÖ
TIIVISTELMÄ

Ala	Koulutusohjelma
Tekniikan ja liikenteen ala	Osaamisen johtaminen -jatkokoulutusohjelma
Tekijä(t) Simo Perttu Tapani Tauriainen	
Työn nimi Ohjelmistotestauksen kehittäminen	
Vaihtoehtoiset ammattiopinnot	Ohjaaja(t) Arto Karjalainen, Asko Kinnunen
Aika 30.5.2005	Sivumäärä 84

Tiivistelmä

Tämän opinnäytetyön tavoitteena oli kehittää ohjelmistotestausta. Opinnäytetyössä analysoidaan työntilaajan, Ponsse Oyj, käytössä olevaa ohjelmiston testauskäytäntöä. Nykykäytännöstä etsittiin heikkoudet ja vahvuudet ja otettiin huomioon yrityksen toimintaympäristön sanelemat vaatimukset ohjelmistotestaukselle.

Opinnäytetyön teoriaosuudessa käsitellään ohjelmiston elinkaariajattelua ja ohjelmistotestauksen käytäntöjä ja käsitteitä. Yrityksessä käytössä olevasta testauskäytännöstä paikallistettiin heikkoudet ja vahvuudet teoriataustaan ja yrityksessä vallitseviin käytännön vaatimuksiin nojautuen. Edellä mainittujen tietojen avulla ohjelmistotestaukseen luotiin työn tilanneeseen yritykseen soveltuva ohjelmistotestauksen toimintamalli. Toimintamalli koostuu kahdesta eri vaiheesta, joita ovat järjestelmätestausvaihe ja hyväksymistestausvaihe. Uuden toimintamallin käyttöönotto vaati jakamaan ohjelmistomuutoksien suoritukset eri kokoluokkiin ja kehittämään näille kokoluokille omat kehitys- ja testauskäytännöt. Ohjelmistotestauksen suoritus on sitä tehokkaampaa mitä rajatummalle kokonaisuudelle testauksen pystyy suorittamaan. Opinnäytetyössä kehitettiin myös toimintamalli funktio testaukseen. Ohjelmistotestauksen kannalta funktio on ohjelmiston pienimpiä toiminnallisia kokonaisuuksia, mitä ohjelmistosta voi irrallaan testata.

Opinnäytetyön tuloksena syntyi Ponsse Oyj:lle uusi ohjelmistotestauskäytäntö, sekä sen suoritukseen tarvittavat uudet ohjelmistokehitysmallit.

Luottamuksellisuus	Julkinen
Hakusanat	Ohjelmistotestaus
Säilytyspaikka	Kajaanin ammattikorkeakoulu / kirjasto

Kajaani Polytechnic

ABSTRACT

Faculty Technology, Communication and Transport	Degree programme Degree Programme in Competence Management Second-Cycle Polytechnic Degree
Author(s) Simo Perttu Tapani Tauriainen	
Title Developing software testing	
Alternative professional studies	Instructor(s) Arto Karjalainen, Asko Kinnunen
Date 30.5.2005	Total number of pages 84

Abstract

The purpose of this thesis was to develop software testing. The thesis analyses the software testing practices of Ponsse Oyj, who commissioned the study. The strengths and weaknesses of current practices were assessed, taking into account the requirements set for software testing by the operating environment of the company.

The theoretical part of the thesis discusses the notion of software life cycle and explores the concepts and methods of software testing. The strengths and weaknesses of the company's current testing practices were localized using the theoretical background and taking into account the company's prevailing practical requirements. On the basis of the above information, a software testing model suitable for the company was created. The model consists of two phases, system testing and acceptance testing. The new model required classification of software changes according to size and establishment of separate development and testing practices for the different size categories. The more confined the entity to be tested, the more efficient the software testing. The thesis also introduces a model that was developed for function testing. In software testing, a function is one of the smallest functional units that can be tested separately.

The thesis study resulted in a new software testing procedure for Ponsse Oyj and new software development models required for its implementation.

	Public
Keywords	Software testing
Deposited at	Kajaani polytechnic library

SISÄLLYS

TERMIT

Ekvivalenssiluokka

Testiaineiston valintamenetelmä, jossa testin suorituskertoja vähennetään jakamalla testiaineisto luokkiin. Yksi testattava arvo vastaa kokoluokan sisältöä ohjelmistotestissä.

Happotestaus

Regressiotestauksen muoto, jossa luodaan erillinen testisarja, jolla voidaan varmistaa ohjelman perustoiminnallisuuden oikeellisuus.

Harmaalaatikko-testaus

Testausmenetelmä, jossa käytetään ohjelmiston rakenteen tuntemusta ja ohjelmistosta tehtyjä määrityksiä apuna virheitä etsittäessä.

Hyväksymistestaus

Asiakkaan suorittama (avulla suoritettava) testaus ohjelmiston todellisessa käyttöympäristössä.

Integrointitestaus

Komponenttien rajapintojen ja keskinäisen kommunikoinnin testaustaso.

Järjestelmätestaus (systeemitestaus)

Järjestelmän testaus koko laajuudessaan, käyttötarkoitusta vastaavassa ympäristössä.

Kenttätesti

Testausinsinööri suorittama/valvoma testi ohjelmistolle, sen todellisissa käyttöolosuhteissa.

Konttoritesti

Ohjelmistokehittäjän suorittama testaus joka tapahtuu konttorilla. Pitää sisällään yksikkö- ja integrointitestausta.

Lasilaatikko-testaus

Testausmenetelmä, jossa käytetään ohjelmiston rakenteen tuntemusta apuna virheitä etsittäessä.

Mustalaatikko-testaus

Testausmenetelmä, ohjelmistosta tehtyjä määrityksiä käytetään apuna virheitä etsittäessä. Testataan syötteiden ja vasteiden avulla testataan toiminnan oikeellisuus.

Muutos- ja kokeilunseurantailmoitus

Ponsse Oyj:n toiminnanohjausjärjestelmän dokumentti, jossa kerrotaan tuotteeseen tulevasta muutoksesta ja siihen liittyvästä testistä.

Ohjelmistojen muutosilmoitus

Ponsse Oyj:n toiminnanohjausjärjestelmän dokumentti, jossa kerrotaan ohjelmistomuutoksista. Ilmoitus on myös merkki organisaatiolle, että ohjelmisto on testattu ja sitä voidaan alkaa jakaa asiakkaiden laitteistoihin.

Ohjelmisto-relaease

Release-jakson aikana kehitetty ohjelmisto, joka pohjautuu edellisessä release-jaksossa kehitettyyn ohjelmistoon .

Raja-arvoanalyysi

Testidatan valintamenetelmä, jossa dataan valitaan luokkien rajoilla olevia arvoja.

Regressio-testaus

Varmistaa sovelluksen oikeellisuuden uudelleen sen jälkeen, kun virheitä on korjattu tai toiminnallisuutta on lisätty tai muutettu.

Release-jakso

Jakson aikana kehitettään ja testataan ohjelmiston uusia ominaisuuksia ja korjataan tunnettuja ohjelmistovirheitä. Release-jakson lopussa ohjelmisto julkaistaan asiakkaiden käytettäviksi.

Testiajuri

Jäljittelee muiden yksikköjen antamia syötteitä, ei sisällä ”älyä”.

Tynkä

Ohjelmistotestissä olevan ohjelmiston toiminnallisuutta jäljittelevä osa.

Tynkäpeti

Testattavan moduulin rajapintoja jäljittelevä osa.

V-malli

Ohjelmistokehityksen malli, joka jakaa ohjelmistoprojektin suunnittelu-, toteutus- ja testaustasoihin ja esittelee tasojen väliset suhteet.

Yksikkötestaus (moduulitestaus)

Ohjelmistokehittäjän suorittama testaus joka tapahtuu yksikkö- tai moduulitasolla.

XP

Extreme Programming

0-sarja testi

Ponsse Oyj:n tuotantotiloissa tapahtuva testaus. Testauksen on tarkoitus varmistaa ohjelmiston toimivuus metsäkoneen valmistuksen (tuotannon) näkökannalta.

1 JOHDANTO

Tämän opinnäytetyön tavoitteena oli kehittää Ponsse Oyj:n ohjelmistotestauskäytäntöä. Opinnäytetyössä tutustuttiin yrityksen ohjelmistotestauksen nykykäytäntöön ja analysoitiin siinä oleviin heikkouksia ja vahvuuksia. Näiden tietojen ja ohjelmistotestauksen teorian avulla kehitettiin käytännönläheinen toimintamalli yrityksen ohjelmistotestaukseen.

Ohjelmistolle suoritettava määrätietoinen ohjelmistotestaus ennen käyttöönottoa on eräs ohjelmistotuotannon tärkeimpiä osa-alueita. Puutteellisesti testattu ohjelma saattaa aiheuttaa yllättäviä kustannuksia ja aikatauluongelmia. Tiedostetusta ongelmasta huolimatta ohjelmistotestausta yleensä aliarvostetaan.

Ohjelmistotestauksen tavoitteena pidetään ohjelmistovirheiden löytämistä, mutta yksinään tämä ei takaa hyvää testaustulosta ja ohjelmistoprojektin onnistumista. Testauksessa virheellisyyden osoittaminen ei ole itsetarkoitus, vaan perimmäisenä pyrkimyksenä on virheiden löytämisen jälkeen saada ohjelmakoodi korjattua virheettömäksi ja oikealla tavalla toimivaksi.

Ohjelmistotestauksessa on useita eri tasoja (vaiheita), jotka ovat toisistaan riippuvia. Ensimmäisessä vaiheessa testataan yksittäistä ohjelmamoduulia, seuraavissa vaiheessa testissä mukana on jo useita ohjelmistomoduuleja. Joissakin vaiheissa keskitytään pääasiassa poistamaan suoranaisia ohjelmistovirheitä esimerkiksi hakemaamaan virhetoimintoja, jotka aiheuttavat ohjelmisto kaatumisia. Ohjelmistotestauksen loppuvaiheessa ei enää pitäisi löytyä edellisen kaltaisia virheitä, vaan jäljellä voi olla enää

väärin ymmärrettyjen määrittelyjen mukaan tehtyjä virhetoimintoja. Kaikkien näiden vaiheiden järjestelmällisellä suorituksella pyritään varmistamaan ohjelmiston laatu.

Tavoitteena opinnäytetyön toiminnallisessa osuudessa on analysoida ja kehittää työn tilanneen organisaation ohjelmiston testauskäytäntöä. Opinnäytetyön tarkoituksena on myös raottaa hieman ohjelmistotestauksen verhoa ja pyrkiä ymmärtämään sen tärkeys. Unohtamatta kuitenkaan käytännönläheisyyttä ja niitä rajoituksia, joita ympäristö ja käytössä olevat resurssit asettavat koko ohjelmistokehitykselle.

Työn yhtenä tärkeänä vaiheena oli käytännönläheisen lähdemateriaalin valinta. Lähdemateriaalin tuli sisältää hyväksi havaittuja tekniikoita ja vinkkejä käytännön työhön ohjelmistotestauksen osalta. Työn tärkeimmät lähdemateriaalit muodostavat kirjat ”Software engineering: A Practitioner’s Approach” kirjoittaja Roger S. Pressman, ”Testing Embedded Software” kirjoittajina Bart Broekman ja Edwin Notenboom sekä Ilkka Haikalan ja Jukka Märijärven kirjoittama Ohjelmistotuotanto. Varsinkin kahdesta ensin mainitusta kirjasta on ollut paljon apua haettaessa apua ongelmiin, joita tulee vastaa jokapäiväisissä työtehtävissä.

Mielenkiinto Pressmanin teokseen heräsi, koska hänet mainittiin nimeltä useissa eri luentomateriaaleissa ja yleensäkin useissa ohjelmistotuotantoon liittyvissä materiaaleissa. Tutustuttuani hänen käytännönläheiseen tapaan lähestyä ongelmia, valitsin hänen teoksensa lähdemateriaaliksi. Hänellä on pitkä kokemus teollisuuden parissa tapahtuvasta ohjelmistokehityksestä ja se näkyykin teoksessa hyvin. Teos on oivallinen apuväline ohjelmistotuotannon maailmaan.

Koska osa opinnäytetyön tilanneen yrityksen projekteista suuntautuu sulautettuun ympäristöön, Broekman ja Notenboomin teos oli mielekäs valinta lähteeksi. Teoksen aihepiirin rajaus käy hyvin yksiin opinnäytetyön aiheen kanssa. Tekijöiden vankka kokemus ohjelmistotestauksesta ja määrätyllä tavalla ”insinöörimäinen” lähestymistapa aihealueelle, olivat syitä miksi päädyin teokseen.

Nämä kaksi teosta sopivat hyvin ammattikorkeakoulun tekniikan jatkotutkinnon linjalle. Tarkoituksena ei ole keksiä mitään mullistavia uusia teorioita, vaan tehdä uusia

sovelluksia sen pohjatyön avulla, minkä tiedemaailma on pitkäjänteisellä työllään saanut aikaan.

1.1 Tutkimuksen tarkoitus ja ongelman asettelu

Opinnäytetyössä kehitettään Ponsse Oyj:ssä käytössä olevan ohjelmiston testauskäytäntöä. Nykyisessä ohjelmiston testauskäytännössä on havaittavissa tiettyjä puutteita ja ongelmia, joita ovat muun muassa aikataulujen venyminen ja testaustyön epätasainen jakautuminen. Opinnäytetyön tarkoituksena on löytää ongelmia aiheuttavat tekijät ja löytää nykykäytännöstä asiat, jotka tulee tiedostaa kehitettäessä ohjelmistotestausta. Opinnäytetyön tavoitteena on kehittää työn tilanneen yrityksen toimintaympäristöön soveltuva ohjelmiston testauskäytäntö (toimintamalli), ottaen huomioon toimintaympäristön ja käytännön asettamat vaatimukset ja rajoitukset.

Opinnäytetyö alkaa ohjelmistotestauksen teoriaan tutustumisella. Teoriasta saadun tiedon avulla lähestytään yrityksessä vallitsevaa käytäntöä ja pyritään löytämään nykyisestä toimintatavasta mahdolliset puutteet ja kehittämiskohteet ohjelmistotestauksen osalta. Työn tilanneessa yrityksessä ei ole tällä hetkellä käynnissä muita ohjelmistotestaukseen liittyvää kehitystyötä, joten vertailevaa tutkimusta ei voida suorittaa tämän opinnäytetyön aikana.

Haasteellisin osa työssä on ratkoa ongelmat käytännönläheisellä tavalla ottaen tukea teoriasta ja soveltaa sitä nykyiseen käytäntöön siten, että opinnäytetyön tulokset voidaan ottaa helposti käyttöön jokapäiväisessä työssä.

Opinnäytetyön nimi on ”Ohjelmistotestauksen kehittäminen”. Työn tavoite voidaan purkaa yrityksen tarpeiden mukaisesti seuraavaan kysymykseen: Mitkä ovat Ponssen Oyj:n tämän hetkiset heikkoudet ja vahvuudet ohjelmistotestauksessa ja miten ne voidaan ottaa huomioon kehitettäessä uutta ohjelmiston testauskäytäntöä?

2 OHJELMISTOTESTAUS

Ohjelmistotestauksen tarkoituksena on parantaa ohjelmiston laatua poistamalla siitä ohjelmistovirheitä. Ohjelmistovirhe on poikkeama määrittelystä sekä havaittu eroavaisuus ohjelmiston toiminnassa ja sille tuotetuissa määrityksissä. Sovelluksessa olevan virheellisen kohdan suorittaminen aiheuttaa vian. Vika on myös virheellinen toiminto, prosessi tai tiedon määritys tietokoneohjelmassa. Vika ei kuitenkaan aina aiheuta ongelmia sovellukseen, koska se voi korjautua itsestään toisen toiminnon tai virheen seurauksena. Joissakin tapauksissa saattaa vian seurauksena syntyä häiriö, joka näkyy järjestelmän ulkoisessa toiminnassa. Vian voi aiheuttaa väärä tai puuttuva ohjelmiston osa. Vika saattaa aiheuttaa yhden tai useampia toimintahäiriöitä. (Haikala & Märijärvi, 2004)

Virhe on ihmisen tekemä erehdys, tekeminen tai tekemättä jättäminen, joka aiheuttaa vian. Suurin osa virheistä johtuu tuotteen määrittelyn virheellisyydestä. Ohjelmiston kehityksessä virhe voi aiheuttaa vikoja vaatimuksissa, määrittelyissä, ohjelmissa tai testeissä. Erehdys on ihmisen teko, joka aiheuttaa väärän lopputuloksen. (Haikala & Märijärvi, 2004)

Virheen määritelmä voidaan kuvata tuotteen määrittelyn kautta. Alla on lueteltu viisi eri vaihtoehtoa, joista yhdenkin toteutuessa virhe esiintyy.

1. Ohjelmisto ei tee jotakin, jota sen määrittelyn mukaisesti kuuluisi tehdä.
2. Ohjelmisto toimii tavalla, jonka määrittely kieltää.

3. Ohjelmisto toimii tavalla, jota määrittely ei mainitse.
4. Ohjelmisto ei tee jotakin, jota määrittely ei mainitse, vaikka se pitäisikin mainita.
5. Ohjelmisto on hankala ymmärtää, vaikeakäyttöinen, hidas tai käyttäjän mielestä selkeästi toimii väärin. (Patton, 2001)

Yksi virheiden syntymiseen johtava syy on ohjelmoijien huono ammattitaito. Se ei ole ihme, sillä joissakin yrityksissä on ohjelmoijina kokemattomia ensimmäisen vuoden opiskelijoita. Eräs syy virheisiin on se, että ohjelmistoilla ratkotaan erittäin vaikeita ja monimutkaisia ongelmia. Kokeneinkaan ohjelmistoammattilainen ei pysty niitä kaikkia hallitsemaan. Kolmas syy ongelmiin on ohjelmointikielten ja työkalujen hankalakäyttöisyys. Muita ongelmaksi koettuja asioita ovat muun muassa ohjelmistotyön lopputuloksen näkymättömyys, ohjelmiston muunnettavuus, ainutkertaisuus ja skaalautumattomuus. Ohjelmiston työmäärään liittyvä näkymättömyys on ollut aikamoinen yllätys monissa projekteissa. Kun on esimerkiksi arvioitu, että ohjelmistotuote on jo 90-prosenttisesti valmis, työaikaa kuluukin vielä saman verran kuin siihen mennessä on käytetty. (Haikala & Märijärvi, 2004)

Ohjelmistojen muunnettavuus on edellytys pitkäikäisyydelle, sillä vaatimukset tarkentuvat ja muuttuvat jo kehitysaikana ja ylläpidon aikana. Ohjelmiston ainutkertaisuuden vuoksi samantapaista ohjelmaa ei ehkä ole tehty aikaisemmin, joten ei tunneta hyvää ja toimivaa ratkaisumallia, vaan se muotoutuu vuosien saatossa. Skaalautumattomuus liittyy ohjelmistoprojektin kokoon. Joissakin pienissä projekteissa hyväksi havaitut ratkaisut eivät toimi enää suurissa sovelluksissa. (Haikala & Märijärvi, 2004)

2.1 Ohjelmistotestauksen tarpeellisuus

Ohjelmistotestauksen tarkoituksena on toimia virheiden eliminointikeinona sekä parantaa ohjelmiston laatua ja toiminnallisuutta. Täysin kattavaa ohjelmistotestausta on kuitenkin käytännössä mahdoton suorittaa. Kuitenkin järjestelmällisellä, rationaalisella testauksella voidaan välttää suurimmat ongelmat. Mitä aiemmin virhe löydetään, sen edullisempaa ja riskittömämpää sen korjaaminen on. Katselmointi on myös osa testaamista ja jo määrittelyvaiheen katselmoinnilla saadaan kustannussäästöä

aikaan. Määrittelyvirheet ovat ohjelmistotuotannon testauksen kannalta keskeisimmät haettavat virheet. (Haikala & Märijärvi, 2004)

Määrittelyvirheen pääsy lopputuotteeseen ja sen korjaaminen on tuhansia kertoja kalliimpaa verrattuna siihen, mitä virhe olisi tullut maksamaan, jos virhe olisi löydetty jo katselmoitaessa määrityksiä. Suunnitteluvaiheessa löydetyn määrittelyvirheen korjaamisen hinta on noin nelinkertainen verrattuna siihen, mitä se olisi ollut löydettäessä määrittelyvaiheessa. Suunnitteluvaiheessa tapahtuneiden virheiden korjauskustannuksen ovat noin puolet siitä, mitä määrittelyvaiheessa tapahtuvien virheiden korjaus maksaa. (Haikala & Märijärvi, 2004)

Ohjelmistojen virhetoimintojen löytyminen ja niiden korjaaminen on tärkeä osa ohjelmistotuotantoprosessia. Alan kirjallisuudessa on todettu, että jopa puolet ohjelmiston kehityskustannuksista voi olla testauksesta johtuvia kuluja. Testauksesta aiheutuvien kulujen määrää vaihtelee ohjelmistoittain. Testauskulujen osuus riippuu ympäristön haastavuudesta, ympäristön vaatimuksista, mihin ohjelmisto on kehitetty. Ääri esimerkkinä voi olla ydinvoimalaitokseen tuotettu ohjelmisto, tällöin testauksesta aiheutuvat kulut voivat olla jopa yli 90 % koko projektin budjetista. Tulevaisuudessa testauksen osuus tule kasvamaan entisestään myös vähemmän kriittisissä sovelluksissa, varsinkin jos testausta ei kehitetä muun ohjelmistokehityksen ohessa. Ohjelmistot tulevat olemaan ja ovat jo tänäkin päivänä olennainen osa nykypäivän laitteiden toiminnallisuutta. Tästä johtuen ohjelmistotestauksen osuus tulee siirtymään myös enemmän laitteiden turvallisuuden varmistamiseen. (Broekman, Notenboom 2003)

Ohjelmistotestauksen järjestelmällisellä suorittamisella pyritään viemään tuote entistä nopeammin markkinoille. Samalla varmistetaan tuotteen luotettavuus, jotta se vastaisi asiakkaiden odotuksia. Ohjelmiston luotettavuus ja asiakkaan kannalta oikein tehdyt toiminnot parantavat asiakastyytyväisyyttä. Mitä aikaisemmassa vaiheessa mahdolliset ohjelmistovirheet löydetään, sitä alhaisemmiksi ohjelmiston kehityskulut yleensä jäävät. (Stenberg, 2004)

2.2 Ohjelmistotestaus osana ohjelmistoprojektia

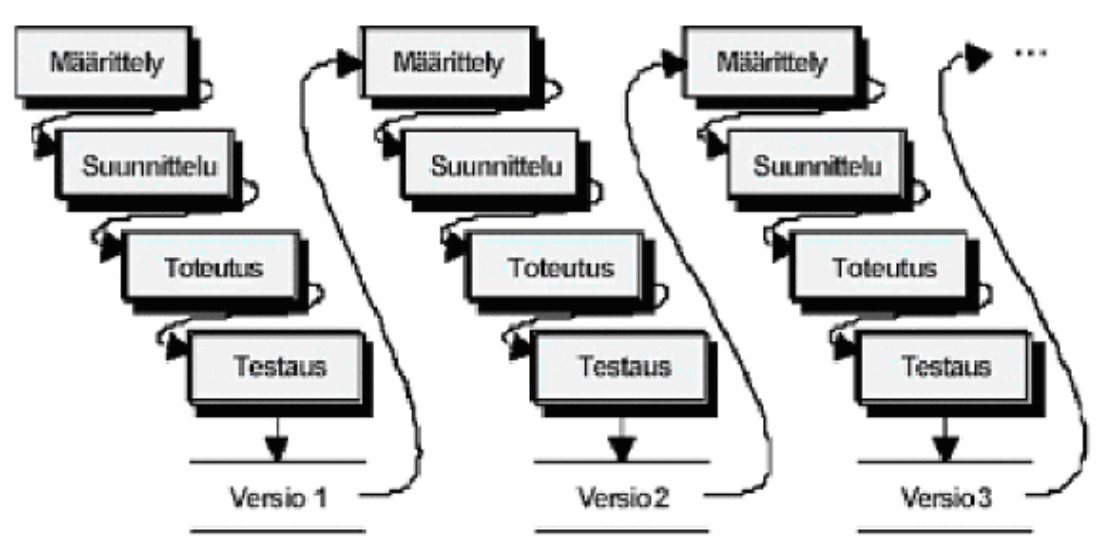
Ohjelmistotestaus on yksi olennainen osa ohjelmistoprojektia, ei irrallinen toimenpide ohjelmiston elinkaaren aikana. Ohjelmistotestausta kehitettäessä ja analysoitaessa, olisi sen osuus ohjelmistoprojektista kyettävä ymmärtämään. Testauksen päätavoitteena on löytää ohjelmistosta virheitä mahdollisimman aikaisin ja varmistaa niiden korjaaminen.

Ohjelmiston kehitysprosessia kuvataan elinkaarella, joka esittää ohjelmiston vaiheet kehittämisen aloittamisesta aina käytöstä poistamiseen asti. Elinkaarta kuvataan vaihejako- eli prosessimalleilla. Näistä eri vaihejakomalleista on useita muunnelmia, mutta yleisinä osina kaikista on erotettavissa määrittely-, suunnittelu-, toteutus- ja testausvaiheet. (Chan & Tanik, 1991)

Kirjallisuudesta ja Internetistä löytyy vaiheenjakomalleja lukematon määrä. Malleihin tutustuttaessa käy ilmi, että suurimassa osasta näitä malleja löytyy paljon yhteneväisyyksiä. Tässä työssä esiteltävät vesiputous-, spiraali-, big-bang- ja XP- vaihejakomallit ovat selkeästi tarvittaessa hyödynnettävissä opinnäytetyön tilanneessa organisaatiossa.

Vaihejakomallit voidaan jakaa karkeasti kahteen eri tyyppiin: lineaarisiin ja iteroiviin malleihin. Lineaarinen malli etenee askelmaisesti vaihe vaiheelta eteenpäin, eikä edelliseen vaiheeseen palata saman projektin puitteissa. Iteroiva malli, nimensä mukaisesti, palaa edelliseen vaiheeseen tai tarvittaessa hyppää useamamman vaiheen yli. (Pressman, 1997)

Näitä kahta mallia yhdistelemällä on luotu lukematon määrä erilaisia vaiheenjakomalli variaatioita. Hyvä esimerkki aiheesta on kuvassa 1. näkyvä EVO-malli.



