VAASAN YLIOPISTO
TEKNILLINEN TIEDEKUNTA
TIETOTEKNIIKAN LAITOS

Esa Nakkila

OHJELMISTOTESTAUS KETTERÄSSÄ OHJELMISTOTUOTANTOPROSESSISSA

Tietotekniikan
pro gradu -tutkielma

TEVITI koulutusohjelma

VAASA 2007

ALKUSANAT

Tämä työ on tehty Vaasan yliopistolle 1.9.2006 – 17.9.2007 välisenä aikana. Työtä on osittain rahoittanut Ixonos Oyj, jolle haluan esittää kiitokseni taloudellisesta tukemisesta. Työn valvojina toimivat professori Merja Wanne sekä lehtori Hannu Niinimäki Vaasan yliopistosta, joita haluan kiittää saamastani opastuksesta ja tuesta.

Työn ohjaajina Ixonos Oyj:llä toimivat Tapani Stjernvall, Riitta Hunnakko ja Jani Lehtinen. Heille haluan esittää kiitokseni kaikesta saamastani tuesta sekä erityisesti asiantuntevasta avusta.

Lopuksi haluan esittää erityiskiitokset puolisolleni Petra Karvoselle hänen osoittamastaan suuresta avusta, tuesta ja kärsivällisyydestä, jota ilman tämän työn tekeminen olisi ollut huomattavasti vaikeampaa. Lisäksi erityiskiitoksen ansaitsee poikani Leevi Nakkila, jonka aurinkoinen ja elämäniloa täynnä oleva olemus piristi jaksamaan loppuun saakka.

Oulussa 18.9.2007

Esa Nakkila
Huurrekuja 1 A 1
90630 Oulu

LYHENNELUETTELO

Taulukko 1.

Lyhenne	Selitys
BCS	British Computer Society. Vuonna 1957 perustettu kansainvälinen tietotekniikka-alan järjestö, jossa on nykyisin yli 60 000 jäsentä yli sadasta maasta.
CMM	Capability Maturity Model. Yritysten tuotantoprosessien kehittämiseen luotu menetelmä. CMMI-menetelmän edeltäjä.
CMMI	Capability Maturity Model Integration. Yritysten tuotantoprosessien kehittämiseen luotu menetelmä. Pohjautuu aiempaan CMM-menetelmään.
GG	Generic Goals. CMMI mallin yleinen tavoite.
GP	Generic Practises. CMMI mallin yleinen toimintatapa.
IEEE	Institute of Electrical and Electronics Engineers. Kansainvälinen elektroniikka-alan järjestö.
ISEB	Information Systems Examinations Board. BCS:n toimielin, joka myöntää sertifiointeja tietotekniikka-alan ammattilaisille.
NCC	National Computing Centre. Vuonna 1966 perustettu englantilainen järjestö, jonka tarkoituksena on edistää tietotekniikka-alan kehittymistä Iso-Britanniassa.
OMG	Object Management Group. 1989 perustettu konsortio, jonka tarkoituksena on edistää oliotekniikoita hajautettujen järjestelmien käyttöön.
SEI	Software Engineering Institute. Yhdysvaltalainen tietotekniikan alan tutkimusta ja kehittämistä harjoittava järjestö.
TMap	Test Management Approach. Ohjelmistotestauksen prosessien kehittämiseen luotu metodi.
TPI	Test Process Improvement. Ohjelmistotuotannon testausprosessien kehitystarpeiden arviointiin käytetty metodi.
UML	Unified Modeling Language. OMG:n vuonna 1997 standardoima graafinen mallinnuskieli, jossa kaavioiden avulla kuvataan olio-ohjelmista tehtyjen mallien rakennetta, käyttäytymistä ja vuorovaikutuksia.
XP	Extreme Programming. Ketteriin menetelmiin pohjautuva ohjelmistotuotannon metodi.

TIIVISTELMÄ

VAASAN YLIOPISTO
Teknillinen tiedekunta
Tekijä: Esa Nakkila
Tutkielman nimi: Ohjelmistotestaus ketterässä ohjelmistokehitysprosessissa
Ohjaajan nimi: Merja Wanne
Tutkinto: Kauppatieteiden maisteri
Laitos: Tietotekniikan laitos
Oppiaine: Tietotekniikka
Opintojen aloitusvuosi: 2004
Tutkielman valmistumisvuosi: 2007 **Sivumäärä:** 61

TIIVISTELMÄ:

Tutkielman lähtötilanteena oli Ixonos Oyj:n *Tietoliikenne*-yksikön ketteriin menetelmiin kuuluvien Scrum ja XP-metodeihin pohjautuva SW Design and Implementation – ohjelmistotuotantoprosessi, johon ei ollut määritelty vielä omaa testauksen aliprosessia.

Tutkimusongelmana oli laatia olemassa olevaan ohjelmistotuotantoprosessiin testaukselle oma aliprosessi niin, että se noudattelee ketterien menetelmien periaatteita ja soveltuu yrityksen tarpeisiin.

Tutkimusmenetelmä oli teoreettinen. Aluksi tutustuttiin alan kirjallisuuteen sekä yrityksen aiheeseen liittyvään omaan dokumentaatioon. Lisäksi haastateltiin yrityksen asiantuntijoita. Saadun tiedon pohjalta analysoitiin muutamien erilaisten menetelmien käyttämistä itse prosessin laatimiseen. Lopulta itse testausprosessi laadittiin ja esiteltiin yrityksen edustajille.

Tutkielman jälkeen tehtiin johtopäätökset. Todettiin, että TMap-menetelmä ei sovellu ketteriin menetelmiin perustuvan testausprosessin kehittämiseen. Lisäksi todettiin, että testausprosessissa ei kaikissa tilanteissa voida täysin puhtaasti noudattaa ketterien menetelmien Scrum ja XP-metodien periaatteita. Testausprosessi, joka yritykselle luotiin, todettiin kuitenkin hyväksi ja käyttökelpoiseksi.

AVAINSANAT: ketterät menetelmät, ohjelmistotestaus, prosessi, SCRUM, extreme programming

ABSTRACT

UNIVERSITY OF VAASA
Faculty of technology
Author: Esa Nakkila
Topic of the Master's Thesis: Software Testing in Agile Software Development Process
Instructor: Merja Wanne
Degree: Master of Science in Economics and Business Administration
Department: Department of Computer Science
Major subject: Computer Science
Year of Entering the University: 2004
Year of Completing the Master's Thesis: 2007 **Pages:** 61

ABSTRACT:

Starting point for this Master's Thesis was to define a testing subprocess based on SW Design and Implementation Process of Ixonos Plc's Telecommunication business unit. SW Design and Implementation Process was created based on Agile methods called Scrum and XP (Extreme Programming).

The research problem was to define testing subprocess so that it applies Agile methods principles and also is suitable for company's needs.

The research method was theoretical. In the beginning of research related litterature was used as well as company's own documentation. Also the experts from the company were interviewed. Afterwards it was analysed if it would be possible to use some of the existing process development methods for this work. The testing subprocess was created and presented for the company's representatives.

Finally the conclusions of the research were made. It was discovered that TMap method is not suitable to be used with processes based on Agile methods. Also it was discoverred that within testing processes it is not always possible to apply Agile methods, Scrum and XP, principles. However the testing subprocess created for the company was found out to be good and very usefull.

KEYWORDS: Agile methods, software testing, process, SCRUM, extreme programming

1. JOHDANTO

Tutkielma kertoo miten uusi ohjelmistotestausprosessi suunniteltiin Ixonos Oyj nimisen yrityksen (31.1.2007 asti Tieto-X Oyj) *Tietoliikenne*-yksikölle, jossa tutkielman tekijä työskentelee. Ixonos Oyj tekee alihankintana asiakasyrityksilleen erilaisia ohjelmistotuotantoprojekteja sekä omia ohjelmistotuotteitaan.

Tietoliikenne-yksikössä oli jo laadittu ohjelmistotuotantoprosessi, jota yrityksessä kutsutaan nimellä SW Design and Implementation Process. Tähän prosessiin ei kuitenkaan oltu vielä määritelty testauksen osalta omaa aliprosessiaan. Tämä oli lähtötilanne tutkielman tekemiselle.

Tutkielmassa kuvataan aluksi tarkemmin lähtötilannetta, jonka jälkeen kuvataan niitä erilaisia työkaluja ja menetelmiä, joita suunniteltiin käytettäväksi testausprosessin luomisessa. Lopuksi kerrotaan millainen testausprosessi määriteltiin ja miten hyvin sen on arvioitu soveltuvan Ixonos Oyj:n toimintaan.

2. TUTKIELMAN ONGELMA JA TAVOITTEET

Tutkimuksen lähtötilanne oli Ixonos Oyj:n *Tietoliikenne*-yksikössä kehitetty ohjelmistotuotannon prosessi, jota yrityksessä kutsutaan SW Design and Implementation -prosessiksi. Periaatteiltaan se noudattelee niin kutsuttuja *ketteriä* (Agile) ohjelmistotuotantomenetelmiä, erityisesti Scrum ja XP-metodeja (Extreme Programming), mutta joiltain osin siinä on Ixonos Oyj:n toiminnalle sovellettuja osa-alueita perustuen Ixonos Oyj:n uuteen palvelukonseptiin.

Uusi ohjelmistotuotantoprosessi ei sisältänyt ohjelmistotestaukselle määriteltyä prosessivaihetta tai sen eri mekanismeja, joten yrityksellä oli tarve saada tähän itse luomaansa SW Design and Implementation -prosessiin integroitua testausprosessi ja sen eri mekanismit.

Tutkijan tarkoituksena oli analysoida, miten ohjelmistotestaus tulisi tehdä noudattaen jo aiemmin luodun ohjelmistotuotantoprosessin periaatteita. Tämän analyysin perusteella oli tarkoitus laatia ja kuvata uusi testausprosessi, joka soveltuisi parhaiten SW Design and Implementation -prosessiin sekä uuteen palvelukonseptiin ja noudattelisi myös ketterien menetelmien periaatteita. Lopputulos esiteltiin yrityksen edustajille ja heiltä saatavan palautteen perusteella tehtiin joitain korjauksia. Lopuksi luotiin viimeistelty materiaali, jonka avulla testausprosessi voidaan ottaa projektien käyttöön.

Tutkimuksen tavoitteet voidaan jakaa seuraaviin osiin:

1. Tunnistaa ja analysoida millä tavoin ohjelmistotestaus tulisi suorittaa Ixonos Oyj:n *Tietoliikenne*-yksikön SW Design and Implementation -prosessin mukaisissa ketteriä menetelmiä noudattelevissa ohjelmistotuotantoprojekteissa.
2. Laatia ja kuvata SW Design and Implementation -prosessin sekä uuden palvelukonseptin mukainen uusi testausprosessi.
3. Laatia ohjeet ja kuvaus testausprosessista.

Tutkimusongelmaksi voidaan määritellä uuden ohjelmistotestausprosessin laatiminen. Erityiseksi ongelmaksi voidaan kuvata testausprosessin näkökulmasta katsoen, millä tavalla asiakkaiden testaukselle asettamat vaatimukset otetaan huomioon. Joissain tapauksissa asiakkaalla voi olla tarve käyttää muita kuin Ixonos Oyj:n itse laatimia testausprosesseja tai sen vaiheita, jolloin Ixonos Oyj:n omien prosessien täytyy voida joustaa asiakkaan vaatimusten mukaisesti.

3. TUTKIMUSMENETELMÄ

Tutkimusmenetelmä on teoreettinen. Tutkielman aikana tutustuttiin aluksi alan kirjallisuuteen sekä artikkeleihin. Lisäksi yrityksen oma dokumentaatio ja asiantuntijoiden haastattelut lisäsivät teoreettista tietämystä aiheesta.

Teoreettisen tiedon hankkimisen jälkeen aloitettiin yleinen prosessin kehityksen kartoittaminen CMMI-menetelmän (Capability Maturity Model Integration) avulla. CMMI:n avulla kartoitettiin testausprosessin kehittämiselle tarvittava viitekehys, josta käy esimerkiksi ilmi, mitkä muut yrityksen prosessit vaikuttavat testausprosessin sisältöön tai sen suorittamiseen. Tavoitteena ei siis ollut luoda täydellistä CMMI:n mukaista mallia ja tasoluokitusta nyt tehdylle testausprosessille. CMMI on SEI:n (Software Engineering Institute) luoma prosessien kehittämiseen suunniteltu menetelmä, jonka avulla yritykset voivat muuttaa tai luoda uusia ohjelmistotuotannon prosesseja. CMMI:stä on kerrottu tarkemmin luvussa 6.

Tiettyjen testausprosessin osioiden ja määrittelyjen luomiseen käytettiin pohjana ISEB (Information Systems Examinations Board) Testing Foundation kurssin ja Testing Practitioner kirjan aineistoa. Testing Practitioner kirja perustuu ISEB:n Testing Practitioner kurssin aineistoon. ISEB ohjelmistotestaus kurssin ja kirjan aineisto kuvaa erilaisia testauksen menetelmiä, tekniikoita ja vaiheita. Tämä aineisto perustuu BCS:n (British Computer Society) ohjelmistotestausstandardeihin. BCS:n ohjelmistotestausstandardeista sekä ISEB:stä kerrotaan tarkemmin luvussa 8. Lisäksi testausprosessia suunniteltiin kehitettäväksi TMap-menetelmän (Test Management Approach) avulla. TMap on testausprosessien kehittämiseen luotu menetelmä, joka myös pohjautuu BCS:n ohjelmistotestausstandardiin. TMap-menetelmää on kuvattu luvussa 7.

Testausprosessin täytyi soveltua ketteriin menetelmiin, koska SW Design and Implementation -prosessi perustuu erityisesti XP- (Extreme Programming) ja Scrum metodeihin, jotka ovat eräitä ketteriä menetelmiä. Ketterät menetelmät poikkeavat

perinteisistä menetelmistä ja malleista, kuten esimerkiksi vesiputous- tai V-malli, hyvin erilaisten lähtökohtien ja näkökulmien vuoksi. Kun tekninen ympäristö muuttuu tuotekehityksen aikana jatkuvasti, järjestelmän kaikkia yksityiskohtia on mahdotonta määritellä alkuvaiheessa täsmällisesti. Tällöin kehitystä tehdään iteratiivisesti ja muutosten aiheuttamaa työkuormaa minimoidaan esimerkiksi vähentämällä kehityksen aikaisen dokumentoinnin tarvetta tehokkaalla sisäisellä viestinnällä. Ketterissä menetelmissä on olennaista, että testausta tehdään koko ajan, esimerkiksi joka yö käännetään uusi toimiva versio ohjelmasta, joka automaattisen testausjärjestelmän avulla testataan heti käännöksen jälkeen.

Testausprosessin luonnissa käytettiin hyväksi yrityksen *Järjestelmäpalvelut* sekä *Tietoliikenne* -yksiköiden muiden testausprosessien määrittelyjä. Yrityksessä on jo aiemmin luotu kaikille testausprosesseille yhtenäiset dokumenttipohjat. Näitä samoja dokumenttipohjia käytetään tässä tutkielmassa laaditussa testausprosessissa yrityksen käytännön mukaisesti.

Yrityksessä työskentelevien ohjelmistotestauksen asiantuntijoiden tietämystä käytettiin testausprosessin suunnittelun hyväksi. Pohjana tutkimukseen tarvittavalle aineistolle oli yrityksen jo olemassa oleva ohjelmistotuotantoprosessi eli SW Design and Implementation -prosessi ja siihen liittyvä dokumentaatio sekä muiden yrityksen testausprosessien kuvaukset ja niihin liittyvä dokumentaatio.

Muuta aineistoa kerättiin kirjallisuuden, yrityksen muun oman dokumentaation ja yrityksen testauksen asiantuntijoiden haastattelujen kautta.

4. KETTERÄT MENETELMÄT

Ixonos Oyj:n *Tietoliikenne*-yksikön SW Design and Implementation -prosessi pohjautuu *ketteriin* (*Agile*) menetelmiin, mistä syystä myös siihen luotavan testausprosessin tulee noudattaa *ketterien* menetelmien periaatteita.

Ohjelmistotuotannossa *Agile* eli *ketterät* menetelmät esiteltiin ensimmäisen kerran vuonna 2001, jolloin internetissä julkaistiin sivusto nimellä Agile Software Development Manifesto (Abrahamsson, Salo, Ronkainen & Warsta 2002: 9). Kiinnostus ketterien menetelmien soveltamiseen ohjelmistotuotannossa syntyi siitä, että perinteiset ohjelmistotuotantomallit tuntuivat kankeilta ja joustamattomilta tietyn tyyppisissä ohjelmistoprojekteissa. Ohjelmistotuotantoprojekteihin haluttiin tehokkuutta ja joustavuutta lisää.

Ketterät menetelmät pyrkivät tehostamaan ohjelmistotuotantoa vähentämällä dokumentaatiota, lisäämällä projektiorganisaation sisäistä kommunikaatiota ja luomalla dynaamisemman työympäristön ja menetelmät.

Ketterien menetelmien ominaispiirteet poikkeavat perinteisistä ohjelmistotuotannon menetelmistä. Näitä ominaispiirteitä on viisi. *Ensimmäisenä ominaispiirteenä* voidaan kuvata organisaatiossa työskentelevien henkilöiden sijoittamista työtiloihin. Istumapaikat pyritään jakamaan niin, että yhdessä huoneessa olisi aina kahdesta kahdeksaan työntekijää. Tällä pystytään saavuttamaan tehokkaampaa kommunikointia ja yhteenkuuluvuuden tunnetta. (Abrahamsson 2002.)

Toinen ominaispiirre on paikan päällä tapahtuva asiantuntijoiden konsultointi. Projektin sisällä ei pyritä luomaan esteitä eri kokonaisuuksista vastaavien tiimien ja ryhmien väliselle kommunikoinnille. Erityisesti pyritään välttämään kommunikoinnin keskittämistä pelkästään ryhmien esimiesten välille. Tavoitteena on, että jokainen projektin jäsen voi avoimesti kommunikoida suoraan juuri sen alan asiantuntijan kanssa, jota esimerkiksi hänen ongelmansa koskee. Tämä mahdollistaa nopean ja dynaamisen

tiedonkulun projektin sisällä sekä ylläpitää keskusteluyhteyttä ryhmien ja henkilöiden välillä. (Abrahamsson 2002.)

