Suvi Honkela

TESTAUSPROSESSI JA SEN HALLINTA – AUTOMATISOINNIN NÄKÖKULMA

Tietojärjestelmätieteen
pro gradu -tutkielma
16.5.2005

Jyväskylän yliopisto
Tietojenkäsittelytieteiden laitos
Jyväskylä

TIIVISTELMÄ

Honkela, Suvi Marjaana
Testausprosessi ja sen hallinta – automatisoinnin näkökulma / Suvi Honkela
Jyväskylä: Jyväskylän yliopisto, 2005.
105 s.
Tutkielma

Tutkielmassa tarkastellaan ohjelmiston testausprosessia ja sen hallintaa automatisoinnin näkökulmasta. Automatisoinnille asetetaan usein suuret tavoitteet kuten prosessin tehostaminen ja laadun parantaminen. Näihin tavoitteisiin pääseminen on kuitenkin osoittautunut ongelmalliseksi. Testauksen automatisointia voidaan kuitenkin helpottaa, jos siihen liittyviä ongelmia ja etuja tunnetaan paremmin.

Näistä lähtökohdista on tutkielman tavoitteena muodostaa kuvaa kirjallisuuden pohjalta testausprosessiin ja sen hallintaan liittyvistä seikoista, jotka vaikuttavat automatisoinnin onnistumiseen. Aiheen käsittely tässä tutkielmassa perustuu lisäksi tapaustutkimuksen keinoin saatuihin tuloksiin pilottiprojektista.

Tutkimuksen keskeinen tulos on, että testauksen automatisointi on vaikeaa ja automatisoinnin yhteydessä esiintyy paljon erilaisia ongelmia, mutta automatisoinnin avulla testausprosessia voidaan tehostaa, mikäli toistoja testien ajossa tehdään useita kertoja. Erityisen ongelmallisena pidettiin välineen käytön aloitusta, jolloin testien ajaminen tuotti ongelmia ja todettiin, että testaus kannattaakin aloittaa lyhyillä testeillä ja pienellä testitapausten määrällä. Käyttäjät tarvitsevat myös koulutusta välineen käytöstä.

AVAINSANAT: testaus, testausprosessi, testauksen hallinta, testauksen automatisointi, ohjelmiston laatu

SISÄLLYSLUETTELO

1 JOHDANTO

Ohjelmistojen toiminnallisuuden ja oikeellisuuden testaaminen on aikaa ja resursseja vaativaa työtä, mutta hyvin toteutettu testausprosessi palkitsee yrityksen asiakkaan luottamuksella ja jatkotilauksilla. Testauksen merkitys on viime aikoina kasvanut suurien ja vakavien ohjelmistovirheiden kasvamisen myötä. Lisäksi testausprosessia pyritään kehittämään monella tavalla, jolloin yhtenä merkittävänä tekijänä nousee esille testauksen automatisointi. Tämä tutkielma on kohdistettu yrityksille ja tutkijoille, jotka kaipaavat tietoa testausprosessin ja sen hallinnan automatisoinnista.

Testaus kuuluu oleellisena osana ohjelmistoprosessiin. Ohjelmistoprosessin aikana syntyneiden suunnitelmien pohjalta voidaan suorittaa testausprosessi. Myers (1979, 5) määrittelee testauksen prosessiksi, jossa suoritetaan ohjelmaa tarkoituksena löytää virheitä. Myöhemmin testauksen määritelmää on hieman tarkennettu. Virheen löytäminen ei uudemmissa määrityksissä nouse yhtä selkeästi esille kuin vanhoissa määritelmissä. Oleelliselle sijalle nostetaan sen sijaan ohjelmiston määritysten täyttäminen. Virheiden havaitsemisen ohella testauksen tarkoituksena on varmistaa, että ohjelmisto täyttää sille asetetut vaatimukset (British Computer Society SIGIST 1998a).

Tutkimuksessa tarkastellaan testausprosessia ja testauksen hallintaa automatisoinnin näkökulmasta. Testausprosessin ja sen hallinnan automatisoinnin yhteydessä eri työvaiheita avustetaan työvälineiden avulla. Testausprosessin työvaiheisiin kuuluu Fewsterin ja Grahamin (1999, 13) mukaan testattavien ominaisuuksien tunnistaminen, testauksen suunnitteleminen, testitapausten määrittely, testiympäristön rakentaminen, testien suorittaminen sekä tulosten analysointi. Testauksen hallintavälineen avustavan ohjelmiston eri osilla voi hallinnoida testiprosessia organisoimalla testauksen dokumentteja, suorittamalla testitapauksia, generoimalla testiraportteja (Giraudo & Tonella 2003) ja pitämällä kirjaa erilaisista metriikoista

(Hetzel 1988). Näin testausprosessista ja sen kehittymisestä voidaan saada ajantasaista tietoa ja koko prosessi on paremmin uudelleenkäytettävissä ja ylläpidettävissä.

Työvälineiden avulla on parhaimmillaan pystytty vähentämään testauksen kustannuksia 80 % (Fewster & Graham 1999, 3). Yleensä automatisointi vie kuitenkin aikaa 3 – 10 kertaa enemmän kuin manuaalisen testin kertasuoritus ja varsinainen säästö saadaan testauksen toistoista (Kaner 1997). Automatisointi ei ole suinkaan ongelmatonta. Monessa organisaatiossa on yllätytty siitä, että automatisointi on tullut kalliimmaksi kuin testaaminen manuaalisesti. Testitapaukset ja työvälineet tulisi valita harkiten. Testauksen automatisoinnin yhteydessä tulisi seurata, kuinka taloudelliseksi automatisointi tulee, ja onko menetelmä kehityskelpoinen.

Vaikka apuvälineen toteutus tai hankinta olisikin tehty huolellisesti, ei ole mitään takeita, että työkalun käyttämisessä onnistutaan. Fewsterin ja Grahamin (1999, 283) mukaan on paljon yrityksiä, jotka ovat menestyksellisesti valinneet ja hankkineet työkalun, mutta suurin piirtein puolet näistä yrityksistä ei ole kokenut saaneensa välineestä mitään hyötyä. Nämä edellä mainitut haasteet testauksenhallinnan automatisoinnin onnistumisen osalta ovat olleet syynä tämän tutkimuksen toteuttamiselle.

Tutkielman tutkimusongelma on selvittää, mitkä ovat testausprosessin automatisoinnin menestystekijät ja ongelma-alueet. Tutkimuksen lähestymistapana toimii käytännönläheinen deduktiivinen päättely. Deduktiivinen päättely lähtee tutkielmassa liikkeelle siitä, että käydään läpi testausprosessia ja sen hallintaa koskevaa teoriaa. Päättely nojaa aikaisempaan teoriaan, jota tutkielman alussa on käyty läpi. Päättelyn keinoin halutaan verrata tehtyä tutkimusta olemassa olevaan teoriaan ja hakea kohdeyrityksestä kerätyille tutkimustuloksille selityksiä. Aikaisemmista tiedoista ja pilottiprojektissa kerätyistä tuloksista pyritään tekemään päätelmiä testauksen hallintavälineen

toimivuudesta sekä toimivuuteen vaikuttavista tekijöistä kohdeyrityksessä. Tutkielman empiirisen osuuden tapaustutkimus on luonteeltaan kuvailevaa ja tulkitsevaa.

Tutkielman kirjoittajan omaksi kontribuutioksi jää tulosten analysointi ja syy-seuraus -suhteiden löytäminen pilottiprojektista löytyneiden ongelmien ja onnistumisten selvittämiseksi. Tutkimuksen keskeisenä tuloksena selvisi, että testauksen hallintaväline voi tehostaa testaamista. Tosin välineen käyttöönoton onnistumiseen vaikuttaa mm. käyttäjien koulutus sekä toistojen määrä testiajoissa. Välineen tärkeänä ominaisuutena pidetään tehokkuutta, mutta tehokkuutta ei saisi liiaksi kehittää muiden ominaisuuksien kuten luotettavuuden kustannuksella.

Tutkielma on jäsennetty kuuteen lukuun. Johdannon jälkeisessä luvussa tarkastellaan testausprosessin ja testauksen hallinnan erityispiirteitä. Kolmannessa luvussa käsitellään testausprosessin ja hallinnan automatisoinnin avulla saavutettavia tavoitteita, tavoitteiden saavuttamiseksi tarvittavia työvälineitä sekä ongelma-alueita. Neljännessä luvussa kuvaillaan testauksen hallintavälineen käyttöönottoprosessia sekä tässä yhteydessä tarvittavaa metriikkapatteristoa laadun arvioinnista. Viides luku käsittelee välineen laadun arviointia kohdeyrityksessä. Kuudennessa luvussa käydään läpi tiivistetysti tärkeimmät tulokset sekä tutkielman perusteella tehdyt johtopäätökset.

2 TESTAUSPROSESSI JA SEN HALLINTA

Tässä luvussa tutkitaan aluksi lyhyesti testauksen taustaa ja sitten testausprosessin kannalta oleellisimmat työvaiheet sekä testauksen hallinta. Tarkoituksena on nostaa esille erityisesti testauksen automatisointia tukevat osa-alueet. Luvussa on esitelty erityisesti Kauppisen ja Tainan (2003) luomaa mallia testausprosessista ja sen hallinnasta, mutta myös Bain ym. (2001) sekä Desain (1994) käsitykset prosessista ovat hyvin lähellä valittua mallia. Eri malleissa puhutaan hieman eri nimillä asioista, mutta niistä nousee samat vaiheet esille. Kauppisen ja Tainan (2003) luoman mallin hyvänä puolena verrattuna muihin malleihin voidaan pitää prosessin iteratiivisuuden esille tuomista.

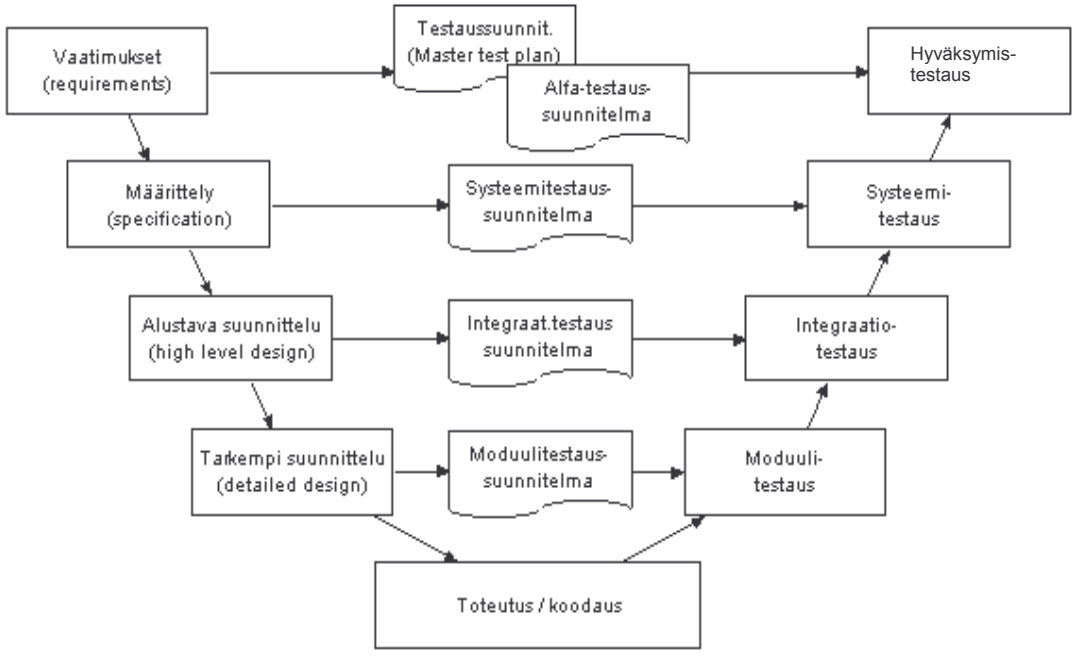
2.1 Testaus osana ohjelmistokehitystä

Testaus voidaan jakaa pienempiin ja paremmin hallittaviin tasoihin. Tason sisällä suoritetaan jatkuvaa testaustoimintaa tietyn tyyppisen testaustavoitteen ja testauskohteen ympärillä. Testaustoiminnasta tekee oleellisesti jatkuvaa vaatimus uusintatestauksesta muutoksen osalta. Taso ei ole välttämättä sama asia kuin testauksen vaihe. Testausvaihe on tason sisäinen tai useiden testaustasojen yhteinen tehtäväkokonaisuus, joka on tyypiltään kertaluonteinen eikä jatkuva (Pyhäjärvi 2005). Testaus, kuten koko ohjelmointiprosessikin, suoritetaan vaiheittain. Jokaisen vaiheen pitää olla looginen jatkumo edelliselle vaiheelle testaustason sisällä. Testauksen tasot suoritetaan V-mallin osoittamalla tavalla (KUVIO 1).

Testauksen tasoja ovat moduuli-, integraatio-, systeemi- sekä hyväksymistestaus (Boehm 1979, 712). Hyväksymistestaus voidaan jakaa vielä pienempiin vaiheisiin, jolloin testauksen ensimmäisestä vaiheesta voidaan käyttää nimeä alfa-testaus. Moduulitestaus on prosessi, joka testaa yksittäisiä aliohjelmia, alirutiineja tai yhteen kuuluvia toimintosarjoja (British Computer Society SIGIST 1998b). Moduulitestauksen tavoitteena on löytää selvien ohjelma-

virheiden lisäksi ristiriitoja moduulin määrittelyn ja toiminnan välillä. Moduulitestauksessa testattavana on aina yksittäinen moduuli. (Myers 1979; Davis 1988)

Integrointitestauksessa yksittäiset moduulit puolestaan kasataan arkkitehtuurisuunnitteluvaiheessa laaditun teknisen määrittelyn mukaiseksi kokonaisuudeksi. Jokaisen uuden moduulin liittämisen jälkeen tulee uuden moduulin rajapinnat ja niiden toimivuus testata (Davis 1988, 1099; Hetzel 1988, 130). Systeemitestauksessa testataan valmistunut järjestelmä kokonaisuudessaan ennen kuin se lähetetään asiakkaalle. Päätavoitteena on tutkia, täyttääkö valmis ohjelmisto sen määrittelyn asettamat vaatimukset loppukäyttäjän näkökulmasta. Järjestelmätestauksen suorittajina pitää olla kehitystyöstä mahdollisimman riippumattomia testaajia. Järjestelmätestauksen aikana suoritetaan yleensä käyttöliittymätestaus. (Mynatt 1990, 300)



KUVIO 1. Testauksen tasot osana ohjelmistokehitystä. Kuvio perustuu Boehmin (1979, 712) sekä Kauton (1996) kuvaukseen aiheesta.

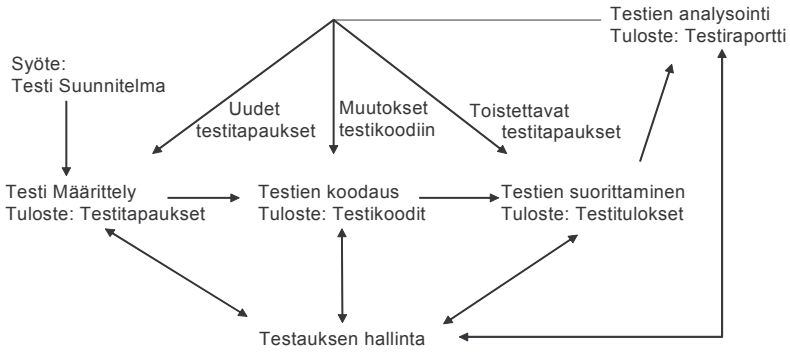
Hyväksymistestausta ei juuri koskaan automatisoida. Hyväksymistestaus suoritetaan asiakkaan tiloissa tuotteen oikeassa ympäristössä, jolloin varmistetaan, että tuote toimii, niin kuin sen on suunniteltu valmiina toimivan. Hyväksymistestauksen suorittaa yleensä asiakas itse. (Myers 1979)

V-mallin mukaisissa testauksen vaiheissa testaus nähdään eri näkökulmista ja eri vaiheet paljastavat erilaisia virheitä testauksen kohteesta. Testausvaiheiden menetelmien tulee olla mahdollisimman hyvin vaiheensa tyypillisiä virheitä paljastavia sillä, mitä ylemmälle tasolle virheet testausmallissa kulkevat ohjelmassa mukana, sitä kalliimmaksi ja monimutkaisemmaksi niiden korjaaminen tulee. (Boehm 1979, 711) V-mallista on muovattu monia erilaisia malleja testauksen kehityksestä, mutta perusajatukset testauksen vaiheista ovat usein hyvin samankaltaisia (Marick 2000).

2.2 Testausprosessin työvaiheet

Muiden ohjelmistotuotantoprosessien tapaan myös testausprosessi sisältää erilaisia työvaiheita kuten edellä kerrottiin. Jokaisessa testaustason vaiheessa ohjelmistoa testataan sekä eri yksityiskohtaisuuden tasolla että eri päämäärän kannalta. Tästä huolimatta kukin taso sisältää samat työvaiheet.

Testaus alkaa vaatimusten pohjalta tehtävän suunnitelman kirjoittamisella (KUVIO 2). Suunnittelua seuraa määrittelyvaihe, jossa määritellään yksittäiset testitapaukset. Testitapausten määritelmän pohjalta kirjoitetaan tarvittava koodi. Koodi suoritetaan ja tulosteena saadaan testitulokset. Kun testauskierroksella havaitaan virheitä, ne raportoidaan ja tulosten analysoinnin jälkeen annetaan kehittäjille korjattavaksi. Testikierrosten toistaminen päätetään, kun ollaan tyytyväisiä korjattuun ohjelmistoon. Testauksen hallinta tukee testausprosessia jokaisen vaiheen aikana. (Kauppinen & Taina 2003)



KUVIO 2. Testauksen hallintavälineen tukema testausprosessi (Kauppinen & Taina 2003, 9).

2.2.1 Suunnittelu ja määrittely

Hetzelin (1988, 34) mukaan testausprosessin alussa suunnitellaan ensin yleinen lähestymistapa testaukselle, jossa määritellään testauksen kohde. Tämän suunnitelman pojalta määritellään varsinaiset testitapaukset (Hetzel 1988, 34). Myös Fewster ja Graham (1999, 14) jakavat testauksen alkutoimet ensin yleisempään suunnitelmaan, jonka pohjalta varsinaisia testitapauksia sitten myöhemmin muodostetaan. He kuvaavat testausprosessin ensimmäiseksi aktiviteetiksi testausolosuhteiden identifioinnin, jossa kartoitetaan, mitä voidaan testata. Fewster ja Graham eivät juurikaan puutu aktiviteeteista syntyviin dokumentteihin. Heidän mukaansa dokumentointi voidaan miettiä järkeväksi kunkin projektin tarpeiden mukaan.

Testaussuunnitelma sisältää yleisen kuvauksen järjestelmästä. ANSI/IEEE (1998) standardi määrittelee, että hyvän testaussuunnitelman tulisi selvittää testauksen puitteet, lähestymistapa, resurssit sekä testausaktiviteettien aikataulut. Se identifioi testattavat yksiköt, piirteet, suoritettavat testaustehtävät, näistä vastaavat henkilöt ja kyseiseen suunnitelmaan sisältyvät riskit (ANSI/IEEE 1998).

Testaussuunnitelman pohjalta määritellään testitapaukset. Testitapaukset sisältävät tiedon siitä, mikä ohjelman osa testataan ja miten se tulee testata. Se sisältää myös usein tarkan tiedon toiminnoille annettavista syötteistä, jolloin testitapaukset olisivat helposti toistettavissa. Kullekin tapaukselle on myös määritettävä alku- ja lopputila, jotta voidaan todentaa toimintojen aiheuttavan oikean tapahtuman. (ANSI/IEEE 1998; Fewster & Graham 1999, 16)

Eri testaustasoille ja testausmenetelmille, kuten moduuli-, integrointi- ja järjestelmätestaukselle, suunnitellaan erilaiset testitapaukset. Testitapausten suunnittelun on oltava järjestelmällistä. Järjestelmällisyydellä tarkoitetaan sitä, että jokainen testitapaus suunnitellaan ottaen huomioon erilaiset syöteyhdistelmät. Testaus tulee keskittää sinne, missä virheiden löytymisen todennäköisyys on suurin ja virheet ovat vakavia. Testauksen priorisointi on tärkeää, koska kaikkien syöteyhdistelmien testaaminen on mahdotonta rajallisten resurssien vuoksi (Fewster & Graham 1999). Testauksen automatisointi toimii toki suurena apuna, jotta voidaan suorittaa suuri määrä johdonmukaisia ja toistettavia testitapauksia.

Testausta suunniteltaessa tulee ottaa huomioon testiympäristön ja testityövälineiden asettamat vaatimukset (Fewster & Graham 1999). Esimerkiksi testauksen suorittamisen yhteydessä suoritetuista testitapauksista kirjautuu tietoja testilokitiedostoon. Testauksen hallintaväline voi arkistoida tarvittaessa virheellisten testien lokitiedot kunkin testitapauksen osalta, mutta tämä järjestely voi vaatia suunnittelua ja oikeanlaisen lokitiedoston muodostamista.

2.2.2 Toteutus ja suoritus

Testausta toteutettaessa luodaan testiskriptejä. Skripti on ennalta ohjelmoitujen komentosarjojen joukko (Järvinen 1999, 508), jonka avulla testi voidaan suorittaa. Testiskriptien koodauksen ohella on usein tarpeellista tehdä ohjelman suoritukselle apuvälineitä, kuten testiajureita, tynkiä tai testikirjastoja, jotta

testitapaukset voitaisiin suorittaa (Fewster & Graham 1999, 16). Edellä mainittujen apuvälineiden muodostamaa rakennetta kutsutaan testausympäristöksi. Sen tulisi vastata ohjelmiston normaalia toimintaympäristöä.

Testausympäristön rakenne tulee dokumentoida testaussuunnitelmaan tai erilliseen dokumenttiin, jotta vastaava ympäristö voidaan pystyttää uudelleen (Myers 1979, 121). Lisäksi sen avulla voidaan havaita ympäristön muuntamisesta aiheutuvat ohjelman virheet. Tällöin tulee myös todettua ja dokumentoitua vaatimusten mukainen yhteensopivuus. Ympäristöstä kannattaa dokumentoida tiedot käyttöjärjestelmästä, kääntäjästä, tietokannoista, laitteistoista, käytetyistä työkaluista ja muista testauksen apuvälineistä.

Kun testausympäristö on toteutettu, voidaan testitapaukset suorittaa. Suoritus voidaan tehdä manuaalisesti skriptejä yksittäin suorittamalla tai ajaa skriptit automatisoidusti. Manuaalisessa testauksessa testaaja joutuu siis ajamaan testiskriptit itse ja antamaan testauksen syötteet käsin. Hän seuraa tulosteita ja kirjaa niistä raportoitavat tiedot ylös. Automatisoinnin yhteydessä testaaja usein vain käynnistää työvälineen ja ilmoittaa minkä testijoukon hän haluaa suorittaa. Työväline suorittaa ja raportoi itse tulokset. (Fewster & Graham 1999, 17)

2.2.3 Analysointi

Testausprosessin viimeisessä vaiheessa verrataan testauksen suorittamisen jälkeen saatuja tuloksia odotettuihin tuloksiin. Työvälineen avulla voidaan suorittaa tämä vertaus ja muodostaa raportit. Raporttien perusteella voidaan havaita ohjelman virheet. Lisäksi sen avulla voidaan analysoida testejä ja päättää, pitääkö niitä parantaa. Työvälineen avulla ei voida suorittaa varsinaista analysointia. Työväline voi verrata tuloksia toisiinsa, mutta se ei voi tietää onko tuloksissa vikaa, joka aiheutuu esim. testausympäristöstä. Tämän analyysin joutuu siis tekemään testaaja. (Fewster & Graham 1999, 17)

Virheiden jäljitys ja niiden korjaaminen kuuluu olennaisesti ohjelmisto-prosessiin. Virheiden korjaamisen jälkeen suoritetaan testitapaukset uudelleen eli suoritetaan niin kutsuttu regressiotestaus, jonka perusteella saadaan uusi raportti. Automatisoitu testausprosessi toimii siis iteratiivisesti. (Kauppinen & Taina 2003) Jossain vaiheessa nämä testauskierrokset on kuitenkin lopetettava. Lopettamispäätös tehdään yleensä jonkin ennalta määrätyn lopetusehdon täyttyessä. Tässä vaiheessa tulee olla syvä luottamus siihen, että testit ovat hyviä ja ne ovat paljastaneet hyvin virheitä.

Päätös testauksen lopettamisesta tulee tehdä harkitusti. Testien suorituksesta laadittu raportti on yhtä luotettava kuin itse testitapaukset. Eli mikäli testitapausten suunnittelemiseen ei ole käytetty tarpeeksi aikaa, voi raportti antaa täysin vääristyneen kuvan ohjelman toimivuudesta. Valitettavan usein testaus lopetetaan, kun rahat ja aika loppuvat. Toisaalta voidaan ajatella, ettei ohjelmiston testaaminen lopu koskaan. Tietyssä vaiheessa testaaminen vain siirtyy asiakkaan tehtäväksi.