KUVA 1. EVO (Haikala & Märijärvi, 2004)

EVO-mallissa Jokainen ohjelmistoversio luodaan yksitellen lineaarisen mallin avulla. Lopullinen ohjelmisto on kokoelma iteroituja versiokierroksia. (Haikala & Märijärvi, 2004)

Vesiputousmalli (lineaarimalli) sopii hyvin pieniin ja ennalta hyvin tunnettuihin projekteihin eli sellaisiin, joiden määrittelystä saadaan kerralla kattava. Onko määrittelystä mahdollista saada yhdellä kerralla täysin kattavaa, onkin jo sitten toinen kysymys. (Pressman, 1997)

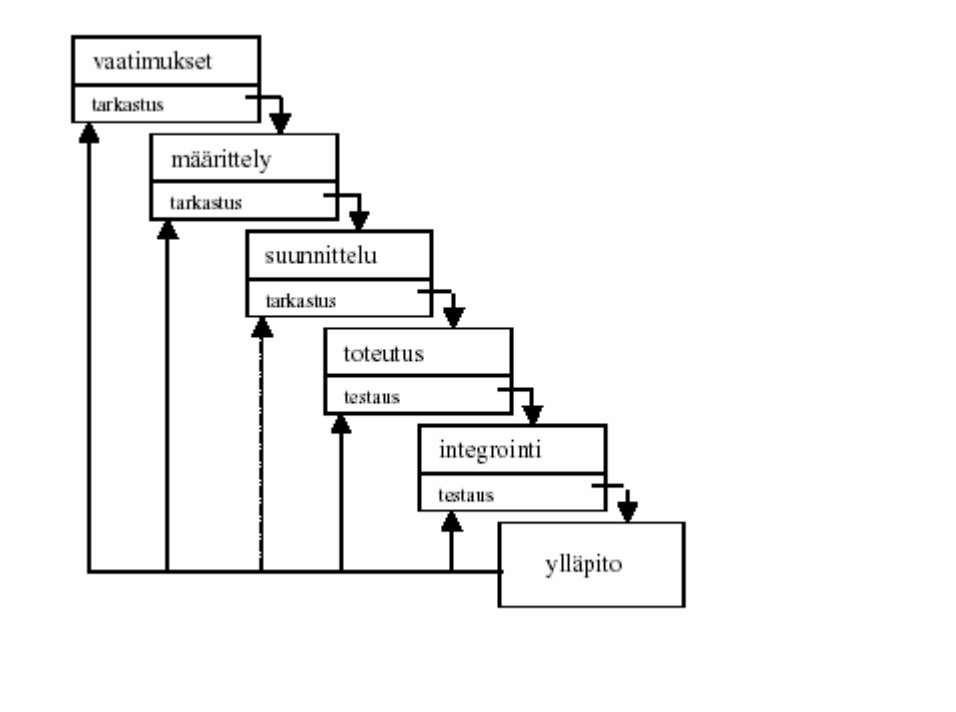
Big-Bang on mukana antamassa ajatuksia ja ehkä tuomassa eräänlaista kärjistettyä esimerkkiä, kuinka asioita voidaan hoitaa. Koska kyseinen vaihejakomalli on kevyt dokumentoida, täytyy malliin tutustua. Kevyt ja toimiva dokumentointi on hyvä tavoite ohjelmistotuotannossa. (Patton, 2001)

XP-vaihejakomalli on mukana teoriaosuudessa sen käytännönläheisen lähestymistavan takia. Mallissa on määritetty kevyt ja toimiva dokumentaatio ja suuri määrä testejä. XP - vaihejakomallissa on myös valmius määrittely muutoksille. (Beck, 1999)

2.2.1 Vesiputousmalli

Vesiputousmalli on ohjelmistoprojektin vaihemalleista yksinkertaisin ja samalla kuitenkin toimiva esimerkki lineaarimallista. Tässä vaihemallissa edetään portaittaisesti vaiheesta toiseen aina projektin synnystä sen elinkaaren loppuun. Jokaisen askeleen lopussa tarkastellaan, joko ollaan valmiita seuraavaan vaiheeseen vai tulisiko samaa vaihetta jatkaa ennen siirtymistä seuraavaan. Ennen seuraavaan vaiheeseen siirtymistä tapahtuva katselmointi on tärkeää, sillä useissa projekteissa mallia sovelletaan pelkästään lineaarisena, jolloin mahdollisuutta palata edellisiin vaiheisiin ei ole. (Broekman & Notenboom, 2003)

Vesiputousmallissa testaukselle varataan vain yksi vaihe. Koska kaikki nämä erilliset suoritusvaiheet ovat irrallisia, ei päällekkäisyyksiä niiden tehtävissä ole. Kuvassa 2. näkyvä vesiputousmalli toimii hyvin projekteissa, joissa vaatimukset ymmärretään hyvin ja niitä myös noudatetaan. Tämä malli on toimiva, kun suunnitellaan pientä ohjelmistoprojektia. (Pressman, 1997)

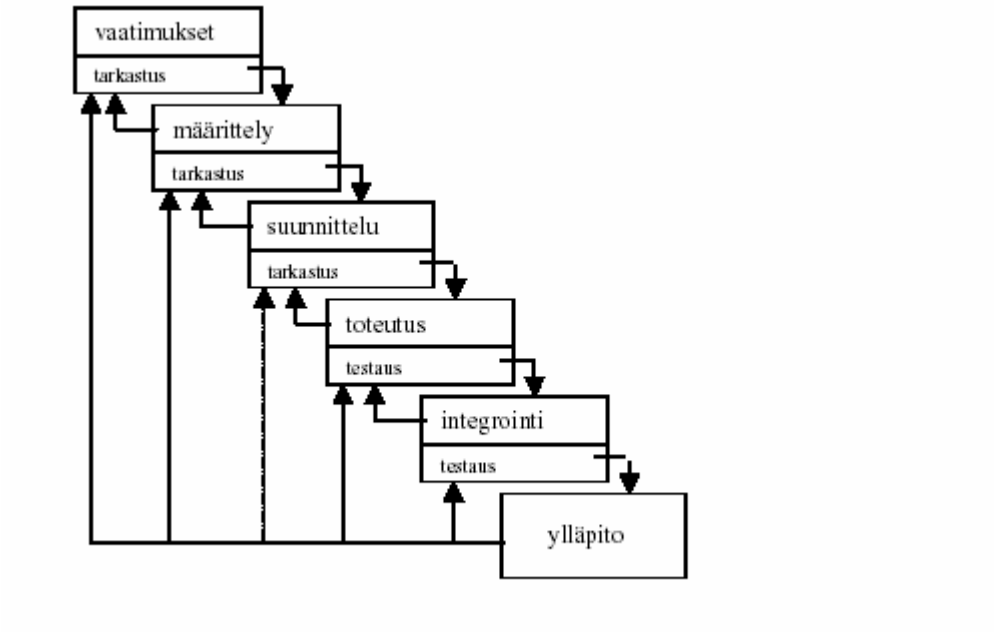


KUVA 2. Vesiputousmalli (Patton, 2001)

Mallia noudatettaessa on lopullinen suunnittelu jo tehty saavuttaessa testausvaiheeseen, jolloin testiryhmä näkee selkeästi, mitkä ovat virheitä ja mitkä haluttuja ominaisuuksia. Tosielämässä asiakas harvoin tietää kaikkia vaatimuksiaan projektin alussa, joten yleensä suunnitelmat tarkentuvat tai muuttuvat yleensä myöhemmin. Huonona puolena tässä mallissa on testauksen suorittaminen vasta projektin lopussa, jolloin virheiden korjauskustannukset ovat huomattavasti suuremmat kuin määrittely- tai suunnitteluvaiheessa. (Patton, 2001)

Tämä vesiputousmalli on kuitenkin paljon käytetty ja se ohjaa kehitysprosessia määrittäen kaikille tehtäville selkeän järjestyksen. Malli on siis projektin vetäjien osalta yleensä helpoiten hallittavissa. Vesiputousmallista on johdettu monia muunnoksia ja kehitetty kokonaan uusia malleja. (Pressman, 1997)

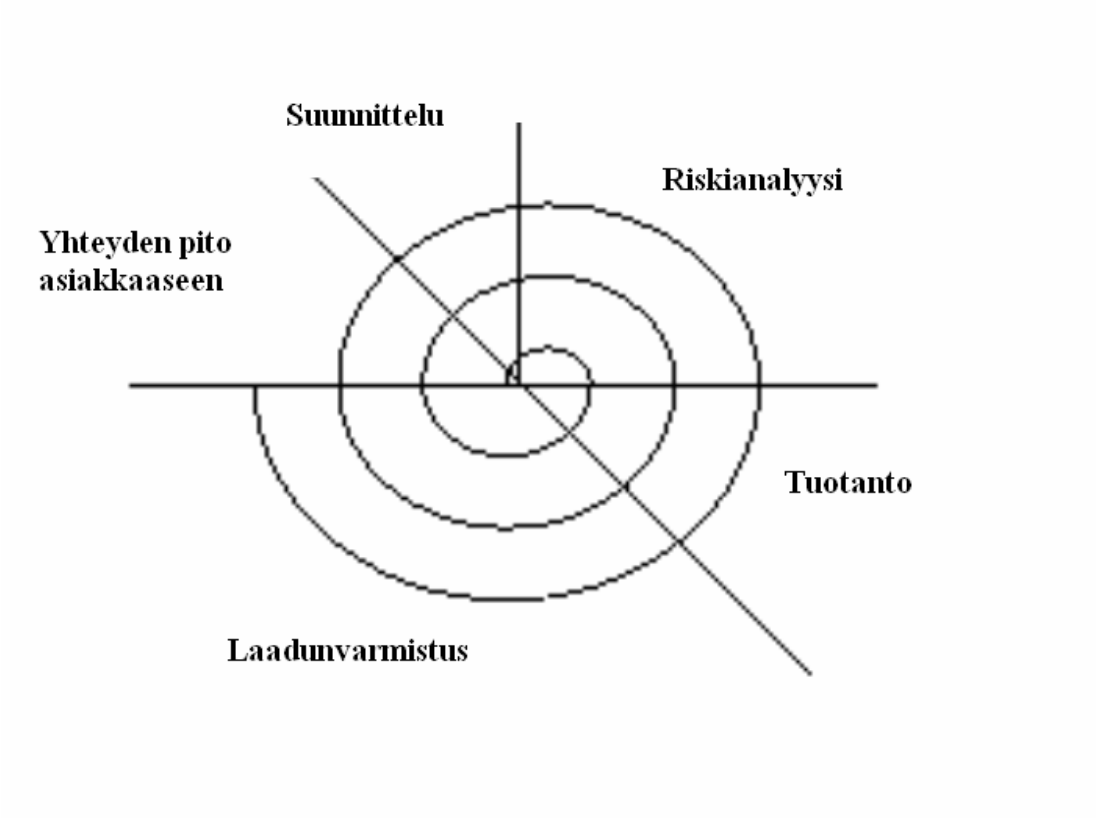
Mallista on myös luotu muunnos, jossa ehdottomasta lineaarisuudesta voidaan hieman joustaa. Esimerkiksi saavuttaessa toteutusvaiheeseen (kuva 3.) huomataan tilanne, että suunnittelussa on ollut puutteita. Tällöin voidaan palata suunnitteluvaiheeseen. Kun suunnitteluvaihe on uudestaan hyväksyttävästi katselmoitu, palataan toteutusvaiheeseen. (Spillner, 2002)



KUVA 3. Iteroivan vesiputousmalli (Spillner, 2002)

2.2.2 Spiraalimalli

Kuvassa 4. näkyvä spiraalimalli on yleinen iteroiva vaihejakomalli. Malli on toimiva nopeatempoisessa kehitystyössä. Spiraalimallin etuna on mahdollisuus suunnitella ensin tärkeimmät ja kriittisimmät osiot, toteuttaa ne ja saada niistä palaute asiakkaalta. Tämän jälkeen voidaan aloittaa uusi kierros tarkentaen prosessia. Asiakas voi tällöin arvioida vaihetuotetta ja antaa palautetta, joka voidaan hyödyntää nopeasti seuraavalla kierroksella. (Pressman, 1997)



KUVA 4. Spiraalimalli (Pressman, 1997)

Spiraalimallin suoritusjärjestys:

1. Yhteydenpito asiakkaaseen, vaatimukset
2. Suunnittelu
3. Riskianalyysi
4. Tuotanto
5. Laadunvarmistus, testaus

Testaajat ovat myös mukana kaikissa projektin vaiheissa, jolloin he näkevät vaatimukset ja halutut tulokset. Spiraalimallin yhtenä hyvänä puolena on se, että virheet havaitaan jo aikaisessa vaiheessa, sillä testaus tapahtuu joka iterointikierroksella. Tällöin korjauskustannukset saadaan pidettyä mahdollisimman alhaisina. (Pressman, 1997)

Spiraalimallin käyttökelpoinen muunnelma on komponenttimalli. Malli muistuttaa hyvin pitkälti spiraalimallia, mutta ainoa poikkeus tulee tuotantovaiheessa. Tuotantovaiheessa tunnistetaan kehitettävän komponentin ominaisuudet. Näitä ominaisuuksia käytetään hyväksi etsittäessä valmista komponenttia, joka implementoidaan ohjelmistoon. Komponentti voidaan etsiä kaupallisilta markkinoilta tai ohjelmistoa kehittävän yrityksen omasta komponenttikirjastosta. Jos valmista ohjelmistomoduulia ei ole saatavilla, tarvittava moduuli joudutaan luonnollisesti ohjelmoimaan ja testaamaan. Jos ohjelmistotalo pystyy hyvin hyödyntämään valmiita ja testattuja ohjelmistokomponentteja, keventää se kehitettävän järjestelmän testausta huomattavasti. (Pressman, 1997)

Komponenttimallin avulla saadaan aikaiseksi huomattavia kustannussäästöjä, esimerkiksi kehitettäessä raportteja tuottavia ohjelmistoja. Ajallisesti saadaan säästöä, koska moduulisuunnittelu- ja yksikkötestausvaihe lyhenee. Lisäksi ohjelmiston laatu paranee, koska käytettävät komponentit ovat olleet käytössä useamman kerran ja tämä laadun paraneminen tuo säästöjä koko ohjelmiston elinkaaren ajalle. (Pressman, 1997)

2.2.3 Extreme Programming (XP)

Extreme Programming on kevyt menetelmä ohjelmistojen tuottamiseen, erityisesti pienille kehitysryhmille. Menetelmän peruslähtökohtana on keskittää suurin osa ryhmän voimavaroista asiakkaan kannalta tärkeimpään asiaan eli toimivan ja määrittelyn mukaisen ohjelmakoodin tuottamiseen. Asiakkaan näkökulmasta XP varmistaa sen, että valmiissa ohjelmassa on juuri ne halutut ominaisuudet, jotka ohjelmistoon on määritelty. Extreme Programming on parhaimmillaan, jos alkuvaiheessa ei tiedetä

tarkalleen, mitä nämä halutut ominaisuudet ovat ja tarpeellisia ominaisuuksia tulee esille vasta projektin ollessa käynnissä. (Kolawa, 2002)

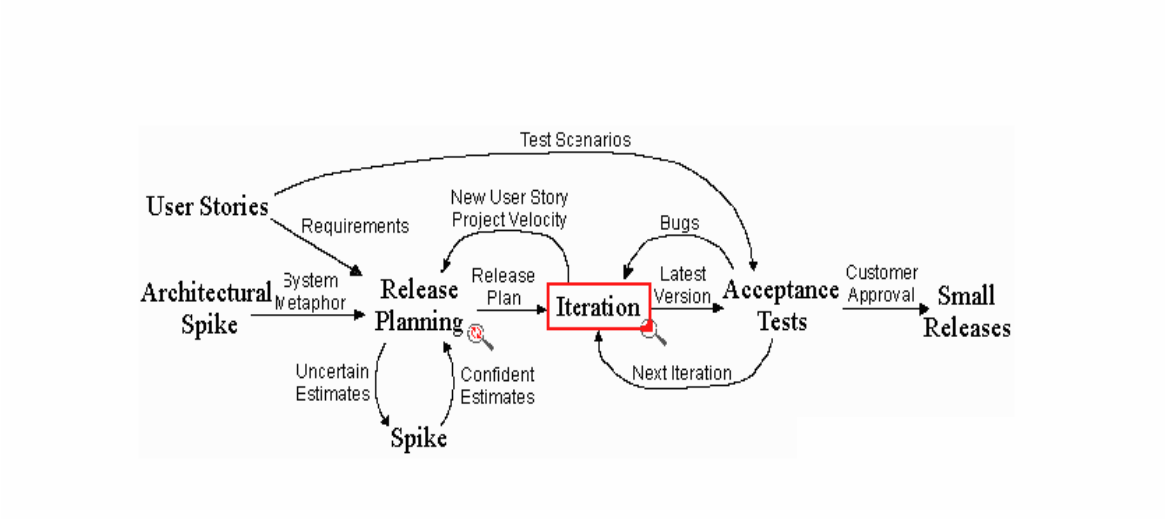
Ero perinteisten vaihejakomallien ja XP:n välillä on siinä, miten todennäköisiin muutoksiin järjestelmässä varaudutaan. Normaalisti järjestelmän arkkitehtuuri ja luokkatason rakenne suunnitellaan siten, että uutta toiminnallisuutta voidaan lisätä pienin muutoksin. Tämä johtaa helposti siihen, että järjestelmästä ja moduulien välisistä rajapinnoista tulee monimutkaisempia, kuin mitä niiden tarvitsisi olla. XP:ssä taas tulevaisuutta ei pyritä ennustamaan, sillä tulevia tarpeita ei joko tiedetä tai vaatimukset voivat muuttua. Sen sijaan järjestelmästä tehdään joka iteraatiossa rakenteeltaan niin yksinkertaiseksi kuin mahdollista. Ohjelmiston kyky muuntua uusien vaatimusten mukaiseksi, pyritään takamaan ohjelmiston yksinkertaisen rakenteen ja helpon muunneltavuuden avulla. (Beck, 1999)

Mallissa määrittely tehdään aluksi keräämällä tietoa ohjelmiston tulevilta käyttäjiltä eli niin sanottuja käyttäjätarinoita. Käyttäjätarinat ovat epämuodollisia kuvauksia siitä, mikä tavoite järjestelmällä halutaan saavuttaa sekä millaiset ovat tähän tavoitteeseen liittyvät pääasialliset käyttöskenaariot. Käyttäjätarinat ovat usein epämuodollisempia ja epätarkempia kuvauksia kuin esimerkiksi UML–mallin käyttötapauskuvaukset. Tästä ylimalkaisuudesta ei ole haittaa, koska XP–mallissa ohjelmiston toimintoja poistetaan ja lisätään jatkuvasti käyttäjiltä saadun palautteen perusteella. Päätökset ohjelmistoon lisättävistä ominaisuuksista tehdään aluksi kuitenkin käyttäjätarinoiden pohjalta. (Beck, 1999)

Käyttäjätarinoiden pohjalta määritetään projektin eteneminen ja ajankäyttö. Niiden perusteella päätetään myös mitä ominaisuuksia ohjelmistoon tehdään. Kehityksen jatkuessa ominaisuuksien tarpeellisuutta arvioidaan jatkuvasti toimivan ohjelmiston perusteella asiakkaan toimesta. XP–mallissa kehitys tehdään niin sanotuissa iteraatioissa, joissa toteutetaan muutama tärkein käyttäjätarina ja tehdään ohjelmistosta toimiva versio. Yksi tällainen iteraatio kestää tyypillisesti muutaman viikon. (Beck, 1999)

Iteraation suoritus alkaa käyttäjätarinoiden jakamisella tehtäviksi, joita ohjelmoijat sitten sitoutuvat tekemään. Kun käyttäjätarinat on saatu muutettua määrittelyiksi, niin

projektissa siirrytään suoraan toteutusvaiheeseen. Kuten kuvasta 5. näkyy, järjestelmän arkkitehtuuri ja yksityiskohtainen suunnittelu syntyvät XP-mallissa jatkuvalla iteraatiolla ja ohjelmakoodin uudelleen läpikäymisellä. (Beck, 1999)



KUVA 5. XP-malli. (Beck, 1999)

Keskeinen metodi XP-projektissa on jo tehdyn ohjelmakoodin läpikäynti. Se tarkoittaa ohjelmakoodin muokkaamista ilman, että ohjelmiston ominaisuudet muuttuvat. Ohjelmakoodista pyritään tekemään aktiivisesti selkeää ja yksinkertaista, helpommin luettavaa. Tällöin myös mahdolliset ohjelmistovirheet ovat helpommin paikallistettavissa lähdekoodista. Ohjelmakoodin yksinkertaisuus ja havainnollisuus ovat XP-metodin merkittävimpiä piirteitä. Edellä mainitut piirteet mahdollistavat XP-mallin dokumentoinnin keveyden. Mallin mukaan toimimalla ohjelmakoodista tulee automaattisesti itse dokumentoivaa. (Beck, 1999)

Testaaminen on yksi tärkeimmistä toiminnoista XP-mallissa. Mallia noudattamalla pyritään mahdollisimman suureen testauksen automatisointiin. Jokainen testi testaa vain omaa kokonaisuuttaan, joten jos jotakin testiä joudutaan muuttamaan, muita testejä ei tarvitse muuttaa tämän vuoksi. Tämä mahdollistaa tarvittaessa kattavat automaattiset testit. (Beck, 1999)

XP:ssä tavoitteena suunnitella yksikkötestit jokaiselle ohjelmistomoduulille ennen kuin varsinaista ohjelmointia on aloitettu. Jokaiselle kehitetylle ohjelmakoodille suo-

ritetaan vähintään yksikkötestaus ja ohjelmakoodin tulee läpäistä luonnollisesti tämä testi ennen julkaisua. Hyväksymistestejä tehdään usein myös pienille julkaisuille. Tämän jatkuvan testaamisen avulla koodissa olevat virheet huomataan heti koodin tuottamisen yhteydessä, eikä esimerkiksi hetkeä ennen julkistamista. (Kolawa, 2002)

2.2.4 Big-bang

Kaoottisin ja samalla ehkä ohjelmistosuunnittelijan helpoiten ymmärrettävä malli on big-bang -malli, joka kuvaa ohjelmistoprosessia maailman alkuräjähdyksen kaltaisena tuotoksena. Ohjelmisto syntyy kokoamalla yhteen tarvittavat resurssit (ihmiset, raha ja aika) ja odottamalla ”räjähdystä”. Suuren räjähdyksen tapahduttua ja savun hälvennyttyä tarkistetaan tulokset. (Patton, 2001)

Tässä mallissa ei suunnitella, dokumentoida eikä tiedetä tulosta ennen sovelluksen valmistumista. Mallissa ei voida myöskään palata aiempiin vaiheisiin, koska niitä ei yksinkertaisesti ole olemassa. Siitä johtuen testaus, jos sellaista suoritetaan, on vain virheiden raportointia asiakkaiden tietoon. Käytännössä asiakkaille välitetään tieto, mitä ohjelmistovirheitä toiset asiakkaat ovat löytäneet ja ohjelmaversio, missä ongelma tullaan poistamaan. Mallin noudattaminen on varsin yksinkertaista. Yleensä lopputulosta ei voida kuitenkaan taata, joten hyvänä mallina tätä ei voida pitää. (Patton, 2001)

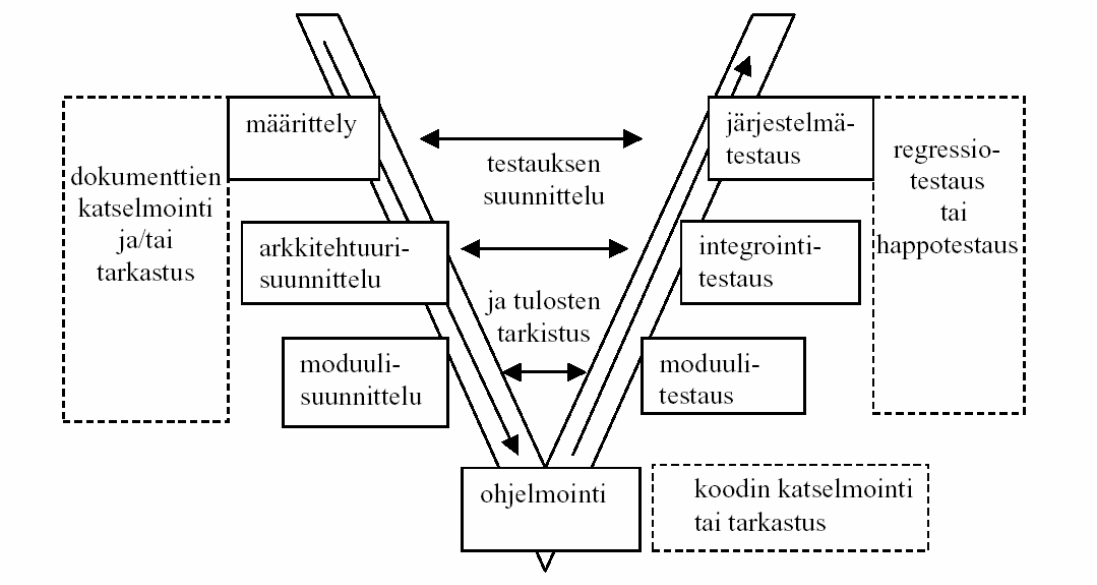
Mallin elinkaaren alussa tapahtuvaa työtä on täysin mahdotonta jäljittää, joten ylläpitovaihe on kohtuuttoman raskas. Malli voi toimia ainoastaan, jos lopputuloksena saatu ohjelmisto on kokoelma jo tunnettuja ja toimiviksi todettuja ohjelmistomoduuleja, joiden rajapinnat ovat kunnossa. Mallin toimivuus vaatii myös sen, että ennen ylläpitovaihetta on pystytty todentamaan ohjelman vastaavan niitä tarpeita, mitä siltä on vaadittu. (Patton, 2001)

3 OHJELMISTOTESTAUKSEN V-MALLI

Ohjelmistotestauksen tarkoituksena on löytää virheitä ohjelmistosta, ennalta määriteltyä systemaattista mallia noudattaen. Mitä enemmän testattavasta järjestelmästä löytyy virheitä, sen onnistuneempi testin katsotaan olevan. Yksi ohjelmistokehityksen perusmalleista on V-malli, joka jakaa ohjelmistoprojektin muun muassa suunnittelu-, toteutus- ja testaustasoihin sekä esittelee tasojen väliset selkeät suhteet. V-malli on metodi, joka on syntynyt vertailemalla ja kehittämällä ohjelmistoprojekteissa käytettyjä menetelmiä. (Britschgi, 2004)

Ohjelmiston toteutuksen tasoja ovat vaatimusten määrittäminen, ohjelmiston määrittely, arkkitehtuurisuunnittelu (järjestelmäsuunnittelu) ja yksikkösuunnittelu (moduulisuunnittelu). Näiden vaiheiden dokumentaatio löytyy V-mallin vasemmalta puolelta. Vastaavasti mallin vasemmalla puolella on testaustaso, jonka avulla pyritään varmistamaan työvaiheen oikeellisuus. (Haikala & Märijärvi, 2004)

Järjestelmätestausta seuraa tarvittaessa erillinen kenttätestaus ja hyväksymistestaus. Muita V-mallia tukevia testausmuotoja ovat mm. dokumenttien katselmointi, regressio-testaus ja happotestaus.[Haikala & Märijärvi, 2004]



KUVA 6. V-malli (Riekkinen & Karvinen, 2004)

V-mallia noudattaen testaus suunnitellaan testaustasoa vastaavalla suunnittelutasolla. Järjestelmätestaus suunnitellaan määrittelyvaiheessa, integrointitestaus arkkitehtuurin suunnitteluvaiheessa ja moduulitestaus moduulisuunnitteluvaiheessa. Suunnitteluvaiheiden ja testaustasojen yhteys näkyy kuvasta 6. Tulokset todetaan oikeiksi vertaamalla niitä tasoja vastaaviin, ennalta tuotettuihin dokumentteihin. (Spillner, 2002)

V-mallia noudatettaessa saadaan kiistatonta etua siitä, miten arkkitehtuurisuunnitelmat ja dokumentaatiot voidaan toteuttaa aikaisessa vaiheessa, ennen varsinaista ohjelmointityötä. Myös tieto V-mallin noudattamisesta kehitystyössä ja testaussuunnitelmassa, auttaa ohjelmoijia tekemään parempaa koodia. Tämä johtuu siitä, että ohjelmointi voidaan aloittaa tutustumalla järjestelmällisesti luotuun dokumentaatioon. (Britschgi, 2004)

Testaussuunnitelman huonona puolena voi olla se, että ohjelmoija pyrkii tekemään koodinsa tavalla, jolla läpäistään ainoastaan testitapaukset. Tähän ongelmaan törmätään myös testauskäytännössä, jossa ohjelmoija suorittaa testauksen. Ilmiö ei ole välttämättä edes tarkoituksellinen, vaan tapahtuu toimesta tahattomasti. Ohjelmoijan laatiessa/noudattaessa testaussuunnitelmaa, suunnitelmasta tulee helposti hänen ohjelmointityylilleen sopiva, jolloin osa virheistä jää löytymättä. (Pressman, 1997)

Moduulitestauksen apuvälineinä käytetään usein tynkäpetiä ja siihen kiinteänä osana liittyviä testiajureita. Apuvälineiden kehittäminen lisää kustannuksia ja niiden ylläpito voi olla hankalaa. Testauksen avuksi kehitetty järjestelmä mahdollistaa testauksen aikaisemman aloittamisen, sillä riittää, että testattavan moduulin ja sen rajapinnan määritykset on tehty. (Broekman & Notenboom, 2003)

V-mallin avulla testausprosessi voidaan jaotella eri vaiheisiin. Näitä vaiheita ovat yksikkö-, integraatio-, järjestelmä- ja hyväksymistestaus. Näiden testausvaiheiden lisäksi suoritetaan usein ohjelman tai järjestelmän koosta ja käyttöympäristöstä riippuen muutakin testausta, kuten tietoturva-, suorituskyky-, kuormitus- ja käytettävyystestejä.