Kolmas ominaispiirre on lyhyiden niin sanottujen *inkrementtien* (increments) luominen. *Inkrementin* tilalla nykyisin käytetään yhä yleisemmin termiä *sprintti*. *Inkrementillä* tarkoitetaan osakokonaisuutta tehtävästä tuotteesta. *Inkrementti* voi sisältää esimerkiksi pelkästään yhden ominaisuuden tai pelkkiä virheiden korjauksia. Tuotetta ei siis aleta tehdä kerralla valmiiksi yhden ison prosessin kautta, vaan sitä tehdään inkrementeittäin palanen kerrallaan. Jokainen *inkrementti* käännetään ja testataan erikseen. Tämä mahdollistaa nopean testauksen sekä nopean reagoinnin virheisiin ja niiden korjaamiseen. Ajallisesti yhden *inkrementin* tulisi kestää maksimissaan kolme kuukautta. Tähän vaikuttaa tuotteen ja asiakkaan asettamat vaatimukset huomattavasti. (Abrahamsson 2002.)

Neljäs ominaispiirre on täysin automatisoitu regressiotestaus. Regressiotestauksessa pyritään varmistamaan tuotteen uuden version sisältämä vanha ja jo aiemmin testattu toiminnallisuus. Ketterissä menetelmissä regressiotestauksella pyritään varmistumaan, että *inkrementtiin* lisätyt uudet ominaisuudet tai virhekorjaukset eivät ole rikkoneet mitään aiempaa toiminnallisuutta. Regressiotestauksessa ei välttämättä testata kaikkea jo aiemmin testattua uudelleen, koska aina siihen ei ole mahdollisuuksia ajallisesti tai se ei ole järkevää. Ketterissä menetelmissä pyritään automaation avulla siihen, että mahdollisimman paljon aiemmasta testauksesta pystytään regressiotestauksessa testaamaan uudelleen ja että testaus suoritetaan nopeasti ja tehokkaasti. Testauksen automatisointia voidaan perustella sillä, että kyse on usein toistuvista samoista testitapauksista. Eräänä tavoitteena on lisäksi se, että regressiotestaus voitaisiin ajaa automaattisesti aina uuden *inkrementin* käännöksen jälkeen. (Abrahamsson 2002.)

Viides ominaispiirre on se, että ohjelmistokehittäjien kokemus kasvaa *inkrementtien* lisääntyessä. Tutkimusten mukaan kokemuksen kasvaessa kehitystyö nopeutuu kahdesta kymmenkertaiseksi verrattuna projektiorganisaation hitaimpaan ohjelmistokehittäjään. (Abrahamsson 2002.)

Craig Larman listaa kirjassaan Agile & Iterative Development – A Manager's Guide kolmetoista ketterien menetelmien periaatetta:

1. Korkein prioriteetti on taata asiakkaan tyytyväisyys toimittamalla tasokkaita ohjelmistoja aikaisen ja jatkuvan toimituksen periaatteella.
2. Muuttuvia vaatimuksia otetaan vastaan, jopa kehityksen myöhäisessä vaiheessa, sekä ketterien prosessien muutoksia asiakkaan kilpailukyvyn eduksi.
3. Toimivia ohjelmia toimitetaan jatkuvasti, kahdesta viikosta muutamaan kuukauteen kestävällä (mieluiten lyhyellä) aikavälillä.
4. Liiketoiminnasta ja ohjelmistojen kehityksestä vastaavien henkilöiden on työskenneltävä yhdessä koko projektin ajan.
5. Projekti rakennetaan motivoituneiden yksilöiden ympärille. Antaa heille sellainen ympäristö ja tuki, mitä he haluavat ja luottaa siihen, että he saavat työn tehdyksi.
6. Kaikkein tuloksellisin ja tuottavin metodi tiedon jakamiseen kehitystiimille ja tiimissä on kasvotusten käytävä avoin keskustelu.
7. Toimiva ohjelma on ensisijaisin edistymisen mittari.
8. Ketterät menetelmät edistävät kannattavaa kehitystä.
9. Tukijoiden, kehittäjien ja käyttäjien tulisi pystyä ylläpitämään tasaista vauhtia määrittelemättömästi.
10. Jatkuva teknisen erinomaisuuden ja hyvän suunnittelun huomioiminen kehittää ketteriä menetelmiä.
11. Yksinkertaisuus – tekemättömien töiden määrän maksimoimisen taito – on olennaista.
12. Parhaat arkkitehtuurit, vaatimukset ja suunnitelmat tulevat esiin itsensä organisoivista tiimeistä.
13. Säännöllisin väliajoin tiimi harkitsee, miten tulla tehokkaammaksi. Pohdinnan jälkeen tiimi virittää ja säätää toimintaansa .

Ketterät menetelmät voidaan jakaa *metodeihin*. Ketterät menetelmät on yleisempi nimitys erilaisille metodeille, jotka noudattavat ketterien menetelmien periaatteita, mutta joilla on joko tiettyyn ohjelmistotuotannon osa-alueeseen tarkempi

lähestymistapa tai kokonaan erilainen lähestymistapa ohjelmistotuotantoon kokonaisuudessaan. ”Termi Agile viittaa ohjelmistokehityksen filosofiaan” (Martin Fowler 2005). Fowler on listannut artikkelissaan seitsemän ketterien menetelmien metodia. Metodeja on useampiakin, mutta Fowler on valinnut seitsemän suosituinta ja kiinnostavinta. Tähän tutkielmaan näistä on valittu viisi lähempään tarkasteluun, koska nämä viisi liittyvät olennaisesti ohjelmistokehitykseen. Kaksi Fowlerin listasta pois jätettyä metodia ovat Lean Development ja (Rational) Unified Process. Lean Development liittyy enemmän teollisen tuotannon kuin ohjelmistotuotannon kehittämiseen. Toinen metodi, eli (Rational) Unified Process, on taas kiistanalainen, josta on käyty tietotekniikka-alan erilaisissa foorumeissa paljon keskustelua ja väittelyä siitä, onko se yhteneväinen *ketterien* menetelmien kanssa vai täysin oma itsenäinen menetelmänsä.

4.1. Agile Manifesto

Ensimmäinen julkaistu ketteriin menetelmiin pohjautuva metodi oli kuvattu *Agile Software Development Manifeston* internet sivuilla. Tätä on pidetty ketterien menetelmien ensimmäisenä versiona, jonka pohjalta muut ketteriin menetelmiin pohjautuvat metodit ovat kehittyneet. Tämän manifestin taustalla oli joukko olio-ohjelmointiin perehtyneitä ohjelmistokehityksen asiantuntijoita, jotka halusivat kehittää joustavamman ja tehokkaamman tavan tuottaa ohjelmia. Tämä metodi pohjautuu aiemmin tässä luvussa kerrottuihin ketterien menetelmien perusperiaatteisiin. (Fowler 2005.)

4.2. XP

XP on kenties kaikkein tunnetuin ja käytetyin metodi ketteristä menetelmistä. *XP-metodi* jakautuu viiteen arvoon: kommunikointi, palaute, yksinkertaisuus, rohkeus ja kunnioitus. Nämä edelleen jakaantuvat yhteensä neljääntoista periaatteeseen, jotka edelleen jakaantuvat 24:ään toiminta tapaan. *XP:n* ideologia onkin, että nämä 24

toimintatapaa ovat konkreettisia ohjeita, joiden mukaan projektiorganisaatio työskentelee päivittäin. Toimintatavat on luotu siksi, että arvoja on helpompi hahmottaa ja sisäistää konkreettisen tekemisen kautta. Toimintatapojen ja arvojen välillä on kuitenkin iso kuilu, joten metodin neljätoista periaatetta on luotu määrittelemään arvojen ja toimintatapojen välistä yhteyttä. Erityisesti *XP:ssä* kiinnitetään huomiota testaukseen. Testaus on tässä metodissa tulkittu tietyssä mielessä koko ohjelmistokehityksen perustaksi, jonka päälle voidaan rakentaa hyvä tuote. *XP-metodin* mukaan jokainen kehittäjä osallistuu myös testaukseen luomalla aina testin itse tekemälleen koodille. (Fowler 2005.)

4.3. Scrum

Scrum metodissa erityispiirteenä on sen keskittyminen erityisesti ohjelmistokehityksen hallinnolliseen näkökulmaan. Tämä korostuu ammatillisesti vahvojen tiimien ja niiden välisen kommunikaation kautta. Tavoitteena on kommunikaatiota korostamalla pyrkiä vähentämään byrokratiaa ja dokumentointia. *Scrumissa* pyritään tekemään kehitystä noin kolmenkymmenen päivän *iteraatioissa*, joita kutsutaan *sprinteiksi*. Lisäksi ohjelmistokehityksen tarkkailua ja kontrollia pyritään tehostamaan päivittäisillä kokouksilla. Tätä kautta korostetaan vähemmän käytännön kehitystyötä ja sitä kautta pyritään tehostamaan sitä. (Fowler 2005.)

4.4. Crystal

Crystal metodin erityispiirteenä on ensisijaisesti se, että se pyrkii muokkaamaan toimintatapansa tiimin koon mukaan. *Crystal* metodin eräs pääolettamuksia on, että kaikki menetelmät eivät sovellu samoilla säännöillä erikokoisille tiimeille, vaan niitä täytyy soveltaa tiimin koon mukaan. *Crystal* metodille on määritelty kolme prioriteettia: turvallisuus, tehokkuus ja elinkelpoisuus. Elinkelpoisuudella tarkoitetaan sitä, että kehitystyötä tekevät pystyvät omaksumaan *Crystal* metodin toimintatavat. *Crystalille* on määritelty myös ominaisuuksia, joista kolme tärkeintä ovat: jatkuva toimitus, joustava

kehitys ja läheinen kommunikaatio. Pyrkimyksenä on luoda prosessi, jossa voidaan mahdollisimman vähäisellä prosessilla saada aikaan menestyksekäs tiimi. Tarkoituksena on helpottaa työntekijöiden elintilaa projektissa vähentämällä prosessikuria ja sääntöjen määrää. (Fowler 2005.)

4.5. Context Driven Testing

Tämä metodi perustuu testauksen näkökulmaan. Lähes kaikki ketterien menetelmien metodit pohjautuvat ohjelmistokehittäjän näkökulmaan, mutta *Context Driven Testing* metodissa menetelmää lähdetään luomaan ensisijaisesti testauksen näkökulmasta. Perusajatuksena on, että ensin luodaan testitapaukset, jotka perustuvat tuotteen määrittelyihin, ja näiden testitapausten perusteella lähdetään vasta tuottamaan itse ohjelmakoodia. (Fowler 2005.)

5. SW DESIGN AND IMPLEMENTATION PROCESS

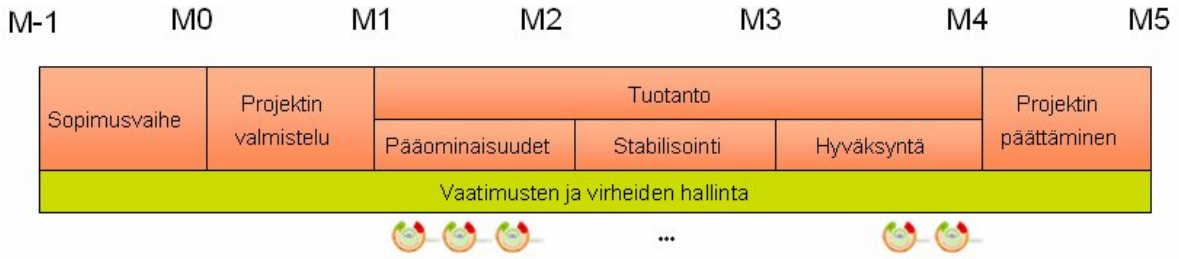
Ixonos Oyj:n Tietoliikenne-yksikössä havaittiin vuoden 2005 aikana tarve muodostaa uusi ohjelmistotuotantoprosessi. Syy tähän haluttuun muutokseen oli se, että yrityksen omissa tiloissa tehtävissä projekteissa prosessit eivät olleet riittävän tehokkaita ja dynaamisia. Tätä tilannetta korjatakseen yritys loi uuden markkinointistrategian ja siihen soveltuvan palvelukonseptin, jonka pohjalta uusi ohjelmistotuotantoprosessi tehtiin. Yrityksen tavoitteena on ollut muodostaa uudenlainen taloudellisesti ja ajallisesti tehokkaampi menetelmä ohjelmistotuotantoon. Vuoden 2006 keväällä valmistui *ProjectCannon* -niminen ohjelmistoprosessia kuvaava malli, josta kuitenkin puuttui ohjelmistotestaukseen liittyvät määrittelyt. Myöhemmin tämä prosessi sai nimekseen *SW Design and Implementation Process.* Tässä prosessissa pohjana on käytetty *ketterien* menetelmien *XP* ja *Scrum* nimisiä metodeja. (Lehtinen 2007.)

SW Design and Implementation -prosessissa projekti jaetaan kuuteen *työvaiheeseen*:

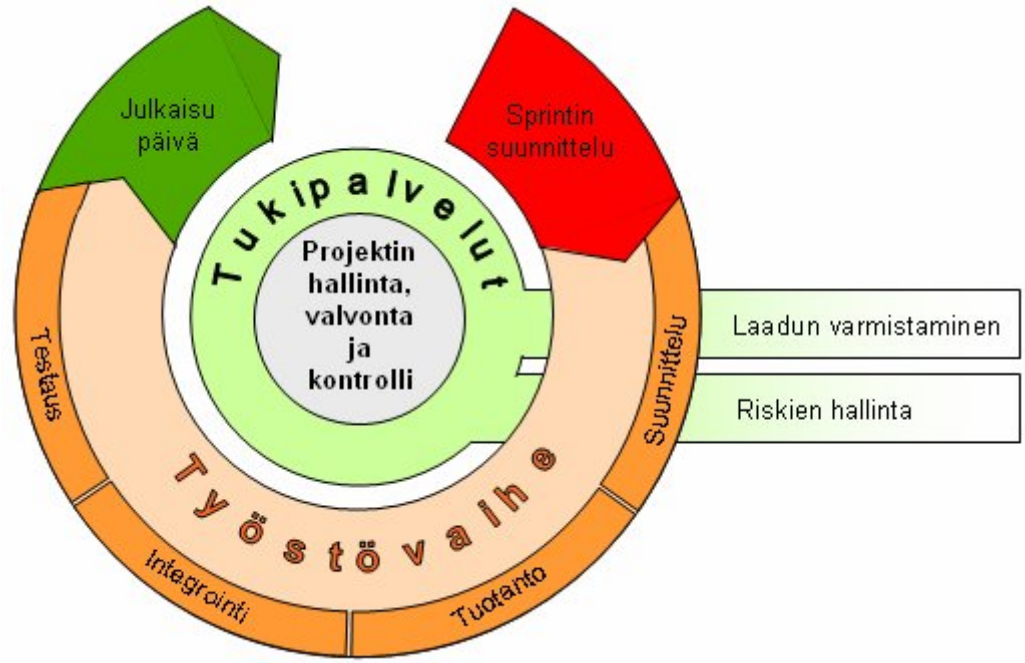
1. Sopimusvaihe (Contracting)
2. Projektin valmistelu (Ramp-up)
3. Tuotanto – pääominaisuudet (Implementation – Priority features)
4. Tuotanto – stabilisointi (Implementation – Stabilizing)
5. Tuotanto – hyväksyntä (Implementation – Acceptance)
6. Projektin päättäminen (Ramp-down).

Osat 1 ja 2 sisältävät pääasiassa projektin suunnitteluun ja käynnistämiseen liittyviä työvaiheita. Osat 3, 4 ja 5 sisältävät tuotantoon liittyvän konkreettisen tekemisen ketterien menetelmien avulla, joissa työvaiheita suoritetaan *sprinteissä*. Osa 6 sisältää projektin päättämiseen liittyvät työvaiheet. Työvaiheet on jaettu ajallisesti seitsemään *etappiin* (milestones), joiden mukaan projektille määritellään aikataulu sekä seurataan projektin työvaiheiden etenemistä. Kuten projekteissa yleensä, uutta työvaihetta ei voida aloittaa ennen kuin edellinen on suoritettu hyväksytysti loppuun. Hyväksynnän yleensä suorittaa Ixonos Oyj ja asiakas yhdessä tarkistettuaan, että kullekin etapille ennakolta

määritellyt kriteerit ovat täyttyneet. Nämä kriteerit määritellään projektisuunnitelmassa. (Lehtinen 2007.)



Kuva 1. *SW Design and Implementation* -prosessin vaiheiden ja etappien kuvaus.



Kuva 2. *SW Design and Implementation* -prosessista tehty ketteriin menetelmiin perustuvan tuotanto-vaiheessa toteutettavan yksittäisen *sprintin* malli.

5.1. Sopimusvaihe

Sopimusvaiheessa Ixonos luonnostelee sopimuksen asiakkaansa kanssa tehtävästä työstä sekä siihen liittyvistä yksityiskohdista. Luonnollisesti sopimuksen syntyminen on edellytys projektin käynnistämiselle. (Lehtinen 2007.)

Sopimusvaiheen aikana, kun varmuus on saavutettu sopimuksen syntymisestä, käynnistyy projektin *suunnitteluvaihe*. Suunnittelussa laaditaan perinteiset ohjelmistotuotantoprojektille tyypilliset asiat, joista tärkeimmät ovat alustava projektisuunnitelma, alustava tuotteen vaatimusten määrittely, tutkimus projektin toteutettavuudesta, työaika- ja työmääräarviointi, riskien arviointi, aikataulu sekä projektin päärooli en resursointi. (Lehtinen 2007.)