Myers (1979, 126) esittää, että virheiden havaitsemisen määriä listaamalla voidaan aikaansaada ns. virheiden havaintokäyrä. Aluksi virheitä löytyy paljon, mutta jossakin vaiheessa virheiden esiintymistiheys harvenee ja lopulta niitä alkaa löytyä yhä vähemmän. Virheiden havaintokäyrä antaa hyvän suunnan testauksen lopettamiselle.

Testauksen lopettamispäätöksen apuna voidaan käyttää myös erilaisia kattavuusmittareita. Kattavuusmittarit voivat mitata mm. lausekattavuutta, ehtokattavuutta tai päätösehtokattavuutta. Lausekattavuus kertoo niiden koodirivien määrän, jotka on suoritettu vähintään kerran. Päätöskattavuutta mittaamalla voidaan todeta, mitkä ehtolauseet on suoritettu niiden kaikilla arvoilla. Hieman samankaltainen mittari on ehtokattavuus, joka kertoo onko kaikkien alkeisehtojen vaihtoehdot tosi ja epätosi läpikäyty. Kattavuuden tutkimiseen on olemassa valmiita työkaluja. (Weller 1994)

2.3 Testausprosessin hallinta

Testausprosessin hallinta on kaikkien testaukseen kuuluvien toimenpiteiden koordinointia, tukea ja johtamista (Hetzel 1988, 178-179). Testauksen hallinnan yleiset määritelmät ovat kirjallisuudessa melko laajoja. Niiden näkökulmasta katsottuna testauksen hallinnan työväline tukee testauksen hallintaa melko suppeasti, vaikkakin se tukee työvaiheiden suorittamista hyvin (Smith 2000). Tutkielman hallintaosuudessa käsitellään testiaineiston organisointi ja mittaaminen sekä edut, joita näiden hallintatoimien avulla saavutetaan. Prosessinhallinnan tehostamisen vuoksi tulisi kehittää mittaristo, jonka avulla testattavasta järjestelmästä ja sen tilasta saadaan tietoa.

Tässä tutkielmassa pyritään tuomaan esiin testauksen hallintatyövälineen prosessia tukevat ominaisuudet. Testauksen hallinnan -työvälineet on kehitetty automatisoimaan testauksen suunnittelua, suoritusta, dokumentointia ja raportointia (Giraudo & Tonella 2003). Markkinoilla on tarjolla ohjelmia ja ohjelmistoja, joilla luodaan oma ympäristökokonaisuus testauksen hallintaan. Tyypilliset hallintaympäristöt helpottavat testauksen organisointia ja tekevät erilaisia raportteja (Rathburg, 1993; Giraudo & Tonella 2003). Lisäksi Hetzel (1988) painottaa tässä yhteydessä myös metriikoiden keräämisen tärkeyttä.

2.3.1 Dokumenttien organisointi

Tainan (2002) mukaan IEEE:n standardia (ANSI/IEEE 1998) voidaan käyttää testausprosessin hallitsemiseen. Standardi määrittelee kahdeksan erilaista dokumenttia, joita voidaan soveltaa kaikille testauksen tasoille yksikkö-testauksesta hyväksymistestaukseen suunnittelussa, määrittelyssä, suorittamisessa ja raportoinnissa. Standardi ei kuitenkaan määrittele, mitä dokumentteja tulee missäkin testaustasossa ja työvaiheessa tuottaa. Tämä päätös jää jokaisen organisaation pohdittavaksi. Testauksen hallinnan oleellisena

tehtävänä on tukea näiden edellä mainittujen dokumenttien ja raporttien organisointia.

Organisaation tulee standardin mukaisen dokumentaation käyttöönottoa harkittaessa päättää, mitä dokumentteja kullakin testauksen tasolla vaaditaan. Myös standardin tarjoamien dokumenttien sisältöä tulee muokata ohjelmatyypin ja testauksen vaiheen mukaan organisaation käyttöön sopiviksi (ANSI/IEEE 1998). IEEE:n standardissa (ANSI/IEEE 1998) kehotetaan ensimmäisessä vaiheessa ottamaan käyttöön suunnittelu- ja raportointi-dokumentaatio. Testaussuunnitelma luo pohjan koko testausprosessille ja raportointidokumentit puolestaan auttavat testauksessa saadun olennaisen informaation tallentamista organisoidulla tavalla (ANSI/IEEE 1998).

Testaussuunnitelma määrittelee testauksen puitteet, lähestymistavan, resurssit sekä testausaktiviteettien aikataulun. Se identifioi testattavat yksiköt, piirteet, suoritettavat testaustehtävät, näistä vastaavat henkilöt ja kyseiseen suunnitelmaan sisältyvät riskit (ANSI/IEEE 1998). Automatisoinnin edellytyksenä voidaan pitää tarkkaa suunnitteludokumentaatiota, jotta voidaan ylipäätään huomata, mitä testitapauksia kannattaa automatisoida. Dokumenttien organisoinnin hyödyt tulevat parhaiten esiin silloin, kun testaus on suunniteltu hyvin ja organisoituja dokumentteja voidaan uudelleenkäyttää myöhemmissä projekteissa.

Testauksen suunnittelun jälkeen päästään määrittelyvaiheeseen, johon standardi ANSI/IEEE (1998) tarjoaa kolme dokumenttia. Testaussuunnitelman täsmennys tarkentaa testauksen lähestymistapaa sekä identifioi ne piirteet, jotka tällä suunnitelmalla ja siihen liittyvillä testitapauksilla ja menettelyillä testataan. Testitapausmäärittely määrittää suoritettavan yksittäisen testitapauksen: testattavan yksikön, sen syöte- ja tulostiedot sekä testiympäristön. Lisäksi testitapausmäärittelyyn dokumentoidaan yksikön testaamiseen vaadittavat erityismenettelyt ja kyseisen testitapauksen kytkennät muihin

testitapauksiin, esimerkiksi tieto siitä, mitkä testitapaukset tulee suorittaa ennen tätä. Testausmenettelyn selostus erittelee testitapausten joukon suoritusvaiheet tai yleisemmin ohjelmayksikön analysoinnin vaiheet. (ANSI/IEEE 1998)

Testauksen raportointiin on tarjolla neljä dokumenttia. Testauksen toimitusraportti sisältää listan testattavaksi toimitetuista kohteista. Testiloki on kronologinen nauhoite relevanteista testin suoristusaikaisista yksityiskohdista. Testitapausraportti dokumentoi testausprosessin aikana ilmenneen ja tutkimusta vaativan tapahtuman ja testausyhteenvetoraportti sisältää yhteenvedon suunniteltujen testaustoimenpiteiden suorituksesta, testauksen kulusta sekä tuloksiin perustuvan arvioinnin testatuista kohteista. (ANSI/IEEE 1998)

IEEE standardi (ANSI/IEEE 1998) ei ole miellyttänyt kaikkia tutkijoita. Kaner (2001) kritisoi voimakkaasti standardia IEEE 829. Hänen mukaansa standardi olettaa käytettäväksi vesiputousmallin mukaista tuotekehitystä ja käytännön projekteissa aina ilmenevät muutostarpeet tekevät luodun dokumentaation ylläpidosta erittäin kallista. Hänen mukaansa standardin yhteydessä ei useinkaan tiedosteta tai keskustella suuren dokumenttimäärän tuottamisen ja ylläpidon kustannuksista. Dokumentointiin käytetty aika on aina poissa henkilön varsinaisesta lisäarvoa tuottavasta työstä. Lisäksi standardin käyttö korostaa dokumentaation määrää sen laadun kustannuksella. (Kaner 2001)

Suuri dokumentaatio myös kätkee helposti virheitä. Esimerkiksi olennaisia testitapauksia on voinut jäädä pois, mutta asian todentaminen on hankalaa dokumentaation käsittäessä satoja sivuja. Standardin soveltamisen sijaan Kaner (2001) suosittaakin käyttämään vähemmän formaalia dokumentaatiota: lyhyiden listojen ja taulukoiden kokoelmia, tilaraportteja ja huolellisesti laadittuja virheraportteja. Tarvittava kommunikointi voidaan hoitaa myös säännöllisillä ryhmäkokoontumisilla. (Kaner 2001)

Kun automatisoinnin yhteydessä puhutaan tuotosten organisoinnista, niin tarkoitetaan näiden edellä mainittujen erilaisten dokumenttien, raporttien, lokitiedostojen sekä testiskriptien organisointia järkevästi järjestelmään, jotta ne olisivat paremmin uudelleenkäytettävissä ja ylläpidettävissä.

Usein testauksen hallintaympäristö luodaan web-pohjaiseksi, jolloin se on kaikkien projektin työntekijöiden käytettävissä, heidän maantieteellisestä sijainnistaan riippumatta. Jokaiselle käyttäjälle määritellään oikeudet, joiden mukaan hän saa hallinnoida erilaisia tuotoksia järjestelmässä. Näin voidaan rajata kullekin työntekijälle pääsy juuri sellaisiin paikkoihin kuin järkeväksi nähdään. Yleensä projektipäällikkö vastaa oman projektinsa käyttäjien hallinnasta, siten että kullakin käyttäjällä on oikeudet toimenpiteisiin, jotka hänen kohdallaan katsotaan järkeviksi.

Testausprosessia pystytään siis ohjaamaan erilaisten dokumenttien organisoinnilla ja työntekijöiden oikeuksia säätelemällä. Testauksen hallinta-työkaluun voidaan parhaimmillaan integroida vaatimus- ja suunnitteluvaiheen työkalut siten, että asiakasvaatimusten ja testitapausten välille syntyy yhteys, jolloin jäljitettävyyden tukeminen on mahdollista. Tällöin voidaan seurata jokaisen asiakasvaatimuksen täyttymistä testitapausten avulla.

2.3.2 Mittaaminen

Mittaaminen voidaan määritellä prosessiksi, jossa luvut tai symbolit liitetään reaalimaailman olioiden ominaisuuksiin, jolloin luvut kuvaavat ominaisuuksia selvästi määriteltyjen sääntöjen mukaan (Fenton 1994, 199). Ohjelmistotuotannossa tarvitaan mittaustietoa, jotta ymmärretään, mitä ohjelman kehityksen ja ylläpidon aikana tapahtuu. Lisäksi mittarit mahdollistavat projektien kontrolloinnin ja sitä kautta prosessin ja myös tuotteen parantamisen (Fenton & Pfleeger 1996).

Fentonin ja Neilin (2000, 361) mukaan metriikoiden käyttämisen kaksi päämotivaatiota ovat halu ennustaa kehittämisprosessin hyödyt sekä kustannukset ja tulevien ohjelmistotuotteiden laatu. Fenton (1994) toteaa, että ohjelmistojen mittaamisessa tulisi pitäytyä tieteellisissä periaatteissa, mikäli tavoitteena ovat yleisesti hyväksyttävät ja kelpaavat tulokset.

Ohjelmistometriikat voidaan luokitella monella eri tavalla. Erilaisia viitekehyksiä metriikoiden luokittelusta löytyy useita kymmeniä. Tutkijan kulloisestakin mielenkiinnon alueesta tai näkökulmasta katsottuna metriikat jäsentyvät heidän tarpeidensa mukaan. Hyvin yleinen tapa on jakaa metriikat kolmeen kategoriaan: tuotemetriikoihin, prosessimetriikoihin ja resurssimetriikoihin (Fenton & Neil 2000, 361).

Tuotemetriikat ovat tuotteen ominaisuuksiin liittyviä mittareita. Esimerkkejä näistä ovat koodin koko ja kompleksisuus. Prosessimetriikoita voidaan puolestaan käyttää parantamaan ohjelmiston kehittämistä ja ylläpitoa. Esimerkkinä mainittakoon virheen poistamisen tehokkuuden parantaminen. Resurssimetriikoilla taas kuvataan projektiin kuluvia resursseja. Tällaisia ovat mm. ohjelmistokehittäjien lukumäärä, kustannukset, aikataulu ja tuottavuus. (Fenton & Neil 2000, 361)

Toinen tärkeä luokittelu ohjelmistometriikoille on jako ennustaviin metriikoihin ja arvioiviin metriikoihin. Arvioivan mittaamisen avulla päätellään ominaisuuden tämän hetkinen arvo (esim. suuruus). Ennustava mittaaminen taas riippuu matemaattisesta mallista, jossa ominaisuuden arvo suhteutetaan aikaisempiin mittauksiin. (Harsu 2003, 12) Kolmas yleinen luokitteluperuste ohjelmistometriikoille on jakaa metriikat luokkiin niiden käyttötarkoituksen perusteella. Tällöin metriikat voidaan ryhmitellä esim. tuottavuuden metriikoihin, laadun metriikoihin sekä teknisiin metriikoihin (Harsu 2003, 10).

Fentonin ja Pfleegerin (1996) mukaan testausprosessin eri osien välistä tehokkuutta voidaan vertailla esim. kussakin vaiheessa löydettyjen virheiden

määrällä. Kun tietyssä vaiheessa löydettyjen virheiden määrä suhteutetaan kunkin virheen löytämisen kestoon ja kustannuksiin, saadaan myös kuva siitä, onko kyseinen testausprosessin osa kustannustehokas. Prosessin tehokkuuden lisäksi on tärkeää myös löytää mittari, jonka perusteella voidaan tehdä päätös tuotteen siirtämisestä seuraavaan testauksen vaiheeseen.

Pienikin projekti tarvitsee metriikoita tehokkaaseen projektin hallintaan (Weller 1994; Ogasawara, Yamada ja Kojo 1996; Bassin, Kratschmer, Santhanam 1998; Fenton & Neil 2000). Weller (1994) korostaa, että projektin laadun, tuottavuuden ja seurattavuuden hallintaan tarvitaan metriikoita. Ogasawara ym. (1996) suosittelevat erityisesti käyttämään metriikoita, jotka ovat helposti automatisoitavissa. Heidän näkökulmastaan erityisesti ohjelmiston laatua voidaan nostaa varhaisessa vaiheessa, jos käytettävissä on metriikkajärjestelmä, jolla saadaan luotettavaa tietoa projektin tilanteista prosessin jokaisen kierroksen aikana. Juuri näihin tavoitteisiin pyritään automatisoimalla testauksen hallinnan yhteydessä metriikkatiedon kerääminen. Kerätyistä tiedoista voidaan hallinnan tueksi laskea automaattisesti erilaisia asioita ja tulostaa graafisia kuvioita.

Daskalantonakis (1992) tuo tutkimuksessaan esille käytännön näkemyksiä ohjelmiston mittaamisesta Motorola yrityksessä. Tavoitteena yrityksellä oli mitatun tiedon perusteella parantaa ohjelmiston hallittavuutta. Tutkimuksen tuloksena löytyi useita etuja ohjelmiston hallittavuudelle. Daskalantonakis (1992) toteaa kuitenkin, että organisaatioissa olisi tärkeää ymmärtää, ettei mittaaminen sinänsä ole mikään tavoite. Tavoitteena on hallinnan parantaminen mitattujen tulosten, niistä tehtyjen analyysien ja palautteen avulla.

2.4 Yhteenveto

Luvussa on käsitelty varsinaista testausprosessia ja sen hallintaa. Testausprosessi alkaa Fewsterin ja Grahamin (1999, 13) mukaan testattavien

ominaisuuksien tunnistamisella. Tätä vaihetta ei kuitenkaan automatisoinnin yhteydessä useinkaan tuoda esille. Kauppisen ja Tainan (2003) mukaan testausprosessin on todettu alkavan vaatimusten pohjalta tehtävien suunnitelmien kirjoittamisella. Suunnittelua seuraa määrittelyvaihe, jossa määritellään yksittäiset testitapaukset. Testitapausten määritelmän pohjalta kirjoitetaan tarvittava koodi. Koodi suoritetaan ja tulosteena saadaan testi-tulokset. Kun testauskierroksella havaitaan virheitä, ne raportoidaan ja tulosten analysoinnin jälkeen annetaan kehittäjille korjattavaksi. Testikierrosten toistaminen päätetään, kun ollaan tyytyväisiä korjattuun ohjelmistoon. (Kauppinen & Taina 2003)

Testauksen hallinnan on todettu tukevan koko testausprosessia. Tässä tutkielmassa pyritään tuomaan esiin testauksen hallintavälineen prosessia tukevat ominaisuudet. Testauksen hallintavälineen avustavan ohjelmiston eri osilla voi hallinnoida testiprosessia organisoimalla testauksen dokumentteja (Giraudo & Tonella 2003) ja pitämällä kirjaa erilaisista metriikoista (Hetzel 1988). Näin testausprosessista ja sen kehittymisestä voidaan saada ajantasaista tietoa ja koko prosessi on paremmin uudelleenkäytettävissä ja ylläpidettävissä. Testausprosessin ja sen hallinnan käsittelyn jälkeen herää kysymys siitä, millaisia etuja automatisoinnin avulla on saavutettavissa. Tätä asiaa ja siihen liittyvää problematiikkaa käsitellään seuraavassa luvussa.

3 TESTAUKSEN AUTOMATISOINTI

Tämän luvun tarkoituksena on käsitellä testauksen automatisoinnin tavoitteita, työvälineitä sekä ongelma-alueita. Automatisoinnille asetetaan usein suuret tavoitteet ja niihin pyritään kehittämällä erilaisia työvälineitä. Testauksen automatisointi on kuitenkin vaikeaa ja automatisoinnin yhteydessä esiintyy paljon erilaisia ongelmia. Ongelmiin on pyritty löytämään luvun lopussa joitakin ratkaisuja.

3.1 Automatisoinnin edut ja tavoitteet

Testauksen hallinnan automatisoinnin tavoitteina voidaan pitää parempaa testiympäristön, dokumenttien sekä virheiden hallintaa. Edellä mainittuihin tavoitteisiin voidaan päästä juuri organisoimalla dokumentteja sekä mittaamalla erilaisia asioita. Testiympäristön hallintaan liittyy oleellisesti testien versionhallinta sekä järjestelmän muuttuvat konfiguraatiot. Virheidenhallinnan avulla puolestaan päästään käsiksi virheen elinkaareen, jolloin raportteihin voidaan kerätä tietoja tietyn virheen muuttuneista tiloista, luokitteluista ja priorisoinnista.

Lisäksi seuranta eri osa-alueilla helpottuu mitattavien tietojen perusteella. Onnistunut testauksen hallinta edellyttää kuitenkin koko testaustiimin kehittämistä. Testauksen koordinaattorin työtä voidaan helpottaa jakamalla hallinnoinnin työt järkeviksi työkokonaisuuksiksi ja ajoittamalla ne oikein. Avainasia on se, kuinka seurattavaksi testausprosessi voidaan tehdä testien suorituksen aikana sekä ennen ja jälkeen varsinaisia testiajokertoja. Tavoitteena on päästä reaaliaikaiseen seurantaan, jolloin projektin todellinen tilannetieto olisi koko ajan saatavilla.

Hayesin (1995, 494) mukaan automatisoinnin avulla voidaan saavuttaa testin toistettavuus, laajennettavuus ja kertymä. Laajennettavuudella tässä yhteydessä tarkoitetaan mahdollisuutta suorittaa testejä, jotka ovat manuaalisesti lähes

mahdottomia. Kertymä (accumulation) puolestaan ymmärretään mahdollisuutena suoriutua sovelluksen muutoksista vähemmällä määrällä testitapauksia, kuin manuaalisessa testauksessa. Viimeinen etu saavutetaan Hayesin (1995, 494) mielestä, kun testitapaustietokanta suunnitellaan niin hyvin ylläpidettäväksi, että ohjelmiston piirteiden määrän lisääntyessä testien määrä ei lisäänny. Tämä lienee kuitenkin käytännön elämässä hyvin harvinaista.

Fewster ja Graham (1999) ovat Hayesin kanssa pitkälti samoilla linjoilla automatisoinnin hyötyjen suhteen. Kuten aikaisemmin jo todettiin, ohjelmistojen testaaminen vaatii erittäin paljon resursseja sekä aikaa. Tästä syystä on järkevää vähentää manuaalisesti suoritettavan testauksen osuutta automatisoimalla testausprosessia. Erityisesti toistettavat testitapaukset tulisi toteuttaa automatisoinnin avulla, jolloin resurssien säästöt tulee parhaiten esille. Ajamalla regressiotestejä ohjelman uusissa versioissa voidaan suorittaa useampia testitapauksia vähemmässä ajassa (Fewster & Graham 1999, 9; Giraudo & Tonella 2003). Joitakin testitapauksia voidaan myös pyrkiä ajamaan muissa projekteissa lähes samalla tavalla, jolloin uudelleenkäytettävyyden hyöty voidaan maksimoida.

Toisaalta nähdään myös, että esimerkiksi säästö testausbudjetissa ja testausajan lyhenemisessä ovat vaikeita saavuttaa, koska juuri näiden tavoitteiden saavuttaminen edellyttää riittävän monta testauskierrosta. Giraudon ja Tonellan (2003) mukaan suurin osa automatisoinnin tavoitteista saavutetaan pitkällä aikavälillä.

Fewsterin ja Grahamin (1999, 9) mukaan testauksen automatisointi voi parantaa ohjelmistojen laatua lisäämällä testaajien aikaa perehtyä testaamaan toimintoja, joita ei voida automatisoida. Eli samalla määrällä resursseja voidaan lisätä testauksen määrää ja kattavuutta testauksenhallinnan automatisoinnin avulla. Lisäksi tietokoneet voivat ajaa testitapauksia esim. öisin sekä viikonloppuisin, jolloin testaajien olisi hankalampi testata vastaavia testitapauksia. On kuitenkin

muistettava, että testauksenhallinnan automatisointijärjestelmän ja -ympäristön luominen on useissa tapauksissa monin verroin resursseja vaativa toimenpide verrattuna manuaaliseen testaamiseen. Pitkällä aikavälillä testauksenhallinnan automatisointi on kuitenkin usein järkevää.

Automatisoinnin avulla saavutetaan myös testien yhdenmukaisuus. Tästä on etua standardien tarkistamisen yhteydessä. Automatisoinnin tuella on helposti tarkastettavissa, että samantapaiset toteutukset ja testaukset suoritetaan yhdenmukaisesti. Testauksen suorittaminen onnistuu lisäksi usein helpommin eri ympäristöissä valmiiden testitapausten avulla eli testien siirrettävyys on saavutettavissa helpommin. (Fewster & Graham 1999, 9-10) Testauksen yhdenmukaisuudella saavutetaan myös ennustettavuuden etu (Fewster & Graham 1999, 346). Esim. automatisoitujen testitapausten suorittaminen voidaan arvioida aikaisemman suorituksen perusteella. Tällöin myös koko testausprosessiin kuluva aika on helpommin ennustettavissa.

Testauksen automatisoinnin avulla usein tehostetaan erilaisia toimintoja, mutta myös suoritetaan toimenpiteitä, joita manuaalisesti olisi hyvin hankala tai jopa mahdotonta toteuttaa. Fewsterin ja Grahamin (1999) mukaan tällaisia testaustoimenpiteitä ovat mm. suorituskykytestaus sekä kuormitustestaus.

Tehokkain tapa varmistaa tuotteen laatu ja toimintavarmuus, niin että tuote saadaan markkinoille ajoissa, on järjestää kuormitustestaus (Elbert ym. 1994, 357). Kuormitustestauksessa (stress testing) tavoitteena on mittareiden avulla arvioida ohjelmiston luotettavuutta erilaisten kuormien määrillä (Chan 1995, 23; Elbert ym. 1994, 357). Liiallisen kiireen vuoksi ohjelmistolle saattaa jäädä tekemättä asianmukaiset kuormitustestaukset ja se osoittautuu usein pahaksi virheeksi tuotaessa tuotetta markkinoille. Suuren kuormituksen seurauksena ohjelmisto voidaan joutua poistamaan käytöstä kokonaan ennen kuin siitä on tehty toimiva suurille käyttäjämäärille.

Kuormitustestauksella saadaan hyödyllistä tietoa jo kehityksen alkuvaiheessa eri komponenttien käyttäytymisestä ja mahdollisten pullonkaulojen sijainnista tietojärjestelmässä (Chan 1995, 23). Näin voidaan laatia strategia siitä, mitä palveluja ja komponentteja monistetaan ja miten ohjelmiston ympäristövaatimukset kehittyvät. Jo palveluiden tuottamisvaiheessa tulee kuormituksen kehittymistä seurata, jotta palvelutason laskiessa tiedetään ryhtyä korjaaviin toimenpiteisiin.