3.1 Yksikkötestaus

Yksikkötestausta kutsutaan usein myös moduulitestaukseksi. Moduuli koostuu yleensä noin 100–1000 ohjelmarivistä. Moduulin toimintaa verrataan moduuli- ja arkkitehtuurisuunnittelun tuloksiin, tavallisimmin tekniseen määrittelydokumenttiin. Moduulitestaus priorisoi ohjelmiston toiminnot ja testaa kriittisimmät osat perusteellisimmin ja hallitummin jo yksikkötasolla. Muusta testauksesta poiketen testauksen suorittaa yleensä moduulin toteuttanut ohjelmoija, vaikkakin parempi vaihtoehto olisi Extreme Programming -mallin mukainen niin sanottu kaveritestaus, jossa kehittäjät testaavat toistensa työt ristiin. (Britschgi, 2004)

Testaustekniikkana yksikkötestauksessa käytetään yleensä lasilaatikko-testausta, jossa testauksessa käytettään hyväksi tietoa yksikön sisäisestä rakenteesta ja pyritään paikallistamaan mahdolliset virheet tätä kautta. Testauksessa käytetään yleensä erilaisia ajureita, jotka antavat syötteitä yksikölle ja sen jälkeen tarkistavat, että yksikkö palauttaa halutun arvon. Joskus käytetään myös niin sanottuja tynkämoduuleja, joiden tehtävänä on simuloida muita testauksessa mahdollisesti tarvittavia moduuleja. (Pressman, 1997)

Mustalaatikko-testaus on myös yksi testaustekniikka. Tällöin ohjelamakoodia testataan tuntematta testattavan moduulin tietorakennetta ja ohjelman sisästä syntaksia. Testauksessa huomioidaan käyttäjän näkökulma, jos se vain on tällä testaustasolla mahdollista. Moduulin toimintaa verrataan moduulista tehtyyn määritykseen. (Virkkunen, 2002)

Testausta voidaan suorittaa myös positiivisena ja negatiivisena testauksena. Positiivisen testauksen tarkoitus on todentaa ohjelmiston toimivan oikein, kun sitä käytetään oikein. Negatiivisen testauksen tarkoituksena on selvittää virhetilanteiden käsittelyn toimivuutta. Negatiivisen testauksen aikana ohjelmistoa käytetään tarkoituksella väärin. (Stenberg, 2004)

Ohjelmiston testaus voidaan suorittaa myös dynaamisena testauksena. Tällöin ohjelmaa testataan suorittamalla sitä tai sitten moduuli testataan staattisesti lukemalla lähdekoodia. Yksikkötestausta varten on luotava kehitysympäristö (ajurit, tynkämoduulit), sillä yksiköitä ei aina voida testata valmiissa ”aidossa” ympäristössä. Näin ollen pienen kokonaisuuden testaus voi vaatia runsaasti valmisteluja. (Pressman, 1997)

Yksikkötestausta pidetään yleensä ohjelmointia tukevana työvaiheena ja sitä suoritetaankin usein samanaikaisesti ohjelmoinnin kanssa. Testaus aloitetaan, kun ohjelmoitavan yksikön lähdekoodi on kirjoitettu ja päällisin puolin tarkistettu. Yksikkötestausta on mahdollista yksinkertaistaa, jos yksiköt suunnitellaan niin, että niillä on mahdollisimman vähän tehtäviä. Yksikkötestauksen etuna on esimerkiksi testattavan osan pieni koko. Pienestä sovelluksen osasta virheet löytyvät helpommin ja nopeammin. Samoin korjauskustannukset ovat suhteellisen pienet tällä tasolla, sillä koodia on vielä vähän eivätkä korjaukset vielä vaikuta muuhun sovellukseen. (McConnell, 1998)

Yksikkötestaus kannattaa tehdä pieninä paloina. Yksikkötestausta voidaan pitää pienten ohjelmapalasten testaamiseksi. Testattavia kohteita ovat metodit, tietyt ohjelmapolut ja funktiot. Ohjelmoija tekee koodin pieninä osasina, jolloin on luontevaa testata jokainen osa erikseen. Seuraavaan osaseen siirrytään vasta, kun työn alla olevan osasen toiminnallisuus on saatu todennettua. Ohjelmoijan on helpompi hallita pienempiä askeleita ja sitä kautta saatu lopputuloskin on parempi. (Hunt, 2002)

3.2 Integrointitestaus

Integrointitestausvaiheessa yhdistellään moduuleita/yksiköitä tai moduuliryhmiä (osajärjestelmiä). Vaiheen tarkoituksena on testata yksittäisten komponenttien välistä vuorovaikutusta sekä tiedonsiirtoa ja toiminnallisuutta niiden välillä. Tärkeintä on kiinnittää huomio moduulien välisten rajapintojen toimivuuden oikeellisuuteen. Integrointitestauksen tuloksia verrataan tekniseen määrittelyyn. Sulautettujen ohjelmistojen integrointitestauksessa testataan ohjelmasovellusten välisen toiminnan lisäksi ohjelmisto–laitteistorajapintaa ja laiteläheisiä ajureita. (Broekman & Notenboom, 2003)

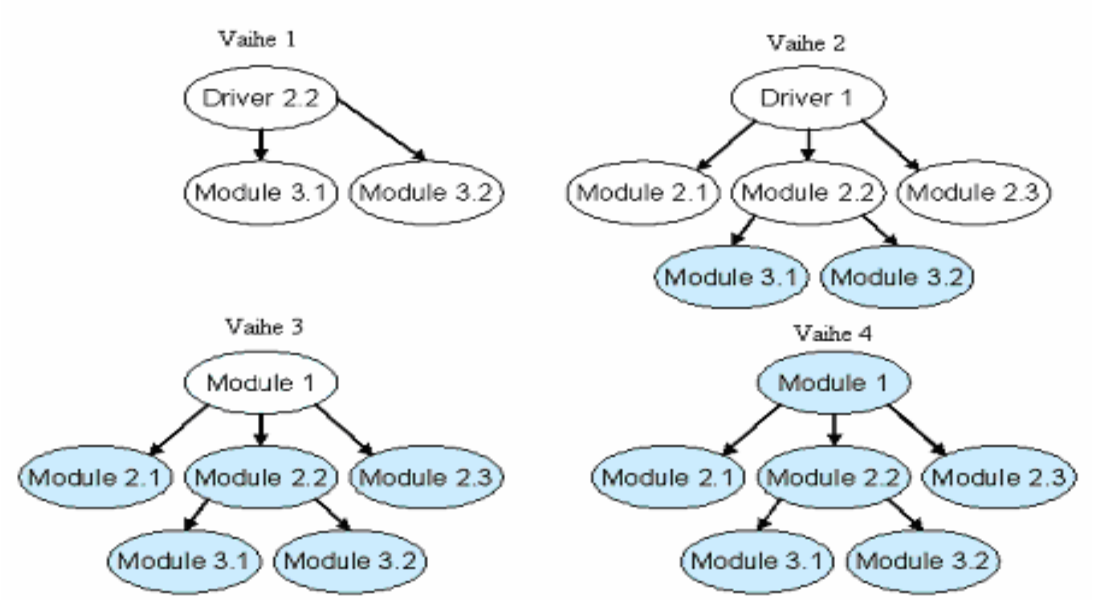
Tämä testausvaihe voidaan käynnistää tarvittaessa heti, kun järjestelmän ohjelmistomoduuleita on ohjelmoitu testattavan kokonaisuuden verran. Integroitavien yksiköiden yksikkötestauksen täytyy olla hyväksyttävästi tehtynä ennen testin aloittamista. Samalla kun testattavaan kokonaisuuteen liitetään uusia osia, varmistetaan se, että testattavan moduulijoukon perustoiminnallisuus säilyy tai kertaalleen testatut toiminnot toimivat. Integraatiotestauksen ongelmana on löytyneiden virheiden paikantaminen. Moduulien väliset rajapinnat ovat monimutkaisia ja saattaa olla vaikea todentaa, missä on lopullinen virheen aiheuttaja. Virhe voi olla esimerkiksi sisäinen virhe tai vika moduulin rajapinnan ohjelmoinnissa vai onko virhe peräti moduulin sisällä. Kyseistä virhettä ei ole vain löydetty aiemmissa testausvaiheissa. (Haikala & Märijärvi, 2004)

Integrointitestausta voidaan suorittaa kahdella eri menetelmällä, joko ei-lisäävällä tai lisäävällä. Ei-lisäävässä- menetelmässä kaikki osat kootaan kerralla testattavaksi kokonaisuudeksi. Ei-lisäävä- menetelmä ei kuitenkaan ole kovinkaan tehokas, koska löydetyn virheen varsinaisen syyn löytäminen on sitä vaikeampaa mitä suurempi testattava kokonaisuus on. (Pressman, 1997)

Yleisemmin käytetään lisäävää menetelmää, jossa järjestelmään lisätään osia vasta, kun edelliset on saatu hyväksytysti testattua. Lisäävässä integroinnissa käytetään

alhaalta-ylös (bottom-up) ja ylhäältä alas-(top-down) testaustapoja. (Broekman & Notenboom, 2003)

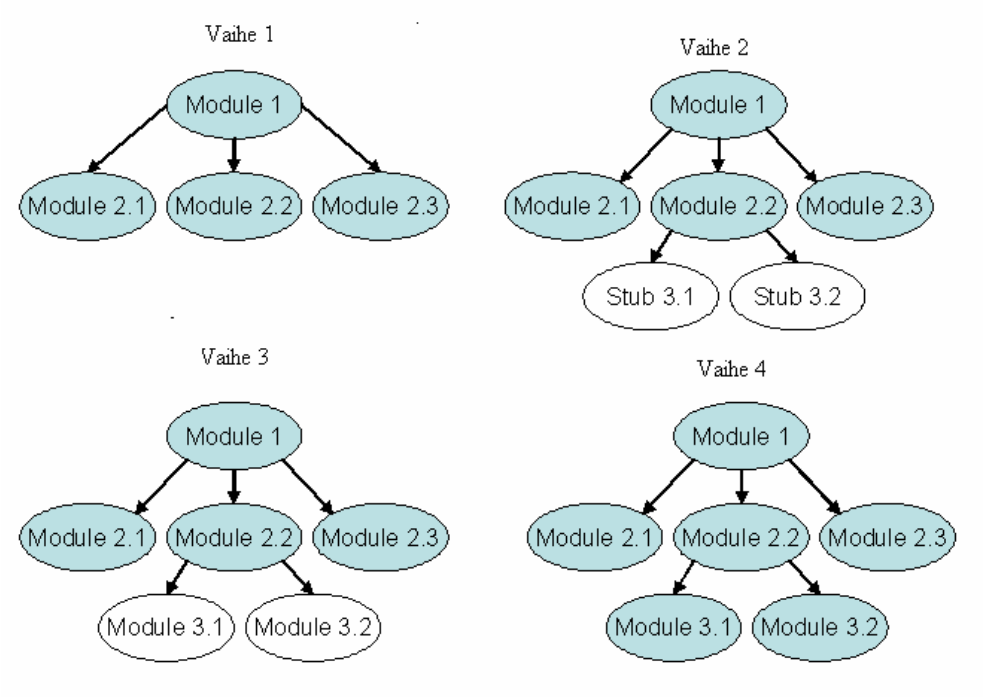
Alhaalta-ylös testauksessa testattavien moduulien ylemmät kerrokset korvataan ajurilla (driver). Kuvassa 7. esitetään yksinkertaistettuna alhaalta-ylös testauksen eteneminen. Aluksi yksiköiden tarvitsemien syötteiden tekemiseen käytetään ajuria (ajuri 2.2). Siirryttäessä toiseen vaiheeseen, testatut moduulit on merkitty tummalla taustalla. Toisessa vaiheessa on edellisen vaiheen ajuri 2.2 korvattu oikealla yksiköllä 2.2 ja uusi ajuri 1 luo syötteet uusille testattaville yksiköille. Kolmannessa vaiheessa taas edellisen vaiheen ajuri korvataan asiaan kuuluvalla yksiköllä 1. Viimeisessä eli neljännessä vaiheessa integrointitestaus on suoritettu ja voidaan siirtyä eteenpäin testausprosessissa. (Broekman & Notenboom, 2003)



KUVA 7. Alhaalta-ylös testaus (Broekman & Notenboom, 2003)

Ylhäältä-alas testauksessa testattavaa osaa alemmat moduulit korvataan tynkämoduuleilla. Kuvassa 8. esitetään ylhäältä-alas testauksen perusvaiheet. Ensimmäisessä vaiheessa valmiina olevat yksiköt testataan toisiaan vastaan. Toisessa vaiheessa testataan yksikkö 2.2, johon liittyvät yksiköt (module 3.1 ja 3.2) eivät vielä ole testauskunnossa. Puuttuvat yksiköt korvataan tynkämoduuleilla (Stub 3.1 ja 3.2), jolloin

saadaan yksikön toiminta testattua ja osa virheistä korjattua tarvitsematta odottaa muiden osien ohjelmointia. Kolmannessa vaiheessa, kun tarvittavat yksiköt on saatu ohjelmoitua, tynkämoduulit korvataan niillä ja kokonaisuus testataan uudelleen. Viimeisessä eli neljännessä vaiheessa integrointitestaus on suoritettu ja voidaan siirtyä eteenpäin testausprosessissa. (Broekman & Notenboom, 2003)



KUVA 8. Ylhäältä-alas testaus (Broekman & Notenboom, 2003)

Integrointitestauksessa testaustekniikkana käytetään yleensä harmaalaatikkotestausta. Tekniikassa on osioita lasilaatikko-testauksesta ja mustalaatikkotestauksesta. Testitapauksia laadittaessa käytetään hyväksi moduuleista laadittuja määrityksiä, ohjelman sisäisen rakenteen tuntemista ja esimerkiksi moduuleista laadittuja vuokaavioita. (Mosley & Posey, 2002)

3.3 Järjestelmätestaus

Järjestelmä- eli systeemitestaus suoritetaan ennen ohjelmiston luovuttamista asiakkaalle. Järjestelmätestauksen tarkoituksena on todentaa toimiiko sovellus, kuten se on määritetty. Testauksen tarkoituksena on myös tunnistaa sekä korjata määrittelyn ja tarkasteltavan systeemin välisiä ristiriitoja. Testauksessa tarkastelun kohteena on koko järjestelmä ja tuloksia verrataan määrittelydokumentaatioon (ohjelmiston toiminnalliseen määrittelyyn) ja asiakasdokumentaatioon (käyttöohjeisiin). Sovellusta tarkastellaan loppukäyttäjän näkökulmasta käyttäen apuna erilaisia käyttötapauksia. (Haikala & Märijärvi, 2004)

Järjestelmätestaus tulisi suorittaa ympäristössä, joka vastaa mahdollisimman paljon järjestelmän todellista käyttöympäristöä. Järjestelmätestauksen tavoitteena on löytää puutteet, joita ei voida löytää testauksen aikaisemmissa vaiheissa. Järjestelmä on enemmän kuin osiensa summa ja jotkin viat voidaan havaita vasta, kun kaikki järjestelmän palaset yhdistetään. (Pressman, 1997)

Tämä testausvaihe voidaan jakaa kahteen eri tyyppiin, toiminnalliseen ja ei toiminnalliseen testaukseen. Ei toiminnallinen testaus sisältää muun muassa kuormitustestit, luotettavuustestit, asennustestit ja käytettävyystestit. (Haikala & Märijärvi, 2004)

Tekniikkana järjestelmätestauksessa voidaan käyttää mustalaatikko-testausta. Toiminnallinen testaus suunnitellaan käyttäjän näkökulmasta. Testissä käytetään syötteenä määrittelyistä saatua informaatioita. Samoista määrittelyistä on oltava luettavissa myös saatu vaste, edellyttäen kuitenkin, että dokumentaatio on riittävällä tasolla. (Beizer, 1995)

3.4 Hyväksymistestaus

Hyväksymistestauksessa asiakas testaa sovelluksen käytettävyyttä ja sopivuutta itselleen. Tämä onkin välttämätöntä, sillä ohjelman kehittäjillä ei yleensä ole käytännön kokemusta sovelluksen aihealueesta eikä loppukäyttäjien yleisistä toimintatavoista. Hyväksymistestaus paljastaa usein sovelluksen huonon käytettävyyden ja dokumentaation puutteellisuuden. Tämä johtuu siitä, että käyttäjät saattavat ymmärtää väärin

käyttöohjeita, antaa vääriä syötteitä tai odottaa erilaisia vasteita. (Haikala & Märijärvi, 2004)

Hyväksymistestauksen ensimmäistä vaihetta kutsutaan alfa-testaukseksi. Alfa-testauksen suorittavat todelliset käyttäjät yleensä sovelluksen kehittäneessä yrityksessä tai muuten kontrolloidussa ympäristössä. Testauksessa käytetään asiakkaan luomaa tai mukanaan tuomaa testiaineistoa, eikä aiempaa testausta varten luotua testidataa. Aineiston vaihdoksesta johtuen voidaan testissä havaita uusia virheitä, koska yleensä todellinen aineisto on suurempi ja monipuolisempi kuin muissa testausvaiheissa käytetty. Järjestelmätestausvaiheen testaajat ovat mukana tarkkailemassa testausta koko ajan ja tarvittaessa he voivat neuvoa alfa-testin suorittajia. Lisäksi käyttäjien kaikki toiminnot huomioidaan. Näin saadaan selville toiminnassa tapahtuvat virheet ja käyttöongelmat. Alfa-testauksessa voidaan myös havaita toiminnallisuuksia, jotka eivät vastaa asiakkaan toivomuksia tai käsityksiä sovelluksesta. (Pressman, 1997)

Beta-testaus on testauksen vaiheista viimeisin ja sillä pyritään takaamaan, että sovellus on valmis julkaistavaksi. Beta-testauksen suorittavat loppukäyttäjät omassa käyttöympäristössään, eikä kehittäjä enää ole paikalla tarkkailemassa/neuvomassa testausta. Beta-testaus on ulkoista testausta, jossa sovellus lähetetään lopulliselle käyttäjäryhmälle, mahdollisimman hyvin edustavalle joukolle, testattavaksi todellisessa ympäristössä. (Pressman, 1997)

Rajattu käyttäjäryhmä kokeilee sovellusta ja raportoi säännöllisin väliajoin sovelluksen toiminnasta ja mahdollisesta toimimattomuudesta, jotka ovat jääneet aiemmissa testeissä havainnoimatta. Osa raportoiduista virheistä johtuu usein käyttäjien omista epäilyistä, väärinkäsityksistä ja käyttötottumuksista. (Pressman, 1997)

4 PONSSE-KONSERNI

Ponsse - konserniin kuuluvat emoyhtiö Ponsse Oyj sekä tytäryhtiöt Ponsse AB Ruotsista, Ponsse AS Norjasta, Ponssé S.A. Ranskasta, Ponsse UK Ltd. Isosta-Britanniasta ja Ponsse USA Inc. Amerikan Yhdysvalloista.

Konserni suunnittelee, valmistaa ja markkinoi ympäristöystävällisiä ja tehokkaita tavaralajimenetelmään perustuvia metsäkoneita sekä puunkorjuuseen liittyvää tietotekniikkaa. Tuotteet ovat ja niiden tulee myös jatkossa olla tehokkaimpia ja kestävimpiä metsäkoneita markkinoilla. Metsäkoneissa on ja tulee myös jatkossa olla enemmän lisäarvoa tuovia ominaisuuksia kuin kilpailevissa merkeissä. Näistä merkittävin on informaatiotekniikan tehokas hyödyntäminen.

Ponsse-konsernin liikevaihto ja tulos kehittyivät hyvin vuonna 2004. Liikevaihto kasvoi 14,4 prosenttia 190,0 miljoonaan euroon (2003, 166,0 miljoonaa euroa) ja liikevoitto 37,7 prosenttia 19,6 miljoonaan euroon(2003, 14,3 miljoonaa euroa). Osakekohtainen tulos oli 0,97 euroa (0,65 euroa) ja omavaraisuusaste 37,5 prosenttia (55,7 prosenttia). Ponsse Oyj:n osakkeet noteerataan Helsingin Pörssin päälistalla.

Työntekijöitä Ponsse-konsernissa oli vuoden 2004 lopussa 688 henkilö. Suomessa työntekijöitä oli 517 ja ulkomailla 171. Tietojärjestelmä yksikössä Kajaanissa, josta opinnäytetyö tilattiin, työntekijöitä on 50 henkilöä.

Tuotantoa Ponsse Oyj:llä on tuotantoa Kajaanissa ja Vieremällä. Kajaanin yksikössä kehitettään ja valmistetaan metsäkoneiden ja sen toimilaitteiden sulautettuusjärjestelmien elektroniikka ja ohjelmisto. Lisäksi Kajaanissa kehitetään ja valmistetaan myös muita metsäkoneissa ja sen toimilaitteissa tarvittavia piirikortteja. PC-elektroniikan kehitys ja valmistus tapahtuu myös Kajaanin yksikössä. Kaikissa Pons-

se Oyj:n tuotteissa oleva ohjelmisto, PC- ja sulautetunjärjestelmien ohjelmisto, kehitettään Kajaanissa. Metsäkoneiden valmistus ja mekaniikan- ja hydrauliikan tuotekehitys sijaitsee Vieremällä.

Metsäkoneiden huolto- ja myyntiorganisaation on maailmanlaajuinen ja se kasvaa kokoajan. Suurin osa huollosta ja myynnistä tapahtuu Ponsse-konsernin tytäryhtiöiden kautta. Loput konsernin tuotteiden myynnistä ja huollosta tapahtuu Ponsse-konsernin ulkopuolisten dealereiden kautta.

4.1 Ohjelmistotuotekehitys Ponsse Oyj:ssä

Ponsse Oyj:ssä kehitetään ohjelmistoja sekä sulautettuun ympäristöön että PC-ympäristöön. Ohjelmistotuotekehityksessä on tällä hetkellä töissä 11 vakituista ohjelmoijaa. Lisäksi yksi henkilö on tekemässä käyttöohjeita ja kaksi testausinsinööriä suorittaa ohjelmistotestausta kentällä. Ohjelmistotuotekehitys tapahtuu Ponsse Oyj:n Kajaanin yksikössä.

Sulautetun järjestelmän ohjelmointi tapahtuu C-kielellä. Sulatettu järjestelmä ohjaa pääasiassa metsäkonetta sekä siihen liittyviä toimilaitteitta. PC-sovelluksia on jo huomattavasti laajemmassa toimintaympäristössä. Yksi sovellusalue on mittalaite, jonka avulla hallitaan puunkäsittelyä hakkuukoneessa. Tähän liittyy olennaisena osana hakkuumäärien raportointi ja saadun tiedon käsittely sekä harvesterissa että konttorilla. Hakkuutapahtuman optimoimiseksi on olemassa useita eri sovelluksia, joiden avulla voidaan simuloimalla tehostaa hakkukoneen toimintaa. Logistiikkasovelluksien avulla ohjataan materiaalivirran (puun) kulkua. Sovelluksien avulla puun vaiheita voidaan seurata ostotapahtumasta paperitehtaan tai sahan portille. PC-sovellukset kehitettään VB6-sovelluskehittimellä.

Ohjelmistotuotekehityksellä on sekä ulkoisia ja sisäisiä asiakkaita. Ulkoisia asiakkaita ovat muun muassa kaikki suuret suomalaiset metsäyhtiöt. Ulkoisiin asiakkaisiin kuuluu myös ulkomaalaisia metsäyhtiötä, sahateollisuuden yrityksiä ympäri maailmaa, yleensäkin puunjalostukseen keskittyneitä pieninä ja suuria yrityksiä. Suoranai-

seen puunkorjuuseen liittyviä asiakkaita ovat metsäkoneurakoitsijat ja puutavararekkaliikennöitsijät. Osa Ponsse Oyj:n ulkomaanmyynnistä tapahtuu dealereiden avulla, jotka myös tilaavat maakohtaisia ominaisuuksia ohjelmistoihin, joten heidätkin voidaan laskea ulkoisiksi asiakkaisiin. Sisäisiä asiakkaita ovat muun muassa Ponssen huolto-organisaatio maailmalaajuisesti, metsäkoneiden ja sen komponenttien valmistuksesta vastaavat tuotantolinjat sekä metsäkoneen mekaniikasta ja hydrauliikasta vastaava tuotekehitysyksikkö. Ohjelmistotuotannon asiakkaiden koko vaihtelee laajalla skaalalla, alkaen yksittäisestä metsäkoneurakoitsijasta päättyen globaaliin metsäyhtiöön. Tässä on yksi Ponsse Oyj:n ohjelmistotuotannon haasteista, kuinka kyetä palvelemaan tehokkaasti ja laadukkaasti näinkin vaihtelevaa asiakaskuntaa.