5.2. Projektin valmistelu

Kun sopimus on syntynyt, voidaan aloittaa projektin valmistelu konkreettisesti. Tämän vaiheen tarkoitus on saavuttaa projektin infrastruktuurille riittävän korkea maturiteettitaso, jotta voidaan aloittaa varsinainen tuotekehitystyö. Tarkoituksena on siis luoda sellainen työympäristö ja projektitiimin kompetenssi, joiden avulla varsinaista tuotetta voidaan alkaa suunnittelemaan ja työstämään. Maturiteettitaso varmistetaan yleensä erilaisilla etukäteen määritellyillä vaatimuksilla ja tarkistuslistoilla. Joskus myös asiakas saattaa tehdä tämän varmistuksen esimerkiksi auditoimalla yrityksen työtilat ja –prosessit. Lisätavoitteena on saavuttaa projektisuunnitelmalle sekä tuotteen vaatimusmäärittelyille hyväksyntä asiakkaalta. (Lehtinen 2007.)

5.3. Tuotanto – pääominaisuudet

Varsinainen tuotekehitystyö tehdään tuotannon työvaiheissa, joita kutsutaan *sprinteiksi* (Sprint). Jokainen *sprintti* sisältää vain yhden laajan tai muutaman pienen uuden ominaisuuden tuotekehitystyön sekä mahdollisia virhekorjauksia. *Sprintti* voi toisinaan

sisältää pelkästään virhekorjauksia. *Sprintin* sisältö määritellään aina sen mukaisesti, miten tärkeistä ominaisuuksista tai korjauksista on kulloinkin kyse. (Lehtinen 2007.)

Sprintti voidaan jakaa kolmeen eri työvaiheeseen:

1. Sprintin suunnittelu (Sprint Planning)
2. Työstövaihe (Working Days)
3. Julkaisuvaihe (Release Day).

Sprintin aikataulu on tarkasti ennalta määrätty. SW Design and Implementation -prosessissa yhden *sprintin* aikatauluksi on määritelty 10 työpäivää. (Lehtinen 2007.)

Tuotannon pääominaisuudet -vaiheessa keskitytään tuotteen toiminnallisuuden kannalta kriittisimpien ominaisuuksien tekemiseen. Nämä määräytyvät tuotteen käyttötarkoituksen ja tuotteelle suunnitellun loppukäyttäjäsegmentin perusteella. Pääominaisuuksiksi lasketaan lisäksi ne, joita ilman tuote ei voi toimia lainkaan. Tavoitteena on, että jokainen sprintti tuottaa täysin toimivan version tehdystä tuotteesta, joka tarvittaessa voidaan luovuttaa asiakkaan käyttöön. (Lehtinen 2007.)

5.4. Tuotanto – stabilisointi

Tämän vaiheen tavoitteena on saada kaikki vaatimusmäärittelyssä listatut ominaisuudet valmiiksi sekä saada viimeinen julkaistu versio toimitettua asiakkaalle. Tässä vaiheessa virheiden korjaukset ovat oleellisessa asemassa. Työ suoritetaan edelleenkin sprinteissä samojen periaatteiden mukaisesti kuin edellisessä vaiheessa. Tavoitteena tässäkin vaiheessa on, että jokainen sprintti tuottaa täysin toimivan version tehdystä tuotteesta, joka tarvittaessa voidaan luovuttaa asiakkaan käyttöön. (Lehtinen 2007.)

5.5. Tuotanto – hyväksyntä

Hyväksynnällä tässä vaiheessa tarkoitetaan nimenomaan projektiorganisaation omaa sisäistä hyväksyntää tuotteesta tehdylle viimeiselle versiolle. Kun kaikki suunnitellut ominaisuudet on tehty ja tuote on suunnitelmien mukaisesti testattu, varmistetaan projektisuunnitelmassa määriteltyjen kriteerien perusteella, että tuotteen viimeinen versio vastaa kaikkia sille tehtyjä määrittelyjä ja että tuote vastaa muiltakin osin sitä, mistä asiakkaan kanssa on sovittu. (Lehtinen 2007.)

5.6. Projektin päättäminen

Tämän vaiheen tärkeimpänä tavoitteena on saavuttaa asiakkaan hyväksyntä tehdylle tuotteelle. Hyväksyntätestauksen kautta asiakas voi varmistaa, että tuote vastaa juuri sitä mistä Ixonos Oyj:n ja asiakkaan välillä on sovittu. (Lehtinen 2007.)

Lisäksi tavoitteena on varmistaa asiakkaalle tietyn ylläpidon saatavuus tehdylle tuotteelle sopimuksessa määritellyn ajanjakson aikana. Tämä velvoittaa Ixonos Oyj:tä pitämään projektista saatua tietämystä saatavilla sovitun ylläpitojakson aikana. Projektista saatua tietämystä voidaan lisäksi hyödyntää uusissa projekteissa. Projektista kerätään ns. ”lessons learned” tietoa projektiorganisaatiolta eli arviointeja siitä, mitä asioita tehtiin hyvin, mitä olisi voitu tehdä paremmin sekä konkreettisia parannusehdotuksia. Projektin päättyessä erilaisten mittareiden sekä palautteen avulla kerätään tilasto- ja metriikkatietoa, jonka perusteella arvioidaan projektissa käytettyjä prosesseja mahdollista prosessien jatkokehitystyötä varten. (Lehtinen 2007.)

6. CMMI – PROSESSIEN KEHITYSMETODI

CMMI, eli Capability Maturity Model Integration, on Software Engineering Institute (SEI) nimisen organisaation luoma metodi, joka on tarkoitettu yritysten erilaisten prosessien kehittämiseen. Tätä metodia käytetään myös arvioitaessa yrityksen prosessikäytäntöjen maturiteettia, mikäli yritys on sopinut noudattavansa menetelmän mukaista prosessinhallintajärjestelmää. SEI:n järjestämien auditointitilaisuuksien avulla yritysten prosesseja arvioidaan *CMMI*-menetelmiin perustuen. Analysoinnin perusteella yritykselle annetaan erityinen tasoluokitus. Tasoluokituksen skaala on 0-5, jossa 0 tarkoittaa heikointa tasoa ja 5 kaikkein parasta. *CMMI*-menetelmää on mahdollista käyttää työkaluna kokonaan uuden prosessin luomiseen ja se valittiin tähän tutkielmatyöhön, koska se on Ixonos Oyj:n käyttämä prosessien kehitystyökalu.

CMMI kehitettiin vuonna 2002. Se perustui sitä edeltävälle SEI:n kehittämälle CMM-menetelmälle (Capability Maturity Model), joka oli joiltain osa-alueiltaan huono eikä soveltunut täysin ohjelmistotuotannon prosessien kehitykseen. (Chrissis, Konrad & Shrum 2003: 5-7.)

CMMI perustuu neljään osa-alueeseen:

1. Järjestelmän rakentaminen
2. Ohjelmiston rakentaminen
3. Integroitu tuotteen ja prosessin kehittäminen
4. Toimittajien valinta. (Chrissis ym. 2003: 7-9.)

Jokaiselle neljälle osa-alueelle on tehty lista prosessialueiden komponenteista, joita on tarkoitus kehittää. Komponentteja on lueteltu näissä listoissa useita kymmeniä. Näiden listojen perusteella voidaan kartoittaa yrityksessä jo olemassa olevat prosessit ja määritellä, mitä *CMMI*:n mukaisia komponentteja yrityksessä hoidetaan ja millä tavoin. Tätä kartoitusta varten on olemassa valmis dokumenttipohja, johon kirjataan yleisiä

tavoitteita (GG = Generic Goals) ja yleisiä toimintatapoja (GP = Generic Practises). Nämä numeroidaan jokaiselle komponentille dokumenttipohjaan ja kuvataan jokaisen komponentin osalta. Jokainen numeroitu tavoite tai toimintatapa otsikoidaan kuvaukseen perustuen. (Chrissis ym. 2003: 39-51.)

Lisäksi prosessialue jaetaan *CMMI*:ssä neljään osaan, jotka ovat:

1. Prosessin hallinta
2. Projektin hallinta
3. Rakentaminen
4. Ylläpito.

Näiden pohjalta tehdään kuvaus eri prosessialueiden välisistä relaatioista ja niiden välillä tapahtuvista aktiviteeteista. (Chrissis ym. 2003: 55.)

Kun yllämainitut kuvaukset ovat valmiina, tehdään näistä yhteenveto. Tähän *CMMI* tarjoaa kahta eri vaihtoehtoista tapaa esittää prosessin rakenne: *jatkuva* (Continuous) tai *vaiheittainen* (Staged). Molemmat esitystavat esittävät siis täysin samoja komponentteja, mutta eri näkökulmista. Tätä menettelyä suositellaan, koska useimmiten pelkästään toinen esitystapa ei sovellu kaikkiin tapauksiin. Erona näissä esitystavoissa on se, että *jatkuva* kuvaa prosessia sen pätevyyden näkökulmasta, kun taas *vaiheittainen* sen maturiteetin näkökulmasta. Näille molemmille näkökulmille on *CMMI*:ssä olemassa omat tasomäärittelyt asteikolla 0-5. Nämä tasomäärittelyt kuvaavat kunkin prosessin komponentin valmiusastetta jo olemassa olevassa prosessissa. Yllä mainittujen kuvausten perusteella tehdään tasomäärittely jokaiselle komponentille, jonka jälkeen pisteytyksen perusteella voidaan tulkita, mikä tietyn prosessin komponentin taso on ja onko se riittävä. (Chrissis ym. 2003: 73-96.)

7. TMAP – OHJELMISTOTESTAUSPROSESSIEN KEHITYSMETODI

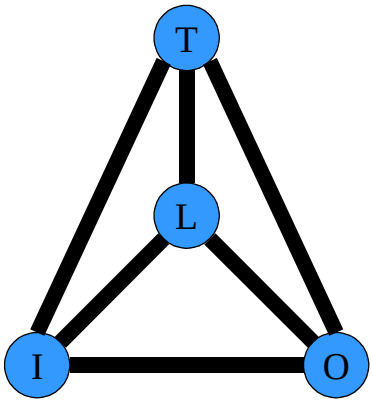
TMap tulee englannin kielen sanoista Test Management Approach. TMap-menetelmän on kehittänyt Sogeti niminen hollantilainen tietotekniikan alan yritys. *TMap* on tunnettu testausprosessien kehittämiseen suunniteltu menetelmä. Se on kehittynyt kohti standardisointia Benelux-maissa ja Saksassa ja se on suosittu myös muissa Keski-Euroopan maissa. *TMap*-menetelmää on käytetty menestyksekkäästi telekommunikaatioalan yrityksissä, mistä syystä tämä menetelmä valittiin yhdeksi työkaluksi tähän tutkielmatyöhön. *TMap*-menetelmää ei ole aiemmin kokeiltu Ixonos Oyj:llä, mistä syystä haluttiin tämän tutkielman yhteydessä selvittää, millä tavoin menetelmä sopisi yrityksen käyttöön.

TMapin avulla pyritään vastaamaan testauksen osalta sellaisiin kysymyksiin testauksen osalta kuin mitä, milloin, miten, missä ja kuka. Se on suunniteltu selvärakenteisen testauksen kehittämiseen. Tätä varten menetelmässä käytetään hyväksi kahta työkalua ja niille tehtäviä määrittelyjä. Toinen niistä on *TMapin* neljä kulmakiveä ja toinen yksi kulmakivistä eli testausprosessin elinkaarimalli.

TMap-menetelmä perustuu neljää ns. kulmakiveen:

1. Testauksen aktiviteeteilla on kehitysprosessiin liittyvä elinkaarimalli (L)
2. Luja organisaatiollinen sisäistäminen (O)
3. Oikeat resurssit ja infrastruktuuri (I)
4. Käyttökelpoiset tekniikat eri testauksen aktiviteeteille (T).

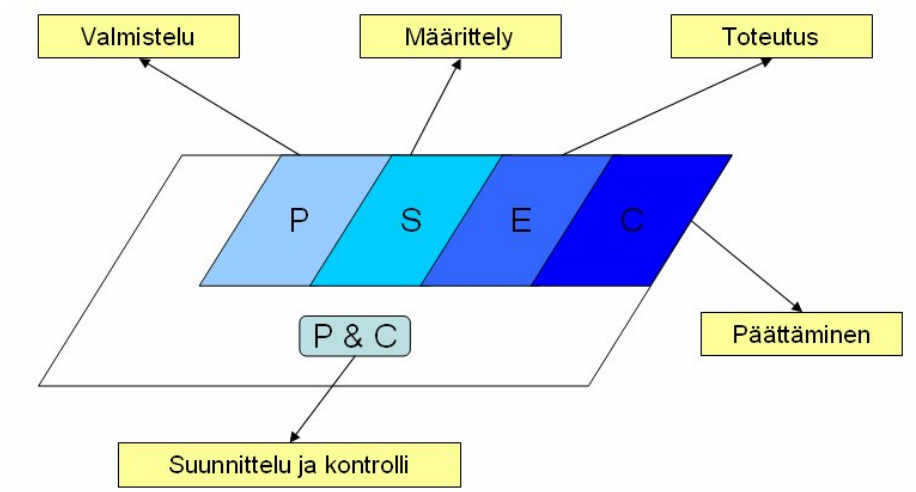
Näistä käytetään lyhenteitä L, O, I ja T. (Sogeti 2006).



Kuva 3. *TMap*:n neljä kulmakiveä. (Pol, Teunissen & van Veenendal 2002: 35.)

TMap kuvaa testausprosessia omalla elinkaarimallillaan, joka on yksi kulmakivistä. Loput kolme kulmakiveä määritellään elinkaarimallin avulla. Elinkaarimalli on jaettu viiteen eri vaiheeseen:

1. Suunnittelu ja kontrolli (Planning and control)
2. Valmistelu (Preparation)
3. Määrittely (Specification)
4. Toteutus (Execution)
5. Päättäminen (Completion).



Kuva 4. *TMap*:n mukainen testausprosessin elinkaarimalli. (Pol ym. 2002: 37.)

Elinkaarimalli pyritään määrittelemään testausprosessin jokaiselle testaustasolle erikseen, jotka edelleen kuvataan nivoutuneena ohjelmistotuotantoprosessin muihin eri vaiheisiin. Testausprosessin luonti *TMapin* avulla tukee siis erityisesti perinteisiä ohjelmistotuotantomalleja, kuten esimerkiksi V-mallia.

Elinkaarimallin eri vaiheille *TMap*-menetelmässä on määritelty tietyt perusaktiviteetit, jotka tulee suorittaa jokaisen vaiheen aikana. Näillä aktiviteeteilla pyritään varmistamaan, että testausprosessissa otetaan kattavasti huomioon tärkeimmät asiat testauksen suunnittelussa ja toteutuksessa. Vaiheiden aktiviteetteja on yhteensä 34. (Pol ym. 2002).

TMap-menetelmässä selvitetään vielä yksityiskohtaisemmin, miten aktiviteetit sekä kulmakivet määritellään ja dokumentoidaan. Näiden pohjalta luodaan lopullinen testausprosessi ja sen kuvaus, jota sitten pyritään noudattamaan testauksessa. (Pol ym. 2002)

8. BCS JA ISEB

BCS tulee sanoista British Computer Society. *BCS* on Iso-Britannian tietoteknisen alan kattojärjestö. Järjestö on perustettu vuonna 1957 ja siihen kuuluu nykyisin yli 60 000 jäsentä yli sadasta maasta. (British Computer Society 2007.)

BCS:n toiminta perustuu siihen, että se pyrkii luomaan tietotekniikan alalle standardeja sekä varmistamaan tietotekniikan alan työntekijöiden ammatillista tasoa erilaisten koulutusten ja niihin liittyvien sertifikaattien avulla. Lisäksi järjestö pyrkii yhteistyöhön muiden saman alan organisaatioiden kanssa sekä avustamaan ja tukemaan eri maiden tietotekniikkaan liittyviä strategioita. (British Computer Society 2007.)

ISEB tulee sanoista Information Systems Examinations Board ja on Iso-Britanniassa toimiva organisaatio, joka järjestää erilaisia tietotekniikan alan koulutuksia sekä niihin liittyviä sertifiointitestejä. Sen katsotaan syntyneen vuonna 1967 jolloin brittiläinen *NCC* (National Computing Centre) ja *BCS* alkoivat yhteistyössä järjestää tietotekniikka-alalle ensimmäisiä sertifiointikursseja. Nykyisin *ISEB*:n kautta on myönnetty yli 155 000 tietotekniikan alan sertifikaattia. (British Computer Society 2007.)

ISEB:n sertifikaatteja voi suorittaa seuraaville tietotekniikan alueille:

1. Projektinhallinta ja tuotteiden sekä projektien tukitoimet
2. Liiketoiminnan analysointi ja muutokset
3. Ohjelmistotestaus
4. Tietojärjestelmien kehitys ja arkkitehtuuri
5. Tietotekniikkapalveluiden hallinta
6. Varallisuuden ja infrastruktuurin hallinta
7. Tietotekniikkaoikeus ja sen soveltaminen
8. Tietoturvallisuus.

Näille aloille on mahdollista suorittaa useita eri sertifikaattitasoja. *ISEB*:n sertifikaattitasot ovat:

1. ISEB Aloittelijataso (ISEB Foundation Qualifications)
2. ISEB Perustaso (ISEB Essentials Qualifications)
3. ISEB Keskitaso (ISEB Practitioner Qualifications)
4. ISEB Ylätaso (ISEB Higher Level Qualifications)
5. ISEB Ammattilaistaso (ISEB Professional Level).

ISEB:n ohjelmistotestaussertifikaatit ovat kansainvälisesti hyvin tunnettuja ja tunnustettuja. Yksi syy tähän on se, että järjestön ohjelmistotestauskurssit ja sertifikaattikokeet perustuvat *BCS*:n luomiin ohjelmistotestausstandardeihin jotka ovat BS 7925-1 ja BS 7925-2. *ISEB*:n antamat ohjeistukset testauksen dokumentointiin perustuvat *IEEE*:n (Institute of Electrical and Electronics Engineers) standardiin IEEE 829. Testausalan terminologiaan antamat ohjeistukset perustuvat samaan *IEEE*:n standardiin. (Grove Consultants 2005; British Computer Society 2007.)