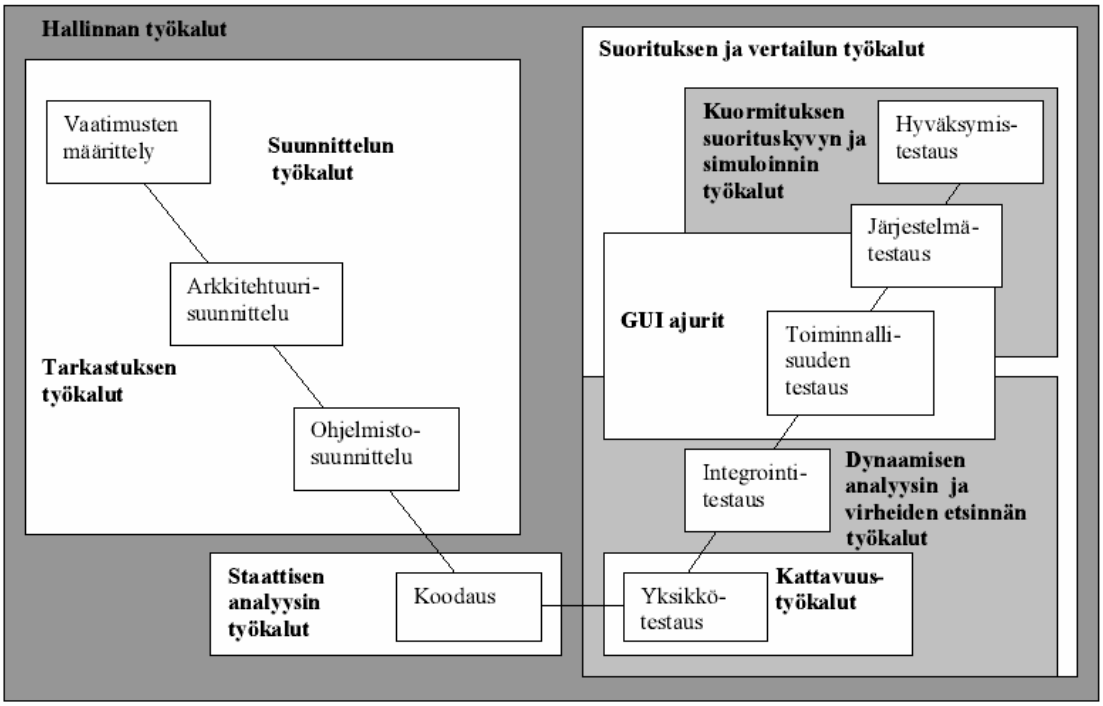
Tavoitteena on varmistaa, että sovellus toimii niin tehokkaasti kuin se on suunniteltu toimivaksi eli varmistetaan tuotteen toimintavarmuus sekä luotettavuus erilaisissa tilanteissa (Chan 1995, 23). Usein unohdetaan, että kuormitusmittauksella ostetaan myös aikaa. Halutaan siis selvittää, kuinka kauan kyseisellä järjestelmällä voidaan tulla toimeen, ennen kuin järjestelmää on ryhdyttävä laajentamaan. Mittausten avulla saadaan selville mm. websivujen latautumiseen kuluva aika sekä kuormituksen vaikutus tietokannan toimintaan.

Jotta kuormitustestaus olisi tehokasta, tarvitaan hyvät ja oikeanlaiset työkalut. Järjestelmän testauksessa tehdään mahdollisimman todenmukaisia käyttötilanteita. Järjestelmän kuormitustestauksessa voidaan arvioida ja tarkentaa pullonkaulojen parantamismahdollisuuksia. (Chan 1995, 26)

3.2 Työvälineet

Ihminen on edelleen testausjärjestelmän tärkein osa. Ennen kuin voidaan automatisoida, täytyy toimiva manuaalinen testausjärjestelmä olla olemassa. Järjestelmän tulee sisältää ensinnäkin yksityiskohtaiset testitapaukset sekä odotetut tulokset, jotka perustuvat vaatimusmäärittelyihin ja suunnitteludokumentteihin. Toisena vaatimuksena on erillinen testausympäristö, jossa on testitietokanta. (Zambelich 1998)

Ohjelmistotuotannon eri tasoilla on käytössä erilaisia testausta tukevia työkaluja. Kuvion 3 avulla on pyritty hahmottamaan testauksen automatisointia aikaisemmin esitetyn V-mallin avulla. Kuvion 3 V-malliin on liitetty testityökalujen käytön sijoittuminen ohjelmistotuotannon tasoille. Rajat eri ryhmien välillä ovat kuitenkin epäselvät vaikka kuviossa ne onkin rajattu selkeästi omiin ryhmiinsä. Jollakin työkalulla voi olla piirteitä useastakin eri ryhmästä. Kuviota tarkastellaan jatkossa tarkemmin.



KUVIO 3. Testauksen työvälineiden sijainti ohjelmistokehityksen elinkaaressa. (Fewster & Graham 1999, 7) Kuvion on suomentanut Pohjolainen (2003)

Hallinnan työkalut ovat käytettävissä koko ohjelmiston kehityksen elinkaaren ajan. Tämä työväline on rajattu kuvaan harmaalle pohjalle. Suunnittelun ja tarkastuksen välineitä käytetään vaatimusmäärittelyssä, arkkitehtuurisuunnittelussa ja ohjelmistosuunnittelussa. Suorituksen ja vertailun välineitä voidaan käyttää koko V-mallin oikeassa puoliskossa. Dynaamisen analyysin työkalut kuuluvat puolestaan yksikkö-, integrointi- ja toiminnallisuus-testaukseen. Ne arvioivat järjestelmän toimintaa ajon aikana. Kattavuustyökalut ovat erikoisesti

moduulitestaukseen suunniteltuja (Hall ym. 1997). Systeemi- ja hyväksymistestauksessa käytetään kuormitus- ja suorituskykytyökaluja. GUI (Graphical User Interface) -ajureilla on monia muiden ryhmien piirteitä mutta ne ovat selkeästi oma ryhmänsä. Ne ovat käyttökelpoisia koko testauksen toteutus- ja arviointialueella.

Aktiviteetit, joita voidaan tukea automatisoinnin avulla, ovat testauksen ehtojen tunnistaminen, testitapausten suunnittelu, testien rakentaminen, testien suorittaminen ja saatujen tulosten vertailu odotettuihin tuloksiin. (Fewster & Graham 1999, 13) Automatisointia varten kootaan yleensä ryhmä projektin henkilöistä. Ryhmässä on oltava kokemusta ohjelmoinnista, testauksesta ja automatisoinnista. Ryhmässä harkitaan tarkkaan, mitä ryhdytään automatisoimaan. Kuormitustestaukset sekä regressiotestaus, ovat yleensä otollisia automatisoinnin kohteita, koska niitä joudutaan toistamaan lähes samanlaisina useita kertoja (Fewster & Graham 1999). Oleellista on myös pohtia tarvittavien työvälineiden kustannuksia. Joskus kustannukset nousevat niin korkeiksi, ettei automatisointi yksinkertaisesti kannata.

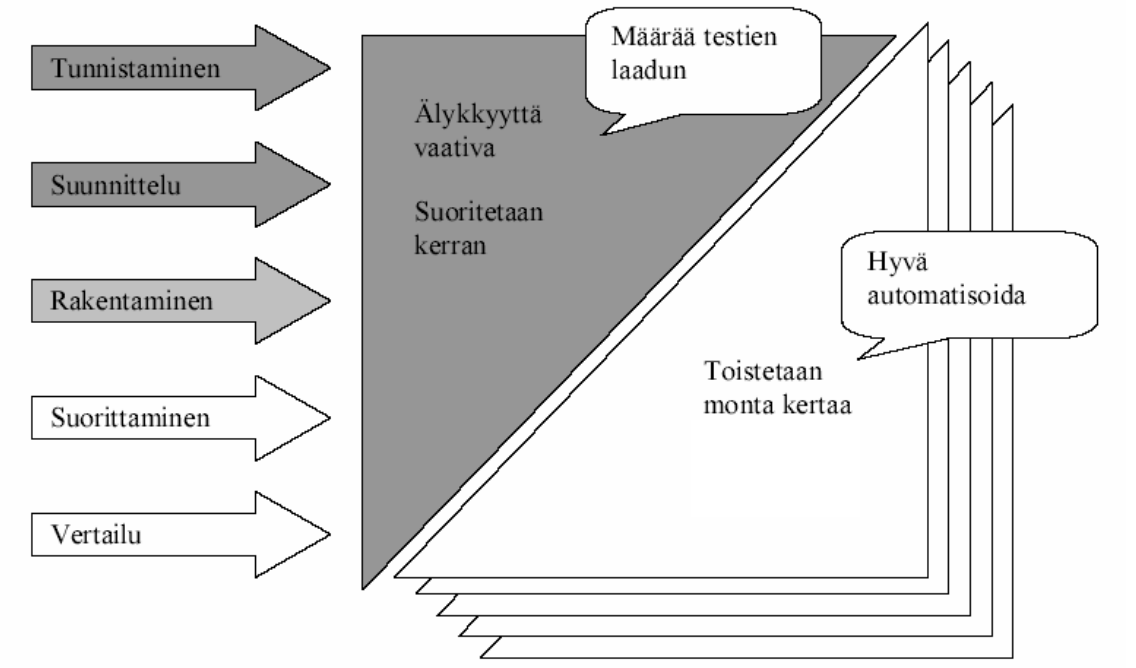
3.2.1 Testauksen hallinnan työvälineet

Hallinnan työkalujen tarkoituksena on automatisoida osittain testauksen suunnittelua, suorittamista, analysointia, dokumentointia ja raportointia. Näillä apuvälineillä voidaan luoda omia ympäristökokonaisuuksia. Tällaiset ympäristöt helpottavat paljon testauksen organisointia, analysointia ja raportointia. (Kautto 1996)

Testauksen hallinnan työkalut sijoittuvat koko systeemin elinkaaren alueelle (KUVIO 3). Esimerkkeinä testauksen hallintavälineistä ovat Test Manager ja Test Director (Kautto 1996). Test Manager mahdollistaa testauspaketin rakentamisen, suorittamisen ja hallinnan. Test Directorilla onnistuu puolestaan testien määrittely, suunnittelu, ajo, analysointi sekä raportointi.

3.2.2 Suunnittelun työvälineet

Fewsterin ja Grahamin (1999) mukaan älykkyyttä eniten vaativia toimintoja testauksessa ovat testauksen ehtojen tunnistaminen ja testitapausten suunnittelu (KUVIO 4). Älykkyyttä vaativat toiminnot määräävät pääasiallisesti testitapausten laadun. Kaksi viimeistä toimintoa, testitapausten suorittaminen ja testin tulosten vertailu, ovat puolestaan luonteeltaan yksinkertaisempia suorittaa. Tästä syystä ne sopivat hyvin automatisoitaviksi. Testitapausten suunnittelun automatisointi ei kuitenkaan aina ole järkevää, sillä se toteutetaan yleensä vain kerran. (Fewster & Graham 1999)



KUVIO 4. Testausprosessin jakautuminen eri aktiviteetteihin (Fewster & Graham 1999, 18)

Suunnittelun ominaisuuksista huolimatta on paljon tapauksia, joissa ainakin osa testitapausten suunnittelusta voidaan ja kannattaa automatisoida. Tällaisissa tilanteissa voidaan käyttää apuvälineinä testitapausten generaattoreita. Ne ovat hyödyllisiä monissa yhteyksissä, mutta eivät voi koskaan täysin korvata ihmisen rakentamia testitapauksia. Ongelmana testitapausgeneraat-

torin käytössä on, että ne voivat tuottaa todella suuren määrän testejä, joista suuri osa on tarpeettomia. Jotkut välineet osaavat tosin minimoida testitapausten määrää käyttäjän antamien kriteerien mukaan. Ongelmaksi kuitenkin jää, että testitapauksia tulee silti liikaa (Korel & Al-Yami 1998; Fewster & Graham 1999, 18). Monet valmistajat lupaavat kuitenkin jo nykyään apuvälineensä minimoivan testitapausten määrän, ja samalla maksimoivan niiden kattavuuden.

Apuvälineitä testauksen suunnittelun automatisointiin löytyy markkinoilta runsaasti. Esimerkkinä mainittakoon Inferno, joka on testitapausten generointiväline. Sen avulla saadaan toistettavia testitapauksia. (Pohjolainen 2002) Testitapauksia voidaan generoida rajapintoihin, koodiin tai määrittelyihin pohjautuen. Rajapintaan perustuva testigenerointi tuottaa testejä pohjautuen johonkin hyvin määriteltyyn rajapintaan, kuten graafisen käyttöliittymän rajapintoihin. Mikäli näyttö koostuu useista valikoista, painikkeista ja tarkistusruuduista, apuväline voi generoida testitapauksia, jotka käyvät tarkistamassa jokaisen. Generaattorin tuella voidaan luoda esimerkiksi testitapauksia, joiden avulla käydään tarkistamassa toimiiko ohje jokaisessa kentässä. (Fewster & Graham 1999, 20-21)

Koodiin pohjautuva menetelmä generoi testitapaukset tutkimalla koodin rakennetta. Toimintojen muodostama polku ohjelman läpi koostuu lohkoista, jotka määräytyvät haarautumisten mukaan jokaisessa ehtokohdassa. Näin voidaan tuottaa automaattisesti jokaisen polkulohkon tarvitsemat loogiset ehdot. Tämä on hyödyllinen piirre kattavuuden mittauksessa. Määrittelyyn pohjautuvalla menetelmällä voidaan generoida sekä testitapaukset että odotetut tulokset. Määrittelyjen on oltava sellaisessa muodossa, että apuväline voi tulkita niitä. Mukana testauksessa voi olla myös teknistä tietoa, esimerkiksi tiloja ja niiden muutoksia. Tietokantojen testaus on myös mahdollista. (Fewster & Graham 1999, 19-22)

Testitapausgeneraattorien ohella myös testisuunnitelmia voidaan generoida. Esimerkiksi Test Director -työvälineeseen kootaan testattavan tuotteen vaatimukset sekä testitapaukset oheistietoineen (Pohjolainen 2002). Testitapaukset linkitetään vaatimuksiin ja työvälineen avulla pystytään seuraamaan, onko kaikille vaatimuksille olemassa testitapaukset. Työväline tukee testauksen hallintaa ja sillä pystytään tarvittaessa tulostamaan koottujen testitietojen pohjalta testisuunnitelma.

3.2.3 Staattisen analyysin työkalut

Staattisen analyysin työkalujen tarkoitus on löytää virheitä ohjelmakoodista ilman varsinaisen ohjelman suorittamista (Adrion, Branstad & Cherniavsky 1982, 165). Yksinkertaisin esimerkki staattisen analyysin apuvälineestä on kääntäjä, joka ohjelmakoodin kääntämisen yhteydessä ilmoittaa koodissa havaituista virheistä. (Kautto 1996) Kääntäjä käy ohjelman lähdekoodia läpi rivi riviltä ja tuottaa tuloksena suorituskelpoisen ohjelman, eli ohjelmointikielellä kirjoitetusta lähdekielisestä ohjelmasta tuotetaan konekielinen ajettava.

Kääntäjä analysoi käännettävää lähdekoodia ja kykenee tuottamaan virheilmoituksia ja varoituksia koodissa esiintyvistä käytettävän ohjelmointi-kielen syntaksin vastaisista virheistä ja myös yksinkertaisimmista ohjelmakoodissa esiintyvistä ohjelmarakenteellisista virheistä (Adrion ym. 1982, 165). Kääntäjät siis kykenevät ilmoittamaan ohjelmakoodissa ilmenevistä käännösaikaisista virheistä.

Ohjelman mennessä käännöksestä läpi, siinä voi olla kuitenkin loogisia virheitä, joiden takia ohjelma ei toimi sille tarkoitetulla tavalla. Staattiset virheen-jäljitystyökalut löytävät osan ohjelmakoodissa olevista loogisista virheistä, esimerkiksi ehtolauseet, jotka ovat aina tosia, tai ohjelmarakenteessa olevan saavuttamattoman koodin. Ohjelmaan voi kuitenkin jäädä myös koodia, joka ei tee mitään tai aiheuttaa virheitä ajon aikana, mutta on silti täysin

ohjelmointikielen syntaksin mukaista. Tällaisten virheiden havaitseminen voi tapahtua vasta koodin suorittamisen aikana.

3.2.4 Testauksen suoritus- ja vertailutyökalut

Periaatteeltaan testauksen suoritus- ja vertailutyökalut ovat ohjelmia, jotka suorittavat testin käyttämällä testattavaa sovellusta samaan tapaan kuin testaajakin manuaalisesti testaa ohjelmistoa. Testityökalu suorittaa testattavalle ohjelmistolle sille määriteltyjä toimintoja samalla etsien virheitä ohjelman vasteista. Testausväline pystyy lisäksi vertailemaan testin tulosteita odotettuihin tulosteisiin. (Fewster & Graham 1999, 8-9)

Testausväline helpottaa siis testaajan työtä nopeuttamalla ja vähentämällä testaajan työtaakkaa (Chan 2000, 713; Giraudo & Tonella 2003). Testauksen osalta on usein ongelmana suurten ohjelmistojen testikattavuuden saaminen vaaditulle tasolle, johtuen ohjelmien syötteiden ja toimintojen muodostamista lukemattomista kombinaatioista. Testityökalun avulla päästäänkin parempaan testikattavuuteen työkalun mahdollistaessa suuremmat testiaineistot verrattuna testin manuaaliseen suorittamiseen (Giraudo & Tonella 2003). Lisäksi oikein suunnitelluilla testitapauksilla ja testityökalujen käytön kohdentamisella mahdollistetaan testaukseen panostettujen resurssien tehokkaampi käyttö. Työkalut tekevät myös vähemmän virheitä, toimittaessa suurten testiaineistojen kanssa. (Fewster & Graham 1999)

GUI-ajurit

Tähän ryhmään kuuluvat työkalut mahdollistavat automatisoidun käyttöliittymien testauksen. GUI-ajurilla on mahdollista tehdä mm. asiakas/palvelin -järjestelmien kuormitustestaus ja suorittaa nauhoitustoimintoja. Yksi välineistä on TestRunner, joka on tarkoitettu järjestelmätestaukseen. Työkaluilla on piirteitä monista eri ryhmistä ja ne sijoittuvat V-mallissa oikeanpuoleiseen haaraan. (Pohjolainen 2002)

Ajurit käyttävät GUI-karttoja nauhoittamiseen. GUI-kartoilla tarkoitetaan graafisessa käyttöliittymässä olevien komponenttien koordinaatteja eli niiden sijaintia sovellusikkunassa. Sijaintitietojen perusteella ajuri osaa suorittaa testiskriptissä määritellyt toimenpiteet oikeille komponenteille. GUI-kartat talletetaan tiedostona, joka ladataan suoritettavan testin käyttöön testin ajon aikana. Työkalut tunnistavat graafiset komponentit lomakkeelta ja luovat valmiita GUI-karttoja, joita on myös mahdollisuus päästä editoimaan: esim. poistamalla tiedostosta testin suorituksen kannalta tarpeettomia komponent-teja.

Automatisoitaessa GUI-testausta on työkalun valintaan suhtauduttava samalla vakavuudella kuin ohjelmiston kehitystyöhön yleensäkin. Työvälineen tulisi olla helposti ylläpidettävissä siten, että siirryttäessä versiosta toiseen on selvittävä mahdollisimman vähillä päivityksillä. Välineen on lisäksi oltava luotettava. Sen tulosten tulee kohdentua tarkasti testattavaan ohjelmaan. Välineen on oltava myös toimintavarma, jolloin virhetilanteen sattuessa se voi toipua itsenäisesti. (Hendricsson 1999)

Dynaamiset analyysin työkalut

Dynaamisella testauksella tarkoitetaan virheiden etsimistä ohjelmasta ohjelmaa tai sen osaa suorittamalla (Fewster & Graham 1999, 8). Suorituksenaikaisen testauksen ja analysoinnin mahdollistamiseksi tarvitaan ohjelmalle toimiva ympäristö. Yleensä ympäristö on käytettävissä, mutta suoritettaessa yksikkötestiä tai integrointitestiä testattavan ohjelmakomponentin ympäristö muodostetaan yleensä testiajureiden ja testitynkien avulla.

Dynaamisen analyysin työkaluilla saadaan myös tietoa testattavan ohjelman vaikutuksesta ympäristöönsä, esimerkiksi tietoa ohjelman muistin ja resurssien käytöstä (Fewster & Graham 1999, 8). Testaus suoritetaan joko käsin syöttäen testitapahtumia ja ohjelman tarvitsemia testisyötteitä tai antaen testiajurin lukea

annettavat syötteet testattavan järjestelmän ulkopuoliselta laitteelta, joka suorittaa tai syöttää testattavalle ohjelmalle sille määritellyt testitapahtumat.

Kattavuus-, kuormitus- ja suorituskykytyökalut

Koodin kattavuudella voidaan tarkoittaa useaa eri asiaa. Yksinkertaisimmillaan tutkitaan, mitkä koodirivit ohjelmasta on suoritettu. Tätä kutsutaan yleensä lausekattavuudeksi (statement coverage). Lausekattavuus tutkii, että kaikki ohjelman lauseet on suoritettu. Hieman vahvempia kattavuusvaatimuksia ovat päätöskattavuus (branch coverage) ja moniehtokattavuus (multicondition coverage). (Marick 1997)

Päätöskattavuus varmistaa, että kaikki ohjelman vuokaavion kaaret on käyty läpi. Ehtokattavuudessa kaikki ehtolauseet saavat sekä tosi- että epätosiarvot. Moniehtokattavuudessa kaikkien alkeisehtojen vaihtoehdot, tosi- ja epätosiarvojen kombinaatiot käydään läpi. Riippumattomien polkujen kattavuus (independent path coverage) käsittää kaikki polut, alkusolmusta loppu-solmuun, joita ei voida muodostaa muiden polkujen alipoluista. Täydellistä polkukattavuutta on mahdotonta saavuttaa ohjelmassa olevien silmukoiden vuoksi. (Marick 1997) Koodin kattavuutta mittaa esimerkiksi Rational PureCoverage. Se näyttää aina pyydettäessä, mitkä ohjelmarivit ovat vielä testaamatta. (Pohjolainen 2002)

Kuormitus- ja suorituskykytyökalut auttavat toiminnallisuus-, järjestelmä- ja hyväksymistestausvaiheissa. Ne ovat usein myös GUI-ajureita. Eräs näistä apuvälineistä on JavaLoad, joka on tarkoitettu Java-ohjelmien kuormitus-testaukseen. Se antaa raportteja käyttäjistä, testidatasta, keskimääräisistä vastausajoista jne. yhden istunnon aikana ja vertailee eri istuntojen tuloksia. (Pohjolainen 2002)

3.3 Automatisoinnin ongelmat

Tässä kohdassa tarkastellaan ongelmia, joita testauksen automatisointi aiheuttaa ja pohditaan, millaisia ratkaisuja ongelmiin löytyy.

3.3.1 Tunnistetut ongelmat

Testausprojektit epäonnistuvat yhtä todennäköisesti tai todennäköisemmin kuin muut ohjelmistoprojektit, koska yritykset eivät panosta yhtä paljon testausvälineisiin kuin toimitettaviin tuotteisiin. Vieläkin ollaan vaiheessa, jossa ihaillaan sokeasti testauksen automatisointia, tunnistamatta sen vaaroja. Automatisointiin pyritään vain, koska tietokoneita pidetään nopeampina, halvempina ja luotettavampina kuin ihmisiä. (Bach 1999)

Testausta automatisoitaessa törmätään usein monenlaisiin ongelmiin. Ongelmat, jotka tulevat täytenä yllätyksenä järjestelmän kehittäjille ja käyttäjille, ovat useasti hankalimpia käsitellä. Automatisointia ei voida pitää ”hopealuotina”. Hyvin suunniteltuna ja toteutettuna se tehostaa testaamista monella alueella, mutta se ei toimi ratkaisuna kaikkiin testauksen alueella esiintyviin ongelmiin. Epärealistiset odotukset lienevätkin suurin syy testauksen automatisoinnin epäonnistumisiin (Fewster & Graham 1999, 10).

Yleisin esimerkki epärealistisista odotuksista automatisoinnin suhteen lienee se, että automatisoinnin avulla voitaisiin löytää enemmän virheitä. Testaus-työvälineet mahdollistavat asioiden toiston nopeammin, mutta niiden avulla tuskin löydetään sellaisia virheitä, joita ei aikaisemmalla testauskerralla olisi havaittu tai joita manuaalisesti testaamalla ei löytyisi. (Fewster & Graham 1999, 10)

Testauksen automatisointi vaatii erityisen hyvän testitapausten suunnittelun ja toteutuksen. Jos yksikin testitapaus toimii virheellisesti testauksen aikana, niin siitä saatava tulos on yhtä todennäköisesti virheellinen (Fewster & Graham

1999, 10). Tällaisessa tapauksessa automatisointi saattaa johtaa tuloksillaan testaajan harhaan. Lisäksi Ahosen ja Junttilan (2003, 9-10) mukaan moduulitestausvaiheessa esiintyy useita ongelmia liittyen testauksen automatisointiin. Testaajien tulisi tällöin tehdä testiajurit samaan aikaan komponenttien valmistumisen yhteydessä. Aikataulu kaikesta huolimatta osoittautuu yleensä pullonkaulaksi ja kaikkia tarvittavia ajureita ei ehditä kehittämään. Lisäksi hankaluuksia syntyy epätäydellisten testitapausten ja tietojen analysoimisesta. Voi olla vaikea kehittää automatisointia huonon suunnitteludokumentaation avulla.