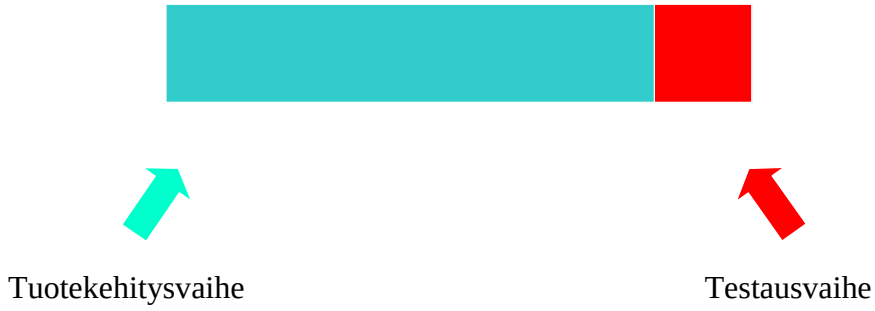
Uudet ominaisuudet ohjelmoidaan ja testataan (konttoritesti konttorilla ohjelmoijien toimesta ja kenttätesti testausinsinöörien toimesta) release-jakson puitteissa. Ohjelmisto-release pitää sisällään useita eri ohjelmia ja jokaisessa release-jaksossa ohjelmisto-releaseen kehitettään myös uusia ohjelmistokomponentteja. Tunnetut ohjelmistovirheet korjataan tarvittaessa myös release-jakson aikana. Tätä puolen vuoden tuotekehitys- ja testauskäytäntö jaksoa kutsutaan release-jaksoksi ja jakson aikana tehdyn ohjelmoinnin työ tulosta kutsutaan ohjelmistoksi, ohjelmisto-releaseksi. Ohjelmisto pitää sisällään PC-sovelluksia metsäkoneisiin, rekkoihin ja konttoreihin sekä ohjelmia sulautettuusjärjestelmään.

Release-jakson aikana kehitytettävä ohjelmisto rakennetaan hyvin pitkälti aina edellisen ohjelmisto-releasen päälle. Ohjelmiston perustoiminnallisuudet pysyvät aina samana. Kehitystyönä ohjelmistoon tehdään uusia ominaisuuksia vanhoihin ohjelmistokomponennteihin ja jatkokehitettään olemassa olevia ominaisuuksia vanhoihin ohjelmistokomponennteihin. Release-jaksolla kehitettään myös uusia ohjelmistokomponentteja sekä korjataan ohjelmistovirheitä.

Ohjelmistotuotekehityksessä yksikkö- ja integrointitestauksen suorittavat (konttoritestaus) ohjelmistosuunnittelijat omissa työpisteissään ja testauslaboratoriossa testauslaitteiston ja muiden tarvittavien laitteistojen avulla. Kenttätestauksen suorittavat testausinsinöörit. He asentavat ohjelmiston koeasiakkaiden koneisiin ja suorittavat perustoiminnallisuustestit sekä testaavat uudet ominaisuudet. Tämän jälkeen ohjelmistot jäävät asiakkaille käyttöön ja he raportoivat mahdollisesta virheistä joko tes-

tausinsinööreille tai suoraan ohjelmistokehittäjille. Ohjelmistotestauksen katsotaan päättyneen, kun ohjelmistosta ei ole löytynyt virheitä pariin viikkoon. Testattavat muutokset sekä ohjelmaversionumerot välitetään kehittäjältä testausinsinöörille muutos- ja kokeilunseurantailmoituksen avulla, suusanallisesti tai sähköpostilla. Muutos- ja kokeilunseurantailmoitus on ainoa dokumentti josta on löydettävissä tietoa ohjelmistotestauksesta ja siinäkin hyvin suppeasti.

Varsinainen hyväksymistesti (kenttätesti) sijoittuu release-jakson loppuun. Aktiivisimmillaan ohjelmistotestaus on juuri tämän release-jakson lopussa, kuten kuvasta 9. on nähtävillä.



KUVA 9. Kuuden kuukauden mittaisen release-jakson toiminnot

Ohjelmiston läpäistyä testausvaiheen hyväksytysti ohjelmistosta tehdään ohjelmistojen muutosilmoitus. Muutosilmoitus sisältää listauksen uuden ohjelmiston sisällöstä ja tarvittavista ohjeista. Ilmoitus on signaali Ponsse Oyj:n organisaatiolle, että uusi ohjelmisto on testattu ja sitä voidaan alkaa asentaa/päivittää asiakkaiden laitteistoihin ja tuotannossa valmistuviin uusiin tuotteisiin.

Muutos- ja kokeilunseurantailmoitus ja ohjelmistojen muutosilmoitus tallennetaan Ponsse Oyj:n toiminnanohjausjärjestelmään. Ilmoitukset ovat osa yrityksen toimintaprosessia, jonka avulla hallittaan muun muassa tuotteiden rakenteita. Tästä kappaleesta eteenpäin Ponsse Oyj:stä käytetään nimitystä Ponsse.

4.2 Release-jaksolla tapahtuva ohjelmistotestaus

Ohjelmistojen kehitysmalli on ollut suoraviivainen, mistä johtuen ohjelmistotestaus ajoittuu pääasiassa ajallisesti release-jakson loppuvaiheeseen. Suurin osa ohjelmistomuutoksista tulee kenttätestaukseen release-jakson lopussa, jolloin löydettyjen ohjelmistovirheiden määrä on suuri. Testausaikaa on varattu kuukausi yhden release-jakson lopussa. Testausaikana kuukausi on juuri ja juuri riittävä, jos suuria ongelmia ei ilmene. Kuukauteen sisältyvät yleensä ohjelmaversioiden korjauskierrokset, tästä johtuen kuukausi ei olekaan riittävän pitkä aika kenttätestauksen suorittamiseksi. Testausaikaan sisältyy lisäksi tuotannossa tapahtuva 0-sarjan testi sekä ulkomailla tapahtuva testaus. 0-sarjan testin avulla todennetaan uuden ohjelmistoversion tuotannolliset näkökohdat, onko se esimerkiksi helppo asentaa tuotannossa uusi laitteisiin. Ulkomaan testien avulla varmistetaan ohjelmiston toimivuus ulkomaiden tarpeet huomioiden.

KUVA 10. Kuukauden pituisen testausvaiheen toiminnot

Kuvassa 10. näkyvä sovellusten kenttätestaus suoritetaan oikeassa ympäristössä, metsäkoneessa, puutavararekassa tai asiakkaan konttorilla. Testiin valittujen metsäkoneiden omistajat ovat urakoitsijoita, joiden kanssa Ponsse on tehnyt yhteistä kehitystyötä jo useamman vuoden ajan ja tätä kautta urakoitsijoiksi on valikoitunut vaativa käyttäjäjoukko. Testausvaiheessa koneille voidaan joutua päivittämään ohjelmistoja useammankin kerran ja osa päivityksistä tehdään ilman, että testausinsinööri käy paikalla. Ladattavan ohjelmistopaketin ollessa pieni, asennuspaketti välitetään harvesteriin suoraan GSM/GRPS – yhteydellä. Koneen kuljettaja asentaa asennuspake-

tin järjestelmään itsenäisesti. Paketin ollessa suuri toimitetaan se asiakkaan konttorille, josta hän siirtää sen esimerkiksi USB- stick:llä harvesteriin.

Vaikka ohjelmistoa on testattu aktiivisesti, löytyy siitä yleensä julkaisun jälkeen ohjelmistovirheitä. Jos esille tullut virhe on vakava, virhe korjataan ja testataan vähintään siinä ympäristössä (asiakkaalla), jossa se on tullut esille. Jos virhettä ei enää esiinny, korjatuista ohjelmamoduuleista tehdään korjauspaketti (ServicePack). Paketti asennetaan julkaisun jälkeen sellaisiin laitteisiin, joissa ongelmia on esiintynyt, sekä laitteisiin, joissa ohjelmistovirheen esiintymisen mahdollisuus on todennäköistä. Tarvittaessa korjauspaketti voidaan päivittää myös kaikki laitteet, joihin virheellinen ohjelmisto on asennettu.

Ilmenevät ohjelmistovirheet voivat olla myös niin pieniä, että ongelmasta annetaan vain tiedote asiasta Ponssen organisaatiolle. Näitä pienempiä virheitä voidaan kasata, kun niitä katsotaan olevan riittävä määrää, julkaistaan korjauspaketti. Korjauspaketilla korjataan senhetkiset tunnetut korjausta vaativat ohjelmistovirheet. Korjauspaketille suoritetaan konttoritesti ja kenttätesti. Ulkomaan testi suoritetaan, jos ohjelmistovirhe on löytynyt ulkomailla ja vastaavasti 0-sarjan testi suoritetaan, jos ongelma on havaittu tuotannossa. Korjauspaketteja tehdään yleensä kaksi tai kolme kappaletta. Korjauspaketeista tiedotetaan ohjelmiston muutosilmoituksella.

4.3 Ohjelmistotestauksen vaiheet

Kun ohjelmistotuotekehityksessä on halutut ominaisuudet ohjelmoitua, ohjelmoijat suorittavat niille tarvittavat konttoritestit. Konttoritesti on ohjelmiston toiminnallisuuden tarkistusta. Tarkistuksen tukena käytetään sitä materiaalia, mitä aiheeseen liittyvät suunnittelijat ovat saaneet: sähköposteja, asiakkailta saatua palautetta ja toiminnan määrityksiä, jos niitä on olemassa. Kun ominaisuus on saatu todennettua toimivaksi tai ohjelmistovirhe on saatu korjattua, suunnittelijat hyväksyvät konttoritestin suoritetuksi. Hyväksyminen tapahtuu täyttämällä muutos- ja kokeilunseuran-

tailmoituksen ja tallentamalla sen toiminnanohjausjärjestelmään. Ilmoitukseen kirjataan testattavat ohjelmaversiot ja niiden testattavat ominaisuudet.

Edellä mainittujen vaiheiden jälkeen kaikista kehitetystä ohjelmistosta rakennetaan asennuspaketti, jonka valmistumisesta lähetetään tieto testausinsinööreille. Asennuspaketti rakennetaan erillisellä builder–koneella, jolloin asennuspaketin eri ohjelmat käännetään lähdekoodista ja saaduista tiedostoista rakentuu automaattisesti asennuspaketti. Tässä vaiheessa tulee yleensä aluksi ongelmia, varsinkin ocx:ien ja dll:ien kohdalla. Eri paikoissa on eri versioita, vaikka kaikki olisi kuinka automaatista tahansa. Tämä onkin jo ensimmäinen testi asennuspaketille, sillä kaikkien ohjelmistomoduulien rajapinnat tulee olla synkronissa keskenään.

Asennuspaketin valmistuttua alkavat kenttätestit. Kenttätestin alussa testausinsinööri asentaa ohjelmiston 2 - 3 harvesteriin. Tässä vaiheessa löytyvät versioristiriidat harvesterin PC:ssä olevien tiedostojen kanssa. Ongelmat korjaantuvat yleensä parilla korjauskierroksella. Samalla löydetään myös mahdollisia puutteita ohjelmiston toiminnallisuudessa. Testausvaihetta voidaan nimittää V-mallin mukaisesti järjestelmätestausvaiheeksi. Järjestelmätestausvaihe vie yleensä 2 - 3 viikkoa riippuen löydettyjen ohjelmistovirheiden määrästä ja vakavuudesta. Ohjelmistosta löytyy tässä vaiheessa yleensä suurimmat ohjelmistovirheet. Virheiden korjausten ja uudelleen testauksen jälkeen ohjelmistot jaetaan vielä 5 - 10 traktoriin, tarvittaessa huollon avustuksella. Vaihetta voidaan jo nimittää V-mallin mukaisesti hyväksymistestausvaiheeksi, vaikkakin ohjelmassa on vaiheen alussa tunnettuja puutteita.

Kenttätestien edetessä ohjelmistot ovat laajemmassa testissä, jossa niistä löytyy yleensä pienempiä puutteita, jotka korjataan ja testaan vaiheen aikana. Kun testausvaihetta on kulunut kaksi viikkoa, lähetetään asennuspaketti ulkomaan tytäryhtiöille testattavaksi. Tytäryhtiöissä on valittu vastuuhenkilöt, joiden kautta testien järjestely hoidetaan. Riippuen testattavien muutoksien kriittisyydestä ja määrästä testien laajuus vaihtelee tapauskohtaisesti. Muutosten koskien juuri kyseisen maan erikoisominaisuuksia testi voi olla hyvinkin laajaa. Sillä ominaisuutta ei välttämättä voi testata kuin kyseisessä kohdemaassa. Vastaavasti, jos muutokset eivät koske kyseisen maan omakohtaisia toimintoja, testaus voi painottua pelkästään käännösten oikeellisuuden tarkistuksiin.

Ulkomaan testien alkaessa käynnistyy myös Ponssen tuotannossa tapahtuva 0-sarjan testi. Testausasennuksen suorittavat tuotannollisen testauksen työntekijät saatujen ohjeiden mukaan. Testissä on mukana myös testausinsinööri. Tuotannollisen testauksen hyöty on opittu kantapään kautta, sillä tuotannon tarpeet ovat erityyppisiä kuin, mitä asiakkaiden tarpeet. Kummankin ryhmän tarpeet tulee ottaa huomioon ohjelmistoa kehitettäessä ja testatessa. Koko hyväksymistestausvaiheen kesto on 4 - 6 viikkoa, jonka jälkeen ohjelmisto on saatu korjattua ja todennettua ohjelmiston toimivuus.

Ohjelmiston läpäistyä kenttätestausvaiheen hyväksytysti ohjelmistosta tehdään ohjelmiston muutosilmoitus.

5 UUSI OHJELMISTO-RELEASEN TESTAUSKÄYTÄNTÖ

Opinnäytetyössä tuli kehittää työntilanneen yrityksen ohjelmiston testauskäytäntöä. Kehittämisen edellytyksenä on tunnistaa heikkoudet ja vahvuudet yrityksessä käytössä olevasta toimintamallista.

Nykyinen testaustoiminta aiheuttaa ongelmia aikataulussa pysymisessä, koska aktiivinen kenttätestaus sijoittuu release-jakson loppuun. Järjestelmätestaus ja hyväksymistestaus ovat yhdistyneenä tässä kenttätestausvaiheessa. Suurin osa testiin tulevista ohjelmistomuutoksista valmistuu testausvaiheen kynnyksellä, joten suurin osa ohjelmistomuutoksista valmistuu yhtä aikaa. Tämä aiheuttaa ohjelmistotestauksessa kiirettä release-jakson lopulla, koska testattavaa on yleensä aika runsaasti. Luonnollisesti muutostarpeita on paljon ja uusia versioita tulee testiin nopealla tahdilla. Koska suurin osa testaamisesta tapahtuu release-jakson lopulla, viivästyttävät löydetyt puutteet yleensä uuden ohjelmiston julkaisua.

Nykykäytännön mukainen kuuden kuukauden release-jakso on asiakkaiden tarpeet huomioon ottaen sopivan mittainen. Esimerkiksi metsäyhtiöt haluavat muutoksia ohjelmistoon usein ja kuusi kuukautta on heidän kannalta hyvä väli ohjelmistojen kehitykselle. Samoin tuotannon tarpeisiin tapahtuvat päivitykset onnistuvat hyvin tällä jaksotuksella.

Testausjakson lyhyys vaikeuttaa ulkomaan testiä. Testiin on varattu aikaa kuukausi ja osa kuukaudesta on jo kulunut varmistettaessa, että ohjelmat ovat sillä tasolla, että niitä voi lähettää ulkomaille testiin. Palautteen saaminen ulkomailta on myös hidasta,

sillä normaalitilanteessa palautteen saaminen kestää 2 - 3 viikkoa. Käytännössä ohjelmistopaketti voi olla jo julkaistu ennen kuin viimeiset palautteet tulevat. Vaarana voi olla, että palautteen perusteella voidaan joutua tekemään korjauksia ja julkistamaan uusi ohjelmistopaketti. Muutenkin ulkomaiden testien kanssa täytyy olla tarkkaan, heillä ei ole muuta tukea kuin puhelintuki ongelmatilanteessa. Ongelmaa on pyritty välttämään valitsemalla ulkomaantestiä suorittamaan kyseisen alueen pätevin/pätevimmät huoltomiehet.

Ulkomailla testiasennuksen suorittavat huoltomiehet normaalin työnsä ohessa. Tästä johtuu ongelma palautteen hitaudesta, jos heillä on paljon töitä juuri testien alkaessa, asennuksien aloitus viivästyy. Saman syyn takia heillä voi mennä paljon aikaa palautteen keräämisessä ja lähettämisessä Suomeen. Kuitenkin käytännön syistä testaus täytyy hoitaa paikallisen organisaation avustuksella, Suomesta ei voida lähettää testaajia joka paikkaan.

Ulkomaan testit ovat kuitenkin kaikesta huolimatta tärkeä osa ohjelmistotestausta, koska yrityksen vienti kasvaa ulkomaille kokoajan. Ulkomaisia asiakkaita pitää pystyä palvelemaan yhtä hyvin kuin kotimaankin asiakkaita, heitäkin koskevat ohjelmistovirheet tulee pystyä karsimaan minimiin. Tietyllä tavalla ulkomailla esiintyvät ohjelmistovirheet ovat vakavampia, mitä Suomessa esille tulevat. Ulkomailla ongelmana on yleensä heikompi tuotetuki metsäkoneille, joten ohjelmistovirheiden esiintyessä heillä ei ole välttämättä yhtä hyviä mahdollisuuksia ratkoa ongelmia kuin Suomessa. Suomessa tilannetta parantaa vielä ohjelmistotuotekehityksen läheisyys.

Testaussuunnitelmien puutteellisuus vaikeuttaa myös testiä. Jos uusista ominaisuuksista ei ole olemassa kattavaa listaa, joku ominaisuus voi jäädä testaamatta. Lisäksi testaajalle on ensiarvoisen tärkeä tietää, miten uuden ominaisuuden toiminnot sijoittuvat järjestelmässä. Mikä on kunkin ohjelmistomoduulin tehtävä järjestelmän toiminnallisuuden kannalta. Ohjelmistomoduulin tehtävä voi olla esimerkiksi pelkästään toimiminen tiedonvälittäjä, jolloin sen toiminta poikkeaa oleellisesti tilanteesta, että se toimisi järjestelmässä toimeenpanevana komponenttina.

Koska osa ohjelmistomuutoksista on lyhyen ajan testissä, jää ongelmaksi pienten, ei kriittisten, ohjelmistovirheiden löytyminen. Sama ongelma korostuu myös puutteelli-

sessa regressiotestauksessa eli testauksessa, joka tapahtuu korjauksen jälkeen. Ohjelmistoon jää puutteita, jotka tulevat esille vasta julkaisun yhteydessä. Vaikkakin ohjelmisto on liian lyhyen ajan testissä testi tapahtuu ohjelmiston kannalta oikeassa käyttöympäristössä. Tämän avulla ohjelmistotestaus on toiminut ja ohjelmistovirheet ovat tulleet paremmin esille todellisessa käyttöympäristössä, ammattilaisten käsittelyssä, mitä laboratorioon rakennetussa testausympäristössä. Ammattilaisilla tarkoitetaan tässä henkilöitä, jotka käyttävät Ponssen tuottamia ohjelmistoja työnään.

0-sarjan testaus tuotannossa on käytännössä hyväksi havaittu testaustapa. Tuotanto on yksi ohjelmistotuotekehityksen sisäinen asiakas ja heidänkin tarpeet tulee ottaa huomioon ohjelmistoja kehitettäessä. Tuotannolla on omat tarpeensa ja tuotantomäärien kasvaessa ohjelmiston asennettavuus ja metsäkoneen parametroinnin helppous/nopeus tuotannossa nousevat merkittäviksi tekijöiksi.

Toiminnanohjausjärjestelmään tallennettavat muutos- ja kokeilunseurantailmoitukset ja ohjelmistojen muutosilmoitukset ovat usein puutteellisia. Ilmoituksista ei ole silloin vastaavaa hyötyä. Kyseiset ilmoitukset ovat myös suunnittelijoiden mielestä hankalasti käytettävissä johtuen osaksi toiminnanohjausjärjestelmän ominaisuuksista, kuinka dokumentteja käsitellään.

5.1 Uusi ohjelmisto-releasen testausmalli

Nykyisessä mallissa ongelmaksi muodostuu aikataulujen pitämättömyys, sillä kentällä tapahtuva testaus keskittyy nykyään liian suppealle ajalle. Syynä tähän on testattavien moduulien testaustapa. Uusia ominaisuuksia ja tunnettujen ohjelmistovirheiden korjauksia ei testata vaiheittain, vaan moduuli toimitetaan testiin yleensä vasta sitten, kun se on täysin valmis. Ongelma korostuu selvimmin, jos kehitettävään/päivitettävään ohjelmistomoduulin on tullut paljon muutoksia. Juuri näiden ongelmien takia on kehitetty seuraavaksi esiteltävä malli. Ongelmiin saadaan ratkaisu jaettaessa ohjelmistomuutokset kahteen eri koko luokaan: pieneen ja suureen. Tällöin päästään tilanteeseen, jossa ohjelmisto on testattavissa kuukauden välein. Ohjelmistoon tullessa paljon muutoksia lisätään uudet ominaisuudet siten, että kuukau-

den välein on olemassa uutta materiaalia kentälle järjestelmätestaukseen. Ohjelmistovirheiden kohdalla yleensä riittää toiminen pienen mallin mukaan, mutta tilanteen vaatiessa suuren muutoksen mukaista mallia voidaan soveltaa. Samat mallit toimivat kehitettäessä täysin uusia ohjelmistokomponentteja tai päivitettäessä ohjelmistoon uusia ominaisuuksia.

Ohjelmisto-releasen suoritus jaetaan kuuteen kuukauden pituiseen testausjaksoon. Ensimmäiset neljä jaksoa ovat järjestelmätestausta, jolloin pyritään painottamaan testiä kotimaan testeihin. Viimeiset kaksi jaksoa ovat hyväksymistestausta, jonka aikana suoritetaan laajemmat testit kentällä ja 0-sarjan testaus tuotannossa. Ulkomaita koskevat muutokset tulesi tehdä kolmen ensimmäisen jakson aikana. Tämä mahdollistaa sen, että ulkomaita koskevia muutoksia pystytään testaamaan vähintään yhden kuukauden ajan kotimaassa. Käytännön sallimissa puitteissa olisi hyvä, jos olisi mahdollisuus lähettää ainakin osa ulkomaita koskevista muutoksista ohjelmistotestiin ulkomaille ennen viidennen testijakson alkamista. Mutta täydellinen paketti tulee kuitenkin lähettää testattavaksi viidennen jakson alussa. Suomessa tapahtuva testaus ei vastaa täysin ulkomailla tapahtuvaa testausta, sillä esimerkiksi Suomen metsälainsäädäntö poikkeaa oleellisesti monien muiden maiden säädännöistä.

Nykykäytännössä ulkomaille lähetyn ohjelmistopaketin testaussuunnitelmaa ei ole tehty. Tästä johtuen osa testattavista asioista voi jäädä testaamatta. Testaajaa ei yksinkertaisesti tiedä kaikkea mitä pitää testata. Hän vain lataa ohjelmaa laitteistoon ja jää odottamaan mitä tuleman pitää. Hän ei välttämättä myöskään tiedä ohjelmistoversion kriittisiä kohtia/tilanteita ja vika voi jäädä piilevänä ohjelmistoon. Tilanne korjaantuu laatimalla järjestelmätestausvaiheen aikana testaussuunnitelma, johon koostetaan ulkomaan testiä varten huomion arvoiset seikat.

Uudessa ohjelmisto-releasen testauskäytännössä on neljä eri järjestelmätestausjaksoa ja ohjelmistot tulee kehittää näiden jaksojen puitteissa. Kuitenkin tilanne, jossa kaikki järjestelmätestausjaksolle suunnitellut tehtävät eivät ole valmistuneet, on mahdollinen. Jokaisen jakson lopussa tulee katselmoida, missä vaiheessa työt ovat. Jos jakson työt katsotaan olevan valmiita seuraavaan järjestelmätestiin, kyseisen jakson ohjelmistotestaus voidaan aloittaa. Tilanne, jossa ohjelmisto ei ole järjestelmätestauskunnossa, voidaan jakaa kolmeen vaihtoehtoon. Ominaisuuksien ollessa ajalli-

sesti alle viikkoa myöhässä, asiaa palataan viikon kuluttua uudestaan ja päätetään järjestelmätestausjakson käynnistyksestä. Toinen vaihtoehto on todeta, että osa ominaisuuksista siirretään seuraavaan jaksoon ja testit aloitetaan alkuperäistä pienemmällä paketilla. Viimeisin vaihtoehto on todeta ohjelmiston olevan niin vajaa, että kyseistä järjestelmätestausjaksoa ei käynnistetä. Tällöin ominaisuudet testataan seuraavalla jaksolla. Ominaisuuksia voidaan joutua karsimaan, jotta pysytään ennalta sovitussa kuuden kuukauden release-jaksossa. Jos ohjelmistoon kehitettään tärkeimmät ominaisuudet ensin, tällöin karsintatilanteessa yleensä joudutaan jättämään vähemmän tärkeitä ominaisuuksia julkaisematta. Ohjelmistomuutosten valmistus järjestys pyrkii estämään tilanteen, jossa ohjelmiston tärkeimmät muutokset jäävät tekemättä kuuden kuukauden release-jakson puitteissa.