Ixonos Oyj:n ohjeistuksen mukaisesti ohjelmistotestauksessa tulisi noudattaa mahdollisimman paljon *ISEB*:n ohjelmistotestauskurssilla esiteltyjä käytäntöjä. Ixonos Oyj:n tavoitteena on saada mahdollisimman usealle testaajalle *ISEB*:n ohjelmistotestauksen aloittelijatason sertifikaatti.

9. TESTAUSPROSESSI

Tässä luvussa kuvataan tutkielman tuottama SW Design and Implementation -prosessille tehty testauksen aliprosessin kuvaus. Testauksen kuvaus on tehty noudattamaan SW Design and Implementation -prosessin mukaista sprinttien toteutustapaa ja etappeja. Lisäksi testausprosessi on suunniteltu huomioiden ketterien menetelmien yleisiä sekä XP- ja Scrum metodien periaatteita aivan kuten SW Design and Implementation -prosessikin. Testausprosessi on suunniteltu siten, että se on mahdollista teettää erillisenä omana projektinaan esimerkiksi jonkin toisen yrityksen luoman tuotekehitysprojektin osana. Tämä tietenkin edellyttää, että tällainen toisen yrityksen tuotekehitysprojekti noudattaa ketteriä menetelmiä ja sprinttien mukaisesti tapahtuvaa tuotekehitystä. Testausprosessin suunnittelussa on huomioitu Ixonos Oyj:n asettamat vaatimukset. Nämä luonnollisesti ovat asettaneet tiettyjä rajoituksia testausprosessin suunnitteluun.

9.1. Testausprosessin CMMI malli

Yrityksen suosituksesta testausprosessin suunnittelussa käytettiin apuna *CMMI*-menetelmää. Testausprosessille ei kuitenkaan luotu kokonaisvaltaista *CMMI* mallia, koska sellainen tehdään koko SW Design and Implementation -prosessille, kun testauksen aliprosessi on valmistunut. Malli arvioi siis jo valmista prosessia ja sen maturiteettia. Tämän työn yhteydessä tehtiin kuitenkin menetelmän avulla kartoitus siitä, millainen viitekehys tällä testausprosessilla on ja mitkä yrityksen muut prosessit vaikuttavat nyt laadittuun testausprosessiin. Tätä kartoitusta on tarkoitus käyttää hyväksi koko SW Design and Implementation -prosessin *CMMI* mallissa myöhemmin.

CMMI:n alueluokituksen mukaan testaus on osa tuotteen ja prosessin integroitua kehittämistä. *CMMI*-menetelmälle määritellyn *Integroitu kehittäminen tuotteelle ja prosessille* -alueen komponenteista testauksen kohdalla tunnistettiin niistä kahdeksan:

1. Projektin valvonta ja kontrollointi
2. Projektin suunnittelu
3. Riskien hallinta
4. Vaatimusten hallinta
5. Kokoonpanojen ja asetusten hallinta
6. Prosessin ja tuotteen laadun varmistaminen
7. Luotettavuus
8. Varmistukset.

Nämä *CMMI*:n mukaiset komponenttien osat on tulkittu olevan tämän kyseisen testausprosessin ulkopuoliset vaikuttajat, jotka otettiin huomioon testausprosessia suunniteltaessa. Pääasiassa nämä vaikuttavat tekijät huomioitiin testausprosessissa kahdella tavalla. Osa niistä osa-alueista, joihin näillä tekijöillä on eniten vaikutusta, ovat testauksen osalta vapaammin määriteltävissä projektikohtaisesti. Joidenkin osa-alueiden määrittelyt ovat yrityksen käytännöissä erityisen tiukasti ennalta määriteltyjä.

Projektin valvonta ja kontrolli sekä *projektin suunnittelu* määräytyy aina projektikohtaisesti. Joitain yleisiä päätason ohjeita ja sääntöjä tietenkin on, mutta projektin koko, luonne ja asiakas määräävät hyvin pitkälle sen, miten nämä *CMMI*:n mukaiset komponentit tulee hoitaa. Tästä syystä testausprosessiin ei haluttu tehdä liian tarkasti näiden komponenttien vaikuttamien osa-alueiden määrittelyjä, kuten esimerkiksi testauksen suunnittelua.

Riskien ja vaatimusten hallinnalla on suuri merkitys testaukselle. Testaukselle tulee tehdä oma riskikartoituksensa. Projektin yleiset riskit sekä projektin ulkopuolelta tunnistettavat riskit vaikuttavat luonnollisesti testausprosessiin. Riskejä ei kuitenkaan voida yleisellä tasolla etukäteen määritellä, koska ne ovat aina projektikohtaisia. Tästä syystä tässä työssä riskien hallinta on mainittu, mutta sitä ei ole tarkemmin määritelty. Riskien hallintaan yrityksellä on oma prosessinsa laadittuna. *Vaatimusten hallinta* on sekin täysin projektikohtaista. Vaatimusten hallintaan vaikuttaa ennen kaikkea yrityksen omat prosessit ja tätä varten valitut työkalut. Usein vaatimusten hallinta on asiakkaalla, jolloin projektiorganisaatio ei pysty itse siihen olennaisesti vaikuttamaan.

Kokoonpanojen ja asetusten hallintaan projektiorganisaatio ja testauksen osalta itse testaajat pystyvät vaikuttamaan eniten edellä listatuista *CMMI*:n komponenteista. Testauksen osalta tämä merkitsee projektille luotua infrastruktuuria, jonka puitteissa projektin testausorganisaatio toimii. Esimerkiksi versionhallinta, tuotteiden hallinta ja testausympäristön hallinta ovat tyypillisiä tämän komponentin piiriin kuuluvia asioita, joilla on suora vaikutus testaukseen. Nämä ovat projektikohtaisia asioita, joten niitä ei voi määritellä testausprosessiin yleisellä tasolla.

Prosessin ja tuotteen laadun varmistaminen on kaikkein olennaisin komponentti tämän työn kannalta. Testaus on laadun varmistamisen yksi tärkeimmistä ja parhaiten näkyvistä osa-alueista. Yrityksen tai asiakkaan luomat vaatimukset laadun varmistukselle on täytettävä. Nämä ovat osittain projektikohtaisia asioita, mutta monet osa-alueet määritellään yrityskohtaisen politiikan kautta. Prosessin laadun varmistaminen on erityisen olennaista tämän työn kannalta. Tämä onkin toteutunut siten, että esimerkiksi yrityksen laatupäällikkö on ollut yhtenä ohjaajana tälle työlle ja että nyt laadittua prosessia tullaan kehittämään lisää, kunhan siitä on saatu ensin kokemuksia pilottiprojekteissa. Yrityksen oman prosessienhallinnan ja -kehityksen kautta pyritään varmistamaan tämän nyt luodun prosessin laatua.

Luotettavuus ja varmistus on tulkittu tässä työssä siten, että testauksen on pystyttävä toimimaan muita sidosryhmiä kohtaan luotettavasti ja testauksen on toiminnallaan varmistettava tuotteen laatu. Testauksen ei saa tuottaa virheellisiä testaustuloksia ja testatusta tuotteesta löydetyt virheet on pystyttävä varmentamaan luotettavalla tavalla. Tämän kautta tuotteen maturiteetti voidaan luotettavalla tavalla selvittää ja tehdä tarvittavat toimenpiteet maturiteetin parantamiseksi.

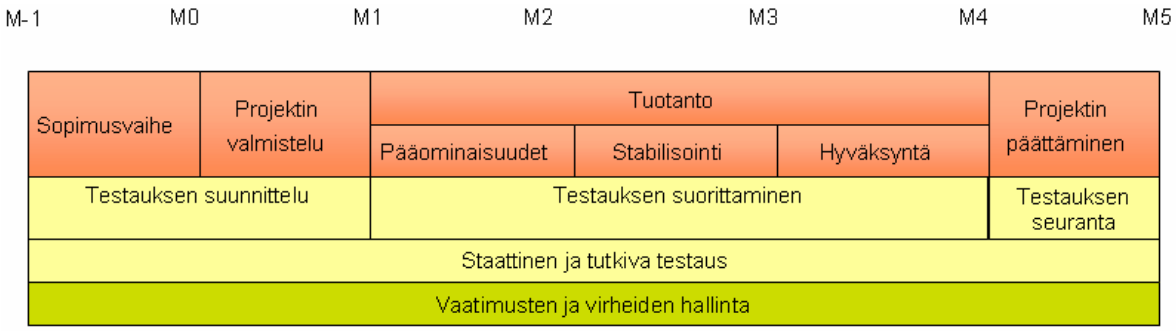
9.2. Testausprosessista yleisesti

Pääsääntöisesti testauksen strategian tulee perustua toiminnallisuuden varmistamiseen ja tästä syystä käyttötapaustestauksen (Use Case Testing) periaatteiden soveltamiseen. Lähtökohtana testaukselle on siis varmistaa, että tuote toimii siten kuin se on

määrittelyissä kuvattu toimivan. Testauksen suunnittelu pohjautuu käyttötapaustestauksen näkökulmaan, jossa lähdetään kartoittamaan loppukäyttäjän näkökulmasta aluksi käyttötapausskenaarioita ja myöhemmin näiden perusteella sitten luodaan testitapaukset. Tämä on hyvin yleisesti käytetty tekniikka niin kutsutussa *”musta laatikko”* -testauksessa (Black Box Testing). Käyttötapausten laatimiseen ja kartoittamiseen on olemassa erilaisia tekniikoita ja työkaluja, kuten esimerkiksi *UML* (Unified Modelling Language). (Copeland 2003.)

Testauksen *strategia ja näkökulma* ovat hyvin projektikohtaisia ja tästä syystä yllä mainittu strategia on vapaavalintainen, joten esimerkiksi käytetyt testausstrategiat voivat vaihdella suuresti projektien välillä. Yleensä projektipäällikkö, testauspäällikkö ja asiakas päättävät yhdessä, millaista testausstrategiaa ja näkökulmaa tullaan käyttämään, mutta testauspäällikön vastuulla on päätöksen jälkeen vastata siitä, että valittua strategiaa noudatetaan.

Varsinaisen testausprosessin suunnittelutyön alussa määriteltiin *testausprosessin vaiheet*, jotka noudattelevat SW Design and Implementation -prosessin vaiheita ja etappeja. Nämä vaiheet ovat *Testauksen suunnittelu* (Test Planning), *Testauksen suorittaminen* (Test Execution) sekä *Testauksen seuranta* (Test Follow-up). Lisäksi suunnittelun tuloksena määriteltiin *Staattinen ja Tutkiva testausvaihe* (Static and Exploratory Testing), joka on aktiivisena koko projektin elinkaaren ajan.



Kuva 5. SW Design and Implementation -prosessin testauksen vaiheet. Testauksen vaiheet on merkitty keltaisella pohjalla.

Staattinen ja tutkiva testaus jakaantuvat koko projektin elinkaaren ajalle. Ennen varsinaista sprinttien testaamista, etappien M -1 ja M1 välillä keskitytään testauksen suunnitteluun. *Testauksen suunnittelu* -vaiheen aikana aloitetaan testauksen valmistavat toimenpiteet. Näistä tärkein on päätason testaussuunnitelman (Master Test Plan) sekä alatasojen testaussuunnitelmien (Test Plan) laatiminen. Lisäksi voidaan laatia muita testausta tukevia suunnitelmia, kuten esimerkiksi testauksen automaatiosuunnitelma. Olennaisena osana Testauksen suunnittelu -vaihetta on testausympäristön ja -laboratorion rakentaminen sekä testaustiimin kompetenssin varmistaminen.

Sprinttien aikana, eli etappien M1 ja M4 välillä, toteutetaan testauksen seuraava vaihe eli *Testauksen suorittaminen*, joka sisältää pääasiassa varsinaisen tuotteelle tehtävän testaamisen. Yksittäisen sprintin testaus voidaan jakaa edelleen neljään osaan: yksikkötestaus, savutestaus, regressiotestaus sekä tutkiva testaus. Sprinttien aikana tehdään lisäksi koko ajan staattista testausta. Erityisesti etappien M3-M4 välillä suoritetaan sisäinen hyväksyntä, joka voi tarvittaessa sisältää myös sisäisen hyväksyntätestauksen.

Lopuksi testauksen aktiviteetit päätetään *Testauksen seuranta* -vaiheen aikana. Tämä vaihe sijoittuu etappien M4-M5 välille. Tämä viimeinen vaihe aloitetaan asiakkaan ulkoisella hyväksyntätestauksella. Viimeisimpinä toimenpiteinä on testausympäristön ja -laboratorion purkaminen sekä varsinaisen testaamisen päättäminen. Viimeisen vaiheen aikana tehdään jonkin verran vielä staattista testausta sekä mahdollisesti myös tutkivaa testausta.

Testausta varten projektiin tulee rekrytoida ainakin muutama hyvin kokenut seniori tasoinen testaaja. Erityisesti tutkiva testausvaihe vaatii, että testaajat ovat erityisen kokeneita ja tuntevat sekä käytettävät teknologiat että testauksen periaatteet syvällisesti. Lisäksi on suositeltavaa, että testaajat tuntevat testattavan tuotteen entuudestaan, mikäli kyse on tuotteen uusien versioiden tekemisestä.

Testaus on alati jatkuva prosessi koko projektin ajan. Jonkinlaista testausta tehdään koko projektin elinkaaren aikana. Jo suunnitteluvaiheessa aloitetaan staattisella

testauksella. Tutkivaa testausta tehdään tuotannon vaiheiden aikana aina, mikäli mitään muuta testausta ei olla tekemässä. Tutkivaa testausta on mahdollista tehdä jo *Testauksen suunnittelu* -vaiheen aikana tietyissä tilanteissa, kuten esimerkiksi silloin, jos ollaan tekemässä jo olemassa olevalle tuotteelle uutta versiota.

Projektiorganisaatiossa ei ole erikseen ohjelmoija- ja testaustiimejä vaan tiimeissä on sekä ohjelmoijia että testaajia. Tällä tavoitellaan erityisesti sitä, että ohjelmoijien ja testaajien keskinäinen päivittäinen kommunikaatio tehostuu. Tämä vaatimus pohjautuu ketterien menetelmien yleisiin periaatteisiin. Tiimin jäsenet voivat keskustella avoimesti ja ilman formaaleja kokouksia tuotteen ominaisuuksista sekä niiden testattavuudesta ja toteutettavuudesta. Tiimin jäsenet voivat myös antaa omista näkökulmistaan vinkkejä toisilleen, esimerkiksi testaaja voi havaita mahdollisen ongelman testauksen näkökulmasta ennen kuin ominaisuus on implementoitu ja tämän kautta avustaa ohjelmoijaa. Ohjelmoija voi avustaa testaajaa keksimällä uusia testitapauksia tai tapoja, miten joitain tiettyjä ominaisuuksia on mahdollista testata. Lisäksi on suositeltavaa, että ohjelmoijat ja testaajat tai ainakin yksittäinen tiimi kokonaisuudessaan, tekevät työtään samassa huoneessa.

Ohjelmoijat osallistuvat testaajien ohella testaukseen. Yksikkötestaus on ohjelmoijien vastuulla kuten yleensäkin ohjelmistotuotantoprojekteissa. Ohjelmoijat kuitenkin osallistuvat testauksen muuhunkin suunnitteluun siltä osin kuin ehtivät varsinaisen ohjelmointityön ohella. On suositeltavaa, että ohjelmoijat osallistuisivat tutkivaan testaukseen ja tekisivät myös testitapauskuvauksia, mikä on ominaista ketterien menetelmien XP-metodille.

Testausta pyritään automatisoimaan niin paljon kuin se on mahdollista. Tässä kuitenkin täytyy muistaa, että automatisointi ei saa olla itsetarkoitus vaan automatisoinnille täytyy olla vankat perustelut. On tarkasti mietittävä, mitä kannattaa automatisoida ja mitä ei. Automatisointiin vaikuttaa erityisen paljon se, millaista tuotetta ollaan testaamassa ja minkälaiset vaatimukset esimerkiksi asiakas on asettanut testaukselle. Automatisointi on järkevää silloin, jos halutaan ajaa yksinkertaisia testejä useita kertoja esimerkiksi stressitestauksessa. Sen sijaan ei ole järkevää automatisoida esimerkiksi kaikkea

käyttöliittymätestausta, jossa vaaditaan ihmisen tunnereaktioihin perustuvaa analysointia tuotteen käytettävyydestä. Mikäli automatisointi on tehokasta ajankäytön ja taloudellisuuden suhteen ja mikäli automatisoinnilla mahdollistetaan testaaminen 24 tuntia vuorokaudessa, voidaan automatisointia pitää perusteltuna. Automatisoidun testiajon ja automatisoinnin ylläpitoon ei saa kulua yhtä paljon tai jopa enemmän aikaa kuin mitä kuluisi testauksen manuaaliseen suorittamiseen. Automatisoinnin tulisi vapauttaa testauksen resursseja muun testaustyön suorittamiseen, jolloin automatisoinnin voidaan katsoa olevan tehokasta.

Jokaiselle sprintille tehdään oma karkean tason testaussuunnitelma, jossa testitapaukset kuvataan vain otsikkotasolla. Suurin osa testitapausten suunnittelusta perustuu tutkivaan testaukseen.

9.3. Testauksen suunnittelu

Testauksen suunnittelu on ensimmäinen kolmesta testauksen päävaiheesta ja ajoittuu SW Design and Implementation -prosessin M-1 ja M1 etappien välille. Testauksen suunnittelu voidaan aloittaa, kun Ixonos Oyj:n ja asiakkaan välinen sopimus on varmistunut. Testauksen suunnitteluvaiheessa suoritetaan alustavat toimenpiteet testauksen aloittamiselle sekä suunnitellaan, millä tavoin, millä välineillä ja kenen toimesta testaus tehdään.