Ahonen ja Junttila (2003, 9-10) näkevät erityisen haastavana markkinoilla olevien testaustyökalujen epätäydellisyyden. Työkalut eivät ole tarpeeksi joustavia erilaisissa ympäristöissä työskentelyyn eikä niillä työskentely ole aivan yksinkertaista (Chan 2000, 714; Ahonen & Junttila 2003, 9-10). Työkalujen käyttöön ja niiden muokkaamiseen kuluu testaajilta paljon aikaa.

Testauksen suunnittelun ja toteutuksen lisäksi testauksen tekniset viat ja ongelmat saattavat aiheuttaa vaikeuksia. Testauksenhallinnan automatisointijärjestelmä on itsessään hyvin monimutkainen ja monenlaista toiminnallisuutta sisältävä kokonaisuus, joten sekään tuskin on täysin immuuni virheille. Pahimmassa tapauksessa testauksen hallintajärjestelmään ei voida liittää testattavaa ohjelmistoa ollenkaan, jos järjestelmä ei esim. tue juuri tarvittavia ohjelmistoja.

Testauksen hallintajärjestelmän käyttö vaatii testaajalta teknistä osaamista. Hänen on hallittava tarkat tekniset tiedot järjestelmän käytöstä. (Fewster & Graham 1999, 10) Jokaisen järjestelmää käyttävän tulisi aina käydä koulutus järjestelmän käytöstä. Automatisoitujen testitapausten ja ympäristön ylläpito vaatii resursseja. Testattavan ohjelmiston muuttamisen yhteydessä saatetaan joutua muuttamaan myös suuri osa testitapauksista. (Fewster & Graham 1999, 10; Chan 2000, 713) Järjestelmän ylläpito on useissa organisaatioissa

osoittautunut yllättävän suureksi työksi. Jos ylläpito arvioidaan testauksen alussa erityisen paljon resursseja vaativaksi, kannattaa harkita automatisoinnin järkevyys kokonaan uudelleen.

Testauksenhallinnan automatisointi ei sovi organisaatioihin tai projekteihin, joissa on huonosti organisoidut testausprosessit, vain vähän dokumentointia testauksen suunnittelusta tai huonosti toteutetut testitapaukset. Ensin on järkevää kehittää testausprosessi laadullisesti järkevälle tasolle ja vasta sitten lähteä tehostamaan prosessin suoritusta. (Fewster & Graham 1999, 11)

Testauksenhallinnan automatisointi ei ole triviaali tehtävä. Se tulisi toteuttaa hallitusti organisaation kulttuuriin. Tarvittavien työkalujen valinta ja käyttäjien kouluttaminen on suoritettava myös asiaankuuluvasti. Lisäksi automatisoitu järjestelmä vaatii yleensä vähintään yhden henkilön yrityksestä, joka pysyy ajan tasalla järjestelmän hallinnan ja kehittämisen suhteen. Hän esim. tekee päätökset siitä, mitä työkaluja järjestelmään milloinkin hankitaan. (Fewster & Graham 1999, 12) Chan (2000, 713) muistuttaa lisäksi, että kokonaisuudessaan tämä prosessi on kallis.

3.3.2 Ratkaisuehdotuksia ongelmiin

Ennen automatisoinnin aloittamista täytyy testauksen automatisointihanke perustella hyvin yritysjohdolle. Johdon on annettava suostumuksensa työkalun hankintaan ja sitouduttava hankkeeseen koko toteutusajaksi. Hyvät selvitykset valitun työkalun ominaisuuksista, arvioinnit sen soveltuvuudesta ja tulevista kustannussäästöistä ovat oleellista tietoa johdon tarpeisiin. (Fewster & Graham 1999)

Yrityksessä on kuitenkin hyvä tiedostaa, että taloudellinen hyöty saadaan vasta suoritettaessa testejä useaan kertaan. Automatisointia on nimittäin mahdotonta kehittää ilman huolellista suunnittelua. Suunnitelman pohjalta toteutettava testausohjelmisto saattaa olla lopulta jopa suurempi kuin testattava ohjelmisto.

Johdon odotukset tulisi suhteuttaa siihen, kuinka nopeasti automatisoinnista saadaan hyötyä. (Kaner 1997)

Suurimpia ja ehkä myös yleisimpiä virheitä, mitä testauksen automatisoinnin yhteydessä tehdään, on koota testausryhmä kokemattomista ohjelmoijista tai pelkästään loppukäyttäjistä (Fewster & Graham 1999; Pettichord 2001). Kanerin (1997) mukaan ryhmässä tulisi olla alan asiantuntemusta, mutta ehdottomasti myös kokemusta. Yrityksen kokenein ohjelmoija tai suunnittelija on sopiva henkilö johtamaan ryhmää.

Testiryhmän kokemattomuuden ohella sopivien testitapausten luominen on hyvin yleinen ongelma. Marick (1998) kertoo kestäneen kauan ennen kuin hän huomasi yrittävänsä automatisoida lähes kaikkia testitapauksia. Vain osa testeistä on sellaisia, jotka kannattaa ottaa mukaan automatisointiprosessiin. Kun on tehtävä päätös jonkin testin automatisoinnista, täytyy arvioida, montako koodimuunnosta tämä testi sietää. Mikäli testi ei kestä monta muunnosta, testin on oltava erittäin hyvä löytämään virheitä, ennen kuin sen automatisointi kannattaa.

Testaus kannattaa aloittaa lyhyillä testeillä ja pienellä testitapausten määrällä. Kun ohjelmisto näyttää toimivan oikein, voidaan testitapausten määrää lisätä tarpeen mukaan (Fewster & Graham 1999). Lisäksi on hyvä muistaa, että ihminen havaitsee sellaisia virheitä, jotka kone ohittaa. Toisaalta ihminen on konetta huonompi tulosten tulkinnassa. Jos virhe sijaitsee esimerkiksi luvun seitsemännessä desimaalissa, automatisoinnin työkalu löytää sen varmasti, mutta ihminen todennäköisesti ei. (Marick 1998) Myös tällaisia testitapauksia kannattaa siis automatisoida.

Fewsterin ja Grahamin (1999) mukaan testitapausten suoritusaika kannattaa pyrkiä pitämään mahdollisimman lyhyenä. Esimerkiksi jos testitapausten suoritus kestää testissä 30 minuuttia ja ensimmäisellä kerralla suoritus epäonnistuu 5 minuutin kuluttua, niin joudutaan suoritus käynnistämään

uudelleen. Toisella yrityksellä saatetaan päästä 10 minuutin kohdalle ja taas tulee virheitä, jolloin suoritus keskeytetään. Kolmannella kerralla päästään 15 minuuttiin asti. Tässä vaiheessa on käytetty jo 30 minuuttia testauksen ajasta, emmekä ole päässeet kuin puoleen väliin testin kokonaissuoritusta, vaikka tarkoituksena oli alun perin suorittaa tässä ajassa koko testi. Lisäksi testitapaukset pitäisi suunnitella sellaisiksi, että niillä on virheen tunnistamiskyky. Pelkkä ilmoitus, että testi ei mennyt läpi, ei riitä vaan pitäisi saada tietoa virheen laadusta ja sijainnista.

Testauksen automatisoinnin ylläpidettävyys osoittautui monissa lähteissä haasteelliseksi, kuten tutkielmassa aikaisemmin jo mainittiin. Testipaketin ylläpidettävyyttä helpottaisi, se ettei paketin anneta kasvaa liian suureksi. Testitapausten lisääminen helpottaa ja varmentaa testausta, mutta vaikeuttaa ylläpitoa. Huolellisella suunnittelulla on tässäkin yhteydessä tärkeä merkitys. Testitapausten määrälle voidaan asettaa tietty yläraja, mutta tämä voi estää hyvienkin uusien testien mukaan pääsyn. On myös hyvä ajoittain käydä läpi olemassa olevat testitapaukset pohtien testitapausten tarpeellisuutta ja päällekkäisyyttä. (Kaner 1997; Fewster & Graham 1999)

Mielestäni testien ylläpidettävyyttä voitaisiin kehittää kokoamalla testitapauksista eräänlaista komponenttikirjastoa. Yrityksissä tehdään usein osastoittain hyvin samankaltaisia projekteja, jolloin aikaisemmin kehitetyt testitapaukset kannattaisi hyödyntää. Esimerkiksi kirjauduttaessa Web-käyttöliittymään sisälle joudutaan tarkistamaan aina käyttäjätunnukset ja salasanat samalla tavalla. Kertaalleen toteutettuja testitapauksia voitaisiin helposti hyödyntää tulevissakin projekteissa.

Fewsterin ja Grahamin (1999) mukaan testitapausten ja skriptien nimeämiseen kannattaa ottaa heti alussa yhteinen käytäntö. Tämä helpottaa huomattavasti niiden löytämistä myöhemmin. Mikäli nimeäminen tapahtuu vapaasti, se johtaa

ennen pitkää kaaokseen. Testitapaukset ja skriptit täytyy dokumentoida hyvin. Tärkeää dokumentoinnissa on laatu, ei sen määrä.

3.4 Yhteenveto

Luvussa kartoitettiin testauksen automatisoinnin tavoitteita, työvälineitä sekä ongelma-alueita. Testauksen automatisoinnille on asetettu paljon erilaisia tavoitteita, joihin työvälineiden avulla pyritään. Selkeästi esille nousivat testauksen nopeampi suorittaminen ja testauksen toistettavuus. Testauksen automatisoinnin avulla on myös pyritty vapauttamaan resursseja testauksen älykkyyttä vaativille osuuksille, jolloin testattavan tuotteen laadun odotettiin parantuvan.

Fewsterin ja Grahamin (1999, 7) esittämä malli työvälineiden luokittelemiseksi osoittaa, että työvälineitä löytyy jokaiseen V-mallin vaiheeseen. Mallia tutkimalla selvisi, että välineillä on paljon päällekkäisiä ominaisuuksia. Riippuu siis työvälineen käyttäjästä ja välineen toteuttajasta pitkälti, mitä ominaisuutta hän painottaa ja mihin luokkaan välineen sijoittaa. Lähes poikkeuksetta kaikissa löytämissäni lähteissä testauksen automatisointia pidettiin erittäin haastavana tehtävänä, jossa oli suuret mahdollisuudet epäonnistua. Kuitenkin hyvin toteutettuna automatisointi voi tehostaa testausta ja se on ehdoton edellytys testauksen tulevalle kehitykselle. Tästä näkökulmasta on tutkimuksen kohdeyrityksessä lähdetty automatisoimaan testausprosessia ja testauksen hallintaa. Välineen valmistuttua on organisaatiossa pystyttävä arvioimaan kehitystyön onnistuneisuutta ja analysoimaan tilannetta tarkemmin. Näihin asioihin perehdytään seuraavassa luvussa.

4 LAADUN ARVIOINTI TESTAUSVÄLINEEN KÄYTTÖÖNOTTOPROSESSIN YHTEYDESSÄ

Tämän luvun tarkoituksena on käsitellä testauksen hallintavälineen käyttöönottoprosessia sekä määritellä käyttöönottoprosessia varten tarvittava metriikkapatteristo laadun arvioimiseksi pilottiprojektissa. Luvussa pohditaan, millaista tietoa prosessin ja välineen laadusta tarvitaan, jotta edellisessä luvussa esiin tulleita etuja ja ongelmia voitaisiin analysoida.

4.1 Välineen käyttöönottoprosessi

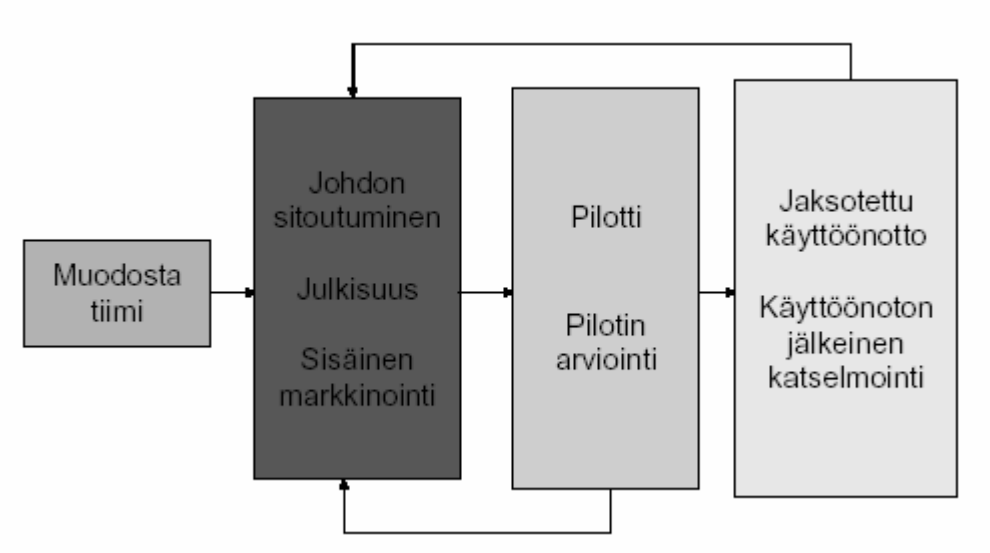
Apuvälineen valmistumisen jälkeen alkaa todellinen työ yrityksessä. Vaikka apuvälineen toteutus on tehty huolellisesti, ei ole mitään takeita, että työkalun käyttämisessä onnistutaan. Fewsterin ja Grahamin (1999, 283) mukaan on paljon yrityksiä, jotka ovat menestyksellisesti valinneet ja hankkineet työkalun, mutta suurin piirtein puolet näistä yrityksistä ei ole kokenut saaneensa välineestä mitään hyötyä.

Kun yritykseen tuodaan uusi testauksen työkalu, se muuttaa tapaa, jolla ihmiset työskentelevät. Ihmiset eivät yleensä pidä muutoksista, mutta on olemassa tapoja, joilla muutosprosessia voidaan helpottaa. Muutosprosessia on mahdollista keventää muun muassa hyvällä koulutuksella. (Fewster & Graham 1999, 285) Tässä luvussa kuvataan testausapuvälineen käyttöönottoprosessia, mutta samat lait koskevat usein kaikkia muutosprosesseja ja siten kyseisen prosessin vaiheita voitaisiin yleistää muihinkin muutosprosesseihin.

Edellisestä näkemyksestä hieman poiketen Kettusen ja Simonsin (2001, 21) mukaan tietoteknologian toteuttajat ja käyttäjät muovaavat tietojärjestelmät toimiviksi välineiksi. Välineen hyötyjen realisoituminen on siis kiinni ihmisistä ja organisaatioista, jotka ottavat käyttöön ja käyttävät välinettä. Samaa tietoteknologiaa voidaan ottaa käyttöön ja käyttää eri tavoin, koska tietojärjestelmän käyttöön, johtamiseen, ylläpitoon ja muuttamiseen on

vaikuttamassa monia tekijöitä ja prosesseja. Tästä syystä tämän tutkimuksen käyttöönottoprosessia tarkastellaan juuri testauksen automatisoinnin yhteydessä esitellyn mallin mukaisesti.

Apuvälineen valintaprosessissa vähennetään vaihtoehtojen lukumäärää asteittain. Käyttöönotto on päinvastainen prosessi. Työkalun hyväksymistä, käyttöä ja saavutettavia etuja laajennetaan asteittain, kuten kuvio 5 osoittaa. (Fewster & Graham 1999, 284) Kuviota tarkastellaan jatkossa tarkemmin.



KUVIO 5. Apuvälineen käyttöönottoprosessi (Fewster & Graham 1999, 284)

4.1.1 Tiimin muodostaminen

Muutosprosessiin liittyy useita rooleja, joissa toimivien henkilöiden valinta voi muodostua muutoksen onnistumisen kannalta olennaiseksi. Oikeat henkilöt oikeissa rooleissa pitävät pitkänkin kehityshankkeen käynnissä ja motivoivat muita kehityksen osapuolia. Pienessä yrityksessä nämä roolit voivat olla kaikki yhdellä henkilöllä, mutta suurissa yrityksissä kutakin roolia saattaa hoitaa eri henkilö. Fewsterin ja Grahamin mukaan (1999, 285-287) tarvittavia rooleja ovat työvälineen puolestapuhuja, muutosagentti, johdon takuumies, työvälineen tekninen hoitaja sekä työvälineen kehittäjät. Näiden roolien avulla pyritään

varmistamaan, että kuvion 5 kaikki vaiheet pystytään totuttamaan turvallisesti, ilman yllätyksiä.

Hankkeen johtajalla on prosessissa erittäin oleellinen osa. Davenportin (1993 , viitannut Forsell 2001, 5) mukaan koko johdon tulee sitoutua ja osallistua organisaation muutokseen, mutta viime kädessä muutoksen pitää olla yhden määrätietoisen ja asialleen omistautuneen johtajan vastuulla. Vastuullisella johtajalla täytyy olla riittävästi valtaa päättää muutosta koskevista asioista. Davenport (1993 , viitannut Forsell 2001, 5) jopa väittää, että johtajan tyytymättömyys senhetkiseen tilanteeseen sekä tyytymättömyydestä johtuva sitoutumattomuus muutokseen on tärkein yksittäinen tekijä onnistumisen kannalta. Vähentääkseen vastustusta hyvän johtajan tulee tiedottaa muutoksen tarpeellisuudesta sekä luoda realistiset ja positiiviset odotukset lopputulokselle. (Forsell 2001, 5) Kaikkien organisaation työntekijöiden on tiedostettava olemassa olevan prosessin ongelmat ja muutoksen tarpeellisuus.

Työvälineen puolestapuhujana toimii henkilö, joka uskoo vahvasti automatisoinnin etuihin. Hänellä on vahva usko hyötyihin, joita automatisointi yritykselle tarjoaa. Kyseisen roolin edustajaa voitaisiin kutsua eräänlaiseksi sanansaattajaksi. Hänen tehtävänään on vakuuttaa muut yrityksen työntekijät työvälineen hyödyllisyydestä. Työvälineen puolestapuhuja tarvitsee tuekseen eräänlaisen johdon takuumiehen, jolla on kokemusta ja valtaa yrityksessä. (Fewster & Graham 1999, 285-287; Forsell 2001, 6)

Muutosagentti puolestaan suunnittelee ja hallitsee muutosprosessia organisaatiossa. Hänen tehtävänään on suunnitella, mitä muutoksia tapahtuu henkilöstölle milloinkin. (Allen 1995; Winston 1999) Hän johtaa ihmiset muutosprosessin läpi. Muutosagentti tarvitsee tuekseen työvälineen teknisen hoitajan. Tekninen hoitaja vastaa välineen teknisestä tuesta. Hän tarjoaa yleisesti apua teknisissä ongelmissa ja konsultoi välineen käyttöönottajia sekä toimii yleisesti muutosagentin apuna. (Fewster & Graham 1999, 285-287)

Agenttien tulee olla innokkaita johtamaan muutosprosessia ja heidän täytyy kyetä sekä teknisesti että poliittisesti ymmärtämään prosessiin liittyviä ongelmia, ja varmistamaan, että tehokkaat ratkaisut toteutetaan. Lisäksi agenteilla täytyy olla muutoksen kohteiden kunnioitus sekä johdon luottamus ja tuki. Agentit muodostavat projektien rajat ylittävän tiimin, jossa kaikki muutoksen kohteet ovat edustettuina. Kaikkien osapuolten edustus varmistaa sen, että kelvollinen suunnitelma saadaan aikaan ja muutos hyväksytään suuremmalla todennäköisyydellä. (Humphrey 1989 , viitannut Forsell 2001, 7). Siinä missä Davenport korostaa johtajan asemaa osana muutosprosessia, Allen (1995) ja Winston (1999, 70) nostavat muutosagentin roolin hyvin merkittäväksi muutoksen läpiviennin kannalta. Muutosagentin tehokkuus työssään näkyy heidän mukaansa koko muutosprosessin tehokkuutena.

Tiimin tulee miettiä keinoja siihen, miten prosessia saadaan aktiivisesti kehitettyä siten, että normaali toiminta ei häiriinny. Usein lähes kaikki edellä mainituista rooleista kuuluvat varsinaisen työvälineen kehittäjille tai valitsijoille. Lisäksi välineen kehittäjien tehtävänä on tulevien tarpeiden mukaan päivittää ohjelmistoa, jotta sitä voidaan onnistuneesti yrityksessä käyttää.

Forsellin (2001) mukaan muutoksen kohteilla ilmenee suurella todennäköisyydellä luonnollista muutosvastaisuutta, jota muutoksen johtamisella tulee vähentää. Muutosvastaisuutta aiheuttavat mm. pelko ja epävarmuus. Ihmiset eivät halua muuttaa vuosien kuluessa opittuja, hyväksi havaittuja toimintatapoja. Keinoja muutosvastaisuuden vähentämiseen ovat tiedotus ja koulutus, sekä muutoksen kohteille suotava mahdollisuus ilmaista mielipiteensä heitä koskevissa päätöksissä. Loppujen lopuksi muutosprosessi on yhteistyötä ja sen onnistuminen riippuu kaikista osallistujista. Jos työyhteisö nähdään oppimisympäristönä ja yksilöt ovat sitoutuneita oppimaan uutta, voidaan muutosprosessin olettaa onnistuvan organisaatiossa. (Forsell 2001)

4.1.2 Käyttöönottoon sitoutuminen ja julkisuus

Johdon tulee sitoutua käyttöönottoprosessiin konkreettisen kädenpuristuksen lisäksi jatkuvalla tuen antamisella käyttöönoton ajan. Varsinkin ongelmallisten tilanteiden aikana on oleellista, että johto on sitoutunut auttamaan sekä taloudellisesti että ottamalla kantaa asioihin tilanteen niin vaatiessa. Työvälineen puolestapuhuja ja johdon takuumies esittelevät liiketoiminta-suunnitelman valitusta apuvälineestä kaikille, jotka ovat asian osallisia tai siitä kiinnostuneita. He esittelevät tiivistetysti välineen hankintaan johtaneet syyt sekä selvittävät välineestä saatavat edut. Lisäksi he esittelevät suunnitelman ja realistisen arvion välineen käyttöönottoprosessista. (Fewster & Graham 1999, 288)

Yleensä ihmiset aliarvioivat julkisuuden merkityksen työvälineen käyttöön-otossa. Yksi esitys suurelle määrälle ihmisiä ei riitä, vaan on valmistauduttava esittelemään välinettä erilaisissa tilanteissa erilaisille yleisöille. Kaikkein mielenkiintoisin julkisuus välineelle saadaan kuitenkin vasta, kun se todellisesti otetaan käyttöön pilottiprojektissa. Pilottiprojektissa saavutetut edut pitää tuoda laajasti esille yrityksessä, jotta mielenkiinto työvälineeseen syntyy. Toisaalta myös ominaisuudet, jotka eivät toimi toivotulla tavalla, on realistisesti osattava tuoda esille, jotta tulevat välineen käyttäjät eivät pety työvälineen ominaisuuksiin liiallisten epärealististen odotusten vuoksi. (Fewster & Graham 1999, 288-289)

4.1.3 Pilottiprojektin valinta ja tavoitteet

Pilottiprojektissa on tarkoitus käytännössä testata ratkaisun toimintaa. Pilottiprojektin valinnassa on kaksi tärkeää kriteeriä: Ensinnäkin projektin tulee olla samankaltainen kuin ne tulevat projektit, joihin tutkittavan kohteen halutaan ulottuvan. Toiseksi projektin onnistumisen todennäköisyyden täytyy olla mahdollisimman korkea. (Fewsterin ja Grahamin 1999; Forsell 2000;

Giraudo & Tonella 2003) Giraudo ja Tonella (2003) korostavat lisäksi sitä, että pilottiprojektissa ei saisi olla kiirettä, jotta mittaukset ehditään suorittaa häiriöttä. Lisäksi automatisointia kannattaa käyttää erityisesti projekteissa, joissa joudutaan toistamaan samoja testejä useaan kertaan. Tällöin projekti ei saa olla lyhytkestoinen. (Fewster & Graham 1999; Giraudo & Tonella 2003)

Yleisesti ottaen ehkä juuri merkittävin ja helpoiten miellettävissä oleva tavoite pilottiprojektilla on hankkia kokemusta työkalun käytöstä. Ensivaikutelma välineen käytöstä on aina kokeellista, mutta hyvä näin. On nimittäin hyvä epäonnistua tärkeissä asioissa ensin pienessä mittakaavassa ja analysoida epäonnistumisen syitä kuin kärsiä tappioita useissa hankkeissa hallitsemattomasti heti aluksi. (Fewster & Graham 1999, 290-292)

Asiantuntemus työvälineen käytöstä rakentuu vasta ajan myötä. Jotta käyttö sujuisi hyvin, tarvitaan pilottiprojektista kokemuksia monen alan osaamisesta ja yhteistyön onnistumisesta. Tietoa tarvitaan mm. skriptaustekniikoista, vertailujen tekemisestä ja siitä, kuinka saada työkalu suhtautumaan erikoistapauksiin. Tulevien käyttäjien tulisi tietää esim. mitä kannattaa välttää. (Fewster & Graham 1999, 290-292)

Oleellista on kuitenkin havaita, että työntekijöiden roolit muuttuvat, sillä testaajan lisäksi tarvitaan automatisoijaa. Roolien muutoksen ohella testauksen suunnittelu muuttuu ja dokumentaatiomuodossakin on huomioitava uudet tarpeet. Tehtävien järjestys on harkittava automatisoinnin yhteydessä tarkoin: Koska päätös testin automatisoinnista tehdään? Kuinka pian aloitetaan testien rakentaminen (automatisointi)? Kuka analysoi virheraportit ja koska? (Fewster & Graham 1999, 290-292)

Muutoksia vaativat testauksen automatisoinnin myötä sisäiset standardit ja käytännöt. Mm. skriptidokumentaation muoto ja sijainti sekä testiaineiston arkkitehtuuri tulee määritellä ymmärrettävästi ja yksiselitteisesti. Lisäksi tulosten oikeellisuuden vertailun lähestymistavat tulee määritellä selkeästi.