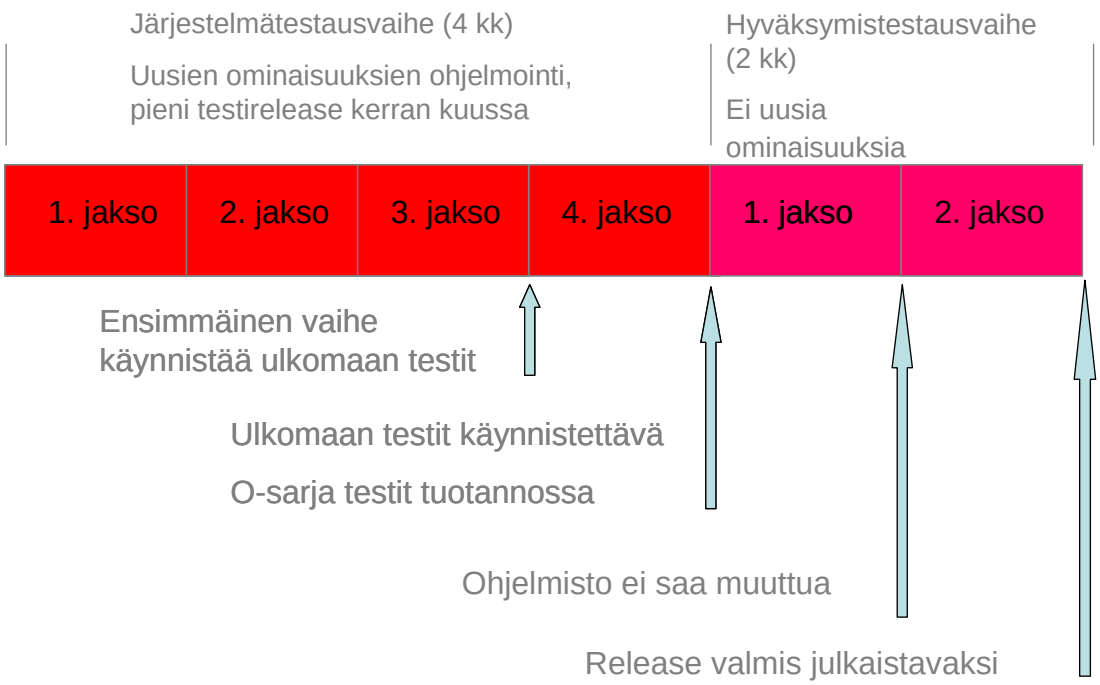
Ohjelmiston tilanteen katselmointi tulee tehdä myös siirryttäessä järjestelmätestauksesta hyväksymistestaukseen. Tarvittaessa hyväksymistestauksen alkua voidaan siirtää kaksi viikkoa, jotta halutut ohjelmistomuutokset saadaan tehtyä. Ohjelmiston ollessa edelleen puutteellinen hyväksymistestaus aloitetaan, niillä ominaisuuksilla mitä ohjelmistossa on sillä hetkellä. Tilanne vaati tarkastelun tehdäänkö puuttuvat ominaisuudet vai ei. Jos ominaisuudet päätetään tehdä, hyväksymistestausvaiheen pituus kasvaa niin kauan kuin tilanne vaatii, tavoite on kuitenkin kahden kuukauden hyväksymistestiaika. Kun ohjelmisto tulee tilanteeseen, jossa siihen on tehty halutut ominaisuudet, tilanteessa pidetään katselmointi ja päätetään onko ohjelmisto tilanteessa, jossa sen kuuluu olla vähintään kuukauden hyväksymistestissä. O-sarjan testi aloittaminen riippuu edellä mainituista muutoksista. Ulkomaan testit käynnistetään tarvittaessa uudestaan. Ohjelmiston saavuttua hyväksymistestausvaiheen loppuun suoritetaan katselmointi, jossa päätetään onko ohjelmisto julkaisukunnossa. Normaalitilanteessa hyväksymistestaus katselmoidaan ensimmäisen kerran hyväksymistestausjakson alussa, toisen kerran ensimmäisen hyväksymistestausjakson lopussa ja viimeisen kerran toisen hyväksymistestausjaksojen lopussa, jossa päätetään onko ohjelmisto-release julkaisukunnossa.

Testien jaksottamisella päästään huomattavasti ajallisesti pidempään testausaikaan, sillä osa ominaisuuksista on ollut jo mukana useassa järjestelmätestausjaksossa ja tällöin todennäköisyys virheen löytämiseen kasvaa. Vaikkakin ohjelmistovirhe havaitaan järjestelmätestauksen lopussa, aikaa regressio-testaukselle on vähintään kaksi

kuukautta. Ongelman tullessa esille jo aiemmin regressio-testausaika luonnollisesti pitenee edelleen.

Järjestelmätestaus tapahtuu neljän ensimmäisen kuukauden aikana, jonka jälkeen alkaa hyväksymistestausvaihe. Työssä kehitetyn uuden mallin rakenne on esitetty kuvassa 11. Hyväksymistestauksen aikana uusia ominaisuuksia ei saa lisätä ohjelmistopakettiin kevein perustein, vaan ohjelmiston sisältö tulee tässä vaiheessa vastata jo rakenteeltaan ja ominaisuuksiltaan ohjelmistoversioita, mikä on menossa tuotantoon. Hyväksymistestausjakson ensimmäisellä viikolla tulee suoritta muutama asennus huollon kanssa. Tässä vaiheessa eteen voi tulla joitakin asennusteknisiä puutteita, ei niinkään varsinaisia ohjelmistovirheitä. Korjaukset liittyvät yleensä asennuspaketin rakenteeseen, jotakin tiedostoja puuttuu tai esimerkiksi asennuspaketti täytyy muokata käyttöliittymän rekisteriasetusten takia.

Huollon kautta tapahtuvien asennusten kanssa tulee käynnistyä samanaikaisesti 0-sarjan testaus. Testivaiheessa tulee jo olla olemassa tarvittavat uudet ohjeet tuotannolle (tehdasasennusohje). Ohjeiden avulla tuotannon henkilökunta asentaa ohjelmiston ja tekee siihen liittyvää parametrointia. Ohjelmiston tärkeitä ominaisuuksia ovat tuotannolliset näkökohdat, ohjelmien asennettavuus ja käytettävyys, joiden tulee olla kunnossa jotta asennus on mahdollisimman helppo. 0-sarjan testissä käydään läpi asetuksia ja muita ohjelmistoon liittyviä tehdasarvoja. Tässäkin vaiheessa esiin tulevat ongelmat liittyvät yleensä asennusongelmiin. Nämä kaksi hyväksymistestin vaihetta tulee olla oleellisena osana release-jakson aikaista testausta. Näissä testeissä saadaan usein esille sellaisia puutteita, mitä järjestelmätestausvaiheessa aiemmin ei ole saatu. Järjestelmätestaus keskittyy pääasiassa juuri toiminnallisten virheiden korjaukseen, jolloin asennettavuustestit jäävät yleensä taka-alalle. Tässä vaiheessa testikoneet (järjestelmätestausvaiheen koneet) ovat käyneet läpi jo niin monta asennuskertaa, ettei asennusongelmia tule niissä enää usein esille. Joten ohjelmia tulee asentaa huollon kautta tämänkin takia ohjelmistotestauksen kannalta uusiin koneisiin.

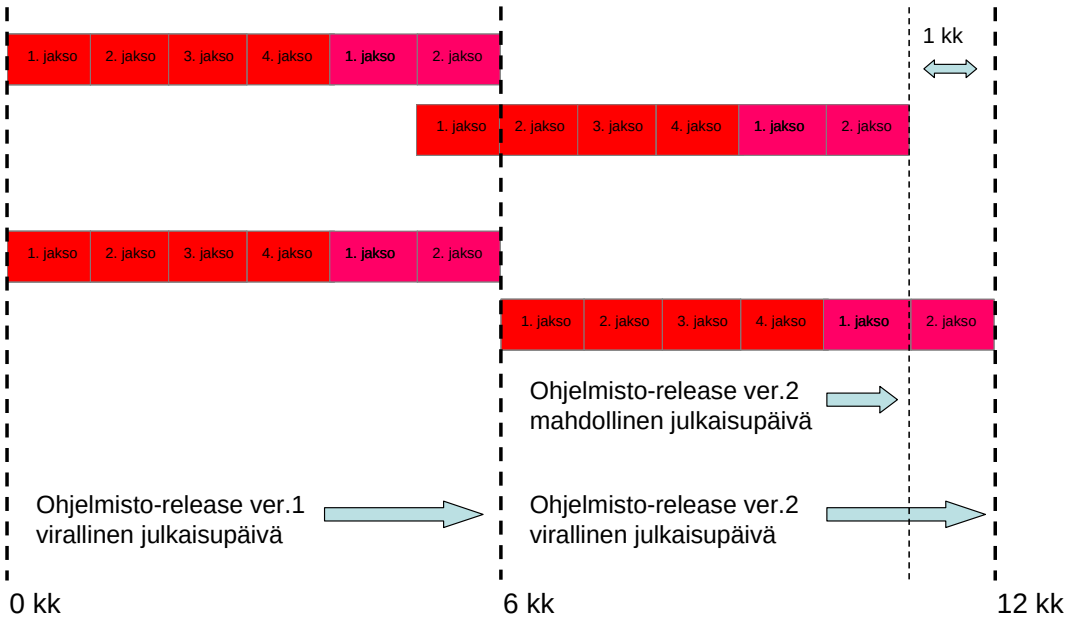


KUVA 11. Kuuden kuukauden release-jakson uusi testauskäytäntö

Kaikki ohjelmistopakettiin liittyvät muutokset, uudet ominaisuudet, tunnetut ohjelmistovirheiden korjaukset, asetukset ja asennuspaketti tulee olla valmiina viimeistään kuudennen jakson alussa. Tällöin hyväksymistestausaikaa on jäljellä kuukausi. Ongelmien tullessa esiin viimeisen jakson aikana, jatkotoimenpiteitä tulee harkita tapauskohtaisesti. Ongelman vakavuudesta riippuen, ongelmat korjataan ja muutokset testaan, mutta testiaikaa ei välttämättä jatketa. Jos löytynyt vika on suuri tai sen mahdollisia seurauksia ei tunneta riittävästi, silloin hyväksymistesti aloitetaan alusta. Tämä johtaa yleensä ohjelmiston julkistamisen viivästymiseen.

Ohjelmiston läpäistyä hyväksymistestausvaiheen hyväksytysti, ohjelmistosta tehdään ohjelmiston muutosilmoitus. Muutos- ja kokeilunseurantailmoitukset ja ohjelmistojen muutosilmoitus tallennetaan toiminnanohjausjärjestelmään nykykäytännön mukaisille lomakkeille.

Uuden releasejakson kehittäminen voidaan aloittaa jo toisen hyväksymistestijakson alussa, jos testeissä ei ole ilmennyt korjaustarpeita. Toinen hyväksymistestausjakson tavoite oli, että ohjelmistoon ei tehdä enää muutoksia. Joten uuden ohjelmisto-relesen kehittäminen voidaan aloittaa jo tässä vaiheessa. Tarvittaessa uuden ohjelmiston kehitys voi alkaa myös korjausten jälkeen toisen jakson lopussa. Uuden ohjelmisto-relasen aloitusvaihtoehdot ovat kuvassa 12. Vaikka uuden ohjelmiston aloittamisessa ei tapahdukkaan limitystä sen julkaisupäivä on kuusi kuukautta edellisen jälkeen. Limityksen avulla ohjelmisto-release voi valmistua kuukautta ennen sovittua julkaisupäivää.. Kuukusi voidaan käyttää tapauskohtaisesti hyväksymistestiin tai ohjelmisto voidaan julkaista kuukauden etuajassa. Ohjelmistotestauksen kannalta olisi paras vaihtoehto käyttää kuukausi hyväksymistestiin jatkamiseen



KUVA 12. Uuden ohjelmisto-releasen aloittaminen.

Kuitenkin on mahdollista, tai erittäin todennäköistä, että ohjelmistoon jää virheitä uuden käytännönkin mukaisesti toimittaessa. Ohjelmistovirheiden vakavuudesta riippuen korjauspaketin voi joutua tekemään ennen seuraavan ohjelmisto-releasen julkaisua. Tällöin kootaan tiedossa, korjausta kaipaavat ohjelmistovirheet yhteen korjauspakettiin. Korjauspaketti testataan vähintään niillä asiakkailla, joilla virhe on tullut esille. Lisäksi korjauspaketti tulisi asentaa, muutamaan laitteistoon, jossa ohjelmisto-

virheen ilmeneminen on todennäköistä. Tällaisia laitteistoja on yleensä löydettävissä, kun ohjelmistovirheen ilmenemisen syy on saatu selville. Korjauspaketin testaukselle on tavoitteena kuukauden testiaika, mutta johtuen ympäristön paineista, kuukausi voi olla liian pitkä aika. Yleensäkin korjauspaketille on hankala luoda tarkkoja ajallisia ja määrällisiä ohjeita. Ongelmatilanteessa täytyy pystyä soveltamaan maalaisjärkeä ja kaikkea ajansaatossa kerättyä tietoa laitteiston käyttäytymistä, jotta ongelmaan saadaan tilanteen huomioon ottaen paras ratkaisu. Korjauspaketissa ei saa olla uusia ominaisuuksia, pelkästään ohjelmistovirheiden korjauksia.

Ohjelmistovirheiden tullessa esiin ohjelmisto-releasen julkaisun jälkeen tarvittaessa voidaan huollon toimesta tapahtuva laitteistojen päivitys asiakkaille kieltää kokonaan. Myös tietyissä ongelman aiheuttavissa olosuhteissa toimivien laitteistojen päivitys voidaan kieltää, kunnes ohjelmistovirhe on saatu paikannettua ja korjattua.

Opinnäytetyössä kehitetyn testauskäytännön toimivuutta voidaan verrata edelliseen toimintamalliin tehtyjen korjauspakettien määrällä. Mitä vähemmän korjauspaketteja joudutaan tekemään, sitä paremmin ohjelmistotestauksessa on kyetty löytämään virheitä. Toinen mittari on ohjelmiston julkaisupäivän siirtyminen, jos sen katsotaan johtuneen ohjelmistovirheiden, myös määritysvirheiden, löytymisestä liian myöhään ja testaus aikaa on jouduttu sen takia jatkamaan. Mittarissa on otettava huomioon, missä määritysvirhe on tehty: onko se tehty Ponssen sisällä, vai onko sen tehnyt joku ulkopuolinen taho. Jos määritysvirhe on tehty Ponssen ulkopuolelle, eikä kukaan Ponssen ohjelmistotuotekehityksestä ole päässyt sitä katselmoimaan, virhettä ei voida pitää Ponssen ohjelmistotestauksen puutteellisuutena. Tilanne on mahdollinen, koska Ponssen toimittamat ohjelmistot toimivat rajapintojen yli monien eri ohjelmistotalojen tuotteiden kanssa.

5.2 Uuden testauskäytännön analysointi

Uusi edellä esitelty testauskäytäntö mahdollistaa hallitumman ohjelmiston testausrutiinien suorituksen. Uudessa testausmallissa varaudutaan uusien ominaisuuksien kohdalla useisiinkin korjauskierroksiin. Tämä takaa uusille ominaisuuksille pidemmän regressio-testausajan ja tarvittaessa korjausta voidaan jopa viivästyttää, jos ei olla varmoja vian aiheuttajasta. Uuden toimintamallin mukaan toimiessa ei enää välttämättä jouduta tilanteessa, että tarvitsisi tehdä jotakin, mutta ei tiedetä mitä pitää tehdä.

Huollon mukanaolo aktiivisesti hyväksymistestauksen aikana mahdollistaa sen, että testiin saadaan mukaan laajempi konekanta. Hyvänä puolena asiassa on se, että testiin saadaan monien kuljettajien erilaisia käyttötottumuksia ja laajempi kirjo alustoista (tietokoneita eri ohjelmisto variaatioilla), joihin ohjelmistot asennetaan. Ensimmäisissä asennuksissa huoltomiehillä tulisi aina olla apuna henkilö tuotekehityksestä tai testausinsinööri. Asennuksessa voi törmätä ongelmiin, joihin huoltomies ei törmää normaali asennuksien yhteydessä. Tällä toimenpiteellä on tarvittaessa saavutettavissa nopea ongelmanratkaisu, joka on sekä asiakkaan että huoltomiehen etu. Testausinsinöörin tai tuotekehityksen tuella huollolle vähennetään arkuutta, jolla uusia versioita aletaan jakaa kentälle. Ensimmäisten asennusten onnistumisen jälkeen huollolle tulee varmuus ohjelmiston ja asennuksen toimivuudesta. Tilanne nopeuttaa myös ohjelmiston nopeampaan leviämistä hyväksymistestin aikana, mikä taas kasvattaa todennäköisyyttä löytää virheitä ennen paketin varsinaista julkaisua.

Kriittinen tekijä testausmallissa on ulkomailta saatu palaute. Siellä havaituista ohjelmistovirheiistä saatu palaute tulee yleensä viiveellä. Palautteen saamista on hankala nopeuttaa, koska ulkomailla ohjelmistotestin suorittavat huoltomiehet. Heillä on myös hoidettava normaalipäivätyönsä, joten tässä vaiheessa voi tulla yllättäviä viiveitä heidän työtilanteesta riippuen. Vakavat virheet voivat olla mahdollisia, koska maakohtaisissa käyttötavoissa on eroja. Ongelmaa pienentää uusien ominaisuuksien aiennettu testaus ulkomailla, mutta se ei poista ongelmaa kuitenkaan kokonaan. Tilannetta voidaan parantaa kasvattamalla testausjaksojen määrää tai aikaistamalla hyväksymistestin aloitukset neljännelle testijaksolle. Tällaiset muutokset eivät kuiten-

kaan ole kokonaisuuden kannalta kannattavia, sillä ohjelmistopaketti olisi silloin ”valmiina” kolme kuukautta, mikä loisi painetta aloittaa ”normaali” päivitykset huollon kautta aikaisemmin. Kehitetyssä testauskäytännössä, ohjelmistopaketin hyväksymistestausvaihe on kaksi kuukautta. Aika on optimi ajatelleen testausaikaa ja tarpeita saada uusi ohjelmistopaketti julkistettua.

Uuden toimintamallin kaksi hyväksymistestijaksoa mahdollistavat tarvittaessa uuden 0-sarja testin suorituksen. Tämä on tarpeellinen ominaisuus, koska tuotannon sujuvuuden varmistaminen nousee laajentuvan yrityksen avaintehtäviksi. Suunnitelmallisesti läpi viety testaus mahdollistaa tavoitteen, jossa prosessi ohjaa testausta, eikä testaus prosessia.

Jotta uusi toimintamalli olisi mahdollinen, työt on kyettävä jaottelemaan yhden kuukauden suoritusjaksoihin (järjestelmätestausjakson pituus). Töiden jakaminen suoritusjaksoihin on uusi käytäntö Ponssella. Töiden jakaminen tulee ottaa käyttöön, jotta uusi release-jakson mukainen testaus voisi toimia. Töiden jakamisesta pienempiin kokonaisuuksiin on keskustelua jo aiemminkin tuotekehityksen sisällä, syynä keskusteluihin ovat olleet liian suuret virheet työmäärä arvioissa. Opinnäytetyössä saatu tulos vahvistaa käsitystä, jonka mukaan ohjelmointityöt tulee suorittaa pienemmissä osioissa. Tällä hetkellä jonkun ohjelmistomoduulin muutostyö voi kestää pisimmillään viisi kuukautta, jonka jälkeen se menee vasta testiin konttorin ulkopuolelle. Uudessa testauskäytännössä on neljä järjestelmätestausjaksoa, joiden aikana uudet ominaisuudet kehitetään. Tämä muutos vaatii jakamaan ohjelmistomuutokset ja testauksen eri kokoluokkiin. Uuden toimintamallin mukainen jako selitetään seuraavassa kappaleessa.

6 UUDEN MALLIN MUKAISET OHJELMISTOMUUTOKSIEN KOOT

Nykykäytännössä ohjelmisto kehitetään ja testataan vesiputousmallin mukaisesti. Kehitystyö tapahtuu hyvin pitkälti kerralla valmiiksi, eikä töitä jaotella paljoakaan pienempiin osiin. Työn kehitysvaihe entisessä käytännössä oli maksimissaan viisi kuukautta, jonka jälkeen sille suoritettiin kenttätesti. Uudessa, kehitetyssä mallissa työt tulee jakaa enintään kuukauden pituisiksi jaksoiksi johtuen kuukauden testaussykleistä (järjestelmätestausjakson pituus). Kuitenkin kestoltaan yli kuukauden työt ovat jo käytännössä aika laajoja. Silloin samaa käytäntöä ei kannata soveltaa työmäärältään pieniin töihin.

Testauksen kannalta ohjelmistomuutoksia on eri laajuisia ja kaikille ei sovi samat testauskäytännöt. Testauksen sekä käytännön toiminnan kannalta eri laajuisia ohjelmistomuutoksia tulisi kyetä jaottelemaan niin ajallisen keston kuin muidenkin ominaisuuksien mukaan. Tämä jaottelu mahdollistaa testausprosessin tehokkaan kehittämisen juuri tietynlaisen muutoksen ympärille. Jaottelu on siten tehty kahteen eri luokkaan: pieni ja suuri ohjelmistomuutos. Jako tehdään ajan perusteella, jonka työn oletetaan ajallisesti kestävän ja seuraavassa kappaleessa esiteltävän kriteerin perusteella.

Pieni ohjelmistomuutos tarkoittaa ajallisesti alle viiden työpäivää kestävää työtä. Aika pitää sisällään sekä ohjelmointityön että konttorilla tapahtuvan yksikkö- ja integrointitestauksen. Kalenterissa tämä viisi työpäivä sijoittuu yleensä 1 - 3 työviikon sisälle. Kalenteriajan suhde varsinaiseen työn kestoon selittyy sillä, että ohjelmoijat

ovat mukana useissa eri projekteissa samanaikaisesti ja hoitavat myös ylläpitotehtäviä.

Pienessä muutoksessa ei muokata kuin yhtä ohjelmistomoduulia. Pienen muutoksen ollessa kyseessä, ohjelmistomoduulin rajapinnan läpi toiseen ohjelmistomoduuliin liikkuva data ei saa oleellisesti muuttua. Esimerkki oleellisesta muutoksesta, ennen rajapinnan yli välitettiin arvoja alueella 0 – 255, vaikka ohjelmistorajapinta sallisi 16-bittisen luvun lähettämisen. Muutoksen jälkeen rajapinnan yli siirretään arvoja esim. 0 - 1000. Esimerkkinä oleva muutos on siis olennainen muutos, joten muutosta ei voida tehdä pienen muutoksen mallin mukaisesti. Pieni ohjelmistomuutos tehdään vesiputousmallin mukaisesti, kerralla valmiiksi.

Suuri ohjelmistomuutos kestää ajallisesti yli viisi työpäivää tai siinä muutettaan useampia ohjelmistomoduuleja tai siinä muutettaan moduulien välisiä rajapintoja. Jos työ on kalenteri ajassa yli yhden kuukauden, työ määritellään yhden kuukauden suoritusjaksoihin. Näitä kuukauden jaksoja on sitten niin monta, kuin ohjelmistolaajuuden määrä vaatii. Jaksojen tarkka määrää selviään vasta kun työstä tehdään tarkempi suunnitelma eli toimitaan opinnäytetyössä kehitetyn suuri ohjelmistomuutos mallin mukaisesti.

6.1 Nykykäytäntö pienen ohjelmistomuutoksen osalta

Ensimmäiseksi tarkastelun kohteeksi on valittu pieni ohjelmistomuutos. Seuraavissa kappaleissa toiminnot käydään läpi nykykäytännön mukaisesti sekä tehdään analyysi ja kehitysehdotus aiheesta. Nykykäytäntö ei tunne virallisesti termiä ”pieni ohjelmistomuutos”, joten tähän työhön on valittu esimerkiksi tapaus, joka vastaa lähtötasoltaan työssä kehitettävän pienen ohjelmistomuutoksen tunnusmerkkejä

Esimerkiksi valitun ohjelmamuutoksen teko kestää alle tunnin ja muutoksen yksikkötestauskaan ei vie tuntia enemmän aikaa. Lopputestaus (kenttätestaus) tehdään todellisessa käyttöympäristössä, metsäkoneessa. Kyseistä ohjelmamuutosta ei testata yksinään kenttätestissä, vaan samalla testataan myös muita ohjelmistossa tapahtuvia

muutoksia. Tämän tyyppisille muutoksille ei yleensä suoriteta kenttätestausta erikseen, vaan ominaisuus testataan jonkun muun testin yhteydessä. Muutokselle suoritetaan kenttätesti viimeistään testausvaiheen (kuva 9.) alussa, jolloin koko ohjelmistorelease on testikunnossa.

Kyseinen ohjelmistomuutos on valittu tarkasteltavaksi, koska se edustaa nopeasti tehtävää ja näennäisesti yksinkertaista muutosta. Lisäksi varsinaiseen tekniseen ongelmaan, vakiokierroskytkimen virhepainalluksen estämiseen, on määritelty ratkaisu valmiiksi tehtäväpyynnössä. Tällöin ohjelmoijalle on jäänyt vain ohjelmointityö eikä hänen tarvinnut ratkaista teknistä ongelmaa.

Tehtävä ohjelmistomuutos on vakiokierrostoiminnan aktivoimiseksi tehtävä 500 millisekunnin ohjelmallinen viive. Palautuvaa kytkintä tulee pitää aktiivisena 500 millisekuntia, jonka jälkeen toiminto aktivoituu. Alkuperäisellä ohjelmalogiikalla toiminto aktivoitui välittömästi, kun kytkintä painettiin. Ohjelmiston logiikkamuutoksen syynä on toiminnon liian herkkä aktivoituminen, esimerkiksi koneen kuljettajan takin hiha voi hipaista tahattomasti kytkintä ja aktivoida toiminnon. Toiminnon tilan vaihtuminen, kuljettajan sitä tiedostamatta, voi aiheuttaa vaaratilanteen. Sama viive toimii vakiokierrostoiminnossa molempiin suuntiin, sekä toiminnon päälle laitossa että toimintoa pois ottaessa.

Toiminnan aktivointi tarkoittaa tässä tapauksessa vakiokierroskytkimen tilan lähetystä dataväylää pitkin solmulta toiselle. Ohjelmamuutos tehdään penkkisolmuun, joka välittää viestin dataväylää pitkin ajovoimansiirron solmulle. Ajovoimansiirto askeltaa kytkimen komennosta kahden tilan väliä, vakiokierrokset päällä ja pois. Tämä tarkoitta lyhyesti sitä, että vakiokierrokset päällä kuljettajan ei tarvitse painaa kaasupoljinta, jotta dieselmoottori kierrokset ovat minimissään esimerkiksi 1700 rpm. Vastaavasti kun tila ei ole aktiivinen kone käy tyhjäkäyntikierroksia (900 rpm) kaasupoljin ylhäällä.

6.1.1 Nykykäytännön dokumentaatio pienestä ohjelmistomuutoksesta

Ohjelmamuutoksen tekopyyntö saapuu ohjelmoijalle sähköpostilla. Koska kyseessä on työmäärältään pieni ohjelmallinen muutos, muita määrityksiä ei tehdä. Lisäksi ohjelmamuutos ei ”koske” kuin yhtä solmua, varsinaista muutosta ei sen kummemmin katselmoida.

Ohjelmamuutoksen valmistettua, tehdään muutos- ja kokeilunseurantailmoituksen. Ilmoituksessa on kerrottu testattava ohjelmaversio ja tehty ohjelmamuutos.

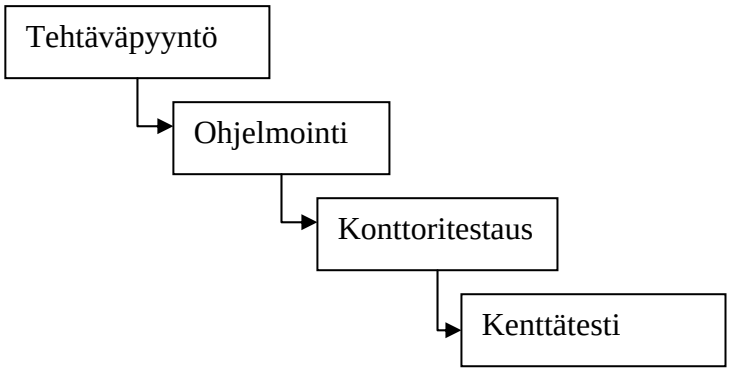
Ilmoituksessa selviää seuraavat asia:

Testattava ominaisuus:
Vakiokierrostoiminnan aktivoimiseksi tehtävä 500 millisekunnin ohjelmallinen viive, ennen kuin kytkimen painallus aktivoi toiminnon.

Muutoksen syy:
Syy muutokseen on estää palautuvan kytkimen tahattomat virhepainallukset.