Testaajien valinnassa tulee kiinnittää huomiota siihen, että vähintään muutama seniori tasoinen testaaja saadaan projektiin mukaan. Lisäksi testaajia rekrytoitaessa tulee kartoittaa heidän kompetenssitasonsa ja sen perusteella suunnitella mahdolliset lisäkoulutukset. Lisäksi testaajien tulee suunnitteluvaiheen aikana tutustua toteutettavaan tuotteeseen ja sen määrittelyihin, käytettäviin teknologioihin sekä työkaluihin. Testausympäristön ja -laboratorion rakentaminen on tehtävä suunnitteluvaiheen aikana ja varmistettava, että ympäristö sekä laboratorio toimivat oikein ja ovat luotettavia.

Koko projektin laajuinen testausmäärittely tulee tehdä heti suunnitteluvaiheen aikana, mikäli sellainen nähdään tarpeelliseksi. Koska projekti tullaan jakamaan sprintteihin, ei välttämättä ole tarpeen tehdä koko projektin kattavaa testausmäärittelyä, kunhan määrittelyt tehdään jokaiselle sprintille erikseen. Suunnitteluvaiheen aikana on tehtävä ensimmäisen sprintin testausmäärittely. Lisäksi täytyy tehdä testauksen automatisoinnin suunnittelu, mikäli testausta on tarkoitus automatisoida. Tätä varten täytyy arvioida automatisoinnin tavoiteltu taso. Arvioinnin täytyy kattaa analysointi siitä, mitkä testattavan tuotteen ominaisuuksien testaukset ja mitkä testauksen vaiheet voidaan automatisoida joko osittain tai kokonaan. Lisäksi tulee arvioida, mitä työkaluja automatisoinnissa voidaan käyttää sekä määritellä automatisointiin ja sen ylläpitoon käytettävä aika ja resurssit koko projektin ajalle.

Suunnitteluvaiheen aikana käynnistetään staattinen ja tarpeen mukaan tutkiva testaus. Staattista testausta tehdään pääasiassa projektin dokumenttien ja ohjelmistokoodien katselmointien kautta. Katselmoinnit suoritetaan yrityksen erikseen katselmoinneille määriteltyjen prosessien mukaisesti. Mikäli projektissa tehdään jollekin jo olemassa olevalle tuotteelle uutta versioita, voidaan myös staattista testausta suorittaa tuotteen olemassa olevalle ohjelmakoodille staattisen analysoinnin muodossa. Tutkivaa testausta voidaan suorittaa jo olemassa olevalle tuotteelle ja sitä kautta tutkia esimerkiksi millaisia parannuksia toiminnallisuuteen voitaisiin tehdä. Tällöin testaus tukee projektille tehtävää vaatimusmäärittelyihin liittyvää suunnittelutyötä.

Tärkein yksittäinen tehtävä suunnitteluvaiheen aikana on päätason testaussuunnitelman luominen. Suunnitelma voi olla oma dokumenttinsa, mutta varsinkin pienikokoisissa projekteissa se on usein varsinaisen projektisuunnitelman osa. Ixonos Oyj:llä on olemassa valmis pohja päätason testaussuunnitelmalle, jota tulisi aina pääsääntöisesti käyttää. Joissain tapauksissa se ei välttämättä ole aina kuitenkaan mahdollista tai kaikkia dokumenttipohjassa määriteltyjä asioita ei tarvitse tai ei voi käyttää uudessa projektissa. Testauksen laadun ja seurattavuuden kannalta on kuitenkin tärkeää, että tietyt tärkeimmät määrittelyt tehdään päätason testaussuunnitelmaan.

Ensimmäiseksi tulisi miettiä, mitä testausstrategiaa käytetään. On tärkeää määritellä esimerkiksi, mikä on testauksen näkökulma, miten testauksen laajuus määritellään ja mitä testaustekniikoita halutaan käyttää? Testausstrategian määritteleminen helpottaa testauksen hallinnointia ja ennustettavuutta olennaisesti. Eräitä hyvin yleisiä testausstrategioita on esimerkiksi suunnitella testaus tuotteen toiminnallisuuden varmistamisen näkökulmasta tai keskittyä nimenomaan virheiden etsimiseen. Testausstrategian tätä prosessia käytettäessä tulisi kuitenkin aina ottaa huomioon ketterien menetelmien erityispiirteet ja periaatteet.

Eräs tärkeä asia määriteltäväksi on, millaisia ulkoisia ja sisäisiä testaukseen vaikuttavia asioita tunnistetaan. On hyvä listata ne projektiorganisaation vaikutusvallan ulkopuolella olevat asiat, joilla voi olla merkitystä testauksen suorittamiseen ja ehkä jopa lopputulokseenkin. Tällaisia asioita ovat esimerkiksi standardit, asiakkaan asettamat vaatimukset ja työkalut, kolmannen osapuolen toimittamat laitteet ja ohjelmistot sekä kontrolloimattomat ympäristöt (esimerkiksi kaupalliset verkkoyhteydet, yhteiskäyttöiset laboratoriot jne.). Erityisen tärkeää on tunnistaa ne asiat, jotka vaikuttavat negatiivisella tavalla testauksen konkreettiseen suorittamiseen tai testien lopputuloksiin. Esimerkiksi jos testauksessa joudutaan käyttämään ulkopuolisen tarjoamaa kaupallista verkkoyhteyttä, ei voida vaikuttaa omalla toiminnalla verkon käyttökatkoksia aiheuttavien ongelmien ratkeamiseen.

Erilaisten testiajovaiheiden kartoittaminen on olennaista tehdä päätason testaussuunnitelmaan. Ennen konkreettisen testauksen tekemistä täytyy miettiä, minkä tyyppiset testiajot ovat testattavan tuotteen ja sen ominaisuuksien kannalta tarpeen. Erilaisia testiajovaiheita ovat esimerkiksi toiminnallisuustestaus, epätoiminnallisuustestaus, integraatiotestaus tai suorituskykytestaus. Tarvittaessa eri testiajovaiheille voidaan laatia omat testaussuunnitelmansa.

Aikataulut, sekä kriteerit, minkä mukaan aikataulut määräytyvät, tulisi aina sisältyä päätason testaussuunnitelmaan. Yleensä testauksen aikatauluja on hyvin vaikea määritellä kovin tarkalla tasolla, koska etukäteen ei voida tietää, miten paljon ja minkä tasoisia virheitä testauksessa löytyy, varsinkin jos testattava tuote on täysin uusi.

Virheiden laatu ja määrä vaikuttavat aina oleellisesti testauksen aikatauluihin. Jos erityisen vakavia virheitä löytyy, saatetaan koko testaus keskeyttää.

Testauksen henkilöresurssien määrä ja tarve pitää olla määriteltynä, sekä heidän roolinsa ja vastuualueensa. Jokaisen testaajan, kuten kaikkien muidenkin projektiorganisaatiossa olevien, tulee tietää tarkasti, mikä heidän roolinsa on ja mitä heiltä odotetaan projektin suhteen. Usein testauksen henkilöresurssit määritellään jo projektisuunnitelmassa, erityisesti pienissä projekteissa.

Testaukseen liittyvät raportointikäytännöt tulee määritellä eli kuka raportoi sekä miten, kenelle ja milloin raportoidaan. On tärkeää, että tämä on erityisen hyvin määritelty, koska testauksen tärkein tuotos on nimenomaan erilaiset raportit ja niihin sisältyvät metriikat ja statistiikat. Raporttien avulla nähdään, miten hyvä tuotteen maturiteetti on ja raporttien perusteella tehdään useita projektin kannalta olennaisen tärkeitä päätöksiä. Näillä päätöksillä on usein suora vaikutus projektin aikatauluihin sekä tuottavuuteen. Lisäksi raporttien perusteella arvioidaan usein testauksen tehokkuutta.

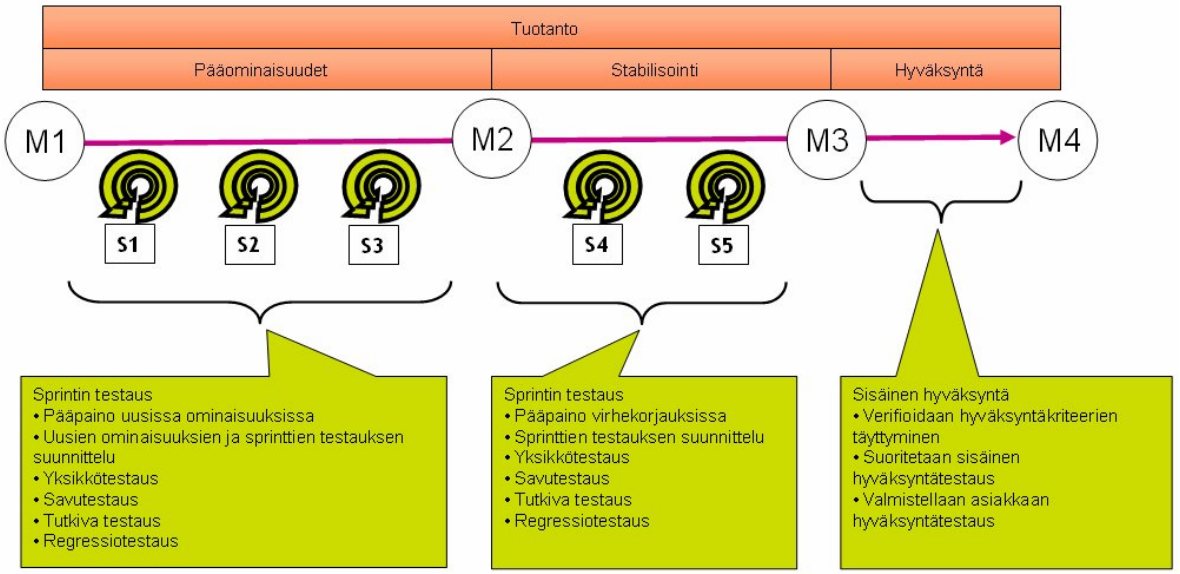
Testauskohtaisten riskien tunnistaminen ja niiden listaaminen on tärkeää. Päätason testaussuunnitelmassa on hyvä olla määriteltynä, mitä projektiorganisaatio tunnistaa riskeiksi testauksen osalta. Tämän kartoituksen avulla testaussuunnittelu, kuten tarvittaessa koko projektisuunnittelu, voidaan toteuttaa niin, että riskien muuttuminen ongelmiksi projektin aikana pystytään hyvissä ajoin estämään tai ainakin niihin pystytään riittävästi varautumaan.

Lopuksi olisi hyvä määritellä testaukselle olennaisia kriteerejä. Näistä tärkeimmät ovat testauksen kriteerit eri etapeille sekä testauksen aloittamis- ja päättämiskriteerit eri testauksen vaiheille. Etapeille määriteltävät kriteerit ovat yleensä testauksenkin osalta osa projektisuunnitelmaa, mutta ne on mahdollista määritellä myös päätason testaussuunnitelmaan. Näillä etappikohtaisilla kriteereillä määritellään toteutumatasot eri testauksen työvaiheille, jotta voidaan hyväksyä etappi testauksen osalta saavutetuksi. Esimerkiksi etapin M4 hyväksymiselle testauksen yhtenä kriteerinä voisi olla kaikkien sprinttien testiajovaiheiden suorittaminen. Yksittäiselle testiajovaiheelle on tärkeää

määritellä aloittamis- ja päättämiskriteerit. Näillä tarkoitetaan sitä, että tietyt ehdot täytyy ensin täyttyä ennen kuin testausta tuotteelle voidaan aloittaa tai ennen kuin se voidaan lopettaa. Esimerkiksi aloittamiskriteereinä sprintissä tehtävälle savutestaukselle voisi olla, että yksikkötestaus on suoritettu loppuun ja ettei yksikkötestauksen yhteydessä ole löytynyt vakavia virheitä. Kriteerit riippuvat luonnollisesti hyvin paljon siitä, millainen projekti, tuote ja asiakas ovat kyseessä.

9.4. Testauksen suorittaminen

Varsinainen testaustyö suoritetaan SW Design and Implementation -prosessin etappien M1 ja M4 välillä. Testauksien suorittamiseen liittyvät aikataulut määritellään sprinttien aikataulujen perusteella. Testauksen osalta voidaan tunnistaa karkealla tasolla seuraavat työvaiheet: sprinttikohtaisen testauksen määrittely, sprintin testaus (staattinen testaus, yksikkötestaus, savutestaus, tutkiva testaus sekä regressiotestaus), virheiden ja virhekorjausten verifiointi sekä raportointi ja testaukseen liittyvän statistiikan ja metriikan kerääminen.



Kuva 6. Testauksen suorittaminen SW Design and Implementation -prosessin Tuotanto -vaiheen aikana.

9.4.1. Sprinteistä yleisesti

Sprinttien käsite tässä prosessissa määräytyy ketterien menetelmien Scrum metodin pohjalta ja pyrkii noudattamaan pääsääntöisesti Scrum metodin periaatteita. Jokaiselle sprintille tehdään karkean tason testaussuunnitelma, jossa määritellään *sprinttiin* tuleville uusille ominaisuuksille testitapaukset otsikkotasolla. Lisäksi testaussuunnitelmassa määritellään yksittäisen *sprintin* testaukseen liittyvät erityisvaatimukset. Pääosin *sprintille* tehtävä testaussuunnitelma noudattelee aina samaa pohjaa. Ixonos Oyj:n määrittelemää testaussuunnitelmapohjaa tulisi pääsääntöisesti käyttää aina, kun se vain projektin suhteen on mahdollista.

Kun yksittäiseen *sprinttiin* suunniteltu toiminnallisuus on saatu toteutettua, voidaan katsoa sprintin olevan valmis. Tämä edellyttää sitä, että tuotteeseen on implementoitu kaikki suunniteltu toiminnallisuus, tuotteesta on onnistuneesti pystytty kääntämään toimiva ohjelmaversio ja se on onnistuneesti testattu. *Sprintin* valmistumisen määrittelyyn voidaan käyttää muitakin kriteereitä. Nämä ovat usein projektikohtaisia. Tässä vaiheessa tutkivaa testausta voidaan edelleen jatkaa. Päättyneen *sprintin* tutkivan testauksen pohjalta luoduista testitapauksista sekä aiempien sprinttien regressiotestauksen testitapauksista ja savutestauksen testitapauksista luodaan seuraavalle sprintille uudet regressiotestauksen ja savutestauksen testisetit.

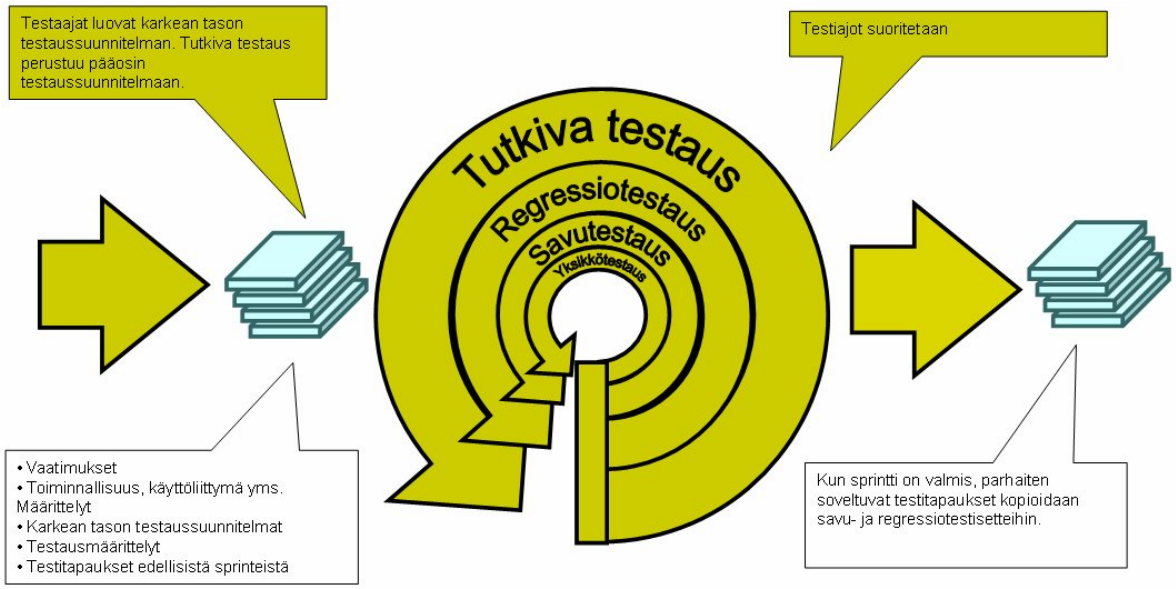
9.4.2. Sprintin testaus

Sprintin testaus (Sprint Testing) korvaa perinteisessä V-prosessissa tai vesiputousmallissa kuvatun järjestelmätestauksen. Sprintin testaus voi sisältää sekä *toiminnallisuus-* (Functional testing) että *epätoiminnallisuustestausta* (Non-functional testing). Epätoiminnallisuustestausta ei kuitenkaan tule aloittaa ennen kuin pääominaisuudet on lisätty tuotteeseen, toiminnallisuustestaus on suoritettu loppuun eikä mitään kriittisen tason virheitä ole avoinna.

Sprintin testaus voidaan tarvittaessa jakaa erilaisiin alatasoihin ja -vaiheisiin. Esimerkiksi regressiotestauksessa voidaan käyttää erilaisia testiajovaiheita kuten

käyttöliittymätestausta ja kuormitustestausta. Päätasolla sprintin testaus sisältää yksikkötestauksen, savutestauksen, tutkivan testauksen sekä regressiotestauksen. Testausta pyritään suorittamaan edellä mainittujen päätason vaiheiden osalta ohjelmien kääntämisprosessin yhteydessä mahdollisimman paljon automatisoidusti, mikäli mahdollista.