(Fewster & Graham 1999, 290-292) Käytännöt tiedostojen, skriptien, testien jne. nimeämisessä skriptisuunnitelmaan on myös hyvä huomioida (Fewster & Graham 1999).

Pilottiprojektin jälkeen arvioidaan saadut tulokset ja kokemukset sekä päätetään mahdollisista jatkotoimenpiteistä, esimerkiksi tuleeko menetelmää vielä kehittää ja miten, vai otetaanko uusi väline yrityksen testausprosessin osaksi. Mikäli pilottiprojekti onnistuu lähes odotusten mukaisesti, voidaan ryhtyä suunnittelemaan työvälineen käyttöönottoa yrityksessä. (Fewster & Graham 1999, 290-292)

4.1.4 Käyttöönotto

Fewsterin ja Grahamin (1999, 293) mukaan käyttöönotto on suuri toimenpide jokaisessa organisaatiossa ja ilman hyvää suunnittelua siitä tuskin tulee menestyksellinen. Pilottiprojektin tulosten analysoinnin pohjalta pyritään löytämään käyttöönottoa tukevaa tietoutta. Käyttöönoton yhteydessä suunnitellaan pilottiprojektissa ilmenneet muutostarpeet ja julkistetaan välineen kokeilun jälkeen pilottiprojektista saadut hyvät kokemukset. Tärkeää on myös muokata yrityksen säännöt ja strategiat, niin että ne huomioivat työvälineen käyttöönoton. Lisäksi tulee varmistaa, että projektipäälliköt huomioivat testauksen automatisoinnin projektisuunnitelmassa sekä laatu- ja testisuunnitelmissa asianmukaisesti. (Fewster & Graham 1999, 293)

Pilottiprojektista kerättyjen tietojen avulla rakennetaan hyvä infrastruktuuri testausmenetelmälle. Dokumentoidaan ja määritellään testauksen suorittamiseen tarvittavat laitteet sekä installoitavat ohjelmat. Määritellään myös, kuinka testiskriptit tulee toteuttaa ja mihin ne tallennetaan, jotta ne ovat järjestelmän käytettävissä. Testien suorittamiseen tarvitaan vakaa, hallittava, edustava testausympäristö. Ympäristön tulee olla eristetty muista ympäristöistä

ja testausvälineet ja toimistotilat tarvitaan testausta tekeville. (Fewster & Graham 1999, 293-294)

Käyttöönoton yhteydessä tähdätään siihen, että automatisointi olisi lopulta helpompaa kuin manuaalinen testaaminen, ja seurataan automatisoinnin tehokkuutta. Testauksen automatisointi voi olla hyödyksi vain, jos automatisoinnin avulla säästetty vaivannäkö on selkeästi suurempi kuin automatisoinnin käyttöön kuluva vaivannäkö. Jotta oikeasta kehityksestä voitaisiin olla varmoja, tulee tehokkuutta seurata mittaamalla. (Fewster & Graham 1999, 293-294)

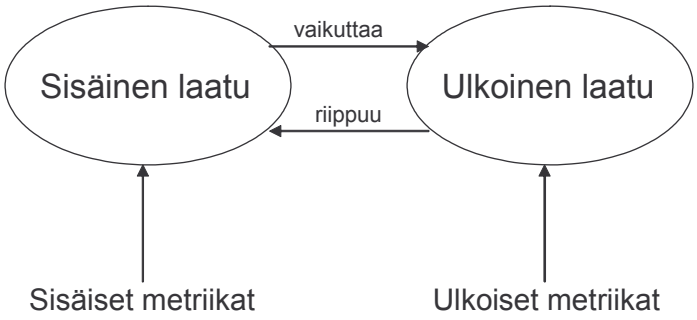
Käyttäjät tulee kouluttaa työvälineen käyttöön. Koulutuksen merkitystä käyttöönoton yhteydessä ei tulisi väheksyä. Asioita voi tehdä monella tavalla ja jotta välineestä saataisiin mahdollisimman suuri hyöty, olisi toivottavaa, että käyttäjät osaisivat käyttää välinettä mahdollisimman tehokkaasti. (Fewster & Graham 1999, 293-294) Tosin Kettusen ja Simonsin mukaan (2001, 21) ei kuitenkaan riitä, että käyttäjät osaavat käyttää järjestelmää. Heidän pitää myös osata liittää välineen käyttö osaksi omaa työtään ja tunnistaa sen välittämä organisatorinen yhteistyö sekä pystyä toimimaan sen avulla poikkeustilanteissa ja kehittää työtänsä (Kettunen & Simons 2001, 21).

4.2 Metriikkapatteristo laadun arviointiin

Laatu voidaan määritellä monella tavalla. ISO/IEC (2002, 3-4) erottaa toisistaan sisäisen ja ulkoisen laadun. Sisäisellä laadulla tarkoitetaan tuotteen attribuuttien täydellisyyttä, millä määritetään tuotteen itsensä kykyä tyydyttää määrätyt ja oletettavissa olevat tarpeet tuotetta käytettäessä. Sisäistä laatua pyritään yleensä arvioimaan tuotteen elinkaaren alkuaikana, sillä se liittyy läheisesti ohjelmakoodin piirteisiin, jolloin tuotetta vasta kehitetään. Näkökulmana on yleensä tuotteen kehittäjän näkemys laadusta suunnitelmien pohjalta.

Ulkoisella laadulla tarkoitetaan mittaa, millä tuote tyydyttää määrätyt ja oletettavissa olevat tarpeet ollessaan käytössä tietyissä olosuhteissa. Ulkoisia metriikoita käytetään laskemaan ohjelmistotuotteen laatua laskemalla sen systeemin käyttäytymistä, missä tuote toimii osana. ISO/IEC (2002, 3-4) Ulkoiset attribuutit näkyvät siis ohjelmiston käyttäjälle ja ne liittyvät lähinnä järjestelmän ja ohjelmiston käyttöön ja sen helppouteen.

Kuvion 6 osoittamalla tavalla ohjelmistotuotteen sisäinen ja ulkoinen laatu ovat tiukassa vaikutussuhteessa keskenään, eikä selvää rajaa niiden välille ole aina vedettävissä ja jokin mitattava attribuutti voi hyvinkin kuulua esim. sekä ulkoiseen että sisäiseen laatuun.

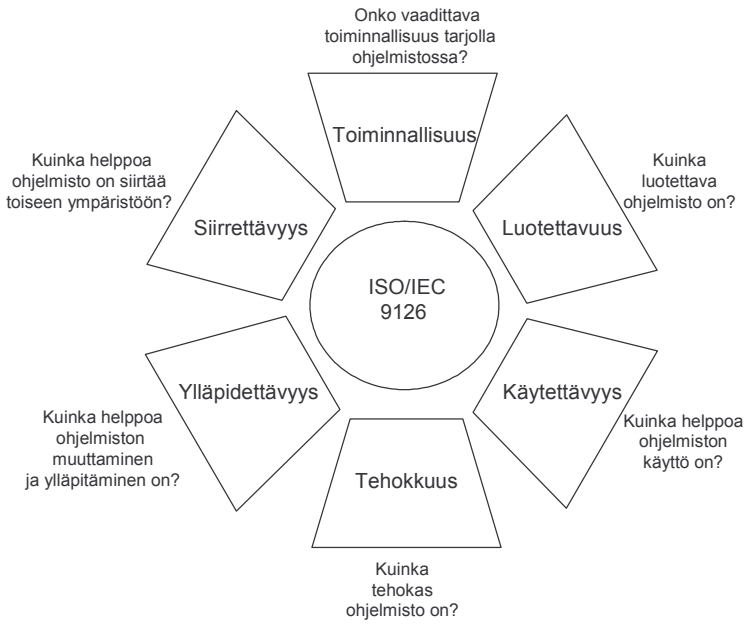


KUVIO 6. Erityyppisten laatumetriikoiden keskinäiset suhteet ISO/IEC (2002, 3) standardin mukaan.

Ohjelmistoille on kehitetty useita laatumalleja (Gueheneuc & Khosravi 2004), joista tunnetuimpiin kuuluvat mm. McCallin (McCall ym. 1976) ja Boehmin (Boehm ym. 1976) laatumallit sekä FURPS (Grady & Caswell 1987) malli ja ISO 9126 (ISO/IEC 9126 1991) laatumalli. Malleja vertailemalla selviää, että niissä on hieman eroja. McCallin ja muiden (1976) laatumalli on kehitetty ensimmäisenä ja Boehm ja muut (1976) ovat tehneet samankaltaisen mallin myöhemmin hieman täydentämällä edellä mainittua mallia. Boehm ja muut (1976) eivät ole liittäneet laadun ominaisuuksiin lainkaan metriikoita. Ortegan ym. (2003) tekemän laatumallien vertailun pohjalta selviää lisäksi, että FURPS-malli sisältää muihin malleihin verrattuna vähiten laadusta mitattavia ominaisuuksia.

Mallin ulkopuolelle jää täysin esim. ohjelmiston siirrettävyys, joka muissa malleissa on huomioitu.

Edellä mainituista laatumalleista uusin on standardisoitu ohjelmiston laatumalli ISO 9126 (KUVIO 7). Kansainvälinen standardointijärjestö (ISO) on kehittänyt ISO 9126 -standardin tuotelaadun arviointia varten. Standardi määrittelee kuusi piirrettä, jotka määrittelevät tuotteen laadun. Kuviossa 7 on esitelty kaikki kuusi piirrettä ja lyhyesti ilmaistu, mitä kukin piirre tarkoittaa.



KUVIO 7. ISO 9126:n laatuominaisuudet (ISO/IEC 2000a)

Fewsterin ja Grahamin (1999, 219-228) malli testauksen automatisoinnin mittaamiseksi on hyvin samankaltainen ISO/IEC 9126 -mallin (ISO/IEC 2000a) kanssa. Fewsterin ja Grahamin (1999, 219-228) malli testauksen automatisoinnin arvioimiseksi soveltuu aiheeseen hyvin, mutta se ei sisällä tarkkaa dokumentaatiota tarvittavista metriikoista. Tarvittavien metriikoiden määrittelyn ohella ISO/IEC 9126 -laatumallin eduiksi tämän tutkimuksen näkökulmasta voidaan laskea ulkoisen ja sisäisen laadun erottelu. Lisäksi malli on hyvin samankaltainen Fewsterin ja Grahamin (1999) luomaan malliin

verrattuna. Näistä edellä mainituista syistä ISO 9126 -malli (ISO/IEC 2000a) on valittu tämän tutkielman laatumalliksi.

Fewster ja Graham (1999, 219-228) suosittelevat mittauksissa päätettäväksi kolmesta neljään mitattavaa ominaisuutta, jotta mittaukset pystyttäisiin näiltä osin toteuttamaan mahdollisimman hyvin. Myös Grady (1993, 67) suosittelee laadun metriikoita mitattaessa pohtimaan, mitkä ovat tuotteen kaikkein tärkeimpiä ominaisuuksia, joista halutaan kerätä tietoa laadun parantamiseksi, ja keskittymään näihin muutamiin ominaisuuksiin.

Tutkielman tapaustutkimusosuudessa esiteltävän pilottiprojektin mitattaviksi ominaisuuksiksi valittiin luotettavuus (reliability), käytettävyys (usability), tehokkuus (efficiency) sekä ylläpidettävyys (maintainability). Niitä pidettiin tuotteen tärkeimpinä ominaisuuksina pilottiprojektin mittauksen kannalta. Tuotteen toiminnallisuus pyrittiin jo tutkimaan järjestelmätestauksen yhteydessä ja siirrettävyyttä ei pidetty testaustyövälineen kannalta tärkeänä ominaisuutena verrattuna muihin valittuihin ominaisuuksiin. Kaikki neljä mitattavaa ominaisuutta löytyvät myös Fewsterin ja Grahamin (1999) mallista, joten myös tämän mallin metriikkatietoutta on tutkimuksessa hyödynnetty. Nämä edellä mainitut neljä ominaisuutta esitellään seuraavaksi tässä kappaleessa.

4.2.1 Luotettavuus

Kypsyys (maturity), toipumiskyky (recoverability) ja vikasietoisuus (fault tolerance) ovat luotettavuuden mitattavia ominaisuuksia. ISO/IEC (2000a, 15) Luotettavuus liittyy Fewsterin ja Grahamin (1999, 222-223) mukaan testauksen automatisoinnin yhteydessä siihen, kuinka usein tulokset ovat epäluotettavia tai tulosten saanti on epäluotettavaa. IEEE 610 (1990) standardin mukaan luotettavuus mielletään toimintavarmuudeksi, jonka yleinen määrittely on

kyky suorittaa vaadittu toiminto määräoloissa vaaditulla ajanhetkellä tai aikavälillä.

Brocklehurstin ja Littlewoodin (1992) mukaan erilaisia malleja ohjelmiston luotettavuuden arvioimiseksi on kehitetty runsaasti, mutta niitä ei silti voida yleisesti suositella, sillä tarkkuus vaihtelee tilanteen mukaan hyvinkin paljon. Luotettavuus on kuitenkin hyvin oleellinen tekijä, kun arvioidaan tuotteen valmiutta siirrettäväksi kehityksestä käyttöön, joten sitä on usein pyrittävä tavalla tai toisella arvioimaan. (Brocklehurst & Littlewood 1992)

Luotettavuutta pidetään yleisesti ottaen hyvin laajana käsitteenä, ja siihen vaikuttaa monet seikat jo hyvin varhaisista prosessin kehitysvaiheista lähtien. Koko ohjelmistoprosessi vaikuttaa siinä kehitettävään tuotteeseen, joten luotettavuutta tulisi arvioida koko ohjelmistoprosessin ajan, aina vaatimusten määrittelystä ylläpitoon asti (Everett ym. 1998; Fenton & Neil 2000, 361; Nagappan 2004).

Mikäli virheitä löytyy kovin paljon, ei testausprosessia, jossa tuote toimii osana, voida pitää vielä kypsänä. (ISO/IEC 2000a, 15-27) Myöskään ajon tuloksiin ei voida liiemmin luottaa. Tämän näkemyksen pohjalta halutaan seurata, kuinka hankala järjestelmä on saada toimimaan virheettömästi prosessissa, eli halutaan seurata välineen kypsyyden ohella prosessin kypsyyttä. Halutaan selvittää, onko uusilla käyttäjillä vaikeuksia saada tehokkaasti luotettavia tuloksia testien ajosta. Tässä testissä mitataan siis tavallaan myös tehokkuutta ja käytettävyyttä. Operaatioiden keskimääräinen vikaväli (Mean time between failures, MTBF) ilmaisee testiajossa löytyneiden virheiden määrän suhteessa testiajon suoritusaikaan kaavan 1 osoittamalla tavalla (ISO/IEC 2000a, 20):

$$MTBF = TOPT/NAFI, \quad (1)$$

missä TOPT on ajon suoritusaika (total operation time) ja

NAFI on löytyneiden virheiden määrä (total number of actually detected failures).

Vikasietoisuus on puolestaan yhteydessä ohjelmiston kykyyn ylläpitää suoritustaso siinä tapauksessa, että järjestelmässä esiintyy virheitä. Vikasietoisuutta mitataan projektin aikana ilmenneiden virheiden määrällä ja järjestelmän kyvyllä selvitä niistä kaatumatta. ISO/IEC (2000a, 22) Kaatumisen välttämissuhde (Breakdown avoidance ratio, BAR) mitataan frekvenssiperusteisella esiintymistodennäköisyydellä, joka kertoo virheiden määrän, jotka eivät ole aiheuttaneet kaatumista, suhteessa kaikkien virheiden määrään (kaava 2) (ISO/IEC 2000a, 22).

BAR= 1-(NB/NF), (2)

missä NB on kaatumisten lukumäärä (Number of breakdowns) ja

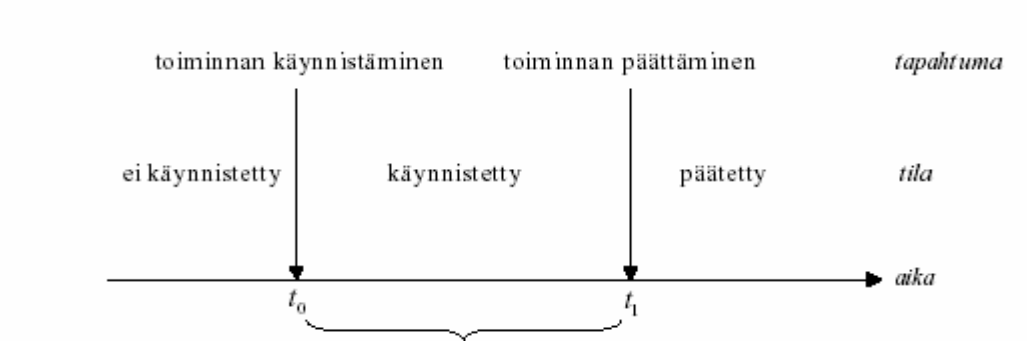
NF on kaikkien virheiden lukumäärä (Number of failures).

4.2.2 Tehokkuus

Tehokkuuden kaksi keskeistä mitattavaa ominaisuutta ovat ISO/IEC:n (2000a, 47) mukaan ajan käytön mittaaminen (time behavior metrics) sekä resurssien hyväksikäytön mittaaminen (resource utilisation metrics). Tehokkuus on suorassa vaikutuksessa rahaan, joka kuluu prosessia suoritettaessa, ja siten se onkin yksi pääsyy sille, miksi testausprosessia halutaan automatisoida (Fewster & Graham 1999).

Mittauksissa arvioidaan testauksenhallinnan työvälineen resurssivaatimuksia sekä ajankäyttöä kuormittamalla järjestelmää eri tavoin. (ISO/IEC 2000a, 47-60) Emuloidaan siis tilanne, missä järjestelmä saavuttaa maksimaalisen kuormituksen tehtävää suoritettaessa ja seurataan tuloksia (ISO/IEC 2000a, 54). Tuloksia arvioidaan suoritusajan (Turnaround time, Tt) perusteella, joka on tehtävän käynnistämisestä sen päättymiseen kulunut aika. (kaava 3)

Suoritusajan laskeminen aloitetaan siitä hetkestä, jolloin syöte on annettu käsiteltäväksi, ja siihen sisältyy järjestelmän reagointiaika. Suoritusajan päättyminen voidaan määritellä kahdella tavalla: se on joko hetki, jolloin järjestelmä aloittaa vastauksen palauttamisen, tai hetki, jolloin vastauksen palauttaminen on päättynyt (ISO/IEC 2000a, 53). Tapaustutkimuksessa suoritusaika mitataan vastauksen palauttamisen päättymiseen eli aikaan, jolloin kaikki testisetit on suoritettu. Testisetillä tarkoitetaan usean, jollain tavalla toisiinsa liittyvän testitapauksen muodostama kokonaisuutta, joka voidaan suorittaa samassa testiajossa, muuttamatta testiympäristöä tai testidataa. Yhdessä testisetissä olevien testitapausten on siis oltava samantyyppisiä, jotta ne muodostavat automaattisesti ajettavan testauskokonaisuuden.



$$Tt = t1 - t0 \qquad (3)$$

ISO/IEC:n (2000a, 56) mukaan siis selvitetään, paljonko kuluu aikaa suoritettaessa järjestelmälle oleellista toimenpidettä, joka tässä tapauksessa on testien suorittamiseen kuluva aika. Suoritusajan perusteella järjestelmää kuormittamalla voidaan arvioida järjestelmän kapasiteettia. Tutkielmassa kapasiteetti halutaan määritellä kuorman avulla ja tässä yhteydessä kuorman yksikkö on yhtäaikaisten testisettien määrä. Järjestelmän suoritustehon ja suoritusajan käyttäytyminen kuorman kasvaessa määrää kapasiteetin (Ylén 2004). Kuorman kasvaessa saavutetaan yleensä piste, jossa suoritusteho ei enää kasva ja järjestelmän muisti kuormittuu ja antaa käyttäjälle virheilmoituksia. Tämä piste määrää käytännössä saavutettavan kapasiteetin. Taitepiste voi

määräytyä myös suoritusajan mukaan. Vaikka suoritusteho vielä kasvaisikin kuorman kasvaessa, voi suoritusaika venyä tasolle, joka ei ole enää hyväksyttävissä.

Lisäksi Fewsterin ja Grahamin (1999, 221) mukaan jotkin testauksen aktiviteetit muuttuvat, kun testauksen automatisointi otetaan käyttöön. Manuaalisessa testauksessa testauksen suorittaminen vie suurimman osan ajasta ja resursseista. Jos nämä testit automatisoidaan, niiden suorittamiseen kuluneen ajan pitäisi selkeästi laskea, mutta vastaavasti saattaa käydä niin, että automatisoinnin suunnitteleminen ja ylläpitäminen vie paljon enemmän aikaa. ISO/IEC (2000b) mukaan aktiviteetteihin kulunut aika on yksi tehokkuuden mittareista. Pilottiprojektissa päätettiin arvioida testauksen tehokkuutta vertaamalla testausaktiviteetteja manuaalisen ja automatisoidun testauksen välillä. Aktiviteetteja verrataan toisiinsa ajan käytön suhteen. Resurssien käytön tehokkuutta ja luotettavuutta pyritään lisäksi arvioimaan laskemalla tietyissä tilanteissa esiintyvien virheiden määrää ja arvioimalla, mikä resursseista muuttuu kuormitettaessa epäluotettavaksi. Näin voidaan löytää parhaita käytäntöjä resurssien tehokkaaseen käyttöön luotettavasti.

4.2.3 Käytettävyys

ISO/IEC (2000a, 29-45) määrittelee käytettävyyden osaominaisuuksiksi ymmärrettävyyden (understandability), opittavuuden (learnability), houkuttelevuuden (attractiveness) ja toimivuuden (operability). Käytettävyyden tutkimusmenetelmänä voi toimia kyselytutkimus. Kysely perustuu SUS-kyselylomakkeeseen (Liite 1). System Usability Scale (SUS) kehitettiin vuonna 1986. SUS:ia on käytetty laajasti ja sitä pidetään yksinkertaisena ja luotettavana. SUS-kyselylomakkeen voidaan katsoa vastaavan yleistä näkemystä käytettävyydestä. SUS:in kysymykset painottuvat pääasiassa tuotteen ymmärrettävyyteen sekä opittavuuteen. (Brooke 1986)

SUS-lomake täytetään sen jälkeen kun käyttäjä on kokeillut tuotetta, mutta ennen kuin käyttäjä on keskustellut tuotteesta jonkun kanssa. Käyttäjien tulisi vastata kyselyn eri kohtiin nopeasti. Kaikkiin kohtiin tulee vastata. Jos käyttäjästä tuntuu siltä, ettei voi vastata johonkin kohtaan, niin silloin valitaan keskimmäinen numero. (Brooke 1986)

SUS-pisteet lasketaan yksittäisille vastauksille. Pisteytys vaihtelee nollasta neljään. Lomakkeen kohdat 1, 3, 5, 7 ja 9 lasketaan vastaajan antamasta arvosta vähentämällä yksi ja kohdat 2, 4, 6, 8 ja 10 lasketaan luvusta 5 vähentämällä vastaajan antama pistemäärä. Lopulliset SUS-pisteet saadaan laskemalla pisteet yhteen ja kertomalla 2.5:llä. Pisteet voivat vaihdella nollasta sataan. (Brooke 1986)

4.2.4 Ylläpidettävyys

Ylläpidettävyys on Lanningin ja Khoshgoftaarin (1994) sekä Bankerin (1993) mukaan yhteydessä ohjelmiston kompleksisuuteen. Liian monimutkaista ohjelmaa on heidän mukaansa hankala ylläpitää ja se jää todennäköisesti suurien kustannustensa vuoksi pois käytöstä.