6.1.2 Nykykäytännön mukainen testaus

Nykykäytännöllä ohjelman vaihejakomalli on suoraviivainen, vesiputousmallin mukainen, kuten kuvasta 13. nähdään. Testaus tehtiin yhdessä osiossa, vaihejakomallin lopussa. Vaadittu ohjelmamuutos tehtiin kerralla valmiiksi, jonka jälkeen se siirtyi konttoritestausvaiheeseen. Luonnollisesti moduulitesti alkoi rinnakkain itse ohjelmointiyön kanssa, jatkuen siitä kenttätestaukseen asti.



KUVA 13. Vanhan testausmallin mukainen käytäntö

Ohjelmoija teki ohjelmistomuutoksen ja testasi kyseisen muutoksen itsenäisesti omassa testiympäristössä. Uusi ohjelma toimi yksikkötestissä ja integrointitestissä, hyvä ja läpäisi konttoritestausvaiheen. Palautuvaa kytkintä oli pidettävä aktiivisena 500 ms, jonka jälkeen toiminto aktivoitui. Kytkin vapautettiin, toiminnon tila ei muutu, aktivointi oli edelleen voimassa. Aktivoinnin toteaminen tapahtui seuraamalla ajovoimansiirron lähettämää kierrosluku pyyntöä dieselille. Toiminnon ollessa aktiivinen pyyntö oli 1700, toiminnon ollessa ei-aktiivinen, pyyntö oli 900 rpm.

Tämän jälkeen ohjelmistosuunnittelija teki muutos- ja kokeilunseurantailmoituksen ja välitti ilmoituksen eteenpäin testausinsinöörille joka suoritti kenttätestauksen oikeassa ympäristössä.

Testausinsinööri testasi muutosta kenttätestissä muiden muutosten ohessa. Hän ”rämpytti” kytkintä nopeasti, jolloin mitään ei tapahtunut. Vastaavasti hitaalla kytkimen tilan vaihdolla ajovoimansiirto askelsi vakiokierroksiaan päälle ja pois oikein. Testausinsinööri totesi muutoksen toimivan ja hän kuittasi testin hyväksytysti muutos- ja kokeilunseurantailmoitukseen. Tällöin voidaan sanoa kenttätestausvaiheen oleen ohi. Sen hän jälkeen toimitti uuden ohjelmaversiota useampaan koneeseen testiin. Kukaan testikoneiden kuljettajista ei kommentoinut ominaisuutta tai puutteita siinä, jolloin muutos tuli seuraavaan julkaistavaan ohjelmistopakettiin.

Ohjelmistomuutoksen ohjelmoinnin konttori- ja kenttätestauksessa tapahtumat on selvitetty haastattelemalla ohjelmistomuutoksen suunnittelijaa ja testausinsinööriä.

Viimeisenä testausvaiheena ennen uuden ohjelmistopaketin julkaisua on aikanaan tapahtuva 0-sarjan testi tuotannossa ja ulkomaantestit. Uusi ohjelmistopaketti asennetaan tuotannossa muutamaan laitteistoon ja katsotaan löytyykö ohjelmistosta puutteita. Muutos käy läpi myös ulkomailla tapahtuvan testin ohjelmisto-releasen mukana, ei yksittäisenä ominaisuutena.

6.1.3 Nykykäytännön analysointi V-mallia vasten

Ohjelmistotestaus tulisi perustua V-mallin mukaiseen jakoon. Esimerkiksi työksi valittu ohjelmamuutos, tulee pystyä jakamaan osiin pohjautuen juuri näihin tietoihin. Näitä tietoja hyväksi käyttäen, voidaan työ jaotella eri työvaiheisiin. V-mallia noudattaen ensimmäinen testaussuunnitelma, järjestelmätestaussuunnitelma (arkkitehtuurisuunnitelma), tulee tehdä määrittelyvaiheen jälkeen. Seuraavana vaiheena on arkkitehtuurisuunnittelu (järjestelmäsuunnittelu), jonka jälkeen tulee tehdä integrointitestaussuunnitelma. Viimeisenä vaiheena ennen varsinaista ohjelmointia on moduulisuunnittelu ja moduulitestaussuunnitelma. (Haikala & Märijärvi, 2004)

Moduulisuunnitteluvaiheen jälkeen tapahtuu ohjelmointi ja V-mallin mukainen ensimmäinen varsinainen ohjelman toiminnallisuutta analysoiva testi, yksikkötestaus. (Haikala & Märijärvi, 2004) Testauksen suorittaa ohjelmistomoduulin ohjelmoija. Integrointitestauksen tekee ohjelmistomoduulin suunnittelija. Järjestelmätestauksen suorittaa testausinsinööri järjestelmätestauksen laitteiston todellisessa käyttöympäristössä. Hyväksymistestaus tapahtuu testausinsinöörin määrittelemissä metsäkoneissa. Tämän jälkeen tapahtuu tuotannossa 0-sarjan testi. Testausinsinööri toimii yhdyshenkilönä testiin osallistuvien asiakkaiden ja tuotekehityksen välillä

Koska ohjelmamuutos on pieni, valitun linja mukaan, dokumentaatioita syntyy vähän ominaisuuden kehityksen aikana. Koska mitään vaihetta ei dokumentoitu, ei minkään vaiheen dokumentaatiota voida katselmoida. Toiminnallisia – ja teknisiä määrityksiä

ei myöskään ole, minkä avulla voitaisiin luoda eri testaustasoille ohjeet, mitä järjestelmän tulee kulloinkin tehdä. Testausinsinööri saa sitten samat ohjeet (tehtäväpyyntö), minkä pohjalta ohjelmoijakin tekee ohjelmistomuutoksen.

Itse ohjelman testaamisessa nykykäytännön mukaan ei ole tapahtunut mitään virhettä, joten sitä ei tarvitse muuttaa. Moduulin suunnittelija suorittaa yksikkö- ja integrointitestauksen. Vaarana kuitenkin on, että ohjelmoija testaa muutoksen omalle ohjelmointityylilleen sopivana, jolloin ohjelma läpäisee testin, mutta ei toimi oikein. Käytännössä tällä hetkellä on kuitenkin mahdotonta suorittaa tämäntyyppistä moduulitestiä muulla tavalla johtuen organisaation koosta ja käytännön järjestelyistä. Näin pientä muutosta ei voi järkevästi testata irrallaan. Viive tehdään ajastettujen ohjelmakierrosten perusteella, 1 kierros 10 ms -> 50 kierrosta 500 millisekuntia. Keskeytysrutiineja ei tarvitse muuttaa ja se vähentää oleellisesti ohjelmistovirheen mahdollisuutta esimerkkitapauksessa.

Integrointitestausta on suoritettu ohjelman kehityksen aikana ja sen jälkeen ohjelmoijan toimesta. Järjestely on hyväksyttävä, koska moduulien rajapintoja ei ole muutettu, eikä rajapintojen yli liikkuvan datan sisältö ole muuttunut. Järjestelmätestaus ja hyväksymistestaus ovat tapahtuneet ohjelmiston kannalta oikeassa ympäristössä, metsäkoneessa. Testaussuunnitelman puute on ongelma, mutta sen korjaa testausinsinöörin tuntemus ympäristöstä, johon uusi ominaisuus on tehty.

6.1.4 Analyysi nykykäytössä olevasta toimintamallista

Kaikkia V-mallin mukaisia työvaiheita ei löydy mallista, johtuen ohjelmistomuutoksen koosta ja käytännön rajoituksista. Esimerkkinä on vaikka katselmointi jokaisen vaiheen jälkeen.

Ennen ohjelmiston luovuttamista asiakkaille on viimeisenä vaiheena testaus, vaihe jossa on yleensä aina kiire. Varsinkin toimittaessa vesiputousmallin mukaisesti, kun uusi ”pieni” ominaisuus on valmis ja se pitäisi saada nopeasti jakeluun, tulee helposti lipsumisia. Ominaisuus voi esimerkiksi valmistua testausvaiheen lopussa, ohjelmis-

torelaesen julkaisun kynnyksellä. Silloin ohjelma voi mennä kenttätestauksesta suoraan jakeluun ja ohittaa tuotannossa tapahtuvan 0-sarjan testauksen. Määrittelyjen puuttuminen on myöskin vakava puute ohjelmistonkehityksessä. Se näkyy testauksessa todella selkeästi, vaikka se ei ole varsinaisesti testausprosessin vika. Katselmointia ei voida suorittaa dokumentaatiosta, jolloin uuden ominaisuuden toiminta ei välttämättä vastaa tarkoitusta. Asiakkaan kannalta ohjelmistovirhe on ohjelmistovirhe, riippumatta kuka sen on tehnyt ja missä vaiheessa.

Testaussuunnitelman puuttuminen juontuu dokumentoinnin keveydestä. Uuden ominaisuuden pienuudesta johtuen varsinaiselle testaussuunnitelmalle ei nähdä tarvetta. Suunnitelman puuttuminen voi tulla esille itse tavassa, miten ominaisuutta testaan. Vaikkakin testaava ominaisuus on pieni, testausympäristö on monimutkainen. Tällöin pienikin muutos kokonaisuudessa voi aiheuttaa muualla ei toivottuja ominaisuuksia. Tätä riskiä voidaan vähentää katselmoimalla pienetkin ohjelmistomuutokset.

Toiminatavasta, jolla pieni ohjelmamuutos nykykäytännön mukaan testataan ja saatetaan tuotantoon, puuttuu oleellinen määrittely ja sen katselmointi (määrittelyn testaus). Muuta testaamiseen tai muutenkaan ohjelmistokehitykseen liittyvässä toimintatavassa ei ole huomauttamista.

6.1.5 Kehitetty pienen ohjelmistomuutoksen toimintamalli

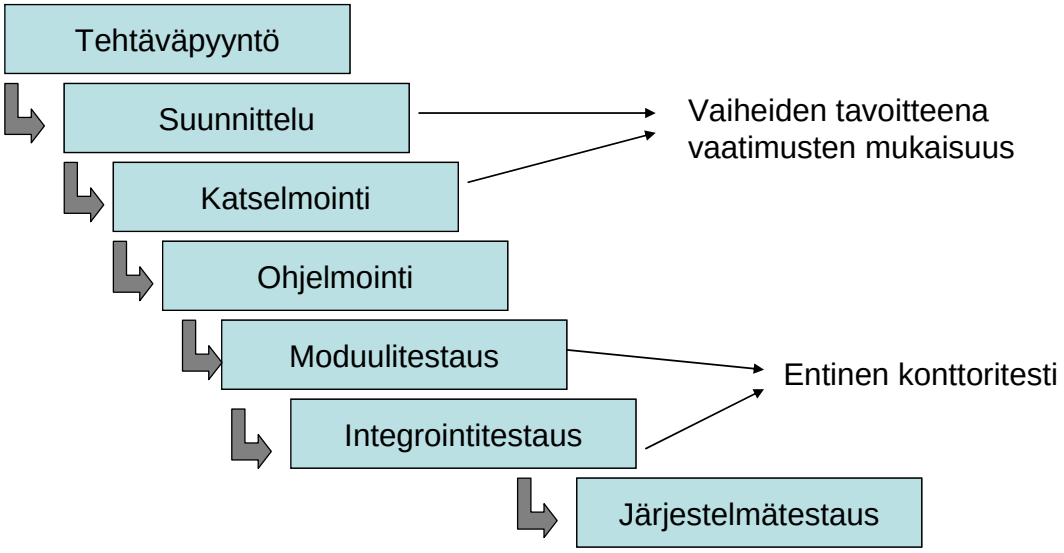
Pienen ohjelmistomuutoksen kriteerit:

Testauksen kannalta ja ohjelmoinnin kannalta pieni muutos on alle viidessä päivässä tehty ohjelmamuutos. Ohjelmistomuutoksen tulee tapahtua vain yhdessä moduulissa. Tällöin ohjelmistomoduulien rajapintoja ei rikota, eikä moduulien välisessä tiedonsiirrossa tapahdu oleellisia muutoksia. Datan siirtyminen moduulista toiseen ei ohjelman kiertokulun kannalta muutu määrällisesti eikä ajallisesti. Lisäksi siirrettävä arvojoukko ei muutu oleellisesti ohjelman suorituksen kannalta.

Pienen ohjelmistomuutoksen toimintamalli:

Suunnittelija, joka saa tehtäväpyynnön, tekee pienimuotoisen määrityksen uudesta ominaisuudesta. Tätä vaihetta kutsutaan suunnitteluvaiheeksi. Sen jälkeen hän lähettää määrityksen sen moduulin suunnittelijalle, johon hänen modifioimansa moduuli lähettää dataa. Tai vastaavasti määritys lähetetään dataa moduulin syöttävän moduulin suunnittelijalle. Kun tehtäväpyynnön saanut suunnittelija on saanut hyväksynnän, hän lähettää määrityksen testausinsinöörille. Kun kaikilta osapuolilta on saatu hyväksyntä, muutos on katselmoitu hyväksytysti. Kuten kuvasta 14. näkyy, katselmointi tulee suorittaa ennen varsinaista ohjelmointityötä. Suunnitteluvaiheeseen päätyvät työt on jo aiemmin määritetty tehtäväksi organisaatiossa, joten työn toteutumisen estävät ainoastaan tässä tai tämän vaiheen jälkeen tulevat ennalta tiedostamattomat ongelmat.

Moduulin suunnittelija suorittaa moduuli- ja integrointitestin, jonka jälkeen testausinsinööri suorittaa järjestelmätestauksen todellisessa käyttöympäristössä. Ennen järjestelmätestauksen alkua suunnittelija tekee muutos- ja kokeilunseurantailmoituksen, johon hän kirjaa testaussuunnitelman. Järjestelmätestauksen jälkeen ohjelma jää hyväksymistestiin vielä todelliseen käyttöympäristöön. Testausinsinööri tekee tarvittavan kommentit muutos- ja kokeilunseurantailmoitukseen, joka toimii virallisena tiedon välityskanavana ja ohjelmistovirheiden seuranta pöytäkirjana.



KUVA 14. Pienen ohjelmistomuutoksen toimintamalli

Jos järjestelmätestauksessa ilmenee ongelmia, tulee ongelmien korjauksen jälkeen testaus suorittaa uudestaan. Virheen korjauksen tehneen suunnittelijan tulee merkitä uuden version muutos- ja kokeilunseurantailmoitukseen ne tekijät, joihin tulee kiinnittää huomioita seuraavassa testausjaksossa. Järjestelmä- ja hyväksymistestaussuunnitelma on sama koska testausinsinööri valvoo testejä. Kommunikointi testausvaiheessa tapahtuu asiakkaan ja testausinsinöörin välillä, joten esimerkiksi testi asiakkaalle jaettavaan hyväksymistestaussuunnitelmaa ei tarvitse sen vuoksi tehdä.

Malli toimii järjestelmätestausvaiheen sisällä, joten mallissa ei oteta kantaa hyväksymistestin eri toimintoihin.

6.1.6 SWOT- analyysi pienen ohjelmistomuutoksen toimintamallista

SWOT-analyysissä on peilattu testausta osana ohjelmistonkehitystä. Testausta ei voida katsella irrallisena tehtävänä, vaan sen tulee nivoutua saumattomasti kokonaisuuteen. Uusi kehitetty pienen ohjelmistomuutoksen toimintamalli on esitelty ohjelmistotestauspalaverin yhteydessä. Esittelyn jälkeen siitä tehtiin SWOT-analyysi. Analyysin tekoon osallistuivat, tuotekehityspäällikkö, testausinsinööri ja suunnittelija.

Ryhmän yleinen mielipide:
Testausprosessin keveys on mielenkiintoinen näkökanta. Jos asiaa käsitellään pelkästään testauksen näkökannalta, keveys tarkoittaa, että voidaan testata nopeasti ja paljon. Mutta se ei vielä vie paljon eteenpäin, jos nopeudella ei saada hyötyä kokonaisuuden kannalta. Kokonaisuuden kannalta hyötyä saadaan siinä, että dokumentaatiota on haettu tarkoituksella keveäksi, mutta tarkoituksen mukaiseksi. Silloin dokumentointi tulee tehdyksi ja siitä on hyötyä monessa eri ohjelmistokehityksen vaiheessa, ei pelkästään testauksessa.

VAHVUUDET
Testaus tapahtuu nopeasti, suorittavia testiajoja vähän

Testattava ominaisuus kerralla valmiiksi -> järjestelmä –ja hyväksymistestissä ”tuorein” versio

HEIKKOUDET
Yksikkötestin suorittaa moduulin suunnittelija, sokeus omille virheille

MAHDOLLISUUDET
Testausprosessin keveys mahdollistaa tarvittaessa nopeankin ohjelmistokehityksen

UHAT
Määrityksistä ei ole tehty dokumentaatioita, joten ominaisuuden katselmointia ei ole ehkä suoritettu tarpeeksi laajasti, ohjelman ominaisuus on pieni, siitä huolimatta aina uuteen ominaisuuteen liittyy mahdollisuus tehdä ohjelmistovirheitä

6.2 Nykykäytäntö suuren ohjelmistomuutoksen osalta

Nykyinen testauskäytäntö suurempien ohjelmistomuutosten osalta on samanlainen, kuin mitä pienempien muutosten kohdalla. Nykykäytäntö ei tunne termiä ”suuri ohjelmistomuutos. Toimintamalli on suoraviivainen ja suuremmat muutokset tehdään kerralla valmiiksi. Erillisiä määrityksiä ja suunnitelmia ei tehdä laajempaan jakeluun. Tästä johtuen suurempiakaan ohjelmistomuutoksia ei katselmoida ja joidenkin ohjelmistomoduulien muutostarpeet huomataan vasta ”kalkkiviivoilla”.

Normaalit määrittelyvirheet liittyvät yleensä uuden toiminnan tuotantoon saattoon. Uutta ominaisuutta on testattu menestyksekkäästi ja tuotantoon vietäessä huomataan, että ominaisuutta ei saada ”asennettua” järjestelmään tuotannossa olevilla sovelluksilla. Tuotantoonottovaiheessa tehdään pikainen päätös ja valitaan helpoiten päivitettävä järjestelmän parametrointiin soveltuva ohjelma ja sitten uuden ominaisuuden vaatimat muutokset tehdään siihen. Uuteen ominaisuuteen liittyvät konfiguroinnit tehdään sovellukseen, mihin ne voidaan tehdä nopeimmin ja helpoimmin. Niitä ei tehdä siihen sovellukseen, mihin konfiguroinnit kuuluisivat parhaiten koko järjestelmän kannalta. Toimintamalli aiheuttaa ongelmia kokonaisuuden kannalta, sillä jär-

jestelmä käyttö ei ole enää välttämättä loogista, vaan asioita on ripoteltu sinne tänne. Edellä mainitun mukaiset ongelmat voidaan pääosin ratkaista määrittelyjen ja suunnitelmien testauksella, katselmoinnilla.

Ongelmana nykyisessä toimintamallissa on se, että yksi ohjelmistomuutos tehdään kerralla valmiiksi. Pitkissä ohjelmistoprojekteissa vesiputousmallin seuraaminen kerralla alusta loppuun johtaa ongelmiin testauksen aikataulutusta suunniteltaessa, koska moduuli täytyy olla täysin valmiina ennen kuin uutta ominaisuutta päästään testaamaan. Varsinainen ongelma ei tule esille, jos testauksen alla ei ole kuin yksi ohjelmistomoduuli. Kuitenkin käytännössä release-jaksolla on testin alla aina useita eri moduuleja. Toimintatapa aiheuttaa luonnollisesti kiireen testauksessa ja testauksesta tulee aivan turhaan väärin kuormitettu. Välillä on rauhallisempaa ja sitten työmäärät nousevat huippuun. Pahin ongelma tulee release-jakson lopussa, kun kaikki uudet ohjelmistomoduulit tulevat kerralla valmiiksi.

Seuraavissa kappaleissa esitellään uuden käytännön mukaiset toiminnot, jonka jälkeen suoritetaan esimerkkiohjelmistomuutos uuden testauskäytännön mukaisesti.

6.2.1 Suuren ohjelmistomuutoksen järjestelmätestaus

Ohjelmistotestauksen työmäärien vaihtelua voidaan tasata järkeistämällä ohjelmistomoduulien kehitysmallia, jolloin työt voidaan jakaa useampaan eri testattavaan vaiheeseen. Jaon mahdollistaa töistä tehtävät suunnitelmat. Suunnitelmien avulla voidaan selvittää suoritettavien tehtävien järjestyksen ja eri ohjelmistomoduulien muutoksien sidonnaisuus toisiinsa. Suunnitteluvaiheessa tehtävästä testaussuunnitelmasta voidaan määrittää toiminnallisuudet eli mitä kulloisessakin järjestelmätestausjaksossa testataan.

Järjestelmä- ja hyväksymistestaussuunnitelma on sama, koska testausinsinööri valvoo testejä. Kommunikointi testausvaiheessa tapahtuu asiakkaan ja testausinsinöörin välillä, joten esimerkiksi testiasiakkaalle jaettavaan hyväksymistestaussuunnitelmaa ei tarvitse sen vuoksi tehdä.

6.2.2 Suuren ohjelmistomuutoksen yksikkö- ja integrointitestaus

Suuren ohjelmistomuutoksen yhteydessä on yleensä mukana toiminnallisuuden suorittava ohjelmistomoduuli, jonka kautta toiminnallisuuden käyttöönotto ja konfigurointi suoritetaan sekä moduli, joka toimii näyttönä toiminnallisuudelle. Yleensä toiminnallisuuden arvoja esittäviä ohjelmistomoduuleja on useita.

Vesiputousmallia noudatettaessa esimerkiksi näyttöjen ohjelmointi jää yleensä viimeiseksi tehtäväksi. Se suoritetaan vasta sitten, kun näytölle tietoa välittävä moduuli on täysin valmis. Jos uuden arvon näyttöjä on vain yksi, ei ongelmaa välttämättä tule esille.

On yleistä, että samaa arvoa näyttää/tallentaa useampi kuin yksi sovellus. Tällöin kaikkien sovelluksien täytyy odottaa toiminnallisuuden valmistumista, ennen kuin niille päästään tekemään yksikkö- ja integrointitestausta.

Testauksen tehostamiseksi tämän tyyppisten ongelmien ratkaisuun on apuna tynkämoduuliajattelun hyväksikäyttö. Näytölle arvoja syöttävää moduulia muokataan sen verran, että se kykenee välittämään näytölle tarvittavat arvot. Tällöin muiden moduuleiden valmistuminen/testaaminen voidaan aloittaa riippumatta itse toiminnallisuuden valmistumisesta.

6.2.3 Suoritus kehitetyn mallin mukaisesti

Esimerkkinä mallin mukaisesta toiminnasta on sulautettuun järjestelmään tehtävä solmujen sarjanumeroiden ja solmun käynnissäoloajan tallennus. Työtä ei ole vielä tehty, joten sille voidaan määrittää entistä toimintamallia parempi malli.

Uuden toiminnallisuuden tuotteistaminen vaikuttaa useaan eri ohjelmistomoduuliin, arvot tallentavaan ohjelmaan sekä solmulla olevaan boot – sovellukseen, joka lukee ja välittää arvot dataväylään. Arvojen näyttämisen ja tallentamisen suorittavat useat eri moduulit.

Ensimmäinen vaihe työssä tehtäväpyynnön jälkeen on tehdä suunnitelma, jossa määritetään uuteen ominaisuuteen liittyvät uudet toiminnallisuudet ja ohjelmistomoduulit, joihin muutokset tehdään. Suunnitteluvaiheen jälkeen tehdään ensimmäinen testaus eli katselmoidaan tehty suunnitelma. Suunnitelman pohjalta tehdään testaussuunnitelma järjestelmätestausvaiheeseen, listataan ominaisuudet sekä mitä pitää testata ja miten. Suunnitteluvaiheeseen päätyvät työt on jo aiemmin määritetty tehtäväksi organisaatiossa, joten työn toteutumisen estävät ainoastaan tässä tai tämän vaiheen jälkeen tulevat ennalta tiedostamattomat ongelmat.

Varsinainen ohjelmointityö aloitetaan tekemällä boot-sovellukseen pieni muutos, jonka avulla boot–sovellus näennäisesti lähettää lukemansa arvot. Todellisuudessa välitettävät arvot on kovakoodattu boot- ohjelmaan, mutta jos arvot välittyvät ennalta sovitussa muodossa, testauksen kannalta tilanne on realistinen.

Arvoja näyttävien ohjelmien kehitys voidaan aloittaa aiemmin, mitä itse toiminnallisuuden ohjelmointi. Toimintamalli mahdollistaa arvoja näyttävien ohjelmistomoduulien tehokkaamman testaamisen, sillä sitä ei ole nyt sidottu itse varsinaiseen toiminnallisuuden kehittämiseen.

Boot-sovellukselle ja sen lähettämien arvojen näyttäville sovelluksille tehdään yksikkötestaus, jonka suorittaa jokaisen moduulin suunnittelija itsenäisesti. Yksikkötestien jälkeen valitaan vastuullinen suunnittelija, jonka vastuulla integrointitestauksen suorittaminen tulee olemaan. Integrointitestaussuunnitelma tehdään tehtäväpyynnössä ja suunnitteluvaiheessa saatujen tietojen perusteella.

Integrointitestauksen jälkeen katselmoidaan ominaisuus ja päätetään, onko uusi ominaisuus siirrettävissä järjestelmätestausvaiheeseen. Katselmoinnin aikana tehdään uuteen ominaisuuteen liittyviä korjauksia järjestelmä- ja hyväksymistestaussuunnitelmaan. Integrointitestausvaihe suoritetaan tarvittaessa uudestaan jos ominaisuudessa huomataan puutteita. Testausvaiheen jälkeen muutos- ja kokeilunseurantailmoitukseen kirjataan järjestelmätestausvaiheen testaussuunnitelmaan tarvittavia muutoksia. Järjestelmätestaus suoritetaan kentällä, tässä esimerkissä olevan uuden ominaisuuden testaussuunnitelma on osa kuukauden hyväksymistestijakson testaussuunni-

telmaan. Testausinsinööri tekee tarvittavan kommentit muutos- ja kokeilunseurantailmoitukseen, joka toimii virallisena tiedon välityskanavana ja ohjelmistovirheiden seurantapöytäkirjana.