Päätavoitteena SW Design and Implementation -prosessin etappien M1–M2 välillä on testata uusi toiminnallisuus sekä vanha toiminnallisuus, jota on muokattu. Etappien M2–M3 välillä päätavoitteena on varmistaa, että virheiden korjaukset toimivat ja etteivät virhekorjaukset ole luoneet uusia virheitä.



Kuva 7. Yksittäisen sprintin testaaminen.

9.4.3. Staattinen testaus

Staattinen testaus (Static Testing) määriteltiin prosessiin mukaan ISEB:n määrittelyjen mukaisesti. ISEB:n mukaan staattista testausta tulee tehdä aina jokaisessa ohjelmistotuotantoprojektissa. Staattista testausta tehdään koko projektin elinkaaren aikana. Staattinen testaus ei sisällä konkreettista tuotteen käyttöä tai käsittelyä, esimerkiksi ohjelman ajamista, kuten Erik van Veenendaal toteaa kirjassaan The Testing Practitioner (2002: 199). Staattinen testaus koostuu pääasiassa ohjelmakoodin ja

projektin dokumenttien katselmoinneista ja tarkistuksista sekä staattisista ohjelmakoodin analyyseista (van Veenendaal 2002: 199-200). Katselmoinnit, tarkistukset ja staattiset analyysit tulee suorittaa Ixonos Oyj:n määrittelemien erillisten prosessien ja työkalujen avulla. Tämä on projektikohtaista, koska ulkoisten vaatimusten vuoksi näistä prosesseista saatetaan joutua poikkeamaan. Staattisen testauksen tarkoituksena on lisäksi kerätä statistiikkaa ja metriikkaa tuotekehityksen ja prosessianalyysien avuksi.

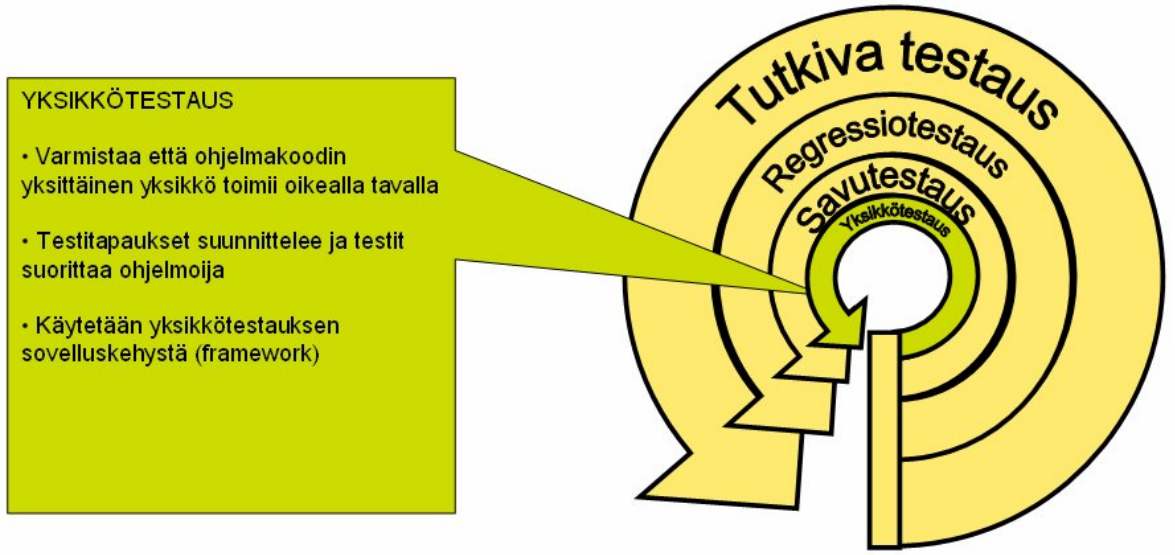
Myös *dynaamista analysointia ja testausta* voidaan tehdä. Tämä ei sinänsä kuulu staattisen testauksen piiriin. Ixonos Oyj:n kanta dynaamisen analysoinnin suhteen ei ollut vielä tätä tutkielmaa tehdessä selvillä, joten dynaamista analysointia ei ole otettu mukaan omana erillisenä osiona tälle testausprosessille. Ixonos Oyj halusi dynaamisen analysoinnin kuitenkin mainita, joten siksi se on tässä työssä mukana. Dynaamisen analyysin avulla pyritään selvittämään esimerkiksi tuotteessa olevan lähdekoodin eri haarojen kattavuus testauksen näkökulmasta. Yleisimmin tällainen verifiointi tehdään yksikkötestauksen yhteydessä, mutta joidenkin tuotteiden kohdalla tällainen analyysi voidaan suorittaa myös muun testauksen yhteydessä. Mikäli dynaamista analyysia käytetään, tulisi kuitenkin muistaa, että se ei saa ohjata testauksen suunnittelua, koska se ei silti anna välttämättä kaikissa tilanteissa todellista kuvaa itse testauksen laadusta ja kattavuudesta. Lähdekoodin eri haarojen kattavuus ei ole sama asia kuin testauksen kattavuus. Esimerkiksi, jos lähdekoodin haarojen kattavuutta mitataan kahden eri testisetin osalta ja todetaan, että suurimaksi osaksi molemmat setit kattavat samoja haaroja, on virheellistä suoraan vetää tästä johtopäätös, että molemmat setit testaavat samoja asioita. Tällaisessa tilanteessa voi kyse olla esimerkiksi siitä, että toinen testisetti testaa vain tiettyjä perustoiminnallisuuksia ja toinen testisetti samojen toiminnallisuuksien erilaisia variaatioita, jolloin lähdekoodin näkökulmasta molemmissa seteissä joudutaan pääsääntöisesti käymään samoissa koodin haaroissa, mutta täysin eri käyttötapausten yhteydessä. Dynaaminen analyysi tulisi olla vain osa projektin prosesseja, jonka avulla testausta ja tuotekehitystä tuetaan.

9.4.4. Yksikkötestaus

Yksikkötestauksen (Unit Testing) tavoitteena on varmistaa, että yksittäinen osakokonaisuus koodista toimii oikealla tavalla. Yksiköllä voidaan tarkoittaa myös yksittäistä pientä ohjelmaa, ohjelman funktiota tai proseduuria. *Yksikkötestausta* varten yleensä jo ohjelmointivaiheessa pyritään luomaan sitä tukeva sovelluskehys itse ohjelman koodiin eli niin kutsuttu *framework*. (Grove Consultants 2005.)

Tällaisen sovelluskehyksen avulla itse *yksikkötestaus* yleensä aina automatisoidaan tai ainakin siihen tulisi pyrkiä. Ohjelmoija on itse vastuussa yksikkötestauksen suunnittelusta ja suorittamisesta. *Yksikkötestaukselle* kerätyt testitapaukset kootaan omaksi setikseen projektissa ennalta määrättyjen kokonaisuuksien mukaan.

Tässä prosessissa *yksikkötestauksella* on erityisen korostunut rooli, sillä tältä osin prosessissa noudatetaan ketterien menetelmien XP-metodia. Eräs XP-metodin periaatteita on vahvasti korostunut yksikkötestaus. Tästä johtuen tuotteelle valitussa ohjelmointimenetelmässä ja -arkkitehtuurissa pitää ottaa huomioon yksikkötestauksen vaatimukset. Koodi täytyy suunnitella testauslähtöisesti niin, että se on helposti testattavissa jokaisen käännöksen aikana yksikkötestien avulla automaattisesti.



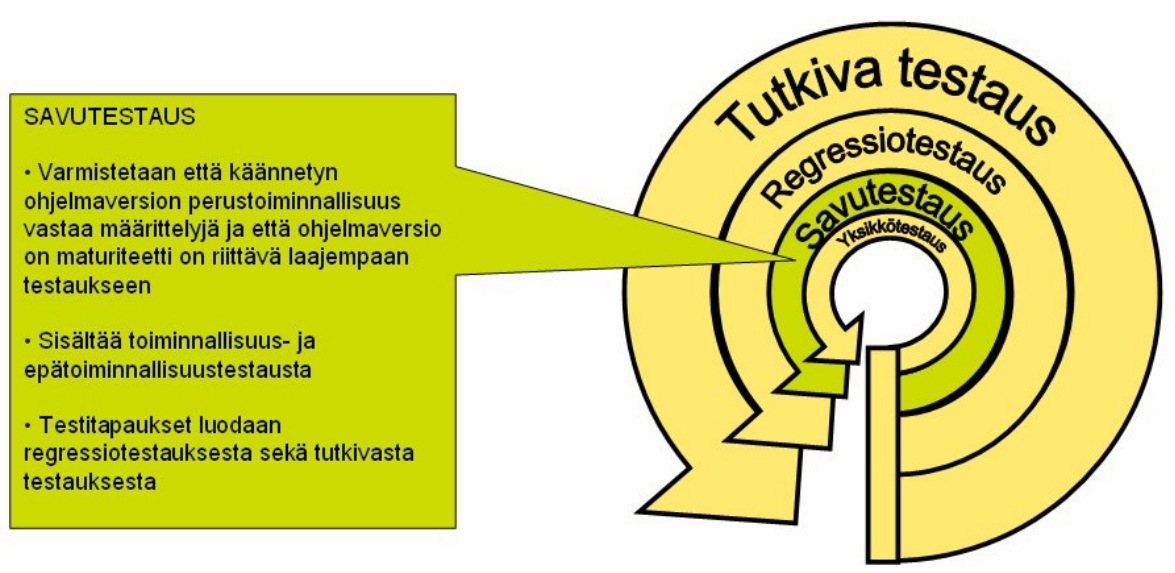
Kuva 8. Yksikkötestaus.

9.4.5. Savutestaus

Savutestauksen (Smoke Testing) tavoitteena on varmistua, että sprintille tehdyn ohjelmaversion perustoiminnallisuus toimii määrittelyjen mukaisesti ja että ohjelmaversion maturiteetti on riittävän korkea sekä testausta varten että myös seuraavan ohjelmaversion pohjaksi. (Rex 2004.)

Savutestaus voi sisältää sekä toiminnallisuus- että epätoiminnallisuustestausta. Testitapaukset savutestaukseen valitaan jokaiselle sprintille erikseen edellisen sprintin regressiotestauksen, tutkivan testauksen ja savutestauksen testitapauksien joukosta. Näin valituista testitapauksista muodostetaan yksittäiselle sprintille oma savutestauksen testisetti.

Testitapausten määrä täytyy pitää alhaisena. *Savutestauksessa* ei saa kuluttaa liian paljon aikaa, jotta muu testaustyö voidaan käynnistää mahdollisimman nopeasti. Testauksen suorittamiseen kuluva aika ei saa ylittää projektille ennalta määrättyä savutestauksen suoritusaikataulua. Yleensä savutestauksen suorittaminen kestää alle neljä tuntia.



Kuva 9. Savutestaus.

9.4.6. Regressiotestaus

Regressiotestauksen (Regression Testing) tavoitteena on varmistaa, että tuotteessa oleva vanha toiminnallisuus, johon ei ole tehty mitään muutoksia meneillään olevan sprintin aikana, toimii edelleen määrittelyjen mukaisesti. Tällä pyritään tarkistamaan, ettei uusi lisätty ominaisuus tai muokattu toiminnallisuus ole rikkonut mitään muuta tuotteessa olevaa toiminnallisuutta tai ominaisuutta. (Rex 2002.) *Regressiotestauksessa* ajetaan uudelleen edellisestä sprintistä valikoidut testitapaukset. Testitapauksia valikoidaan edellisen sprintin savu-, regressio- ja tutkivan testauksen seteistä ja niistä muodostetaan uusi *regressiotestauksen* testisetti. Testauksen aikataulu ei saa ylittää projektille ennalta määriteltyä regressiotestauksen aikataulua. *Regressiotestausta* voidaan suorittaa samanaikaisesti kuin mitä tahansa muuta sprintille suunniteltua testausta.

Kuva 10. Regressiotestaus.

9.4.7. Tutkiva testaus

Tutkiva testaus (Exploratory Testing) pohjautuu erityisesti ketteriin menetelmiin ja korostaa niitä eniten testausprosessin muihin vaiheisiin ja osa-alueisiin verrattuna. *Tutkiva testaus* noudattelee osittain sekä XP että Scrum metodien periaatteita.

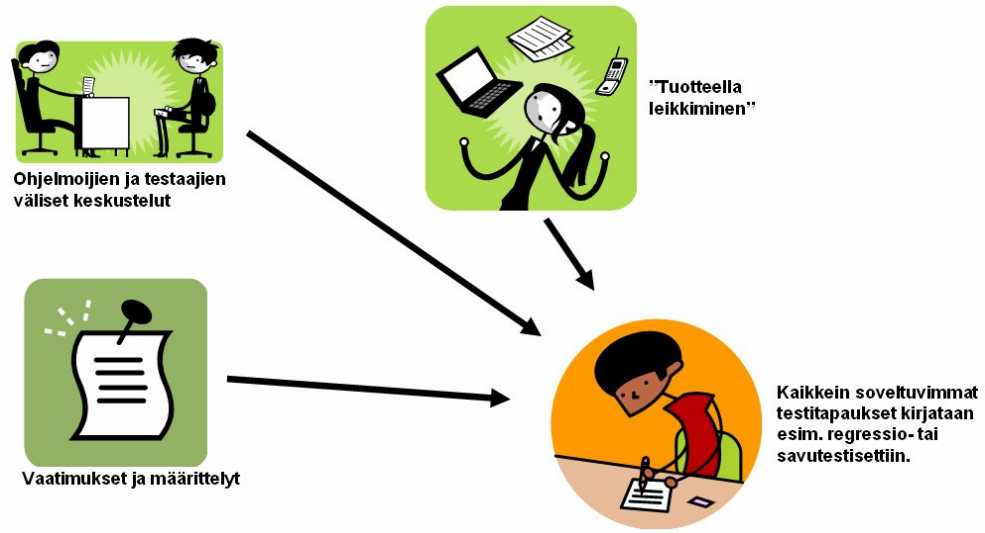
Tutkivaa testausta suoritetaan miltei koko projektin elinkaaren ajan. Tutkiva testaus kuitenkin korostuu erityisesti sprinttien testauksessa. Tuotteen *intuitiivinen testaus* on eräs tutkivan testauksen pääpiirteistä. Tutkiva testaus tapahtuu pääosin tuotteella ”leikkimällä”. Sekä testaajat että ohjelmoijat osallistuvat testaukseen. He ottavat tuotteen käyttöönsä ja alkavat kokeilla, mitä kaikkea sillä voi tehdä. Tarkoituksen on luoda tilanteita, miten heidän mielestään loppukäyttäjä saattaisi tuotetta käyttää. Tällä tavoin pyritään analysoimaan esimerkiksi, mitkä tuotteen alueet ovat erityisen herkkiä virheille. Samoin tällaisen ”leikkimisen” kautta he saavat uusia ideoita, miten jokin ominaisuus voitaisiin tehdä paremmin tai mitä mahdollisia uusia ominaisuuksia kannattaisi tuotteeseen lisätä. (van Veenendaal 2002: 261-273.)

Tutkivan testauksen aikana voidaan virallisten ohjelmapakettien lisäksi käyttää ohjelmasta tehtyjä väliversioita ja epävirallisia käännöksiä. Tutkiva testaus voi sisältää sekä toiminnallisuus- että epätoiminnallisuustestausta. (van Veenendaal 2002: 261-273.)

Ennakoiva testaus on osa tutkivaa testausta. Testaajat keskustelevat ohjelmoijien kanssa jo suunnitteluvaiheesta lähtien tuotteen tekemiseen liittyvistä yksityiskohdista. Näiden keskustelujen ja erilaisten määrittelyjen pohjalta he yhdessä miettivät, millä tavoin mitäkin kokonaisuuksia voitaisiin testata ja mitkä osa-alueet ovat mahdollisesti erityisen virheherkkiä. Lisäksi keskusteluja käydään siitä, millä tavoin vaatimusmäärittelyjen mukaiset toiminnallisuudet voitaisiin toteuttaa itse tuotteeseen niin, että se parhaiten soveltuisi loppukäyttäjän näkökulmasta käytettäväksi.