ISO/IEC (2000a, 29-45) standardin mitattavia ylläpidettävyyden ominaisuuksia ovat analysoitavuus (analysability), muutettavuus (changeability), vakaus (stability) sekä testattavuus (testability). Muutettavuutta pyritään mittaamaan arvioimalla järjestelmän päivittäjän joutuisuutta hänen yrittäessään toteuttaa järjestelmään muutosta. Ohjelmiston muuttamiseen vaadittava aika (Required effort (in person hours) to change software, T) lasketaan jakamalla muutokseen kulunut aika muutetun koodin määrällä ja suhteuttamalla osamäärä tehtyjen muutosten määrään (kaava 4). (ISO/IEC 2000a, 65)

$$T = \text{Sum}\,(A(i)/B(i))/N, \quad (i=1\ldots N) \qquad (4)$$

missä A(i) on muutokseen (i) kulunut aika (Work time spent to change),

B(i) on muutoksessa (i) muutetun koodin "määrä" (Changed software size) ja

N on muutosten määrä (Number of changes).

Vakautta laskettaessa on tarkoituksena puolestaan seurata järjestelmän odottamatonta käytöstä kuvaavia attribuutteja, kun tuotetta käytetään jonkin päivityksen jälkeen. Löytyneiden virheiden tiheys muutoksen jälkeen ((Frequency of encountering failures after change), F kaava 5) voidaan määritellä muutoksesta aiheutuneiden virheiden määrän suhteena tarkkailu-aikaan (ISO/IEC 2000a, 66)

F= Na / Ta, (5)

missä Na on muutoksesta aiheutuneiden virheiden määrä (Number of turns which user encounters failures during operation after software was changed) ja

Ta on tarkkailuaika muutoksen jälkeen (Operation time during specified observation period after software is changed).

4.3 Yhteenveto

Luvussa käsiteltiin laadun mittaamista erityisesti testauksen hallintavälinettä käyttöönotettaessa pilottiprojekti vaiheessa. Luvussa esiteltiin malli välineen käyttöönottamisesta. Ennen välineen asettamista pilottiprojektiin tulee muodostaa tiimi, joka vie hanketta eteenpäin. Tiimin rooleihin valittavat jäsenet tulee valita harkiten, jotta välineen käyttöönotto onnistuisi mahdollisimman hyvin. Tuotetta on lisäksi valmistauduttava esittelemään erilaisissa tilanteissa erilaisille yleisöille. Näiden toimenpiteiden jälkeen voidaan valita pilottiprojekti ja asettaa sille tavoitteet, joiden täyttymistä voidaan arvioida metriikoiden avulla.

Tämän tutkimuksen puitteissa on tarkoituksena testata välineen laatua projektissa sekä tarkentaa käsityksiä prosessin vahvuuksista ja löytää myös alueita ja syitä esiintyville ongelmille. Pilottiprojektin tavoitteena voidaan pitää edellisten ohella työkalun arvon demonstrointia sidosryhmille, korkean tason johtajille sekä potentiaalisille työkalun käyttäjille. Arvoa demonstroidaan mittareiden avulla esim. arvioimalla säästetty manuaalisen testauksen työmäärä. Luvussa esiteltiin metriikkapatteristo mitattavista ominaisuuksista. Tärkeimmiksi mitattaviksi laatuominaisuuksiksi valittiin prosessin sekä välineen luotettavuus, tehokkuus, käytettävyys ja ylläpidettävyys. Seuraavassa luvussa tarkastellaan aiemmin esitettyjen mittausten suorittamista ja tulosten esittelyä kohdeyrityksessä.

5 LAADUN ARVIOINTI KOHDEYRITYKSESSÄ

Tässä luvussa tarkastellaan kohdeyrityksestä kerättyjä tuloksia, joita saatiin siellä suoritetuissa tutkimuksissa. Pilottiprojekti ja siinä mitatut tiedot valittiin yhteistyössä kohdeorganisaation henkilöstön kanssa niiden ajatusten perusteella, joita syntyi teoreettisen osan valmistelun aikana. Luvussa käsitellään työvälineen käyttöönottoa yrityksessä lähinnä pohtimalla käyttöönottoprosessin onnistumiseen vaikuttavia tekijöitä sekä esittelemällä pilottiprojektista saatuja tuloksia.

5.1 Tapaus Yomi

Yomissa oli tarvetta kehittää ympäristö ja tekniset ratkaisut automaattisen testausprosessin läpiviemiseen. Ratkaisuun päädyttiin esitutkimuksesta saatujen tietojen pohjalta. Markkinoilta ei löytynyt välinettä, jossa olisi mahdollista testata yhtäaikaisesti eri ympäristöissä erilaisia ohjelmistoja kohdeyrityksessä vaadittavalla tasolla. Lisäksi välinettä halutaan itse laajentaa yrityksen vaatimusten mukaisesti erilaisiin tarpeisiin ja tämä onnistuu vain, jos kehitettävä väline on itse toteutettu. Ennen välineen suunnittelun aloittamista laskettiin lisäksi kannattavuuslaskelma, jonka perusteella työvälineen valmistaminen katsottiin kannattavaksi hankkeeksi. Kannattavuuteen vaikuttaa mm. se, että monella markkinoilla olevalla välineellä on todella kallis lisenssihinta.

Kohdeyrityksessä kehitetty testauksen hallintaväline Tento toimii itsenäisenä sovelluksena. Välineellä keskitytään automatisoimaan pääasiassa alemman tason testausta eli luokka- ja yksikkötestausta. Tentoon halutaan tulevaisuudessa rakentaa liittymät eri testauksen työkaluihin, jolloin voidaan hyödyntää myös kaupallisten tuotteiden ominaisuuksia. Alkuvaiheessa Tentoon toteutettiin erityyppisten ja usealla eri alustalla ajettavien testitapausten automaattinen ajaminen sekä ajoihin liittyvien tulosten tallentaminen ja

raportointi. Kaupallisten tuotteiden ominaisuuksia voidaan hyödyntää esimerkiksi testauksen suunnittelussa ja testauksessa löydettyjen virheiden hallinnassa myöhemmässä vaiheessa, kun välinettä kehitetään edelleen.

Tapaustutkimuksessa kohdeyrityksessä suoritetaan ensimmäistä pilottiprojektia Tentolle, missä arvioidaan sen hyödyllisyyttä ja toimivuutta sekä mahdollisia ongelmatilanteiden aiheuttajia yleisellä tasolla tähän mennessä kehitettyjen ominaisuuksien osalta. Apuvälineen käyttöönotto-prosessin yhteydessä muodostettiin Yomilla tiimi, joka koostuu useista eri roolin omaavista jäsenistä. Välineen puolestapuhujan roolissa toimii välineen tekninen hoitaja. Hänen tehtäviinsä kuuluu lisäksi välineen eteenpäin kehittäminen, joten luontevana osana hänen työtään on puhua välineen puolesta ja selvittää käyttäjien näkemyksiä tuotteesta ja korjata tuotetta haluttuun suuntaan. Johdon takuumiehenä toimii resursoinneista vastaava johtaja ja muutosagenttina puolestaan toimii testausyksikön vetäjä.

Kohdeyrityksessä järjestettiin kolme demonstraatiotilaisuutta tuleville käyttäjille välineen kehityksen aikana. Julkisuuden hakeminen työvälineelle päätettiin aloittaa hyvissä ajoin ennen työvälineen käyttöönottoa, jotta ihmiset tiedostavat, että jotain on tapahtumassa ja kiinnostuvat aiheesta. Välineestä haluttiin kuulla kunkin yksikön kehitysideoita sekä esitellä sitä tuleville käyttäjille. Esittelijät pyrkivät vastaamaan kysymyksiin ja samalla vakuuttamaan tulevat käyttäjät välineen hyödyllisyydestä.

Sopivien henkilöiden valinnan ohella valittiin yrityksestä pilottiprojekti. Valittu pilottiprojekti täyttää yleiset vaatimukset samankaltaisuudesta ja onnistumisen todennäköisyyttä voidaan pitää myös hyvänä, mutta kiireetöntä projektia ei löytynyt. Yrityksestä valittu pilottiprojekti kestää useita kuukausia ja tässä tutkimuksessa tutkitaan automatisointia projektissa yhden inkrementin aikana. Ensimmäiseksi asetettiin pilottiprojektille ensisijainen tavoite. Tämän tutkimuksen puitteissa on tarkoituksena testata välineen toimivuutta

projektissa sekä tarkentaa käsityksiä prosessin vahvuuksista ja löytää myös alueita ja syitä esiintyville ongelmille. Pilottiprojektin tavoitteena voidaan pitää edellisten ohella työkalun arvon demonstrointia sidosryhmille, korkean tason johtajille sekä potentiaalisille työkalun käyttäjille. Arvoa voidaan demonstroida mittareiden avulla esim. arvioimalla säästetty manuaalisen testauksen työmäärä.

Tässä tutkimuksessa ei oteta kantaa muutoksiin, joita testausprosessi vaatii. Niitä seurataan kyllä projektissa, mutta tämän tutkimuksen näkökulmasta ne jäävät vähälle huomiolle. Tutkimuksessa on painotuttu mittaamaan tuotteen ulkoista laatua, koska tuote on otettu käyttöön pilottiprojektissa. Sisäistä laatua on mitattu tuotteen kehittämisen aikana. Aina ei ole selvästi kuitenkaan määriteltävissä selvää rajaa tuotteen ja sen ympärillä toimivan prosessin välille. Tosin tällaista rajaa ei ole edes tarpeellista tiukasti vetää, sillä tuotteen on tarkoitus toimia oleellisena osana prosessin suorituksessa. Tutkimuksessa on tarkoitus arvioida prosessin sekä välineen kehittymistä prosessista mitattavien ominaisuuksien avulla.

Tento- testauksen hallinnantyövälineen tärkeimmäksi mitattavaksi ominaisuudeksi valittiin luotettavuus. Välineellä on toki lähdetty hakemaan tehokkuutta, mutta tehokkuudesta ei ole mainittavaa hyötyä, mikäli tuotteeseen ei voida luottaa. Lisäksi oleellisia ominaisuuksia tuotteen käyttöönoton kannalta ovat sen käytettävyys sekä ylläpidettävyys. Käytettävyydellä halutaan mitata käyttäjien mielipidettä tuotteen helppokäyttöisyydestä ja ylläpidettävyydellä taas tuotteen ylläpitäjälleen aiheuttamaa kuormaa pilottiprojektin aikana. Metriikoiden avulla tulee voida laskea attribuuttien arvoja systeemin käyttäytymiselle, jossa työväline toimii osana. Prosessi ja työväline eivät ole erotettavissa toisistaan suurimmassa osassa mittaustapauksia.

5.2 Tutkimuksen taustatekijät

Pilottiprojektien mittausten onnistumiseen ja tulosten analysointiin vaikuttavat monet tekijät. ISO/IEC (2002, 42-44) määrittelee tekijöitä, joita tulee huomioida tulosten analysoinnin yhteydessä. Nämä tekijät liittyvät lähinnä välineellä työskentelevien henkilöiden ominaisuuksiin sekä välineen omiin ominaisuuksiin. Lisäksi tässä tutkimuksessa pyritään esittelemään muutosprosessin inhimillinen puoli, kuten työntekijöiden mahdollinen muutosvastaisuus. Muutosprosessiin liittyy useita rooleja, joissa toimivien henkilöiden valinta voi muodostua muutoksen onnistumisen kannalta olennaiseksi. Oikeat henkilöt oikeissa rooleissa pitävät pitkänkin kehityshankkeen käynnissä ja motivoivat muita kehityksen osapuolia tiedottamalla ja neuvottelemalla erilaisten kanavien välityksellä.

5.2.1 Tutkimukseen vaikuttavat taustamuuttujat

Tutkimuksen näkökulmasta pilottiprojektin muuttujat voidaan jaotella mm. taustamuuttujiin ja varsinaisiin tutkimusmuuttujiin. Tutkimusmuuttujat liittyvät välittömästi tutkittavaan ilmiöön; sen sijaan taustamuuttujat antavat yleisempää tietoa tilastoyksiköstä. Käytännössä jako ei ole välttämättä täysin yksiselitteinen. (Paaso ym. 2004) Ennen pilottiprojektin aloittamista kartoitettiin projektin muuttujia, jotka vaikuttavat pilottiprojektin onnistumiseen.

Mitattaessa tuotteen laatuominaisuuksia tulee määritellä tekijät, jotka vaikuttavat mittaustuloksiin: käyttäjät, heidän tavoitteensa ja käytön tarkoitus. Yleensä ei ole mahdollista kartoittaa tarkasti näitä kaikkia tekijöitä, joten on tarpeellista valita tärkeimmät ja edustavimmat tekijät, joiden vaikutusta tuloksiin pyritään arvioimaan.

Käyttäjien ominaisuudet, jotka vaikuttavat heidän suoriutumiseensa tehtävistä pilottiprojektissa tulee määritellä. Tällaisia ominaisuuksia ovat tietämys, taidot,

kokemus, koulutus ja harjoittelu. Lisäksi fyysiset ominaisuudet kuten motoriset kyvyt vaikuttavat tuloksiin. Tässä tutkimuksessa ei kuitenkaan arvioida osallistujien fyysisiä ominaisuuksia. Taulukossa 1 tuodaan esille ominaisuudet, joiden vaikutusta arvioidaan tutkimuksen tuloksia analysoitaessa. (ISO/IEC 2002, 42-44) Kukin arvioitava ominaisuus on selitetty tarkemmin liitteessä 2.

TAULUKKO 1. Testaajien ominaisuudet

Henkilöt	Henkilö 1	Henkilö 2	Henkilö 3
Ikä	29	31	31
Koulutus	FM kesken	FK	Diplomi-insinööri
Rooli projektissa	Testaus (autom.)	Testaus (autom.)	Testaus (manuaal.)
Ammatillinen kokemus	5 vuotta hyvin monipuolista kokemusta alalta	8 vuotta hyvin monipuolista kokemusta alalta	7 vuotta hyvin monipuolista kokemusta alalta
Testaustuotteiden tuntemus	Tuntee joitakin tuotteita ja on myös käyttökokemusta	Tuntee joitakin tuotteita ja on myös käyttökokemusta	Tuntee joitakin tuotteita, ei käyttökokemusta

Pilottiprojektin testaajat olivat iältään 29 – 31 vuotta. Heillä kaikilla on hyvin vankka työkokemus alalta. He ovat tehneet teknisesti vaativia tehtäviä yrityksessä ja heillä on monipuolista osaamista erilaisista ympäristöistä. Lisäksi testauksen automatisoijilla oli jo aikaisempaa kokemusta testauksen hallintavälineistä. Testaajien osalta voisi siis arvioida, ettei pilottiprojekti ainakaan heidän osaamisensa vuoksi epäonnistu.

ISO/IEC (2002, 42-44) standardin mukaan myös laitteisto- ja ohjelmisto-ympäristö tulee määritellä, sillä ne yhdessä vaikuttavat testattavan tuotteen käyttäytymiseen. Lisäksi voidaan määritellä ympäristön mukavuuteen ja kulttuuriin liittyvät seikat, mikäli ne nähdään tutkimuksen kannalta oleellisina. Tässä tutkimuksessa ei arvioida mukavuustekijöiden kuten esim. valaistuksen

vaikutusta eikä kulttuuriin liittyviä tekijöitä. Pilottiprojektin testaukseen tarvittava ympäristö on melko vaatimaton. Ympäristön muodostamiseen tarvitaan ainoastaan kaksi PC-konetta. Toinen koneista toimii palvelinkoneena, johon on asennettu Linux-käyttöjärjestelmä ja tietokanta sekä tarvittavat palvelut. Lisäksi tarvitaan toinen kone, joka toimii asiakaskoneena.

5.2.2 Tutkimuksen kriittiset tekijät

Muutosprosessin onnistumiseen kohdeyrityksessä liittyy useita kriittisiä tekijöitä. Suurin osa näistä tekijöistä on varmasti esillä kaikissa yrityksissä muutosprosessien yhteydessä, mutta tässä luvussa korostetaan erityisesti kohdeyrityksen näkökulmasta ongelmallisia seikkoja. Ehkä tärkein kriittinen tekijä onnistumisen kannalta on johdon tuki. Johdon tuki pitäisi hankkia kaikilla johdon tasoilla, mikä on muutoksen puolestapuhujan tehtävä. Johdon eri tasoille muutosprosessi on perusteltava eri tavoin ja vaikeuksia voi aiheuttaa mm. se, miten keskijohtoa (esim. projektipäälliköt) voidaan motivoida järjestelmän käyttöön. (Forsell 2001) Johdon tehtäväksi jää myös asioiden tiedottaminen ja koulutuksesta vastaaminen työntekijöille.

Johdon tuen lisäksi eräs kriittinen tekijä on oikeiden henkilöiden valitseminen muutosprosessin rooleihin. Erityistä huomiota on kiinnitettävä muutoksen sponsoreiden ja muutoksen johtajan valintaan. Ilman muutoksesta vastuussa olevaa johtajaa muutos jää aina vain suunnitteluasteelle. Useassa kehittämis-hankkeessa on havaittu, että toimiva ja tehokas kommunikaatio on erittäin tärkeätä ja se usein riippuu avainhenkilöistä. (Forsell 2001) Aktiivinen ja asialleen omistautunut muutoksen johtaja pitää muutosprosessin liikkeellä myös vastoinkäymisten aikana ja hän käyttää olemassa olevia välineitä apunaan.

Työntekijöillä ilmenee lisäksi suurella todennäköisyydellä luonnollista muutosvastaisuutta, jota muutoksen johtamisella tulee vähentää. Muutos-

vastaisuutta aiheuttavat mm. pelko ja epävarmuus. Ihmiset eivät halua muuttaa vuosien kuluessa opittuja, hyväksi havaittuja toimintatapoja. Keinoja muutosvastaisuuden vähentämiseen ovat tiedotus ja koulutus, sekä muutoksen kohteille suotava mahdollisuus ilmaista mielipiteensä heitä koskevissa päätöksissä. Loppujen lopuksi muutosprosessi on yhteistyötä ja sen onnistuminen riippuu kaikista osallistujista. Jos työyhteisö nähdään oppimisympäristönä, ja yksilöt ovat sitoutuneita oppimaan uutta, voidaan muutosprosessin olettaa onnistuvan organisaatiossa hyvin. (Forsell 2001)

5.3 Tutkimuksen suorittaminen

Tässä luvussa käsitellään tiedon hankintaa, tutkimusmenetelmää sekä mittausten toteuttamista. Lopuksi tarkastellaan metriikoiden avulla saatuja mittaustuloksia.

5.3.1 Tiedon hankinta ja tutkimusmenetelmä

Tämän tutkimuksen tiedonhankintatavaksi on valittu määrällinen tapaus-tutkimus. Kvantitatiivinen tutkimusaineisto muunnetaan muuttujiksi ja edelleen numeeriseksi havaintoaineistoksi (Hynynen 2000, 49). Mitattavat kohdat pyritään valitsemaan siten, että niiden perusteella saadaan selville jonkinlaisia trendejä siitä, kuinka kyseinen ominaisuus kehittyy pilotin aikana. Tyypillisessä trendi-asetelmassa yhdestä havaintoyksiköstä on useita mittaustuloksia eri aikapisteissä. Hynysen (2000, 49) mukaan kvantitatiivinen tutkimus pyrkii yleistettävyyteen, ennustettavuuteen ja kausaaliselityksiin, joten se sopii tämän tutkimuksen tiedon keräystavaksi. Lisäksi kvantitatiivista aineiston keräämistä puoltaa helpompi vertailun tekeminen myöhemmässä vaiheessa, kun välinettä kehitetään ja tutkitaan sen toimivuutta projekteissa. Tulokset ovat paremmin verrattavissa keskenään, kun tässä tutkimuksessa saatuja tuloksia verrataan aikaisempiin tuloksiin. Pilottiprojektiin tehtävien

mittausten perusteella pyritään suuntaa-antavaan analyysiin. Tuloksilla ei ole hyvää yleistettävyyttä muihin vastaaviin projekteihin.

Tapaustutkimus voidaan määritellä empiiriseksi tutkimukseksi, jossa tutkitaan nykyajassa tapahtuvaa ilmiötä todellisessa elämäntilanteessa, sen omassa ympäristössä (Hynynen 2000, 49; Järvinen & Järvinen 2000). Tässä tutkimuksessa tapaustutkimuksen kohteeksi valittiin pilottiprojekti, joka on mahdollisimman tyypillinen ja edustava projekti kohdeyrityksessä. Tutkimus perustuu pilottiprojektin aikana kerättyyn mittaustietoon. Mittaustiedot kerätään virhekannasta, järjestelmän lokitiedoista, testiraporteista, testaajien tuntikirjauksista sekä käytettävyystestin osalta kyselylomakkeella. Lähtökohtana oli, että mitattavat tiedot on voitava kerätä, niin ettei pilottiprojektin työntekijöiden työ häiriinny.

Tutkimuksessa tutkijan tarkoituksena on johtaa tarkasteltavista teorioista ennusteita ja valita sitten yhdessä kohdeyrityksen henkilöstön kanssa kiinnostavimmat mittarit ja arvioida lopuksi pilottiprojektissa teorioiden toteutumista mitattujen tulosten valossa. Tapaustutkimus perustuu ISO/IEC 9126:n (ISO/IEC 2000a) laatuominaisuuksista muodostettuun metriikka-patteristoon. Tapaustutkimuksen avulla pyritään selvittämään Tenton avulla saavutettavia etuja ja hahmottamaan erilaisia ongelmatilanteita. Näiden tietojen pohjalta pyritään arvioimaan, onko välineen kehittämisessä onnistuttu ja voidaanko sitä pitää hyvänä investointina. Lisäksi tapaustutkimuksen avulla halutaan kerätä tietoa ajoissa tuotteessa olevista heikoista ominaisuuksista, jotta parannuksia voidaan tehdä, ennen kuin väline tulee laajaan käyttöön organisaatiossa. Pilottiprojektista kerättyjen tietojen avulla verrataan testauksen automatisoinnista saatavia etuja ja aiheutuvia ongelmia saavutettuihin etuihin ja ongelmiin. Tarkoituksena on kartoittaa edellä mainittujen seikkojen syitä ja seurauksia.

Mittarin määrittäminen lähtee siitä, että ensin määritellään asia tai ilmiö, jota halutaan mitata. Tämä edellyttää ilmiön täsmällistä käsitteellistämistä. Sitten on kyettävä määrittämään konkreettinen mittari. (Paaso, Mattila & Sivonen 2004) Mittareita ei tässä tutkimuksessa ole kehitetty itse vaan sen sijaan on käytetty valmiita mittareita. Mittarin täytyy lisäksi olla kohteeseensa sopiva ja sen on mitattava sitä asiaa, mitä sillä halutaan mitata. Mittarin validiteetilla tarkoitetaan sen pätevyyttä eli sen hyvyyttä mitata juuri sitä, mitä sen on tarkoitus mitata - tarpeeksi kattavasti ja tehokkaasti. (Paaso ym. 2004) Tämä asia on pyritty varmistamaan valitsemalla standardoitu metriikkapatteristo.

Mittarin on oltava myös luotettava mittauksia toistettaessa eli sillä on oltava pysyvyyttä. Mittarin eduksi luetaan se, että se mittaa kokonaisuudessaan samaa asiaa ts. mittarin konsistenssi on hyvä. Nämä ominaisuudet liittyvät mittarin reliabiliteettiin. Reliabiliteetti-sana voidaan suomentaa sanoilla 'luotettavuus', 'käyttövarmuus' ja 'toimintavarmuus'. Kvantitatiivisen tutkimuksen kielessä sillä tarkoitetaan mittarin johdonmukaisuutta; sitä, että se mittaa aina, kokonaisuudessaan samaa asiaa. (Paaso ym. 2004)

Reliabiliteetissa erotetaan kaksi osatekijää: stabiliteetti ja konsistenssi. Stabiliteetissa on kysymys mittarin pysyvyydestä ajassa. Epästabiilissa mittarissa näkyvät olosuhteiden ja vastaajan mielialan ynnä muiden satunnaisvirheiden vaikutukset helposti. Mittarin pysyvyyttä voidaan tarkastella vertaamalla useampia ajallisesti peräkkäisiä mittauksia. Tällöin aikavälin pituus tulisi osata optimoida: Sen pitää olla tarpeeksi pitkä, jotta vastaaja ei muista vastauksiaan, mutta toisaalta niin lyhyt, ettei todellisia muutoksia asioissa ole ehtinyt tapahtua. Monissa tapauksissa tämä reliabiliteetin mittaustapa ei ole toteuttamiskelpoinen, sillä huono reliabiliteettikerroin voidaan usein helpommin selittää ajassa tapahtuneilla todellisilla muutoksilla kuin epästabiililla mittarilla. (Paaso ym. 2004)

Mittarin konsistenssilla eli yhtenäisyydellä tarkoitetaan puolestaan sitä, että kun useista väittämistä koostuva mittari jaetaan kahteen joukkoon väittämiä, kumpikin väittämäjoukko mittaa samaa asiaa. Tällöin molempien väittämäjoukkojen kokonaispistemäärien välinen korrelaatiokerroin saa suuren arvon. On kuitenkin todettava, että on mahdollista luoda väittämäpatteristo, joka sisältää täysin eri asioita mittaavia, mutta keskenään voimakkaasti korreloivia muuttujia. Toisaalta saman ilmiön osa-alueita mittaavat muuttujat eivät aina välttämättä korreloi keskenään ja kuitenkin niitä on tarpeen tarkastella yhdessä. (Paaso ym. 2004)

Tutkimuksen mittareita ei voida väittää kovin stabiileiksi, sillä tuloksiin pääsevät mittausten aikana vaikuttamaan monet tekijät. Näiden tekijöiden vaikutusta tosin pyritään arvioimaan tulosten analysoinnin yhteydessä. Mittaustulos tutkimuksessa on ikään kuin poikkileikkaus välineen ominaisuuksista. Se kertoo jollakin tavalla rajatut piirteet todellisesta välineestä prosessikäytössä. Tavoitteena on mahdollisimman realistinen kuva.