6.2.4 Kehitetty suuren ohjelmistomuutoksen toimintamalli

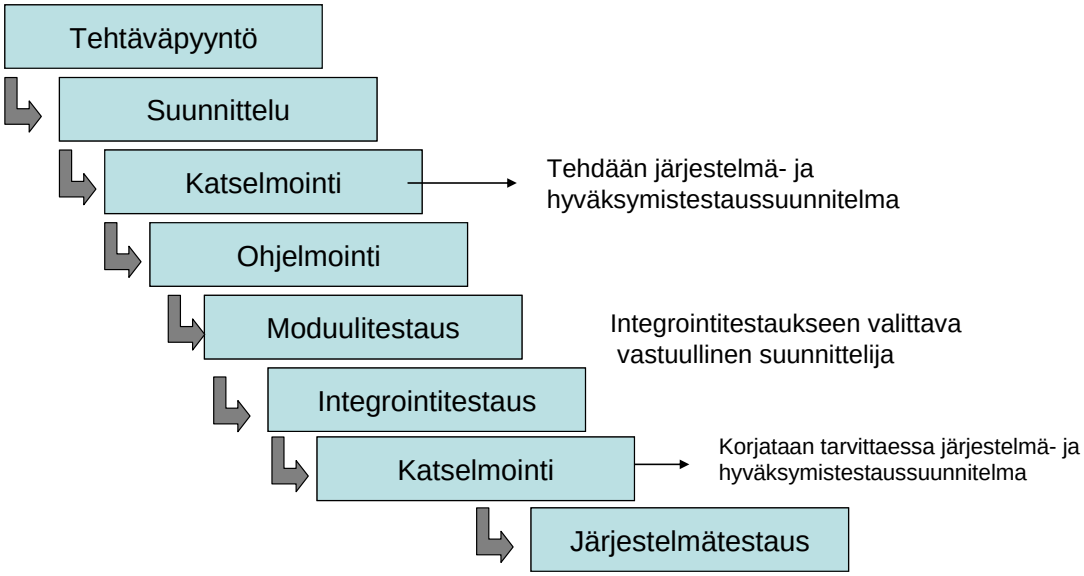
Suuren ohjelmistomuutoksen kriteerit:

Suuri ohjelmistomuutos kestää ajallisesti yli viisi työpäivää tai siinä muutettaan useampia ohjelmistomoduuleja tai muutettaan moduulien välisiä rajapintoja. Viiden työpäivä raja tarkoittaa työn suoritus aikaa, jos ohjelmoija tekee täyspainoisesti pelkästään kyseistä tehtävää. Jos työ on kalenteri ajassa yli yhden kuukauden, jaotellaan työ yhden kuukauden suoritusjaksoihin. Näiden kuukauden jaksoja on sitten niin monta, kuin ohjelmistolaajuuden määrä vaatii.

Suuren ohjelmistomuutoksen toimintamalli:

Suunnittelija, joka saa tehtäväpyynnön, tekee määrityksen uudesta ominaisuudesta. Tätä vaihetta kutsutaan suunnitteluvaiheeksi. Kuvasta 15. näkyy että suunnitteluvaiheen jälkeen määritys katselmoidaan ja sen pohjalta tehdään järjestelmä- ja hyväksymistestaussuunnitelma. Ohjelmointivaiheen jälkeen kukin suorittaa yksikkötestauksen itsenäisesti. Vaiheen jälkeen valitaan vastuullinen suunnittelija, joka organisoi integrointitestauksen.

Testauksen jälkeen saadut tulokset katselmoidaan ja tarvittaessa korjataan järjestelmätestausvaiheen testaussuunnitelmaa. Ennen järjestelmätestauksen alkua, integrointitestauksesta vastaava suunnittelija tekee muutos- ja kokeilunseurantailmoituksen, johon hän kirjaa testaussuunnitelman. Järjestelmätestauksen jälkeen ohjelma jää hyväksymistestiin vielä todelliseen käyttöympäristöön.



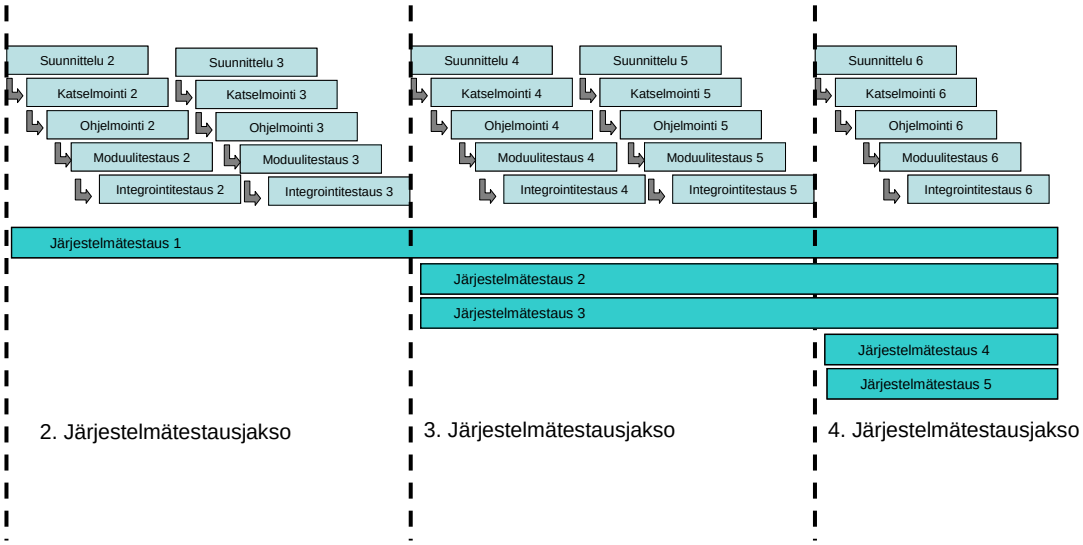
KUVA 15. Suuren ohjelmistomuutoksen toimintamalli

Jos järjestelmätestauksessa ilmenee ongelmia, tulee ongelmien korjauksen jälkeen testaus suorittaa uudestaan. Virheen korjauksen tehneen suunnittelijan tulee merkitä uuden version muutos- ja kokeilunseurantailmoitukseen ne tekijät, joihin tulee kiinnittää huomioita seuraavassa järjestelmätestausjaksossa. Malli toimii järjestelmätestausvaiheen sisällä, joten mallissa ei oteta kantaa hyväksymistestin eri toimintoihin. Testausinsinööri tekee tarvittavan kommentit muutos- ja kokeilunseurantailmoitukseen, joka toimii virallisena tiedonvälityskanavana ja ohjelmistovirheiden seurantapöytäkirjana.

6.3 Ohjelmistomuutokset järjestelmätestausjaksoissa

Kuukauden mittainen järjestelmätestausjakso pitää sisällään useita eri tehtäviä, pienen tai suuren ohjelmistomuutoksen mukaan tehtäviä ohjelmistomuutoksia. Koko release-ajattelun kannalta on tärkeintä pystyä määrittelemään annettujen tehtävien kestot. Kun käydään läpi esimerkkinä olevaa kuvan 16. mukaisia järjestelmätestausvaiheita, huomio täytyy kohdistaa tehtäväpyyntöihin, joita ei näy enää vaiheen suori-

tuksissa. Tehtäväpyynnöt ovat esimäärittelyjä työlle, jotka kuuluu tehdä ennen varsinaista suunnitteluvaihetta. Esimäärittelyn avulla töille valitaan, suuren tai pienen ohjelmistomuutoksen mukainen toimintamalli. Tämä esisuunnitelman perusteella, töiden varsinainen määrittely (suunnitteluvaihe), joka katselmoidaan organisaatiossa.



KUVA 16. Järjestelmätestausjaksojen sisältö

Esimerkiksi valitussa release-jaksossa (kuva 16.) on jo ensimmäinen järjestelmätestausjakso suoritettu. Ensimmäiseen järjestelmätestausjakson aikana on ohjelmoitu tehtävä 1 ja suoritettu sille vaadittavat testit (moduuli ja integrointi), jotka ohjelmisto on jo läpäissyt. Toisen järjestelmäjakson alkaessa tehtävässä 1 vaaditut ohjelmamuutokset/ohjelmisto siirtyy järjestelmätestausvaiheeseen ja pysyy järjestelmätestissä koko järjestelmätestausvaiheen ajan.

Toisessa, kolmannessa ja neljännessä (ei näy kokonaan kuvassa) järjestelmätestausjaksossa suunnitellaan ja ohjelmoidaan aiemmin tehtyjen esiselvitysten perusteella tehtävät 2 - 5. Tehtävät siirtyvät seuraavassa vaiheessa (hyväksynnän jälkeen) järjestelmätestausvaiheeseen ja sitä kautta hyväksymistestausvaiheeseen.

Suuren ohjelmistomuutoksen ollessa kyseessä, suunnitteluvaihe voi olla niin laaja, että suunnittelu voidaan tehdä ennen kuin tehtävä siirtyy suoritettavaksi järjestelmäjaksossa. Esisuunnitteluvaihe ajoitetaan edellisen release-jakson viimeiseen hyväk-

symistestausjaksoon. Tällöin uuden release-jakson käynnistyttyä voidaan ohjelmointi aloittaa välittömästi.

Tämän opinnäytetyön aihe on ohjelmistotestauksen kehittäminen. Työn aikana tuli esille kuitenkin ongelma nykyisessä ohjelmistokehitysmallissa. Nykyisen mallin mukaan toimittaessa ohjelmistotestausta oli käytännössä mahdotonta toteuttaa johtuen määrittelyjen puutteesta. Myöskään nykykäytäntö ei tuntenut tapaa, jolla työt voidaan suorittaa kuukauden jaksoissa, järjestelmätestausjaksojen mukaisessa syklissä. Ohjelmistotuotekehityksessä on ollut ongelmia aiemminkin nykykäytännön kanssa, joten päätös uusien mallien kehityksestä oli helppo tehdä. Tämän syyn takia opinnäytetyössä täytyi suunnitella käytäntö, kuinka ohjelmistomuutokset kyetään vaiheistamaan eri järjestelmätestausjaksoihin. Jotta ohjelmistomuutokset ja uudet ominaisuudet saadaan jaoteltua kuukauden jaksoihin, työt täytyy jakaa eri kokoluokkiin (suuri ja pieni), ja luoda niiden suoritukseen toimintamallit. Näiden rutiinien loppuun asti hiominen ja resurssien jako, esisuunnitellun ja suunnittelun toimintamallit, ohjelmoijien tuottavuuden laskenta, ohjelmoijien ylläpito kuorman määrittäminen järjestelmäjaksoihin aikataulutuksiin, dokumenttien tallennus ja muut tarvittavat toimenpiteet on jätetty pois tästä opinnäytetyöstä. Tämä opinnäytetyö rajattu on ohjelmistotestaukseen ja koska yllämainittujen tehtävien kehittämisessä olisi aihetta erilliseen opinnäytetyöhön, ne eivät ole mukaan tässä opinnäytetyössä.

7 YKSIKKÖTESTAUS FUNKTIO TASOLLA

Valitaanpa ohjelmistonkehitysmalliksi pieni tai suuri, koostuu ohjelmisto aina käytännössä funktioista. Testauksen kannalta parempaan lopputulokseen päästään, mitä rajatumpaa ja pienempää kokonaisuutta kyetään järjestelmällisesti testaamaan. Tällöin ohjelman testaaminen on paras aloittaa suorittamalla testit alkaen funktio tasolta. Testaus alkaa kun ohjelmamoduulin on tullut ohjelmoijan mielestä valmiiksi. Sitten hän alkaa tarkastella ohjelmakoodia läpi funktio kerrallaan. Seuraavissa kappaleissa on hieman mukana teoria, koska selkeyden kannalta kyseinen teoria on parempi tuoda esille tässä vaiheessa, mitä muun teoria yhteydessä työn alussa.

Ensimmäinen testaus tulee suorittaa staattisena testauksena, jolloin käydään läpi funktion nimen ja funktioissa käytettyjen nimien oikeellisuus. Nimien tulee olla kuvaavia funktion toiminnallisuutta ajatellen. Muuttujien nimestä pitää pystyä myös päättelemään muuttujan datatyyppi. Samalla tulee tarkistaa mahdollisten lisäkommenttien tarve. Testausvaiheessa tulee olla esillä moduulista tehty määrittely, jos sellainen on tehty. Testausvaiheessa tulisi myös tarkistaa, että ohjelamakoodissa, kommenteissa ja määrittelyissä käytetyt nimitykset ovat yhtenäisiä.

Toinen testausvaihe on edelleen staattista testausta. Käydään läpi lausekattavuus eli voiko ohjelma saavuttaa jokaisen ohjelmanlauseen suorituksen aikana. (Satukangas, 2003) Testausvaiheessa tutkitaan löytyykö valintatilanteissa jokaiselle valinnalla toimiva vaihtoehto ja onko tunnettujen ja ennakoitavissa olevien virhetilanteiden varalta olemassa virheenkäsittely rutiineja. Käytännössä tämä tarkoittaa esimerkiksi sitä, että onko switch-case rakenteessa default-vaihtoehtoa. Tässä vaiheessa tarkiste-

taan myös silmukoissa olevien rakenteiden oikeellisuus, esimerkiksi voiko for - silmukan aikana käytettävien taulukoiden kokorajat ylittyä. Globaalien muuttujien tilaan kiinnitettään myös huomiota. Tällöin katsotaan myös, että tuleeko tahattomia sijoituksia muuttujiin, vaikka muuttujaa käytetään esim. vertailuarvona.

Seuraava kohta ei ole varsinaista testausta, mutta kuuluu kuitenkin oleellisena osana ohjelmisto tuotekehitysvaiheeseen. Ponssen ohjelmistotuotekehityksessä on jonkun aikaa mietitty ohjelmistokirjaston rakentamista. Hanketta ei ole vielä käynnistetty, eikä mietitty sen pidemmälle. Kuitenkin ohjelmointivaiheessa suunnittelijalla on jo käsitys voiko työn alla oleva funktio olla yleispätevä ja täten siirrettävissä myöhemmin kirjastoon. Merkintä on helpompi laittaa funktioon ohjelmointivaiheessa, mitä erotella funktiota kirjastofunktioiksi myöhemmin. Funktion merkitseminen kirjastofunktioksi, ei palvele vielä tämän päivän tarpeita, mutta helpottaa tulevaisuuden tuotekehitystä, jos/kun kirjasto ajattelu otetaan käyttöön.

Seuraava testausvaihe tulee olemaan jo dynaamista testausta. Funktion testaukseen tarvitaan funktiolle tynkäpeti, jonka avulla funktiolle voidaan antaa tarvittavat syötteet. Syötteiden avulla pyritään tarkastamaan mm. funktion kannalta kriittisimmät kohdat (raja-arvot) sekä todentamaan virheenkäsittelijöiden toimivuus. Syöte voi olla funktioon välitettävä parametri, funktion sisäinen muuttuja tai globaali muuttuja.

Kolmas testausvaihe on jo dynaamista testausta. Se tullaan aloittamaan ekvivalenssi testauksella. Rajatulla syötteiden määrällä pyritään testaamaan koko syöteavaruus. (Virkkunen, 2002) Syötteet pyritään jakamaan kolmeen eri ryhmään. Ensimmäinen ryhmä ovat syötteet, jotka eivät vaikuta funktion käsittelemiin muuttujiin. Toisen ryhmän muodostavat syötteet, joiden muutos vaikuttaa muuttujiin, mutta ohjelman suorittama polku on aina sama. Polku on siis sama syöteryhmän sisällä, saatu tulos vain muuttuu. Kolmas syöteryhmä sisältää muuttujat, joilla ohjelman suorituspolku muuttuu. Tämä ryhmä on kaikkein vaativin testattava, sillä polku voi haarautua useasta eri kohdasta.

Ensimmäisen ryhmän syötteillä on tarkoitus testata funktion virheenkäsittelijöitä, jolloin testaustekniikkana on negatiivinen testaus. Syötteiden avulla funktion toimintaa ohjataan siten, että funktion joutuu vikatilaan.

7.1 Kehitetyn funktiotestauksen suorittaminen

Funktiotestauksen kannalta paras vaihtoehto on suorittaa testaus välittömästi funktion ohjelmoinnin tai muutoksen jälkeen. Funktiotestaus tulee olla normaali toimenpide, joka tapahtuu aina ohjelmoinnin jälkeen. Tietyt testausrutiinit tulee suorittaa aina, joten rutiinit eivät voi olla monimutkaisia tai muuten testit jäävät tekemättä. Yksinkertaisimmillaan funktiotestaus on tarkistuslistan läpikäymistä, jolloin ohjelmoija huomaa funktion olevan puutteellinen. Testauskysymykset tulee laittaa jokaisen funktion yläpuolella tulevaan osioon. Osioissa on tätä ennen ollut pelkästään kuvaus funktion toiminnasta, kommentteja muutoksista sekä viimeksi funktioita muokanneen ohjelmoijan nimi. Testausraportti tule olemaan osa ohjelmakoodia ja silloin se on sähköisessä muodossa ja aina helposti saatavilla. Funktiotestauksessa minimivaatimus on staattinen testausosa. Dynaamisen funktio tason testaus voi olla helpompi suorittaa useamman eri funktion avulla, jolloin vältytään liiallisilta tynkäpetien ohjelmoinnilta.

Funktion ohjelmoinnin jälkeen kysymyksiin vastaaminen muistuttaa hyvin pienistä perusasioista. Tavoitteena on löytää virheet ohjelmasta sekä löytää ohjelmasta mahdolliset debuggaukseen liittyvät ylimääräiset muuttujat, joita ei ole otettu pois käytöstä esimerkiksi ehdollisella kääntämisellä. Dynaamisen testauksen suorittaminen on suotavaa, jos se vain on mahdollista. Tämä vaihe vie enemmän aikaa mitä staattinen testaus, mutta vastaavasti siinä löydetyt ohjelmistovirheet ovat jo oikeita ohjelmiston suoritusta haittaavia ongelmia. Funktiotestauksen on tarkoitus olla nopea toimenpide, ei tuntikausia vievä operaatio.

Alla ohjelmakoodin funktion yläpuolelle tuleva tarkistuslista:

Funktion nimi:
Funktion kuvaus:
Funktioon välitettävät parametri:
Funktion muutoshistoria ja tekijä:

Staattinen testaus, ohjelmakoodia lukemalla

Tarkistettu muuttujien nimien oikeellisuus ja kommentoinnit:
Tarkistettu virheenkäsittelijöiden riittävyys:
Tarkistettu funktion lausekattavuus:
Onko funktio kirjastofunktio:

Dynaaminen testaus, ohjelmakoodia ajamalla

1. ryhmän syötteet, tarkistettu virheenkäsittelijöiden riittävyys:
2. ryhmän syötteet, tarkistettu yksihaarainen polku:
3, ryhmän syötteet, tarkistettu polkukattavuudet:

7.2 Kehitetyn funktiotestauksen analysointi

Testauksen pilkkominen aina pienempiin ja pienpiin osiin parantaa ohjelman testattavuutta. Pienempiä kokonaisuuksia käsiteltäessä virheet ovat selkeämmin havaittavissa. Ohjelmatekniset virheet löytyvät parhaiten pienestä testikohteesta. Kuitenkin väärien määritysten löytyminen on funktiotasolla melkein mahdotonta, mutta jo yksikkötestausta suoritettaessa määrittelyvirheet on jo löydettävissä.

Funktiotestauksen tulee olla mahdollisimman nopeata, jotta se tulisi tehtyä hyvin. Tässäkin näkyy se tosiasia, että mitä yksinkertaisempi ja vähemmän koodia testattava funktio sisältää sitä helpompi se on hallitusti testata. Dynaamisen testauksen suoritus

on hankalin vaihe funktiotason testauksessa. Funktion ollessa yksinkertainen sekin on kuitenkin järjestettävissä yksinkertaisilla järjestelyillä.

Yksikkötestausta ja varsinkin funktiotestausta mietittäessä törmätään selkeästi siihen tosiasiaan, että yksinkertainen on kaunista. Funktioiden ja ohjelmistomoduulien ollessa selkeitä kokonaisuuksia, paranee niiden testattavuus ja uudelleen käytettävyys. Mitä enemmän ohjelmistossa on mukana uudelleenkäytettyä koodia, sitä suuremmalla todennäköisyydellä saatu lopputulos on myös luotettavampi. Sillä useasti käytetyt koodirivit ovat käyneet läpi monet testit ja ohjelmistovirheet ovat karisseet suurimmaksi osaksi matkan varrelle.

8 KEHITETTYJEN MALLIEN DOKUMENTAATIO

Oleellinen osa ohjelmistotuotekehitystä ja testausta on ohjelmistonkehityksen aikana tuotettu dokumentaatio. Dokumentaation avulla voidaan seurata missä vaiheessa release-jakso ja sen erilliset tehtävät (ohjelmistomuutokset) ovat menossa. Ohjelmistomuutoksessa ja release-jakson aikana tapahtuvia virheitä (toimintavirheitä ei ohjelmistovirheitä) voidaan jäljittää dokumentaation avulla. Palaamalla dokumentaatiossa ajassa taaksepäin, tällöin on olemassa mahdollisuus, että mahdolliset virheen aiheuttajat löydetään. Tällöin voidaan mahdollisesti oppia virheistä ja pyrkiä välttämään niitä seuraavan kerran.

Opinnäyte työn edetessä, opinnäytetyön tilaajaa ilmoitti kantansa kuinka dokumentaatioita tulee käsitellä opinnäytetyössä. Valmiita dokumenttipohjia opinnäytetyössä ei tarvitse tehdä, tärkeintä on tiedostaa mitä dokumentteja tuotekehitysvaiheen ja testausvaiheen aikana tulee tallentaa. Seuraavissa kappaleissa määritettään, mitä ja miten dokumentaation täytyy tuottaa ohjelmiston release-jakson eri vaiheissa. Dokumentaatio tallennetaan Ponssen tietojärjestelmään, ei pelkästään suunnittelijoiden omille tietokoneille.

8.1 Pieni ohjelmistomuutos

Jokaisesta uudesta ominaisuudesta ja ohjelmistovirheen korjauksesta on oltava tallennettuna tehtäväpyyntö, esimäärittely dokumentti. Ohjelmistomuutoksen täyttäessä pienen ohjelmistomuutoksen kriteerit, esimäärittelystä ei tarvitse välttämättä olla

erillistä lomaketta, vaan riittää, että sähköposti(t), joka(jotka) liittyvät muutokseen tallennetaan.

Jokaisesta uudesta ominaisuudesta ja ohjelmistovirheen korjauksesta on oltava tallennettuna suunnitelma, määrittely dokumentti. Ohjelmistomuutoksen täyttäessä pienen ohjelmistomuutoksen kriteerit, määrittelystä ei tarvitse välttämättä olla erillistä lomaketta, vaan riittää että, sähköposti(t), joka(jotka) liittyvät muutokseen tallennetaan.

Jokaisesta uudesta ominaisuudesta ja ohjelmistovirheen korjauksesta on suoritettava määrittelyvaiheen katselmointi, määrittelyn katselmointipöytäkirja. Ohjelmistomuutoksen täyttäessä pienen ohjelmistomuutoksen kriteerit, määrittelystä ei tarvitse välttämättä olla erillistä lomaketta, vaan riittää että, sähköposti(t), joka(jotka) liittyvät muutokseen tallennetaan.

Jokaisesta uudesta ominaisuudesta ja ohjelmistovirheen korjauksesta on tehtävä muutos- ja kokeilunseurantailmoitus. Ilmoituksessa on järjestelmätestausohje uudelle ominaisuudelle. Testausinsinööri kommentoi ja hyväksyy/hylkää muutoksen kyseisellä ilmoituksella.

Pienessä ohjelmistomuutoksessa tallennettava dokumentaatio: esimäärittelyn dokumentaatio, määrittelyn dokumentaatio, määrittelyn katselmointipöytäkirja(ei pakollinen), muutos- ja kokeilunseurantailmoitus.

8.2 Suuri ohjelmistomuutos

Jokaisesta uudesta ominaisuudesta ja ohjelmistovirheen korjauksesta on oltava tallennettuna tehtäväpyyntö, esimäärittely dokumentti. Ohjelmistomuutoksen täyttäessä suuren ohjelmistomuutoksen kriteerit, esimäärittelystä tulee olla erillinen dokumentti ja esimäärittelyvaiheessa on pidettävää esimäärittelypalaveri, pelkkä sähköposti ei riitä kommunikointivälineeksi.

Esimäärittelyn suorittava ryhmä päättää katselmoidaanko tehtävä vai ei. Esimäärittelyn katselmoinnista kommentoidaan tarvittaessa esimäärittely dokumenttiin. Kommunikointivälineeksi riittää tarvittaessa pelkkä sähköposti.
Jokaisesta uudesta ominaisuudesta ja ohjelmistovirheen korjauksesta on oltava tallennettuna suunnitelma, määrittely dokumentti. Ohjelmistomuutoksen täyttäessä suuren ohjelmistomuutoksen kriteerit, määrittelystä tulee olla erillinen dokumentti ja määrittelyvaiheessa on pidettävää määrittelypalaveri, pelkkä sähköposti ei riitä kommunikointivälineeksi.

Määrittelyn on katselmoitava. Määrittelyn katselmoinnista laaditaan tarvittaessa ”määrittelyn katselmointipöytäkirja”. Katselmointi voidaan kommentoida myös määrittely dokumenttiin. Pelkkä sähköposti ei riitä kommunikointivälineeksi.

Määrittelyn suorittava ryhmä tekee järjestelmä- ja hyväksymistestaussuunnitelman ja päättää katselmoidaanko suunnitelma vai ei. Testaussuunnitelmasta tallennetaan ja muutos- ja kokeilunseurantailmoitukseen. Pelkkä sähköposti ei riitä kommunikointivälineeksi.

Määrittelyn suorittava ryhmä päättää tehdäänkö yksikkötestaussuunnitelmaa ja katselmoidaanko suunnitelma vai ei. Testaussuunnitelmasta laaditaan tarvittaessa yksikkötestaussuunnitelma. Katselmointi kommentoidaan tarvittaessa yksikkötestaussuunnitelmaan tai tehdään erillinen ”yksikkötestaussuunnitelman katselmointipöytäkirja”. Kommunikointivälineeksi riittää tarvittaessa pelkkä sähköposti.

Jokaisesta uudesta ominaisuudesta ja ohjelmistovirheen korjauksesta on oltava tallennettuna integrointitestaussuunnitelma. Ohjelmistomuutoksen täyttäessä suuren ohjelmistomuutoksen kriteerit, integrointitestaus suunnitelma tulee olla erillinen dokumentti. Suunnitelma laaditaan ”integrointitestaussuunnitelma palaverissa”. Pelkkä sähköposti ei riitä kommunikointivälineeksi. Viimeistään tässä vaiheessa valitaan vastuullinen suunnittelija integrointitestaukseen.

Jokaisesta uudesta ominaisuudesta ja ohjelmistovirheen korjauksesta on oltava tallennettuna ”integrointitestaussuunnitelman katselmointipöytäkirja”. Ohjelmistomuutoksen täyttäessä suuren ohjelmistomuutoksen kriteerit, katselmoinnista tulee olla

erillinen dokumentti ja katselmointi on tehtävä ”integrointitestaussuunnitelman katselmointi palaverissa”, pelkkä sähköposti ei riitä kommunikointivälineeksi.

Jokaisesta uudesta ominaisuudesta ja ohjelmistovirheen korjauksesta on oltava tallennettuna ”integrointitestauksen katselmointipöytäkirja”. Ohjelmistomuutoksen täyttäessä suuren ohjelmistomuutoksen kriteerit, katselmoinnista tulee olla erillinen dokumentti ja katselmointi on tehtävä ”integrointitestauksen katselmointi palaverissa”, pelkkä sähköposti ei riitä kommunikointivälineeksi. Palaverissa korjataan tarvittaessa järjestelmä- ja hyväksymistestaussuunnitelmaa. (muutos- ja kokeilunseurantailmoitus)

Jokaisesta uudesta ominaisuudesta ja ohjelmistovirheen korjauksesta on tehtävä muutos- ja kokeilunseurantailmoitus. Ilmoituksessa on järjestelmä- ja hyväksymistestaussuunnitelman uudelle ominaisuudelle. Testausinsinööri kommentoi ja hyväksyy/hylkää muutoksen kyseisellä ilmoituksella.