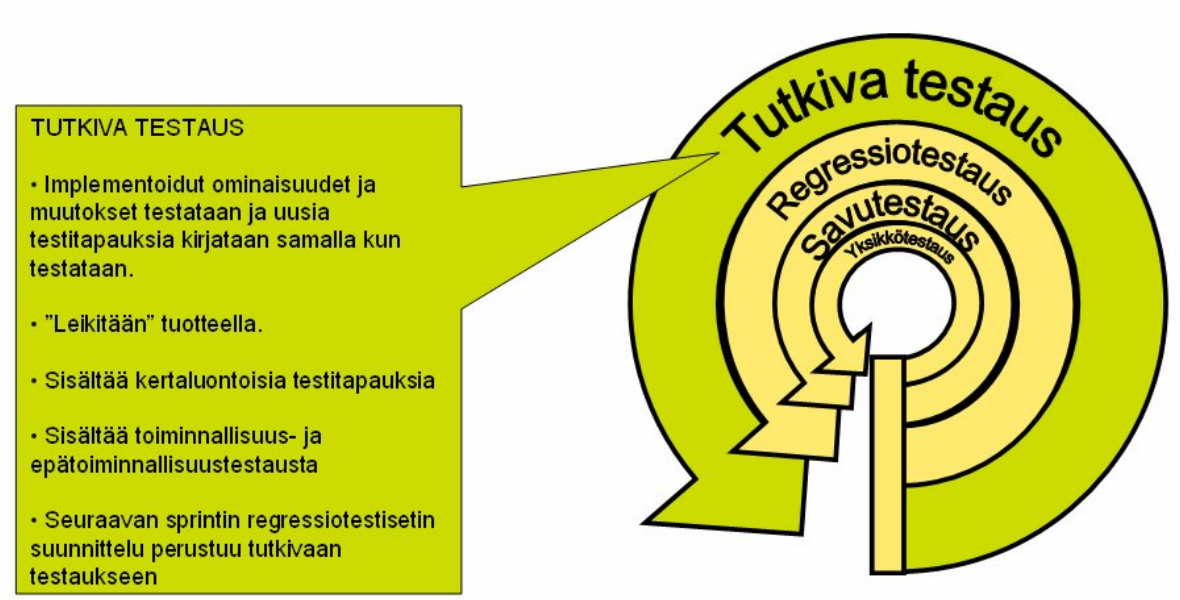
Tutkivan testausvaiheen aikana ei käytetä perinteisiä testitapauskuvauksia lainkaan. Testitapaukset perustuvat uutta toiminnallisuutta kuvaaviin määrittelyihin, testaajien ja ohjelmoijien välisiin keskusteluihin sekä tuotteella ”leikkimisen” kautta saatuihin ideoihin. Testitapaukset määritellään edellä mainittujen asioiden pohjalta pelkästään otsikkotasolla ja vasta itse varsinaisen testauksen jälkeen parhaimmat ja käytännöllisimmät testitapaukset kirjataan tarkemmiksi testitapauskuvauksiksi regressiotestauksen ja savutestauksen testisetteihin. Lisäksi ne testitapaukset, joiden kohdalla löydetään virhe, täytyy kirjata tarkemmaksi testitapauskuvaukseksi toistettavuuden varmistamiseksi. Tutkivan testauksen aikana suoritetaan useita

kertakäyttöisiä testitapauksia, jotka ajetaan vain kerran sprintin ja koko projektin aikana. Tällaisilla kertaluontoisilla testitapauksilla varmistetaan toiminnallisuutta, jonka ei katsota olevan kovin kriittinen tuotteen kokonaisuuden kannalta ja johon ei olla tekemässä muutoksia enää projektin myöhemmissä vaiheissa. Testauksen laajuuden tulisi pääsääntöisesti kattaa sprintin testauksen aikana sprintissä tuotettu uusi toiminnallisuus. Tämän lisäksi tutkivalla testauksella voidaan kartoittaa myös muita tuotteen osa-alueita, mutta näiden testaus tulee olla toissijaista.



Kuva 11. Testitapausten määrittely tutkivan testauksen avulla.

Löydetyt virheet raportoidaan aina ohjelmoijille, mutta ei välttämättä virallisen virheraportointi prosessin kautta. Tämä riippuu siitä, onko virhe löydetty esimerkiksi virallisen vai epävirallisen ohjelmapaketin testauksen yhteydessä. Lisäksi virheen laatu ratkaisee, käytetäänkö virallista virheraportointiprosessia. Jos löydetyn virheen laatu on vakava tai hyvin kriittinen, tulee tällaiset virheet aina tilanteesta riippumatta raportoida virallisen prosessin kautta.



Kuva 12. Tutkiva testaus.

9.4.8. Sisäinen hyväksyntä

Sisäisen hyväksynnän (Internal Acceptance) tarkoitus on varmistaa, että projektiorganisaation omasta mielestä tuotteen laatu ja toiminnallisuus vastaavat sitä, mistä on sovittu asiakkaan kanssa. Kriteerit sisäiselle hyväksynnälle määritellään projektisuunnitelmassa. Yleensä sisäinen hyväksyntä katetaan ennalta määritellyn tarkistuslistan avulla. Tällaiseen tarkistuslistaan määritellään tietyt tavoitteet. Jos nämä tavoitteet on saavutettu, voidaan projektin puolesta antaa sisäinen hyväksyntä tuotteelle. Kaikkien ominaisuuksien implementointi tuotteeseen ja testauksen avulla varmistettu ennalta määritelty maturiteettitaso ovat yleisimpiä hyväksymiskriteerejä.

Joskus voidaan lisäksi suorittaa *sisäinen hyväksyntätestaus*. Tämä ei yleensä ole tarpeen, mutta joissain projekteissa se on suositeltavaa. Sisäistä hyväksyntätestausta ei voida suorittaa ennen kuin kaikki muut sisäiselle hyväksynnälle asetetut kriteerit on täytetty. Tällaista testausta varten täytyy määritellä erillinen oma testisettinsä. Testitapaukset valitaan yleensä sen perusteella, että ne testaavat perustoiminnallisuuden sekä erityisen kriittiset ja virheherkät tuotteen osa-alueet. Ajettavien testien määrän ei

tarvitse olla kovin suuri, koska yleensä tässä projektin vaiheessa tuote on jo moneen kertaan hyvin kattavasti testattu eikä mitään vakavia virheitä enää ole avoinna.

9.5. Testauksen seuranta

Testauksen seuranta ajoittuu SW Design and Implementation -prosessin M4 ja M5 etappien välille. Erityisesti heti M4 etapin jälkeen testauksessa aloitetaan valmistelut asiakkaan järjestämälle *ulkoiselle hyväksyntätestaukselle*. Ulkoisen hyväksyntätestauksen tavoitteena on vahvistaa asiakkaalle, että tuotteen laadun taso ja tuotteessa oleva toiminnallisuus vastaa sitä, mitä Ixonos Oyj:n ja asiakkaan välillä on sovittu. Ulkoinen hyväksyntätestaus voidaan tehdä vasta sitten, kun sisäinen hyväksyntä on saavutettu. Yleensä asiakas itse suorittaa ulkoisen hyväksyntätestauksen, mutta joissain tapauksissa projektitiimi suorittaa ulkoisen hyväksyntätestauksen asiakkaan puolesta asiakkaan valvonnan alaisena. Yleensä asennustiimi (joko Ixonos Oyj:n tai asiakkaan) toimittaa tuotteen ja vastaa asennuksista asiakkaan pilottiympäristöön, jossa hyväksyntätestaus tehdään. Projektin testaajat valmistelevat mahdollisten hyväksyntätestauksessa löytyvien virheiden verifioinnin sekä suorittavat testauksen tulosten analysoinnin. Lisäksi he avustavat asiakkaan testaajia hyväksyntätestauksen aikana neuvomalla ja ohjaamalla esimerkiksi erityisen vaikeasti testattavien osa-alueiden tai hankalasti käytettävien testaustyökalujen suhteen.

Testaajat voivat sovittaessa osallistua projektin jälkeen *ohjelmisto- ja järjestelmätuen* antamiseen. Tässä tilanteessa testauksen seurantavaiheen aikana testaajat valmistautuvat tähän tehtävään ja aloittavat sen suorittamisen heti, kun tuote on asennettu asiakkaalle. Lisäksi testaajat tukevat omaa asennustiimiään, mikäli kyse on tuotteesta, jota käydään asentamassa useille asiakkaille.

Etapin M4 lopussa suoritetaan projektin *testauksen alasajo*. Aluksi testausympäristö ja -laboratorio puretaan. Kaikki testauksen tuottama dokumentaatio (testaussuunnitelmat, testiraportit, testitapaukset yms.) tallennetaan arkistoon. Kuten koko projektista myös testauksen osalta kirjataan *palaute*. Palautteessa pääasiassa kartoitetaan asioita, jotka

menivät projektin aikana erityisen hyvin tai huonosti. Lisäksi myös parannusehdotukset kirjataan palautteeseen. Palautetta käytetään hyväksi seuraavien projektien kohdalla sekä prosessien jatkokehityksessä. Testausta käsittelevä palaute tulisi pyytää koko projektiorganisaatiolta sekä asiakkaalta. Koko projektin viimeisenä vaiheena testauksen osalta vapautetaan testauksen henkilöresurssit uusien projektien käyttöön.

9.6. Roolit ja vastuut

Testauksen roolit määräytyvät pääsääntöisesti Ixonos Oyj:n käytännön mukaisesti. Ixonos Oyj:n henkilöstöpolitiikan mukaisesti testauksen roolit ja niiden vastuut on määritelty *ISEB*:n määritelmiä noudattaen. Testaukselle on määritelty useita erilaisia rooleja ja eri rooleille omat vastuualueet seuraavasti:

Taulukko 2. Roolit ja vastuut. (Ixonos Oyj 2007.)

Rooli	**Vastuut**
Testauspäällikkö (Test Manager)	- testauksen suunnittelu ja hallinnointi - yksityiskohtainen aikataulujen laatiminen, resursointi ja työn johtaminen - päivittäisten hallinnollisten asioiden hoito - raportointi oman vastuualueensa sisällä - katselmoida ja hyväksyä testausmäärittelyjä - tukea ja ohjata testaajia - vastata, että tarvittavat työkalut ovat saatavilla
Testikonsultti (Test Consultant)	- jakaa tietoa testauksesta ja eri tekniikoista - analysoida ja haastaa vaatimuksia ja testitapauksia - kouluttaa yksityiskohtaisesti erityisten testaustyökalujen ja testauskäytäntöjen suhteen
Testauksen statistiikko (Test Statistician)	- raportoida testauksen edistymisestä projekti- ja testauspäällikölle sekä asiakkaalle - kerätä ja muodostaa vaaditut metriikat raportteihin - virheraportoinnin päävastuullinen - osallistua testauksen suorittamiseen
Tekniikkaspesialisti (Technique Specialist)	- analysoida miten tuottaa testitapauksia käyttötapausten, määrittelyjen yms. pohjalta ja tämän perusteella luoda uusien testitapausten otsikot - tutkia tuotteelle asetetut vaatimukset - osallistua testauksen suorittamiseen
Testausympäristöspesialisti (Environment Specialist)	- vastata testausympäristön laitteiden ja ohjelmistojen toimivuudesta - päivittää laitteiden ja ohjelmistojen uudet versiot sekä opiskella niiden uudet ominaisuudet ja niiden asettamat vaatimukset testausympäristölle

	- ratkoa testausympäristössä olevat ongelmat - osallistua testauksen suorittamiseen
Automaatiospesialisti (Automation Engineer)	- analysoi automaattisen testauksen tarpeen ja tämän perusteella luo automatisointisuunnitelman - hallitsee käytössä olevat automaatiotyökalut ja antaa niiden käyttöön tukea tarvittaessa - suunnittelee uudet automaattiset testitapaukset ja varmistaa niiden toimivuuden ennen niiden käyttöönottoa - osallistuu testauksen suorittamiseen
Testauksen analysoija (Test Analyst)	- analysoi testitulokset - analysoi testauksesta löytyneet virheet - raportoi virheet - osallistuu testauksen suorittamiseen
Testaussuunnittelija (Test Designer)	- tekee testausmäärittelyn käyttötapauskuvausten, vaatimusten ja uuden toiminnallisuuden mukaan - tekee testitapauksen testausmäärittelyn perusteella - osallistuu testauksen suorittamiseen
Testauksen suorittaja (Test Executor)	- suorittaa päivittäin testauksen eri aktiviteetteja kuten esimerkiksi testauksen suorittamista, testauksen raportointia, virheiden raportointia yms.

Testaukselle on siis määritelty useita eri rooleja, mutta yksitäisessä projektissa ei välttämättä tarvita kaikkia edellä lueteltuja rooleja. Tämä luonnollisesti riippuu täysin projektista, tuotteesta ja asiakkaasta. Yhdelle testaajalle voidaan asettaa useita eri rooleja ja vastaavasti usealle eri testaajalle voidaan asettaa samanlaiset roolit. Testauspäällikön nimittää projektipäällikkö. Muihin testauksen rooleihin resurssit nimittää joko testaus- tai projektipäällikkö.

9.7. Testauksen raportit

Testauksen raportit kertovat testauksen tuloksen ja sen perusteella pyritään arvioimaan testatun *tuotteen maturiteettia* sekä itse *testausprosessin tehokkuutta ja laatua*. Raportointitavat sekä raporteissa käytetyt *metriikka ja statistiikka* riippuvat yleensä hyvin voimakkaasti asiakkaasta. Pääsääntöisesti sprintin metriikat ja statistiikka sisältyvät yksittäisen sprintin testiraportteihin. Testattujen sprinttien tuloksista voidaan laatia erilaisia *statistiikkoja ja trendejä*.

Testiajon tulokset esittelevä *testiraportti* pitää tehdä heti testiajon päätyttyä. Pääsääntöisesti Ixonos Oyj:n omaa testiraporttipohjaa tulisi käyttää raportointiin. Raportin pääasiallinen tarkoitus on kertoa tuotteen maturiteetti. Maturiteetin määrittely riippuu osittain tuotteesta, projektista ja asiakkaasta. Testiraportissa kirjataan *metriikkaa ja statistiikkaa testiajosta*, mutta testiraportissa täytyy olla myös *testaajan kirjallinen vapaamuotoinen kommentti*, joka kuvaa, mitä mieltä testaaja oli tuotteen maturiteetista. Jokainen yksittäinen ajettu testitapaus pitää myös raportoida. Jokaiselle testitapaukselle, jonka status on jotain muuta kuin ”hyväksytty” (passed), pitää olla kirjallinen lyhyt selitys miksi status on jotain muuta. Mikäli status on ”epäonnistunut” (failed), tulee selitykseen liittää myös virheraportin tunniste. Raportointityökalu riippuu projektista.

Testiraportin lisäksi jokaisesta löydetystä virheestä täytyy kirjoittaa virheraportti virallisen prosessin mukaisesti lukuun ottamatta tutkivan testauksen yhteydessä mainittuja erityistapauksia. Virheraportti tulee kirjata projektin ohjeiden mukaisesti.

9.7.1. Testauksen metriikat ja statistiikka

Testauksen metriikoista tulisi laskea ainakin *testiajoprosentti* (runrate) ja *läpäisyprosentti* (passrate). Testiajoprosentti kertoo, kuinka monta testitapausta pystyttiin suorittamaan verrattuna suunniteltujen testitapausten kokonaismäärään yksittäisen testiajon aikana. Läpäisyprosentti kertoo, kuinka monta testitapausta suoritettiin hyväksytysti verrattuna suoritettujen testitapausten kokonaismäärään yksittäisen testiajon aikana. Muitakin metriikoita usein käytetään, mutta nämä määräytyvät projektikohtaisesti. Asiakkaalla on hyvin usein omat vaatimuksensa laskettavista metriikoista.

Statistiikkaa kerätään yleensä silloin, jos tuloksia halutaan jostain syystä vertailla. Usein tuotteiden eri versioiden testaustuloksia halutaan vertailla, jotta voidaan selvittää miten paljon tuotteen maturiteetti on eri versioiden välillä muuttunut. Vertailua tehdään usein myös, jos samaa tuotteen versioita käytetään eri tuotealustoilla. Vertailu on mahdollista, mikäli samalle tuotteelle eri tuotealustoilla tai saman tuotteen eri versioille suoritetaan samoja testiajoja useita kertoja. Testituloksista kerätään erilaisia metriikoita, joita sitten

tilastollisesti vertaillaan. Regressiotestauksesta kerätään usein erilaisia tilastoja. Regressiotestauksessa ajetaan usein ainakin osittain samat testitapaukset esimerkiksi jokaisen sprintin tuotepaketille. Tämä mahdollistaa referenssitiedon keräämisen sekä testauksen vertailtavuuden eri versioiden ja sprinttien välillä. Tällaiset niin kutsutut trendit ovat usein pohjana tuotteen maturiteetin kehittymisen analysoinnissa.

9.7.2. Testauksen tehokkuuden arviointi

Testauksen tehokkuuden arviointiin on olemassa useita erilaisia mittareita. Nämä luonnollisesti riippuvat voimakkaasti siitä, millaista testausta tehdään, millaisin välinein ja kenen toimesta. Ennen arviointien tekemistä on syytä tarkkaan miettiä millaisten asioiden arviointi palvelee projektia ja yritystä parhaiten. Arviointeja ei pidä tehdä vain arvioinnin vuoksi, vaan niillä tulee olla selkeä päämäärä ja tarve. Testausprosessia tulisi arvioida ennen projektia, projektin aikana ja sen jälkeen. Arvioinnin perusteella tulisi päätellä, onko testaus riittävän tehokasta laadullisesti, teknisesti, taloudellisesti ja ajallisesti. Lisäksi pitäisi arvioida, onko testausprosessissa joitain osa-alueita, joita tulisi kehittää. Testausprosessien arviointeihin on kehitetty erilaisia metodeja kuten esimerkiksi TPI (Testing Process Improvement) analyysi.

Testauksen kattavuus täytyy arvioida. Arvioinnissa täytyy selvittää, ovatko kaikki tuotteelle määritellyt vaatimukset sekä tuotteeseen lisätyt tai muokatut ominaisuudet katettu riittävästi testitapauksilla. Lisäksi voidaan tehdä *dynaamista analysointia*, jonka perusteella pystytään selvittämään, kattavatko testitapaukset kaikki ohjelmiston lähdekoodin eri haarat. Jos testattavalla tuotteella on aiempia versioita, on hyvä arvioida, kattaako testaus riittävästi aiemmissa versioissa havaitut virheherkät tai kriittisiä virheitä sisältäneet alueet. Muitakin kattavuusanalyysejä voidaan tehdä.

Virhevuotoanalyysi on yksi testauksen tehokkuuden mittareista. Tähän olennaisena osana vaikuttaa myös muu tuotekehityksen laatu ja tehokkuus. Jos tuotteessa on suuri määrä virheitä ja testaus on tiukasti rajattua, ei virheiden vuotaminen projektiorganisaation ulkopuolelle ole yksin testauksen tehottomuudesta johtuvaa. Jotta tätä analyysia voidaan hyödyntää tehokkaasti, täytyy testauksella olla projektin

ulkopuolelta raportoidut virheet tiedossa. Jotta tällainen analyysi olisi mahdollista tehdä, täytyy projektin ulkopuolisten virheraportointiprosessien tukea tätä käytäntöä. Mikäli kaikki virheraportit, siis myös projektin ulkopuolelta kirjatut esimerkiksi loppukäyttäjien ilmoittamien virheiden perusteella tehdyt raportit, kootaan samaan virheraporttitietokantaan, pitäisi analyysin tekeminen olla kohtuullisen helppoa. Analyysissa ensin lasketaan, kuinka monta kaikista kirjatuista virheistä on löytynyt projektin oman testausprosessin ulkopuolelta. Tämän jälkeen tehdään analyysi virheiden laadusta ja siitä, missä tuotteen ominaisuudessa tai yksittäisessä ohjelman lähdekoodin komponentissa virhe oli. Lopuksi selvitetään, missä oman projektin testausvaiheessa virhe olisi pitänyt löytyä ja miksi sitä ei löydetty. Tämän analyysin pohjalta voidaan tarvittaessa tehdä korjauksia testausprosessiin.