5.3.2 Mittausten toteuttaminen

Ennen empiirisen tutkimuksen toteuttamista pohdimme yhdessä käyttöönottoprosessia varten perustetun tiimin jäsenten kesken tutkielman teoreettista taustaa vasten, mitkä ovat ominaisuuksia, joita haluamme pilotin toteuttamisen aikana mitata. Näitä ajatuksia listasimme kirjallisuuden ja omien kokemustemme pohjalta listaksi, jota meidän oli tarkoitus hyödyntää myöhemmin mittausten pohjana. Listan avulla muodostimme mittareita, joita voisimme hyödyntää, mutta tarkemmin asiaa pohdittuamme havaitsimme, etteivät mittarit mitanneet aina juuri sitä asiaa, mistä halusimme tietoa, joten päätimme ottaa käyttöön standardoidun metriikkapatteriston.

Ensimmäisen varsinaisen mittauksemme pääsimme aloittamaan vasta syyskuun alussa 2004, jolloin pitkällisen etsimisen jälkeen löysimme sopivan

pilottiprojektin. Mittausdataa kerättiin yhden inkrementin aikana pilotti-projektissa. Inkrementti oli kahden kuukauden mittainen jakso. Projekti vaatii onnistuakseen koko projektiryhmän sitoutumisen, ja parhaat tulokset saadaan aikaan jakamalla selkeät vastuualueet ryhmän jäsenille. Projektissa toimi kolme testaajaa, joille oli pyritty jakamaan saman verran testattavaa. Kaksi testaajaa testasi ohjelmistoa testauksen hallintavälinettä käyttäen ja yksi manuaalisesti. Lisäksi testauksen hallintavälineellä oli ylläpitäjä pilottiprojektin ajan. Projekti aloitettiin pienellä noin 15 minuutin opastuksella, jonka testauksen hallintavälineen ylläpitäjä piti testaajille. Opastuksen jälkeen testaajat muuttivat testitapaukset toimiviksi ja muutenkin päivittivät tarvittavat tiedot ja aloittivat testaamisen.

Tutkimuksessa kerättiin pilottiprojektin toteutuksen aikana edellä mainittujen ominaisuuksien osalta tietoa koko projektin ajan. Ainoastaan käytettävyydestä kerättiin tietoa vaiheittain pilotin aikana. Käytettävyys mitattiin SUS-kyselylomakkeen avulla kahdessa vaiheessa. Ensimmäisessä vaiheessa pilottiprojektin testaajat täyttivät lomakkeet kokeiltuaan ensimmäisen kerran välinettä. Pyrkimyksenä oli kartoittaa heidän näkemyksiään tuotteen käytettävyydestä vaiheessa, jolloin väline on vielä uusi heille.

Toisessa vaiheessa sama kysely tehtiin pilottiprojektin päättyessä uudelleen. Tarkoituksena oli arvioida, ovatko käyttäjien mielipiteet tuotteesta muuttuneet käytön aikana. Tulosta saattaa vääristää se, että käyttäjät muistavat, mitä vastasivat ensimmäisellä kerralla kysymyksiin, ja vastaavat samoin toisella kerralla. Tehokkuutta haluttiin lisäksi arvioida kuormituksen osalta erillisessä testissä, sillä pilottiprojekti ei yksin testeillään pystynyt kuormittamaan järjestelmää tarvittavan paljon.

5.3.3 Tulosten analysointi

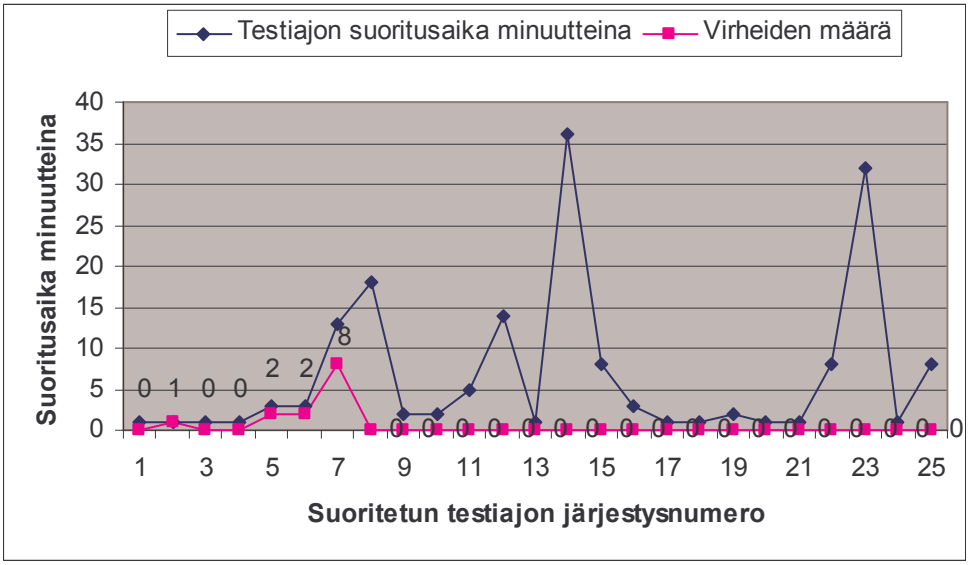
Luotettavuus

Mitattavan ohjelmiston laatu heijastaa myös tuotteen kehityksen laatua. Jos kehitysprosessi oli aikanaan huono, niin todennäköisesti myös tuote on osaltaan epäluotettava ja toisinpäin. (Everett ym. 1998; Fenton & Neil 2000; Nagappan 2004 361)

Luotettavuutta arvioitiin pilottiprojektissa aluksi prosessin ja välineen kypsyyden näkökulmasta. Operaatioiden keskimääräinen vikaväli (MTBF) saadaan sijoittamalla kaavaan 1 (s. 51) kuviossa 8 esitetyt arvot testien suorittamisen aikana löytyneiden virheiden määrästä ja testien ajoon kuluneesta suoritusajasta. Tuloksena saatava keskimääräinen vikaantumisväli ei kuitenkaan paljasta sitä oleellista tietoa, milloin nämä viat ovat pilottiprojektin aikana esiintyneet, joten vikaväli on haluttu esittää ainoastaan graafisessa muodossa. Jossakin toisessa järjestelmässä viat voivat ajoittua tasaisesti koko järjestelmän käytön ajalle, mutta tässä nimenomaisessa tapauksessa näin ei ole käynyt. Keskimääräinen vikaantumisväli antaisi siis harhaanjohtavan tuloksen siitä, että vikoja esiintyy järjestelmässä tietyin ajanjaksoin tasaisesti koko ohjelmiston ajan, vaikka viat ajoittuvat pilottiprojektissa alkujaksoon.

Testiajon suorituksen aikana löytyneiden virheiden määrää kuvataan ensimmäisen kuukauden osalta kuviossa 8, jolloin pilottiprojektissa suoritettiin kaikkiaan 25 testiajoa. Pilottiprojektin toinen kuukausi ei näy kuviossa, koska silloin ei enää ilmaantunut lainkaan virheitä testien suorittamisessa, joten tulosten tarkastelun suhteen projektin ensimmäinen puolisko osoittautui kiinnostavimmaksi. Kuvion 8 perusteella nähdään, että ensimmäisten seitsemän testiajon aikana virheiden määrä korreloi testiajon pituuden kanssa. Virheiden määrä esittävän viivan päällä olevat numerot kertovat virheiden määrän kunkin testiajon kohdalla. Kahdeksannella kerralla testiajon kaikki

testit on osattu päivittää niin, ettei virheitä enää syntynyt. Tästä eteenpäin katsoen havaitaan, ettei virheitä enää testiajoissa ilmaantunut vaikka ajon pituutta lisättiin ja testiajossa saattoi välillä olla kymmeniäkin testitapauksia tarpeen vaatiessa. Järjestelmän vikaantumisväli on siis alussa lyhyt ja projektin loppuvaiheilla vikoja ei taas esiinny enää lainkaan.



KUVIO 8. Testiajojen epäonnistumisen jaksottuminen ajan suhteen.

Tulosten perusteella voidaan päätellä, että testiajureiden ja ympäristön muokkaaminen tuottaa eniten ongelmia alussa. Ajan myötä testien ajaminen sujui pilottiprojektissa suhteellisen helposti ja ongelmitta. Väline ei siis tuottanut testaajille päänvaivaa epäluotettavuudellaan prosessin loppuvaiheessa. Kokonaisuuden kypsyyden voidaan pikemminkin arvioida parantuneen käyttäjien kokemuksen myötä.

Fewsterin ja Grahamin mukaan (1999) testaus kannattaakin aloittaa lyhyillä testeillä ja pienellä testitapausten määrällä. Kun ohjelmisto näyttää toimivan oikein, voidaan testitapausten määrää lisätä tarpeen mukaan. Kypsyysmittauksesta voidaan siis havaita, että jos alussa yritetään ajaa pitkää testiajoa, testitapausten suoritus kestää tarpeettoman pitkään, sillä testien suoritus joudutaan käynnistämään yhä uudelleen löytyneiden virheiden korjauksen jälkeen.

Luotettavuuden toisena ominaisuutena arvioitiin välineen vikasietoisuutta: Kaatumisen välttämissuhteeksi BAR saatiin sijoittamalla kaavaan 2 (s. 52) todetut arvot: NB=1 ja NF=17 kaavan 2 mukaisesti 0,94. Vikasietoisuuden laskeminen perustuu järjestelmän muodostavien laitteiden vikaantumiseen, ympäristön konfiguroinnin vikaisuuteen sekä testitapausten vikaisuuteen. Halutaan selvittää järjestelmän kykyä jatkaa tarkoitettua toimintaa esiintyvistä vioista huolimatta. Välineen vikasietoisuutta voidaan pitää hyvänä. Tosin voidaan todeta, ettei pilottiprojekti rasittanut välinettä kovin paljon, sillä projektin aikana ilmaantui vain kaiken kaikkiaan 17 virhettä koko järjestelmässä. Suuremmalla testimäärällä ja usean projektin yhtäaikaisella testaamisella virheitä löytyisi varmasti enemmän.

BAR = 1-(1/17),

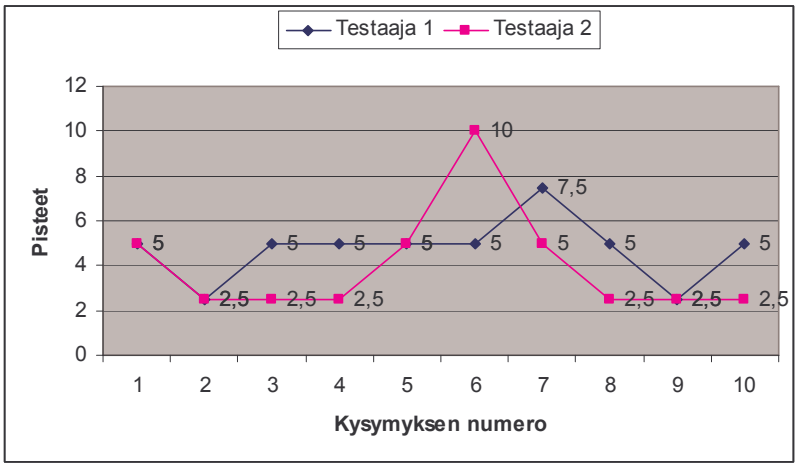
BAR = 0,94.

Käytettävyys

Käytettävyyden osalta SUS-kyselyssä (liite 1) selvisi, että käyttäjät pitivät järjestelmää aluksi melko hankalana ja monimutkaisena, sillä testaajat antoivat vain kerran yli 5 pistettä asteikolla 1 – 10 järjestelmälle ensimmäisten kokeilukertojen jälkeen (Kuvio 9). Testaajat olivat melko samaa mieltä järjestelmän yleisestä käytettävyydestä, mutta kysymyksestä 6 he ovat olleet eri mieltä. Toisen testaajan mielestä järjestelmässä ei ollut liikaa epäjohdonmukaisuuksia, mutta toisen testaajan mielestä väline vaikutti jonkin verran epäjohdonmukaiselta alussa.

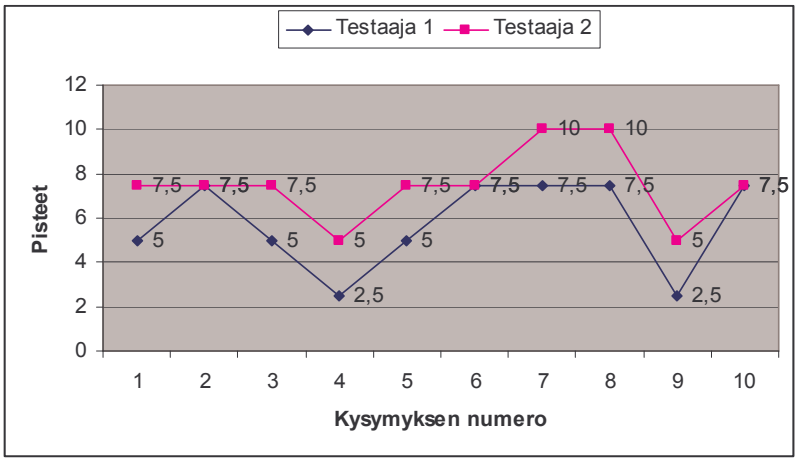
Näkemystä tuotteen hankaluudesta ja monimutkaisuudesta tukee myös edellinen mittaus kypsyydestä. Tulos vastasi pitkälti odotettua tulosta, sillä järjestelmästä on pyritty kehittämään monipuolinen ja sillä pitäisi pystyä testaamaan yrityksessä tehtäviä tuotteita erilaisissa ympäristöissä. Jotta tuotteesta saadaan toimiva kaikissa vaadittavissa tilanteissa, joudutaan

tiedostoja muokkaamaan käyttäjien toimesta kuhunkin tilanteeseen sopivaksi jokaisen projektin alussa. Tämä saattaa tuntua aluksi hankalalta, mutta monipuolisuutensa ansiosta väline taipuu projektin vaatimuksiin tilanteen mukaan.



KUVIO 9. Testaajien alussa antamat pisteet käytettävyydestä

Testaajien lopussa antamat pisteet välineen käytettävyydestä ovat jo paremmat kuin aikaisemmat pisteet (Kuvio 10). Selvimmin on muuttunut testaajan 2 mielipiteet, joka projektin raporttien perusteella on ajanut eniten testejä Tento-työvälineellä. Joka tapauksessa kokemuksen myötä testaajien mielipiteet käytettävyydestä ovat selkeästi samansuuntaisia. Testaajat eivät ole täysin eri mieltä mistään käytettävyyskyselyn asiasta.



KUVIO 10. Testaajien lopussa antamat pisteet käytettävyydestä

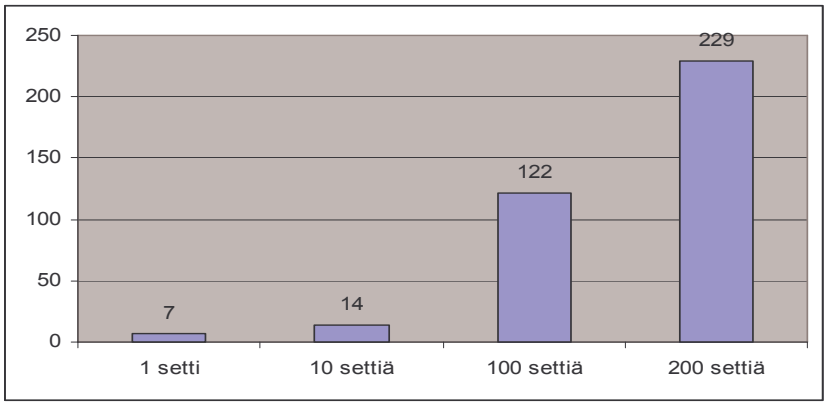
Heikoimmat pisteet on kyselyssä annettu kysymykselle 9: ” Tunsin oloni hyvin varmaksi tuotetta käyttäessäni” ja kysymykselle 4: ”Mielestäni tuotteen käyttö vaatii asiantuntijan opastusta”. Kysymysten 2 ja 8 kohdalla vastaajien mielipiteet olivat muuttuneet eniten kokemuksen myötä. Kysymys 2: ” Tuote on mielestäni tarpeettoman monimutkainen” ja kysymys 8: ” Mielestäni tuotetta on hyvin hankala käyttää” tuntuivat siis kahden kuukauden käytön jälkeen muuttuneen eniten positiivisempaan suuntaan eli käyttäjät kokivat tuotteen yksinkertaisemmaksi ja helpommin käytettäväksi kokemuksen myötä. Vastausten valossa välinettä voidaan pitää varsinkin alussa hieman monimutkaisena, joten käyttäjät tarvitsevat koulutusta käyttöönoton yhteydessä, sillä käyttäjät kokivat koulutuksen tarpeelliseksi vielä silloinkin, kun tuote oli jo heille tutumpi. Koulutuksen myötä testaajat voivat tuntea itsensä varmemmiksi käyttöönoton yhteydessä.

Tehokkuus

Kuormitusmittauksia ei voida tehdä pilottiprojektista, sillä se ei kuormita järjestelmää mittausten vaatimalla tavalla. Tätä mittausta varten järjestetään siis erillinen testi. Tarkoituksena on ensin asteittain kuormittaa yhtä testejä suorittavaa asiakkaana toimivaa PC-konetta järjestelmässä. Kun löydetään maksimiraja, jolla yksi kone vielä pystyy suorittamaan testejä luotettavasti, otetaan käyttöön lisää koneita ja kuormitetaan jokaista tällä löydetyllä maksimimäärällä, niin kauan että järjestelmästä löytyy seuraava heikko kohta. Näin saadaan selville, millaisen määrän järjestelmä pystyy testitapauksia suuressa kuormituksessa ajamaan ja mikä resursseista pettää missäkin tilanteessa. Lisäksi mitataan suoritusaikaa, jonka tietty testitapausten ajaminen vie erilaisissa tilanteissa.

Tehokkuutta mitattaessa selvitettiin aluksi yhden asiakaskoneen suorituskykyä settien ajossa. Testisettiin valittiin 12 testitapausta, jotka koskivat testien tulosten muodostamista. Oikeiden testitulosten muodostaminen suuria testi-määriä

ajettaessa on erittäin oleellinen ominaisuus välineelle. Testien avulla oli tarkoitus tutkia järjestelmän kykyä suoriutua testattavien settien kannasta hakemisessa, järjestelmän kykyä jakaa testejä oikeille asiakaskoneille ympäristövaatimusten mukaisesti, kykyä suoriutua testien suorittamisesta sekä tulosten raportoinnista. Yksi asiakaskone pystyi ajamaan noin 200 settiä (KUVIO 11). Lisättäessä settien määrää asiakas muuttui hyvin epävakaaksi ja saattoi kaatua tai tuli useita virheitä ja varotuksia sen kuormittumisesta. Testin avulla arvioitiin, että asiakaskoneella pystyy ajamaan korkeintaan 200 testisettiä, minkä jälkeen se muuttui epäluotettavaksi, sillä sen muisti kuormittui testeissä liiaksi.



KUVIO 11. Yhden asiakaskoneen suoritusaika Tt kuormitustilanteissa

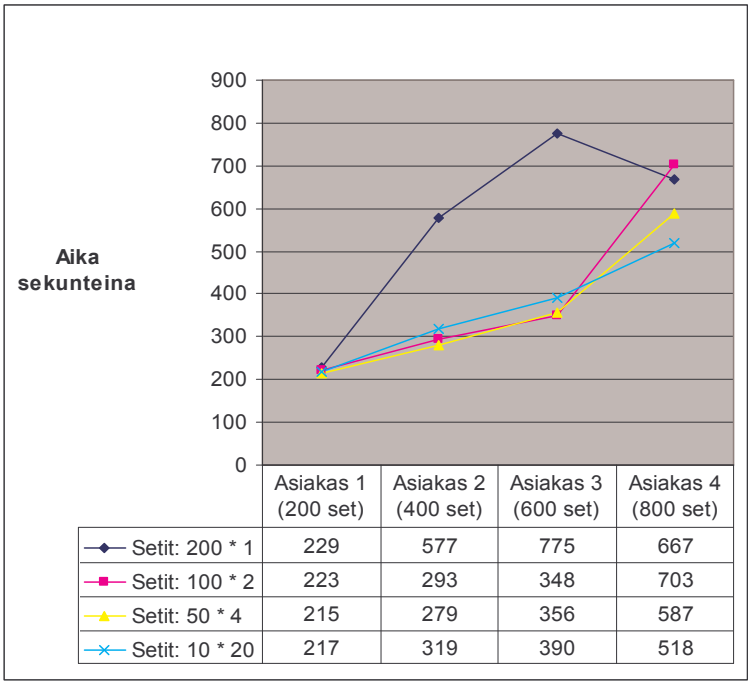
Seuraavaksi selvitettiin, millä tavalla testit kannattaa jakaa testiympäristössä olevien asiakaskoneiden kesken. Aikaisemmassa testissä oli vain yksi asiakaskone ja nyt niiden määrää lisättiin yksi kerrallaan aina neljään koneeseen asti. Tento-järjestelmä antaa kullekin asiakkaalle tietyn määrän testejä ajettavaksi. Jos järjestelmään on kytketty esim. kaksi asiakasta ja tietoihin kirjataan ajettavaksi testit 10 setin erissä, niin järjestelmä antaa ensimmäiselle asiakkaalle kymmenen settiä ajettavaksi ja tämän jälkeen jakaa toiselle asiakkaalle 10 kappaleen erän settejä niin kauan, että kaikki testit saadaan ajetuksi.

Testissä päätettiin käyttää kullakin asiakaskoneella 200 testisettiä, koska aikaisemmassa testissä tällä määrällä asiakaskoneen todettiin vielä toimivan luotettavasti. Testissä haluttiin myös selvittää, että onko sillä merkitystä, millaisissa ryhmissä nämä 200 testisettiä annetaan asiakkaille ajettavaksi. Ensimmäisessä tavassa kaikki 200 settiä annettiin heti asiakaskoneille ajettavaksi. Toisessa tavassa kullekin koneelle annettiin ensin sata settiä ajoon ja seuraavat sata vasta sitten, kun ensimmäiset oli ajettu. Kolmannessa tavassa testit annettiin ajettavaksi neljässä ryhmässä 50 setin erissä ja neljännessä tavassa testisettejä annettiin ajoon 10 setin erissä.

Ensin ajettiin siis 200 settiä vain yhdellä asiakaskoneella. Sitten otettiin yksi asiakaskone lisää testiin ja settien määrää lisättiin 400:an. Näin jatkettiin kunnes lopulta ajossa olivat kaikki 4 asiakaskonetta ja 800 testisettiä. Kunkin kokeen yhteydessä tallennettiin ajoon kulunut aika ja ajon yhteydessä löytyneiden virheiden määrä. Kuviosta 12 voidaan todeta, että jos kaikki 200 testisettiä antaa kerralla ajoon asiakkaalle, niin järjestelmä kuormittuu nopeasti. Neljännen asiakaskoneen käyttöönoton yhteydessä suoritusaika laskee, koska virheiden esiintymisen vuoksi testijärjestelmä vain kuittaa testit nopeasti virheellisiksi, eikä yritä enää suorittaa niitä. Testijärjestelmän virheiden määrän näkee kuviosta 13.