Suuressa ohjelmistomuutoksessa tallennettava dokumentaatio: esimäärittelyn dokumentaatio, määrittelyn dokumentaatio, määrittelyn katselmointipöytäkirja (ei pakollinen), yksikkötestaussuunnitelma (ei pakollinen), integrointitestaussuunnitelma, integrointitestaussuunnitelman katselmointipöytäkirja, integrointitestauksen katselmointipöytäkirja, muutos- ja kokeilunseurantailmoitukset jokaisesta ohjelmistomuutoksesta.

8.3 Järjestelmätestausvaihe

Jokaiselle järjestelmätestausjaksolle tehdään oma pöytäkirja, josta käy ilmi jaksolle valitut tehtävät. Samaan pöytäkirjaan dokumentoidaan jakson lopulla järjestelmäjakson ohjelmistotestauksen läpäisseet ohjelmistomuutoksen, sekä ohjelmistomuutokset jotka eivät läpäisseet testi. Pöytäkirjat laaditaan ”järjestelmätestausjakson seuranta” palavereissa, pelkkä sähköposti ei riitä kommunikointi välineeksi. Pöytäkirjat nimetään kyseessä olevan release-jakson version ja järjestelmätestausjakson numeron

mukaan. Pöytäkirjat laaditaan järjestelmätestausjakson palaverissa, pelkkä sähköposti ei riitä kommunikointivälineeksi.

Tarvittaessa pöytäkirjaan tehdään merkintä ulkomaan testien aloittamisesta, tällöin pöytäkirja täytyy katselmoida, erillistä katselmointipöytäkirjaa ei tarvitse laatia, merkintää riittää että ulkomaantestit voidaan aloittaa. Sähköposti riittää kommunikointiin.

Järjestelmätestausvaiheen lopussa laaditaan ”järjestelmätestausvaiheen lopetus pöytäkirja”. Pöytäkirja laaditaan ”järjestelmätestausjakson lopetus” palaverissa, pelkkä sähköposti ei riitä kommunikointivälineeksi.

Viimeistään ”järjestelmätestausjakson lopetus” palaverissa laaditaan testaussuunnitelma ulkomaan testiä varten. Suunnitelmaan listataan asiat joihin testissä tulee kiinnittää huomiota. Suunnitelma ei saa olla sama minkä avulla testaan Suomessa, sillä siihen tulee poimia koostetusti ulkomaita koskevat muutokset. Kompaktilista helpottaa testausta ja nopeutta palautteen saantia.

Pöytäkirja katselmoidaan ”järjestelmätestausvaiheen lopetuksen katselmointi” palaverissa. Hyväksynnän saatua laaditaan ”hyväksymistestausvaiheen aloitus dokumentti”, pelkkä sähköposti ei riitä kommunikointivälineeksi. Samassa palaverissa myös käydään läpi edellisen ohjelmisto-releasen tehdasasennusohjetta, mitä muutoksia siihen täytyy tehdä uuden ohjelmisto-releasen takia. Uutta ohjetta ei kannata laatia alusta asti, koska kyseiseen dokumenttiin ei tule yleensä paljon muutoksia.

Järjestelmätestausvaiheessa tallennettava dokumentaatio: jokaiselle järjestelmäjaksolle oma pöytäkirja, järjestelmätestausvaiheen lopetus pöytäkirja, ulkomaan testaussuunnitelma ja tehdasasennusohje.

8.4 Hyväksymistestausvaihe

”Hyväksymistestausvaiheen aloitus pöytäkirjaan” merkitään ohjelmistomuutokset, jotka on hyväksytty hyväksymistestausvaiheeseen. Pöytäkirjaan kirjataan päätös 0-sarjan testin aloittamisesta ja ulkomaan testien aloittamisesta. Pöytäkirja laaditaan ”hyväksymistestausjakson aloitus” palaverissa, pelkkä sähköposti ei riitä kommunikointivälineeksi.

Ensimmäisen hyväksymistestausjakson lopussa pidetään ”tuotantoon hyväksytyt ominaisuudet” palaverissa. Pöytäkirjaan merkitään 0-sarjan ja ulkomaantestien tulokset. Pöytäkirjaan tulee merkitä ohjelmistomuutokset, jotka hyväksytään tuotantoversioon ja ohjelmistomuutokset, jotka vaativat jatkotoimenpiteitä. Tavoitteena on laatia pöytäkirja/pitää tuotantoon hyväksytyt ominaisuudet palaveri ulkomailta saadun palautteen jälkeen. Pöytäkirja laaditaan ”tuotantoon hyväksytyt ominaisuudet” palaverissa, pelkkä sähköposti ei riitä kommunikointivälineeksi.

Pöytäkirja katselmoidaan ”tuotantoon hyväksyttyjen ominaisuuksien katselmointi” palaverissa. Katselmointi kommentoidaan ”tuotantoon hyväksytyt ominaisuudet” pöytäkirjaan, pelkkä sähköposti ei riitä kommunikointivälineeksi.

Katselmoinnin jälkeen tehdään päätös jatketaanko mahdollisten ohjelmistovirheiden korjaamista, vai hyväksytäänkö ohjelmistovirheiden takia mahdollisesti alkuperäisiin määrittelyihin verrattuna oleva puutteellinen ohjelmisto. Katselmoinnin tulokset ja mahdolliset jatkotoimenpiteet kirjataan ”tuotantoon hyväksytyt ominaisuudet pöytäkirjaan”, pelkkä sähköposti ei riitä kommunikointivälineeksi.

Hyväksymistestausvaiheen lopuksi pidetään ”ohjelmisto-releasen tuotantoon julkaisu” palaveri, jossa päätetään onko ohjelmisto tuotantokunnossa ja tarvittaessa merkitään pöytäkirjaan ohjelmistomuutokset jotka eivät tule ohjelmisto-releaseen. Samalla päätetään myös jatkotoimenpiteistä ohjelmistomuutosten kohdalla, jotka eivät ole läpäisseet testiä. Jos hyväksymistestausaikaa joudutaan jatkamaan, kaikki asiaan liittymät tapahtumat tulee kirjata ylös ”ohjelmisto-releasen tuotantoon julkaisu” pöytäkirjaan.

Jos ohjelmistossa on kaikki ne ominaisuudet, mitkä ovat mainitut ”tuotantoon hyväksytyt ominaisuudet” pöytäkirjassa, ”ohjelmisto-releasen tuotantoon julkaisu” pöytäkirjaa ” ei ole välttämätöntä katselmoida. Palaverissa päätetään katselmoinnin tarpeellisuus. Jos ohjelmisto-release on vajaa verrattuna edellä mainittuun dokumentaatioon katselmointi tulee suorittaa. Katselmoinnin jälkeen tehdään päätös jatketaanko mahdollisten ohjelmistovirheiden korjaamista, vai hyväksytäänkö ohjelmistovirheiden takia mahdollisesti alkuperäisiin määrittelyihin verrattuna oleva puutteellinen ohjelmisto. Katselmoinnin tulokset ja mahdolliset jatkotoimenpiteet kirjataan ”ohjelmisto-releasen tuotantoon julkaisu” pöytäkirjaan, pelkkä sähköposti ei riitä kommunikointivälineeksi.

Kun ohjelmisto on hyväksytty tuotantoon ja siitä on merkintä ohjelmisto-releasen tuotantoon julkaisu-pöytäkirjassa, ohjelmistosta tehdään ohjelmistojen muutosilmoitus.

Jokainen hyväksymistestausvaiheen pöytäkirja on nimettävä kulloisenkin hyväksymistestaus palaverin ja julkaistavan ohjelmisto-releasen versionumeron mukaan.

Hyväksymistestausvaiheessa tallennettava dokumentaatio: hyväksymistestausvaiheen aloitus pöytäkirja, tuotantoon hyväksytyt ominaisuudet pöytäkirja, ohjelmisto-releasen tuotantoon julkaisu pöytäkirja ja muutosilmoitus.

8.5 Korjauspakettista tuotettava dokumentaatio

Korjauspaktista on oltava pöytäkirja, josta on käytävä ilmi ohjelmistovirheet, mitkä on löydetty ohjelmisto-releasen julkaisun jälkeen. Virheet on priorisoitava kahteen luokkaan: ei korjaustoimenpiteitä välittömästi ja korjattava välittömästi. Sekä asiakkaat ja laitteisto, joissa ohjelmistovirheiden testaus suoritetaan, on merkittävä pöytäkirjaan. Pöytäkirjaan on myös merkittävä lupa korjauspaketin julkaisusta.

Pöytäkirja on nimettävä kulloisenkin ohjelmisto-releasen versionumeron ja siihen tehtävän korjauspaketin sarjanumeron mukaan.

Korjauspaketista on tehtävä ohjelmistojen muutosilmoitus.

Korjauspaketista tallennettava dokumentaatio: Korjauspaketin pöytäkirja ja ohjelmiston muutosilmoitus.

8.6 Funktiontestauksesta tuotettava dokumentaatio

Funktiotestauksessa dokumentointi tapahtuu suoraan ohjelmakoodiin, joten erillistä dokumentaatiota funktiotestauksesta ei tallennu.

9 JOHTOPÄÄTÖKSET

Tutkimuksen pätevyyttä (validiteetti) voidaan pitää hyvänä. Opinnäytetyössä käytetty lähdemateriaali on etsitty alalla paljon näkyvistä julkaisuista. Pätevyyttä voi heikentää käännöksistä johtuvat virheet sekä mahdollinen lähdemateriaalin suppeus. Vastaavasti tutkittavan yrityksen tuottama materiaali on luotu todellisessa toimintaympäristössä, joten tutkimuksessa käytetty tutkimusmateriaali on autenttista materiaalia eli materiaalia, jonka avulla testausprosessi etenee.

Tutkimuksen toistettavuutta (reliabiliteetti) voidaan myös pitää hyvänä, sillä saatu lopputulo/päätelmä ei poikkea olennaiselta osalta ohjelmistotuotekehityksessä vallitsevista käytännöistä. Tällöin voidaan olettaa toistettavuuden olevan hyvää luokkaa.

Yllämainittujen argumenttien avulla saatuja tuloksia ja niiden avulla tehtyjä päätelmiä voidaan pitää luotettavina, soveltuvina ja tarkoituksen mukaisina.

9.1 Kehitystehtävän tulokset koostetusti

Opinnäytetyössä kehitettiin teoriataustaan ja työn tilanneen yrityksen tarpeet ja toimintaympäristön huomioon ottava, ohjelmisto-releasen testauskäytäntö. Ohjelmistotestauksesta on tehty uudessa mallissa luonteva osa ohjelmistonelinkaarta, eikä se ole enää irrallinen toiminto release-jakson lopussa.

Uuden testauskäytännön käyttöönotto vaatii muuttamaan myös yrityksen ohjelmistotuotekehityksessä käytössä olevaa ohjelmiston kehitysmallia. Vanhasta mallista, jossa ohjelmistomuutoksia ja ohjelmistovirheiden korjausta ei määritellä, on ollut huonoja kokemuksia myös muissa yhteyksissä. Ohjelmiston kehitysmallin kehittämistarpeista on käyty keskustelua ennen opinnäytetyötäkin, joten ongelma on ollut tiedostettuna jo jonkin aikaa yrityksen sisällä.

Käytännöntarpeet tiedostaen opinnäytetyössä luotiin kehitys- ja testausmallit (pieni ja suuri ohjelmistomuutos), joiden avulla uudet ominaisuudet ja ohjelmistovirheiden korjaukset tuodaan testiin hallitusti, pienempinä kokonaisuuksina. Tällä toimintatavalla saadaan vältettyä liian suuret muutokset ohjelmistossa yhdellä kerralla.

Luotu järjestelmä edesauttaa automaattisesti regressiotestausta, sillä uudet ominaisuudet ja ohjelmistovirheiden korjaukset tuodaan vähitellen testiin. Tällöin massiivista regressiotestisarjaa järjestelmälle ei tarvitse välttämättä tehdä. Jos yritys vielä pystyy tiedostamaan kriittisimmät muutokset ja tuomaan ne testiin ensimmäisissä järjestelmätestausjaksoissa, voidaan tällöin kriittisille ominaisuuksille taata mahdollisimman pitkä regressiotestausaika uuden testauskäytännön avulla.

Ohjelmisto on modulaarinen rakenne, joka voidaan jakaa yhä pienempiin ja pienempiin kokonaisuuksiin. Opinnäytetyössä päädyttiin siihen, että pienin testattava kokonaisuus on funktio. Funktion testaus tulee suorittaa välittömästi, kun se on ohjelmoitu ja siinä on toiminnallisuudet, jotka siihen on määritetty. Työssä funktiolle määritettiin selkeät testausohjeet. Ohje on tarkistuslista tyyppinen, joka dokumentoidaan kommenttina osaksi ohjelmakoodiin, jolloin se on ohjelmoijan helposti käytettävissä.

Opinnäytetyössä kehitettyä mallia voidaan soveltaa myös muiden kuin työn tilanneen yrityksen tarpeisiin. Mallia voidaan skaalata, lisäämällä ja vähentämällä järjestelmäjaksojen määrää, sekä tarvittaessa kasvattamaan hyväksymisvaiheen pituutta. Tärkeintä skaalauksessa on erotta edellä mainittujen vaiheiden merkitys toisistaan, ohjelmiston on oltava muuttumattoman testissä riittävän ajan. Kehitetty malli soveltuu vanhan ohjelmiston päivitykseen ja myös uuden ohjelmiston luontiin, koska siinä on eroteltavissa selkeästi määrittelyvaiheet ja testaustasot, jotka ovat tärkeitä tekijöitä erityyppisissäkin ohjelmistomuutoksissa. Riippumatta siitä onko kyseessä ohjelmis-

ton ominaisuuksien päivitys, uuden ohjelmiston kehitys tai sitten ohjelmistovirheen korjaus.

Opinnäytetyön kuului vastata kysymykseen: Mitkä ovat Ponssen tämän hetkiset heikkoudet ja vahvuudet ohjelmistotestauksessa ja miten ne voidaan ottaa huomioon kehitettäessä uutta ohjelmiston testauskäytäntöä?

Opinnäytetyössä on analysoitu viidennessä kappaleessa Ponssella käytössä olevan ohjelmistotestauksen heikkoudet ja vahvuudet. Näihin tietoihin perustuen on luotu uusi käytännönläheinen ohjelmisto-releasen testauskäytäntö. Joten opinnäytetyö vastaa työstä laadittuun kysymykseen.

9.2 Pohdinta

Ohjelmistotestaus on aiheena mielenkiintoinen, koska sen mekanismin avulla pyritään konkreettisesti analysoimaan ja varmistamaan ohjelmiston laatua. Testausta ei pidä nähdä pelkästään omana irrallisena tehtävänä ohjelmiston elinkaaren aikana, vaan sen tulee olla mukana koko ohjelmiston ylläpidon ajan. Ohjelmistotestauksen kehittäminen ei saa jäädä yrityksessä pelkästään tähän opinnäytetyöhän, vaan toimintatapaa tulee kehittää jatkossakin aktiivisesti. Uuden käytännön mukainen dokumentaatio luo pohjan jatkuvalle kehitykselle. Tallennetun historiatiedon avulla voidaan seurata tehtyjä muutoksia ja niiden vaikutusta ohjelmistotestauksen kehittymiseen.

Testauksen kehittämiseksi luotavat toimintamallit ovat haasteellisia, koska jokaisessa ohjelmistoprojektissa on aina omia yksilöllisiä piirteitä. Ohjelmistojen laajuudet ja ihanteelliset vaihejakomallit vaihtelevat projekteittain.

Jokaisesta ohjelmistoprojektista on kuitenkin löydettävissä yhteisiä tekijöitä. Näiden tunnusmerkkien avulla kehitettiin uudet toimintamallit (käytännöt), joiden avulla testauksen toimintaa voidaan parantaa. Ennen varsinasta ohjelmointityötä syntyvien dokumenttien (määrittely- ja suunnitteluvaiheen dokumentaatio) katselmointi on yksi kehittämistehtävän tärkeä huomion kohde, koska yleensä varsinkin pienissä tehtävis-

sä yksi suunnittelija vastaa kaikesta. Tällöin määrittelystä on vaarana tulla tietyllä tavalla ”yksipuolinen”. Määrittelyvirheet ovat kiusallisia ja kalliita virheitä, joten niiden välttämiseksi kannattaa tehdä töitä, vaikka jatkuva palavereiden pitäminen tuntuukin joskus joutavanpäiväiseltä.

Opinnäytetyöhön kehitetyssä toimintamallissa ominaisuudet kehitetään ja testaan useammassa jaksossa. Toimintatavassa otetaan huomioon happotestauksen tärkeys, sillä silloin varmistetaan jo kertaalleen tehtyjen ominaisuuksien toimivuus. Katselmointi on tärkeä vaihe ohjelmistonlaadun varmistuksen kannalta, myös suuremmissa kokonaisuuksissa. Tämä tärkeä seikka on myös otettu huomioon opinnäytetyössä.

Järjestelmä- ja hyväksymistestausvaiheissa laadittavien dokumenttien määrä on tehty tarkoituksen mukaisesti pieneksi. Käytännön avulla helpotetaan ohjelmistotuotekehityksen ulkopuolisten tahojen mahdollista ohjelmisto-releasen etenemisen seuraamista. Tarvittava tieto on löydettävissä nopeasti ilman useiden dokumenttien läpikäymistä. Suuren ohjelmistomuutoksen dokumentaatiossa on mahdollisuus laatia tarkasti vaiheittain etenevä dokumentaatio. Toimenpiteellä mahdollistetaan release-jaksolla tapahtuneen virheen paikallistaminen ja tätä kautta mahdollisuus välttää kyseistä virhettä seuraavalla kerralla. Jos järjestelmä- ja hyväksymistestausvaiheissa on tapahtunut virheitä, virheiden selvittäminen ei vaadi niin tarkkaa teknistä dokumentaatiota, kuin ohjelmistomuutoksen määrittely- tai ohjelmointi tapahtunut virhe. Dokumentointi on tärkeä asia ohjelmistotuotekehityksessä mutta siitä ei saa tulla itsetarkoitus. Dokumentoinnin tulee olla tarkoituksen mukaista ja sen on tuettava käytännöntarpeita. Sitä ei saa olla liika ohjelmistomuutokseen laajuuteen nähden, muuten sen tekeminen koetaan hyvin herkästi turhauttavaksi. Dokumenttien laadinnassakin on muistettava käytännön tarpeet. Jos niiden avulla kyetään osoittamaan ohjelmoijille esimerkiksi määrittelyn tarpeellisuus, turhien koodirivien ohjelmoinnin vähenemisenä. He huomaavat sen tarpeellisuuden, on paljon mielekkäämpää tehdä ohjelmointityötä, kun tietää että tekemänsä koodirivien tulevan käyttöön, eikä tehtyä työtä heitetä roskiin määrittelyvirheen takia. Toinen tekijä mikä huomataan pidemmän ajan kuluessa, liittyy uuden ominaisuuden/ohjelmistokomponentin rakenteeseen. Jos tarpeet ovat tiedossa alusta asti, uuden omaisuuden vaatima toiminnallisuus voidaan tarvittaessa optimoida juuri kyseisen ongelman ympärille tai tekemään siitä tarvittaessa yleispätevä, riippuen siitä mitä dokumentaatio pystyy luotettavasti kertomaan.

Opinnäytetyötä tehdessä nousi esille myös muita testaukseen liittyviä asioita. Automatisoitu testaus on yksi olemassa oleva vaihtoehto regressiotestaukseen. Joskin automatisoidun testausjärjestelmän kehittäminen on asiaa tutkineiden mielestä kallista, mutta se voisi sopia hyvin joillekin Ponssen ohjelmistotuotteille. Erityisesti PC-sovelluksille, joiden käyttöliittymä ei usein muutu ja perustoiminnallisuus pysyy samana. Toinen uusi asia oli ohjelmistojen testaukseen erikoistuneet yritykset. Kummatkin edellä mainitut ovat sen verran laajoja kokonaisuuksia, että päätin olla ottamatta niihin kantaa tässä opinnäytetyössä. Pidän kuitenkin kummankin asian esilletuloa tärkeänä tietona mietittäessä Ponssen ohjelmistotestauksen tulevaisuuden vaihtoehtoja.

Tehtävässä luotuja toimintamalleja olisi ehkä voinut olla useampiakin, mutta päädyin näihin vaihtoehtoihin, koska testauksen kehittäminen on uusi asia työyhteisössäni. Uuden asian läpivienti ja omaksuminen on helpompaa, jos asia on selkeää ja yksinkertaista. Tämä ”yksinkertainen on kaunista” tapa on niin monesti tullut todistettua työelämässä oikeaksi toimintatavaksi, niin kuin myös osaamisen johtamisen opinnoissa, että uskon sen toimivan tässäkin kohdassa.

Työssä luodut toimintamallit ovat yksinkertaiset, joten ne eivät varmaankaan ole täydellisiä. Kehittämistehtävässä määritettyjä toimintamalleja on hyvä kehittää eteenpäin, sillä aina on helpompi kommentoida jo olemassa olevia, tunnettuja puutteita, kuin alkaa miettiä puutteita ilman selvää vertailukohtaa. Näiden yksinkertaisten perusmallien (pieni ja suuri muutos) luonti onnistui mielestäni hyvin, sillä löysin tarvittavat yhteiset tekijät, joiden avulla testausta saadaan kehitettyä. Tärkein näistä tekijöistä oli dokumentaation katselmointi ja selkeästi vaiheistettu uusi release-jakson toimintamalli.

Työtä kriittisesti analysoitaessa huomaa mallissa puutteen hyväksymistestauksessa. Mallia ei määritä kuin kaksi hyväksymistestausjaksoa ja on mahdollista, että ohjelmisto ei täytä sille asetettuja vaatimuksia annetussa ajassa. Luotu malli olettaa tilanteen olevan hyvin epätodennäköinen, sillä ohjelmistoa on järjestelmätestattu huomattavasti enemmän, kuin nykykäytännön mukaan toimitaan. Tällöin ongelman realisoituminen pitäisi olla epätodennäköistä. Huomio tulee myös kiinnittää viimeisessä järjestelmätestausjaksossa tapahtuviin ohjelmistomuutoksiin. Ohjelmistomuutoksille ei

ole varsinaista järjestelmätestausjaksoa, vaan ominaisuus siirtyy suoraan hyväksymistestaukseen. Jos ominaisuudet kehitetään kriittisyysjärjestyksessä, viimeisimmät ominaisuudet eivät ole suuria/kriittisiä kokonaisuuden kannalta, joten tilanteesta ei pitäisi tulla ongelmaa. Ongelmien ilmetessä, mallista ei löydy selkeätä vastausta, joten siltä osin mallia pitää vielä jatko kehittää.

Jatkokehitystä kaipaavaa piirre on muutos- ja kokeilunseurantailmoitus lomakkeen hyväksikäyttö koko release-jakson ajan. Kehitetyssä mallissa on jokaisesta järjestelmätestausjaksolla ja hyväksymistestausjaksolla olevasta uudesta ominaisuudesta/muutoksesta on oma muutos- ja kokeilunseurantailmoitus. Käytännön kannalta parasta olisi, jos jokaisella jaksolle olisi yksi kokoomaraporttipohja, joita testausinsinööri täyttää. Raporttipohjia tulisi ehkä olla neljä kappaletta järjestelmätestiin ja yksi hyväksymistestiin. Tällöin voidaan kuitata Ponssen tuotannonohjausjärjestelmään yhdellä lomakkeella ohjelmiston hyväksymistestin suoritetuksi ja samalla annetaan lupa ohjelmisto-releasen julkaisulle.

Tulevaisuudessa on mietittävä myös erillisen hyväksymistestaussuunnitelman tekemistä, luodun mallin mukaan sama testaussuunnitelama käy järjestelmä- ja hyväksymistestausvaiheessa. Erilliselle suunnitelmalle on luultavasti tarvetta konttoriohjelmistolla, ei niinkään metsäkoneympäristössä, jossa testausinsinööri on aktiivisesti mukana koko testien ajan.

Parhaiten onnistunut osio työssä on löydetyt ongelmat release-jakson suorituksessa ja käytännönläheinen tapa, jolla ongelmat voidaan poistaa. Mitään ”poppakonsteja” ei tarvitse käyttää, vaan ongelma voidaan poistaa modernisoimalla jo nykyistä mallia. Asioita tehdään eri järjestyksessä kuin ennen ja hyväksytään ajatus, että uusi kehitettävä ominaisuus voidaan kehittää ja testata pienemmissä kokonaisuuksissa. Tämä avulla ohjelmistotestaus tulee osaksi koko release-jakson toimintaa, kiinteäksi osaksi ohjelmistotuotekehitystä. Eikä se ole enää ole enää kiireellä tehty viimeinen voitelu ohjelmistolle ennen sen julkaisua. Suorittamalla release-jakson vaiheittain, uuden käytännön mukaisesti, ohjelmisto kehitettään pala kerrallaan. Vaiheistuksen tavoitteena on järjestelmällinen ja määrätietoinen toiminta, harkittu askel kerrallaan, ei suuria epämääräisiä harppauksia.

LÄHTEET

Beck K., 2004, Extreme Programming,
www.extremeprogramming.org
Luettu (29.11.2004)

Beizer B, 1995. Black-Box Testing Techniques for Functional Testing of Software and Systems. ISBN 0-471-12094-4

Britschgi J., 2004, Testauksen valmennusohjelma

Broekman B, Notenboom E. 2003. Testing Embedded Software.
ISBN 0-321-15986-1

Bruce P., Daniel M., 2002, Just Enough Software test Automation
ISBN 0-13-008468-9

Haikala I., Märijärvi J, 2004. Ohjelmistotuotanto.
ISBN 952-14-0850-2

Karvinen K., Riekkinen A., 2004, Soveltamiskokemuksia ohjelmistotuotannon menetelmistä: Vaatimusmäärittely, käyttöliittymäsuunnittelu, arkkitehtuurisuunnittelu, toteutus ja testaus
ISBN 951-27-0232-0

Kolawa A., 2002, The ABCs of XP, RAD and PSP
www.stickyminds.com
Luettu (1.5.2005)

McConnel S., 1998, Ohjelmistoprojektit - selviytymisopas,
ISBN 951-762-616-9

Patton R., 2001, Software Testing
ISBN 0-672-31983-7

Pressman R., 1997, Software engineering a practitioner’s approach fourth edition.
ISBN 0-07-052182-4

Murat M., Chan S., 1991, Fundamentals of Computing for Software Engineers
ISBN 0-442-00525-3

Satukangas A., 2003, Yksikkö- ja integrointitestauksen menetelmät ja mittarit

Spillner A.,The W-MODEL- Strengthening the Bond Between Development and Test,2002
www.iti.upv.es/~squac/JTS/JTS2004/docs/Wmodel.pdf
Luettu (30.11.2004)

Stenberg A., Ohjelmiston testaus, 2004,
http://www.pori.tut.fi/~stenberg/
Luettu (1.4.2005)

Thomas D., Hunt A., Learning to Love Unit Testing. The Software Testing & Quality Engineering Magazine, January/February 2002

Virkkunen H., 2002, Ohjelmistojen testaus ja virheenjäljitys