Testauksen suorittamisen tehokkuus on yksi yleisimmistä mitattavista asioista testausprosesseissa. Tätä tehokkuutta voidaan mitata esimerkiksi *testauksen suorittamiseen kuluvan ajan* mittaamisella. *Testauksen automatisoinnin* tasoa voidaan mitata arvioimalla, miten paljon manuaalisesta testauksesta on mahdollista automatisoida. Testauksen suorittamisen tehokkuutta voidaan tutkia lisäksi arvioimalla, miten monta ja minkä laatuisia virheitä yksittäisillä testitapauksilla on löytynyt. Jos on testitapauksia, joilla ei ole löytynyt yhtään virhettä, voidaan analysoida, onko testitapaus lainkaan hyödyllinen. Voidaan myös tutkia, miten monta ja minkä laatuisia virheitä on löytynyt yksittäisistä ominaisuuksista ja tämän perusteella selvittää, onko testauksessa jäänyt joitain osa-alueita kattamatta.

9.8. Testauksen dokumentointi

Testauksen dokumentointi pyritään pitämään mahdollisimman vähäisenä ketterien menetelmien periaatteiden mukaisesti. Testauksessa on kuitenkin tietyt dokumentit, jotka on pakko kirjata tavalla tai toisella, mikä tekee dokumentoinnin vähentämisen haasteelliseksi.

Ixonos Oyj on luonut useita testaukseen liittyviä dokumenttipohjia, joita tulisi käyttää mahdollisuuksien mukaan jokaisessa projektissa. Dokumenttipohjat ovat ohjeellisia ja vapaasti muokattavissa projektien vaatimusten mukaisesti. On kuitenkin tiettyjä dokumentteja, jotka testausprosessin tulisi tuottaa. *Päätason testaussuunnitelma* sekä *sprinttikohtaiset testausmääritelmät* ovat ehdottoman tärkeitä, koska ne muodostavat koko testauksen rungon projektille. Jos projektissa pyritään testauksen automatisointiin, on tehtävä *testauksen automatisointisuunnitelma*, jotta voidaan hahmottaa automatisoinnin asettamat kokonaisvaatimukset projektille. Tällainen suunnitelma voi tosin olla osana päätason testaussuunnitelmaa. *Testitapauskuvaukset* yksikkö-, savu- ja regressiotestaukselle täytyy olla kirjattuna toistettavuuden vuoksi, sillä ainakin osittain samoja testitapauksia tullaan käyttämään useissa sprinteissä. Tutkivan testauksen testitapauksista vain osa kirjataan, sillä ainakin osittain samoja testitapauksia tullaan käyttämään seuraavien sprinttien muissa testauksen vaiheissa. Testauksesta saadut *testitulokset* täytyy kirjata ja niihin perustuen täytyy luoda *testaus- ja virheraportit* sekä niihin sisältyvät *metriikat ja statistiikka*. Näiden avulla määritellään tuotteen maturiteetti sekä informoidaan asiakasta tuotekehitystyön ja testaustyön edistymisestä.

9.9. Testauksen työkalut

Käytettävät testauksen työkalut riippuvat luonnollisesti hyvin vahvasti testattavasta tuotteesta, projektista ja joissain tapauksissa asiakkaasta. Testauksessa käytettävät työkalut voidaan jakaa kahteen osaan: *testauksen hallinnan työkalut* ja *testauksen suorittamiseen käytettävät työkalut*. Pääsääntöisesti tulisi käyttää Ixonos Oyj:n valitsemia työkaluja, mutta hyvin usein asiakkaan vaatimuksesta käytetään asiakkaan valitsemia työkaluja. Useasti testattavassa tuotteessa käytetään sellaista teknologiaa tai laitteita, että tuotteen testaukseen ei ole saatavilla kaupallisia tai avoimeen lähdekoodiin pohjautuvia testityökaluja ja tästä syystä projektissa joko käytetään asiakkaan tekemiä työkaluja tai tehdään ne itse.

Tutkielmaa tehdessä Ixonos Oyj:n testauksen hallinnan työkaluja oltiin uusimassa. Tästä syystä hallinnollisista työkaluista ei ole kirjattu kovin yksityiskohtaisia huomioita.

Testauksen hallintaan käytettävät työkalut ovat usein samat eri projekteissa. Testauksen hallinnalla tarkoitetaan pääasiassa testauksen ajan hallintaan, testauksen suunnitteluun, testauksen dokumentaation ylläpitoon, virheraportointiin, resursointiin ja raportointiin käytettäviä työkaluja. Periaatteessa tähän kategoriaan voi laskea kaikki ne työkalut, joita ei käytetä suorittavan testaustyön tekemiseen. *Testauksen suorittamisen työkalut* ovat miltei aina tuote- tai teknologiakohtaisia ja tästä syystä vaihtelevat voimakkaasti eri projektien välillä.

10. JOHTOPÄÄTÖKSET

Tutkielman lopputuloksena voidaan todeta, että laadittu testausprosessi oli onnistunut. Ixonos Oyj:n edustajat olivat hyvin tyytyväisiä työhön, sillä se todettiin soveltuvan hyvin yrityksen SW Design and Implementation –prosessiin. Testausprosessi esiteltiin lisäksi yrityksen *Tietoliikenne* -yksikön johtajalle, joka hyväksyi lopullisen version. Testausprosessi lisättiin yrityksen myyntiorganisaation käyttöön sekä lisäksi yrityksen intranetissä olevalle prosessikuvaussivustolle, jossa se liitettiin osaksi SW Design and Implementation -prosessin kuvausta. Tutkielman kirjoittaja tulee syksyn 2007 aikana esittelemään testausprosessia Ixonos Oyj:n yksiköiden projekti- ja testauspäälliköille. Ensimmäiset projektit, jotka ottavat tämän testausprosessin käyttöönsä, alkavat syksyn 2007 aikana.

Testausprosessin suunnittelussa oli tarkoitus käyttää apuna TMap-menetelmää. Tutkielman aikana kuitenkin todettiin, että TMap-menetelmä soveltuu huonosti ketteriä menetelmiä noudattavan testausprosessin suunnitteluun. TMap-menetelmä on luotu kehittämään erityisesti perinteisten mallien, kuten V-mallin ja vesiputousmallin, mukaista testausprosessia. TMap-menetelmä perustuu siihen, että projektissa tuotetta tehdään porrasteisesti ja testaus määritellään sen mukaisesti selvärakenteiseksi. Lisäksi TMapin perusolettamuksia on, että tuote tehdään kerralla valmiiksi, ei siis iteratiivisen mallin mukaan. SW Design and Implementation -prosessi on luotu ketterien menetelmien periaatteiden mukaisesti. Prosessi etenee sprintteihin perustuvan tuotekehityksen kautta, minkä mukaan testausta on tarkoitus tehdä lähes koko projektin elinkaaren ajan. Ketteriin menetelmiin perustuva testausprosessi ei ole kaikilta osin selvärakenteinen. Eräs tämän tutkielman tuloksena syntyneen testausprosessin perusajatuksista on ollut käyttää tutkivaa testausta. Tämän tyyppinen testaus ei ole selvärakenteista, koska tutkivassa testauksessa ei ole aina ennakolta määritelty kovin tarkasti, mitä testataan ja millä tavoin. Tietty suunnitelmallisuus on määritelty esimerkiksi testitapausten otsikoiden avulla, mutta tarkempaa määrittelyä tutkivassa testauksessa ei ole. Koska testaus toteutetaan sprinteissä, kuten koko muukin tuotekehitys, olisi TMap:n käyttö edellyttänyt, että malleja olisi jouduttu tekemään

jokaiselle sprintille erikseen, koska testauksen tavoitteet ja tavat vaihtelevat sprinteittäin. Koska lisäksi sprinttien testauksellinen sisältö ei ole aina tiedossa kovinkaan tarkasti etukäteen, tämä johtaisi siihen, että TMap-malli jouduttaisiin luomaan aina uudelleen jokaisen projektin seuraavan sprintin yhteydessä. Tämä olisi ollut liian raskas työvaihe koko prosessia ajatellen ja osittain jopa vastoin ketterien menetelmien periaatteita, sillä esimerkiksi projektin sisäinen byrokratia ja dokumentoinnin tarve olisi oleellisesti lisääntynyt testauksessa TMap-mallintamisen kautta. Näiltä osin ketterät menetelmät ovat ristiriidassa TMap mallintamisen suhteen, eikä sitä voitu tästä syystä tässä työssä soveltaa. Tähän johtopäätökseen tultiin yhdessä yrityksen asiantuntijoiden kanssa.

Tutkivaa testausta määriteltäessä todettiin, että se toteuttaa hyvin konkreettisella tavalla käyttötapaustestauksen periaatteita ja näin ollen voidaan jopa tulkita tutkivan testauksen olevan tietyiltä osin käyttötapaustestausta. Tutkivan testauksen yhteydessä projektitiimi ”leikkiessään” tuotteella pyrkii käyttämään sitä loppukäyttäjän näkökulmasta. Tiimi pyrkii käyttämään tuotetta innovatiivisella tavalla ja sitä kautta ideoimaan, millaisia virhetilanteita loppukäyttäjät voivat toiminnallaan luoda sekä millaisia uusia ominaisuuksia olisi järkevää lisätä tai mitä olemassa olevassa toiminnallisuudessa kannattaisi muuttaa. Näiden ideoiden perusteella prosessissa luodaan myös testitapauksia, jotka siis pohjautuvat projektitiimin käytännön kokeilun kautta luotuihin käyttötapauksiin. Poikkeuksena perinteiseen käyttötapaustestaukseen tutkivassa testauksessa ei kuitenkaan laadita käyttötapaustestaukselle tyypillistä dokumentaatiota, kuten esimerkiksi käyttötapausmalleja tai -kaavioita.

Tutkielman aikana huomattiin, että ketteriä menetelmiä voidaan soveltaa testaukseen kohtalaisen hyvin. Suurin haaste on kuitenkin ketterien menetelmien periaatteet, joiden mukaan dokumentaation määrä tulee pitää mahdollisimman alhaisena sekä välttää byrokratiaa. Testauksen osalta tämä on erityisen vaikeaa, koska testauksen toiminnan tulokset ovat nähtävissä pääasiassa vain dokumentoitujen testiajojen ja testitapausten sekä niiden pohjalta muodostettujen raporttien, metriikoiden ja statistiikan kautta. Testauksen pääasiallinen funktio on varmistaa tuotteen laatu sekä ominaisuuksien vastaavuus asiakkaan kanssa tehtyjen sopimusten suhteen. Tätä varmistusta varten

testauksen on dokumentoitava työnsä tuloksia ja menetelmiä. Tästä johtuen dokumentaatiota ei siis kovin laaja-alaisesti voida vähentää testauksen osalta. Dokumentointia on kuitenkin mahdollista vähentää testauksen suunnitteluvaiheessa ja osittain testiajojen aikana, jota testausprosessin tutkivan testauksen vaihe korostaa. Byrokratiaa on testauksessa vaikea välttää. Jotta testaus tehdään varmasti huolellisesti ja hyvin, täytyy testauksessa noudattaa tarkasti tiettyjä sääntöjä ja määräyksiä. Tämän tyyppinen kurinalainen toimintatapa takaa sen, että toistuvasti ajettavat testitapaukset ajetaan aina samalla tavalla ja että testitapaukset ovat yksiselitteisesti määriteltyjä. Tällä tavoin varmistetaan, että testitulokset esimerkiksi tuotteen eri versioiden välillä ovat vertailukelpoisia. Tämä näkökulma korostuu erityisesti regressiotestauksessa. Tutkivassa testauksessa sen luonteen vuoksi sama byrokraattinen näkökulma ei kuitenkaan toteudu. Tutkiva testaus erityisesti noudattaa ketteriä periaatteita dokumentoinnin ja byrokratian vähyyden suhteen huomattavasti enemmän kuin muut testauksen osa-alueet.

Yleisesti ketteristä menetelmistä tutkija teki huomion, että ne eivät välttämättä sovellu kovin suurikokoisiin projekteihin. Alunperinkin SW Design and Implementation Process on suunniteltu pienille projekteille, joissa organisaation koko on 10 – 15 henkeä. Mikäli projektiorganisaatio on kovin suuri, on tutkijan mielestä erityisen haasteellista organisoida ja ennen kaikkea hallinnoida projektia pelkästään ketterien menetelmien avulla. Ketterien menetelmien periaatteisiin kuuluu, että dokumentaatiota ja byrokratiaa pyritään vähentämään ja että tiedon kulku perustuu pääasiassa organisaation jäsenten avoimeen kommunikaatioon. Mikäli näin toimitaan, on tällaisen kokonaisuuden hallittavuus vaikeaa, jos organisaatiossa on useita kymmeniä henkilöitä tai jopa enemmän. Isoissa projekteissa tulee olla jonkin verran byrokratiaa, minkä avulla projektin kontrolloiminen ja hallinta on selkeämpää ja helpompaa. Tämä toisaalta hidastaa projektin toimintaa ja kykyä reagoida muutoksiin.

Eräänä vaatimuksena testausprosessille oli, että testaus voitaisiin teettää erillisenä projektina tarvittaessa. Tämän suhteen kuitenkin todettiin, ettei se ole ketteriä menetelmiä noudatettaessa helposti toteutettavissa, koska esimerkiksi tutkiva testaus vaatii koko ajan jatkuvaa yhdessäoloa ja kommunikointia testaajien ja ohjelmoijien

välillä. Myös ohjelmoijien ja testaajien yhteisten tiimien luominen olisi haasteellista. Mikäli testaus siis tehtäisiin erillisenä projektina fyysisesti eri tiloissa kuin ohjelmoijien suorittama työ, erityisen vaikea haaste olisi saada tutkiva testaus ja ketterien menetelmien mukainen kommunikointi toimimaan tehokkaasti. Sen sijaan testaus voidaan toteuttaa erillisenä projektina kohtalaisen vaivattomasti, mikäli asiakkaan oma ohjelmistotuotantotiimi noudattaa ketteriä menetelmiä ja testausprojekti toteutetaan osana asiakkaan omaa ohjelmistotuotantoprojektia samoissa tiloissa.

Tutkijan omalta kannalta tutkielman tekeminen oli erityisen hedelmällistä. Tutkijalle ketterät menetelmät eivät tätä enne tutkielmaa olleet kovin tuttuja. Ketterien menetelmien periaatteet olivat tutkijalle täysin uusi tapa nähdä ohjelmistotuotanto ja ennen kaikkea testausprosessit. Tutkijalla oli ennen tutkielman aloittamista kokemusta ja tietoa vain perinteisistä vesiputous- ja V-mallin mukaisista tuotekehitysmenetelmistä.

Tulevaisuudessa SW Design and Implementation -prosessille laadittua testauksen aliprosessia on tarkoitus kehittää edelleen. Tämän prosessin jatkokehitys pohjautuu tulevaisuudessa uusista projekteista saataviin kokemuksiin ja sen kautta kirjattuihin mahdollisiin kehitystarpeisiin. Tutkija tulee toistaiseksi olemaan mukana tässä jatkokehityksessä.

LÄHDELUETTELO

Abrahamsson, Pekka, Outi Salo, Jussi Ronkainen & Juhani Warsta (2002). *Agile software development methods: Review and Analysis.* Espoo. VTT publications. 107s. [siteerattu 25.11.2006]. Saatavana World Wide Webistä: <URL:http://www.inf.vtt.fi/pdf/publications/2002/P478.pdf>. ISBN: 951-38-6009-4.

Black, Rex (2002). *Managing the Testing Process*. USA. Wiley Publishing. 500 s. ISBN: 0-471-22398-0.

Black, Rex (2004). *Critical Testing Processes: Plan, Prepare, Perform, Perfect.* Boston. Addison-Wesley. 571 s. ISBN 0-201-74868-1.

British Computer Society (2007). [siteerattu 3.1.2007]. Saatavana World Wide Webistä: <URL:http://www.bcs.org/>.

Chrissis, Mary Beth, Mike Konrad & Sandy Shrum (2003). *CMMI: Guidelines for Process Integration and Product Improvement.* Boston. Addison-Wesley. 663 s. ISBN: 0-321-15496-7.

Copeland, Lee (2003). *A Practitioner's Guide to Software Test Design*. Norwood. Artech House Publishers. 294 s. ISBN 1-58053-791-X.

Fowler, Martin (2005). *The New Methodology*. [siteerattu 25.11.2006]. Saatavana World Wide Webistä: <URL:http://www.martinfowler.com/articles/newMethodology.html#FromNothingtoMonumentalToAgile>

Grove Consultants (2005). *ISEB Foundation Certificate in Software Testing*. 226 s. Julkaisematon. Grove Consultants.

Ixonos Oyj (2007). *Testing Roles and Responsibilities.* Julkaisematon. [siteerattu 1.6.2007]. Ixonos Oyj.

Larman, Craig (2003). *Agile and Iterative Development: A Manager's Guide*. Boston. Addison-Wesley. 368 s. ISBN 0-131-11155-8.

Lehtinen, Jani (2007). *SW Design and Implementation Process.* 32 s. Julkaisematon. Ixonos Oyj.

Pol, Martin, Ruud Teunissen & Erik van Veenendaal (2002). *Software Testing: A Guide to the TMap Approach.* Norfolk. Addison-Wesley. 564 s. ISBN-10: 0 201 74571 2.

Sogeti (2006). *TMap*. [siteerattu 26.11.2006]. Saatavana World Wide Webistä: <URL:http://eng.tmap.net/Home/archief/931847.jsp>.

van Veenendaal, Erik (2002). *The Testing Practitioner*. Valkenswaard. UTN Publishers. 524 s. ISBN: 0-907-219465-9.