Muut tavat, joissa kullekin asiakkaalle annettiin kerrallaan joko 100, 50 tai 10 settiä ajoon, käyttäytyivät keskenään ajankäytön suhteen melko samalla tavalla. Kuvion 12 perusteella voidaan todeta, että rajoitettaessa asiakaskoneelle kerrallaan ajettavien settien määrää esim. 10 kappaleeseen päästään nopeampiin suoritusaikoihin, koska yksi kone pystyy heti jakamaan tehokkaasti testejä muille koneille, jos sillä on jo testejä ajossa, eivätkä koneet pääse kuormittumaan liikaa. Jos taas testejä on 200 kappaletta yhdellä asiakkaalla kerrallaan ajossa, niin se kuormittuu liikaa, ennen kuin järjestelmä jakaa testejä muille koneille, ja suorituskyky kärsii. Voidaan siis todeta, että rajoitettaessa asiakaskoneelle kerrallaan ajettavien settien määrää esim. 10

kappaleeseen saadaan kapasiteetin käyttöaste järkevälle tasolle. Hyvin suunnitellussa järjestelmässä eri resurssien käyttöasteissa ei ole suuria eroja, eli kuormitus jakautuu tasaisesti eri resursseille — tällöin järjestelmä on tasapainoinen.



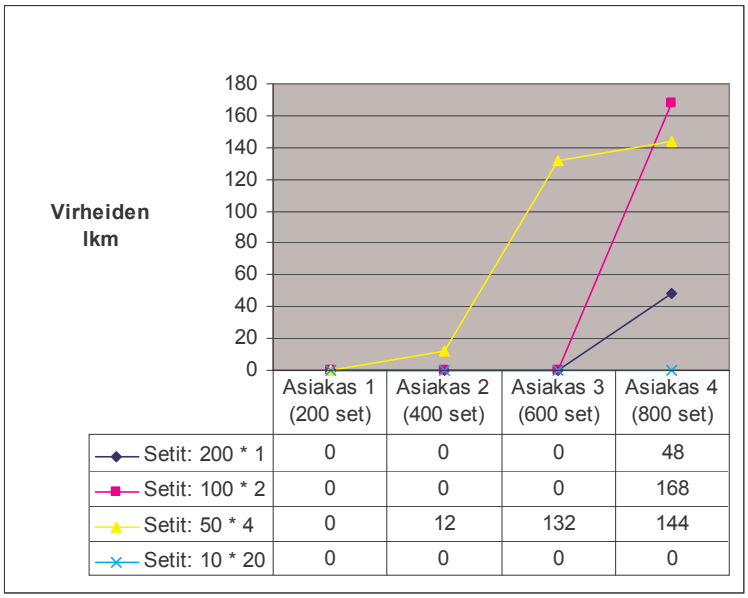
	Asiakas 1 (200 set)	Asiakas 2 (400 set)	Asiakas 3 (600 set)	Asiakas 4 (800 set)
Setit: 200 * 1	229	577	775	667
Setit: 100 * 2	223	293	348	703
Setit: 50 * 4	215	279	356	587
Setit: 10 * 20	217	319	390	518

KUVIO 12. Välineen suoritusaika Tt eri kuormitustilanteissa

Kuviosta 13 voidaan seurata, kuinka paljon virheitä järjestelmään tuli kussakin testissä. Ainoastaan 10 setin yhtäaikainen ajo toimi kaikissa tilanteissa täysin virheettömästi. Sitä voidaan siis pitää tehokkaana ja luotettavana tapana ajaa testejä. Kuviosta nähdään, että mitä enemmän asiakaskoneita otetaan testiin mukaan, sitä enemmän virheitä yleensä esiintyy. Tämä on seurausta siitä, että palvelin kuormittuu usean asiakkaan päivittäessä tietoja yhtä aikaa. Asiakaskoneita lisättäessä tarvitaankin tehokkaampi palvelinkone.

Ensimmäisessä tavassa, jossa kaikki 200 settiä ajetaan yhtä aikaa, syntyy vain yksi säie, joka luo testejä ajettavaksi. Toisessa tavassa syntyy kaksi säiettä, jotka päivittävät kantaa yhtä aikaa. Tämä kuormittaa palvelinta jo melko paljon. Kolmannessa tavassa syntyy neljä yhtäaikaista säiettä, jotka yrittävät päivittää

kantaa ja virheitä syntyy paljon, jo silloin kun otetaan kaksi asiakasta testiin mukaan. Neljännessä tavassa ensimmäisten säikeiden suoritettua jo tehtävänsä vielä uusia säikeitä syntyy ja yksi säie kuormittaa vähemmällä tietomäärällä palvelinta kuin aikaisemmissa tavoissa.



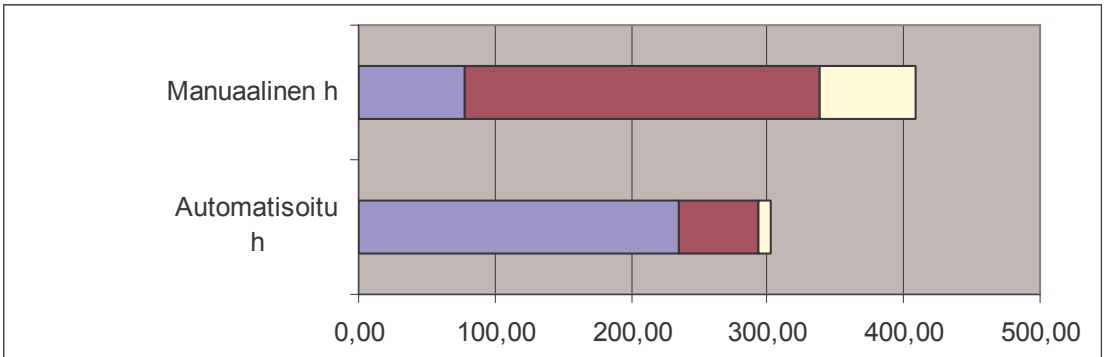
	Asiakas 1 (200 set)	Asiakas 2 (400 set)	Asiakas 3 (600 set)	Asiakas 4 (800 set)
Setit: 200 * 1	0	0	0	48
Setit: 100 * 2	0	0	0	168
Setit: 50 * 4	0	12	132	144
Setit: 10 * 20	0	0	0	0

KUVIO 13. Virheiden lukumäärä kuormitustilanteissa.

Mittaustietoja vertailtaessa tulee huomioida Tento-järjestelmän skaalautuvuus. Skaalautuvuudella tarkoitetaan tässä järjestelmän kykyä sopeutua järjestelmän laajenemiseen. Skaalautuvan Tento-järjestelmän kapasiteettia voidaan lisätä lisäämällä laitteiston määrää tai nopeutta ilman, että ohjelmistoon tarvitsee tehdä muutoksia. Kun kapasiteetti kasvaa suhteessa laitteiston tehoon, voidaan järjestelmän päivitys tehdä helposti kuorman kasvaessa.

Tehokkuutta mitattiin myös pilottiprojektista kerättyjen tuntikirjausten perusteella. Fewsterin ja Grahamin (1999, 221) mukaan jotkin testauksen aktiviteetit muuttuvat, kun testauksen automatisointi otetaan käyttöön. Muutokset ovat nähtävissä myös pilottiprojektin osalta (Kuvio 14). Projektissa testit oli jaettu manuaalisiin ja automaattisiin testeihin. Projektin alkaessa tehtävät pyrittiin jakamaan kolmelle testaajalle, niin että kullakin olisi

testauksen osalta yhtä paljon testattavaa. Testaajat eivät kuitenkaan testanneet samoja testejä, vaan kaikki testisuunnitelman testit oli jaettu testaajien kesken. Yksi testaaja testasi manuaaliset testit ja kaksi testaajaa suoritti automatisoituja testejä.



	Automatisoitu h	Manuaalinen h
Raportointi & Analysointi	9,00	69,00
Suorittaminen	58,25	261,00
Suunnittelu & Ylläpito	235,00	78

KUVIO 14. Testauksen aktiviteetteihin kulunut aika manuaalisessa ja automatisoidussa testauksessa

Testaajien tuntikirjausten perusteella ei voida vetää mitään kovin varmoja johtopäätöksiä aktiviteetteihin yleensä kuluvasta ajasta, mutta tiedot ovat suuntaa antavia ja niistä näkee selkeästi, että automatisoitujen testien suunnittelu ja ylläpito vie enemmän aikaa kuin manuaalisten testien suunnittelu ja ylläpito. Tosin suunnittelussa kulunut aika voidaan voittaa takaisin, mikäli testejä joudutaan toistamaan useasti. Mitä pidempi projekti, ja mitä useammin testejä suoritetaan, sen todennäköisemmin automatisointi kannattaa. Pilottiprojektiksi pyrittiin valitsemaan projekti, jossa testauksen automatisoinnilla on hyvät todennäköisyydet onnistua, ja tässä tutkimuksessa saatujen mittausten perusteella voidaan myös arvioida automatisoinnin kannattaneen hyvin.

Ylläpidettävyys

Tutkimuksen välineessä on paljon ominaisuuksia ja sen on taivuttava monenlaisiin vaatimuksiin, joten sitä voidaan jossain määrin pitää hankalana ja onkin tarpeellista selvittää, osoittautuuko sen ylläpito liian työlääksi. Ylläpidettävyyden osalta oli tarkoituksena tutkia myös muutoksesta aiheutuneiden virheiden määrä välineeseen, mutta pilottiprojektin aikana ei havaittu lainkaan sellaisia uusia virheitä, joita edelliset päivitykset olisivat aiheuttaneet. Lisäksi pilottiprojektissa haluttiin tutkia ylläpidettävyyden osalta tuotteen muutettavuutta ja vakautta. Ohjelmiston muuttamiseen vaadittavaa aikaa T arvioitiin jakamalla muutokseen kulunut aika muutetun koodin määrällä ja suhteuttamalla osamäärä tehtyjen muutosten määrään kaavan 4 mukaisesti (s. 56).

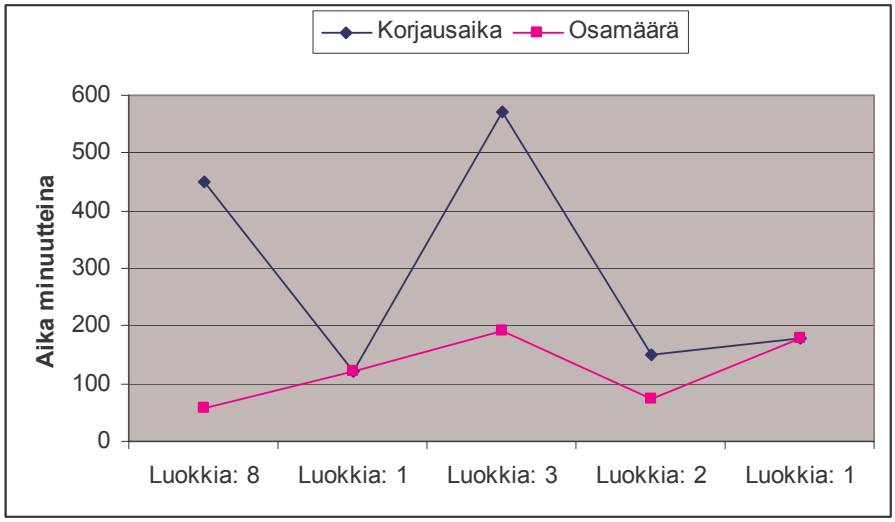
$$T = (450/8 + 120/1 + 570/3 + 150/2 + 180/1)/5$$

$$T = 124 \text{ minuuttia/luokka}$$

Ylläpidettävyyttä arvioitiin pilottiprojektin ylläpitäjän näkökulmasta. Hän kirjasi pilottiprojektin aikana löytyneiden virheiden löytymiseen ja korjaamiseen kuluneet ajat ylös. Näiden aikojen perusteella haluttiin arvioida välineen ylläpidon aiheuttamaa työmäärää. Sijoittamalla ylläpitäjältä saadut tiedot kaavaan 4 saatiin ohjelman muuttamiseen vaadittavaksi ajaksi 124 minuuttia. Eli ylläpitäjältä menee aikaa noin kaksi tuntia virheen löytämiseen ja korjaamiseen, mikäli virhe voidaan korjata muuttamalla yhtä luokkaa. Suurien useampaa luokkaa koskevien virheiden korjaaminen vie ylläpitäjältä enemmän aikaa.

Kuvion 15 perusteella voidaan huomata, että useampaa luokkaa päivitettäessä ylläpitäjältä on kulunut enemmän aikaa virheen korjaamiseen (korjausaikaa kuvaava viiva). Virheiden korjaamiseen ja löytämiseen kuluneet ajat on esitetty kuviossa siinä järjestyksessä, missä ne löytyivät pilottiprojektin aikana. Virheen

korjaamiseen kuluneella ajalla ei ole havaittavissa mitään erityistä trendiä. Kuviosta 15 (osamäärä kuvaava viiva) voidaan lisäksi hahmottaa virheiden laajuutta. Jakamalla muutokseen kulunut aika muutetun koodin määrällä päästään noin 1 – 3 tunnin haarukkaan korjaukseen käytetyn ajan suhteen virhettä kohti.



KUVIO 15. Välineen ylläpitäjän virheiden löytämiseen ja korjaamiseen kulunut aika sekä virheen laajuus korjausta vaativien luokkien osalta.

Vikojen korjaamiseen ei kulunut ylläpitäjän mielestä erityisen kauan aikaa. Ylläpitäjän mukaan löytyneet viat olivat lähinnä käytössä ilmenneitä käytettävyysongelmia tai kosmeettisia vikoja. Tosin hän mainitsee, että korjauksen yhteydessä jokainen muutos on aina testattava ennen kuin korjattu versio voidaan ottaa käyttöön. Joidenkin virheiden korjauksen varmistaminen vie kauan aikaa, kun muutoksen toimivuus pitää varmistaa erilaisissa ympäristöissä. Tästä huolimatta ylläpitäjä ei arvioinut välinettä erityisen hankalasti ylläpidettäväksi.

5.4 Yhteenveto

Luvussa todettiin, että useat tekijät vaikuttavat pilottiprojektista kerättäviin tapaustutkimuksen tuloksiin. Ensimmäisenä seikkana on kiinnitettävä

huomiota valittavaan pilottiprojektiin. Sen tulee täyttää pilotille asetetut vaatimukset mahdollisimman hyvin. Lisäksi mitattaessa tuotteen laatu-ominaisuuksia tulee määritellä muutkin tekijät, jotka vaikuttavat mittaustuloksiin. Yleensä ei ole kuitenkaan mahdollista kartoittaa tarkasti näitä kaikki tekijöitä, joten on tarpeellista valita tärkeimmät ja edustavimmat tekijät.

Projektin kriittiseksi tekijäksi arvioitiin muodostuvan lähinnä oikeiden henkilöiden valitseminen muutosprosessin rooleihin. Erityistä huomiota on kiinnitettävä muutoksen johtajan valintaan, sillä ilman muutoksesta vastuussa olevaa johtajaa muutosprosessi jää aina vain suunnitteluasteelle. Myös johdon tuki on tärkeää. Tulosten tutkimisen jälkeen niitä voidaan analysoida teoreettista taustaa vasten ja pohtia syy-seuraus -suhteita löytyneille ilmiöille.

6 JOHTOPÄÄTÖKSET

Tutkielman kirjallisuusosuudessa tutkittiin testauksen hallintavälineen ongelma-alueita ja saavutettavissa olevia etuja. Tässä luvussa analysoidaan kohdeyrityksessä suoritetun tutkimuksen tuloksia verraten saatuja tuloksia tutkimuksen alun teoriaosuuteen. Analysointi on metriikoiden käyttämisen tärkein vaihe (Fenton & Neil 2000; Kaner 2002). Jopa pienellä määrällä dataa saadaan hyvän analysoinnin ansiosta arvokasta informaatiota kehittämisprosessista. Siksi Fentonin ja Neilin (2000) mukaan onkin tärkeää, että metriikkaprojektit ovat analysointipainotteisia.

Ahosen ja Junttilan (2003, 9-10) tapaan myös kohdeorganisaatiossa nähtiin erityisen haastavana markkinoilla olevien testaustyökalujen epätäydellisyys. Työkalut eivät ole tarpeeksi joustavia erilaisissa ympäristöissä työskentelyyn eikä niillä työskentely ole aivan yksinkertaista (Ahonen & Junttila 2003, 9-10; Chan 2000, 714). Testauksen hallintavälineen käyttöönoton yhteydessä tähdätään kuitenkin siihen, että automatisoitu testaaminen olisi lopulta tehokkaampaa kuin manuaalinen testaaminen. Jotta tähän tavoitteeseen päästäisiin mahdollisimman monessa projektissa, päätettiin kohdeorganisaatiossa kehittää oma testauksen hallintaväline. Oikean kehityssuunnan varmistamiseksi seurattiin pilottiprojektissa mm. tehokkuutta mittaamalla, sillä testauksen automatisointi voi olla hyödyksi vain, jos automatisoinnin avulla säästetty vaivannäkö on selkeästi suurempi kuin automatisoinnin käyttöön kuluva vaivannäkö. (Fewster & Graham 1999, 293-294)

Ohjelmiston mittaaminen tarjoaa kvantitatiivisen näkökulman ohjelmistotuotannon prosesseihin ja yksittäisen ohjelmistotuotteen laadun arviointiin. Erityyppisiä ohjelmistomittoja on kehitelty satoja. Mittatiedon hyväksikäytön ongelmana ei olekaan mittojen puute vaan niiden soveltaminen. Mittaaminen tarjoaa analyyseihin ja prosessien ohjaukseen raakadataa, jota tulisi pystyä soveltamaan myös päätöksenteon tukena. Tutkielman empiirisessä osiossa

kohdeorganisaation testauksen automatisoinnin laatua arvioitiin kansainvälinen standardointiorganisaation (ISO) kehittämän ISO/IEC 9126 standardin (ISO/IEC 2000a) pohjalta. Standardi määrittelee 6 pääpiirrettä ja 27 alapiirrettä, jotka määrittelevät tuotteen laadun. Tämän lisäksi standardi sisältää runsaasti piirteisiin liittyviä metriikoita.

Ohjelmistostandardeihin on yleensä kohdistettu kritiikkiä lähinnä siksi, ettei standardien tueksi löydy tarpeeksi todistusaineistoa ja tutkimustietoa. Pfleegerin ym. (1994) mukaan standardit käyttävät lähestymistapoja, joiden tehokkuutta ei ole täsmällisesti ja tieteellisesti demonstroitu. Chung ym. (2004) pitivät tärkeänä tutkia ISO/IEC 9126 standardin piirteiden kategorisoitumisen oikeellisuutta ja luotettavuutta laadun arvioimisen suhteen. Heidän tutkimuksensa perusteella standardin 6 pääpiirrettä ja 27 alapiirrettä eivät ole selkeitä. Ensinnäkin jotkin alapiirteet eivät ole selkeästi pääpiirteen alle kuuluvia, vaan voisivat kuulua yhdeksi uudeksi pääpiirteeksi. Lisäksi ei ole selvää, miten alapiirteet jakautuvat pääpiirteiden alle. Jokin alapiirre voisi yhtä hyvin kuulua toisenkin pääpiirteen alle, kuin mihin se on standardissa määritelty kuuluvaksi. (Chung ym. 2004)

Edellä mainittu kritiikki ei osoittautunut aivan perättömäksi tutkielman teon aikana. Joidenkin metriikoiden kategorisoituminen tietyn pääpiirteen alle oli kyseenalainen. Esim. jos halutaan selvittää luotettavuutta arvioitaessa, onko uusilla käyttäjillä vaikeuksia saada luotettavia tuloksia testien ajosta, voidaan saman mittaustuloksen avulla arvioida välineen luotettavuutta, mutta yhtä hyvin voidaan analysoida käytettävyyttä tai tehokkuutta saatujen tulosten perusteella. Hieman epäselväksi jää myös se, onko lainkaan järkevää pitää tismalleen samaa laatumallia käytössä laskettaessa sisäistä ja ulkoista laatua. Tai pitäisikö ainakin jossain ohjata käyttäjää, mistä asioista ollaan erityisen kiinnostuneita missäkin tilanteessa. Loppukäyttäjän ja varsinaisen ohjelmiston toteuttajan näkökulmat ovat kuitenkin jossain määrin erilaisia laatua arvioitaessa.

ISO/IEC 9126 standardiin (ISO/IEC 2000a) liittyvä dokumentaatio on lisäksi heikkoa. Dokumentaatio on ylimalkaista ja usein liian suppeata johtopäätösten tekoon. Monet ohjelmistomitoista ovat väljästi määriteltyjä. Aina ei edes mainita, mistä puhutaan kun puhutaan määrästä. Tarkoitetaanko esim. luokkien määrää vai koodirivien määrää. Lisäksi koodirivien määrästä puhuttaessa unohdetaan mainita, onko kommenttirivit ja tyhjät rivit laskettu mukaan vai ei. Mitasta ja mitan laskutavasta riippuen samasta lähdekoodista voidaan saada hyvinkin erityyppisiä tuloksia.

Ideaalinen mitta on objektiivinen, mittaajasta ja mittauksen ympäristöstä riippumaton mitattavan kohteen projektio numeeriseksi arvoksi (Paaso ym. 2004). Erityyppisiä harhoja, jotka vaikuttavat ohjelmistomittojen oikeellisuuteen tässä tutkielmassa, aiheuttavat puutteelliset mittojen määritelmät, resurssien vähäisyys (esim. testaajia oli pilottiprojektissa vain kolme kappaletta) sekä mittojen subjektiivisuus. Harha mittauksessa voi rakentua näiden kaikkien puutteiden yhteisvaikutuksesta. Väljien määrittelyjensä takia mitat voivat sisältää kätkettyä subjektiivisuutta. Käytettyjen mittareiden konsistenssia ei voida pitää kovin hyvänä esim. puutteellisten määritelmien ja epämääräisen kategorisoinnin johdosta. Mittaukset eivät mittaa kokonaisuudessaan aina samaa asiaa. On myös muistettava, että toistettaessa aikaisemmin tehtyjä mittauksia on aina huomioitava ympäristön vaikutukset. Erilaisissa kehitysympäristöissä saatuja tuloksia ei ole mahdollista vertailla keskenään ellei ympäristötekijöiden voimakasta vaikutusta ymmärretä, sillä mittaustulokset ovat harvoin vertailukelpoisia erilaisissa ympäristöissä.

Mittaustiedoissa esiintyvästä harhasta huolimatta pilottiprojektista kerättyjen tulosten perusteella voidaan projektin arvioida onnistuneen hyvin. Onnistumiseen positiivisesti vaikuttavia tekijöitä löytyi useita. Yksi merkittävä tekijä hankkeen onnistumiselle on testaushenkilöiden aikaisempi kokemus IT-alalla käytettävistä välineistä ja projektityöskentelystä. Järjestelmästä saa sitä enemmän irti, mitä tutumpi sen taustalla oleva käyttöjärjestelmä ja laitteet ovat

teknisesti ja toiminnallisesti. Käyttäjät kokivat välineen aluksi hieman monimutkaiseksi, mutta tutkimuksen lopussa heidän mielipiteensä oli jo muuttunut kokemuksen myötä positiivisempaan suuntaan. Tosin pilotissa esiintyi myös onnistumista hankaloittavia tekijöitä, joita tarkastellaan myös tässä luvussa.

Testausta automatisoitaessa törmätään usein monenlaisiin ongelmiin. Ongelmat, jotka tulevat täytenä yllätyksenä järjestelmän kehittäjille ja välineen käyttäjille, ovat useasti hankalimpia käsitellä. Automatisointia ei voida pitää ”hopealuotina”. Hyvin suunniteltuna ja toteutettuna se tehostaa testaamista monella alueella, mutta se ei toimi ratkaisuna kaikkiin testauksen alueella esiintyviin ongelmiin. Epärealistiset odotukset lienevätkin suurin syy testauksen automatisoinnin epäonnistumisiin (Fewster & Graham 1999, 10). Tästä syystä välineen kehitysvaiheessa järjestelmää esiteltiin systemaattisesti tuleville pilottiprojektin käyttäjille. Tiedotuksen systemaattisuus ja suunnitelmallisuus ehkäisevät usein ennakkoluuloja ja pelkoja sekä auttavat hahmottamaan realistisesti etuja, joita välineen avulla voidaan saavuttaa.

Työntekijöillä ilmeni tiedottamisesta huolimatta pilottiprojektin aikana luonnollista muutosvastaisuutta, jota muutoksen johtamisella tulisi vähentää. Muutosvastaisuutta aiheuttavat mm. pelko ja epävarmuus. Ihmiset eivät halua muuttaa vuosien kuluessa opittuja, hyväksi havaittuja toimintatapoja. (Forsell 2001) Muutosvastarinta on mielestäni täysin luonnollista pilottiprojektin yhteydessä testaajilla ja projektipäälliköllä, sillä välineen testaaminen ennen käyttöönottoa saattaa aiheuttaa projektissa lisätyötä ja työntekijät voivat kokea, etteivät he varsinaisesti saa mitään etua siitä, että ottavat uuden välineen koekäyttöön omaan projektiinsa. Kaikki tavallaan toivovat, että joku muu toimisi koekaniinina ja itse saisi vain käyttää valmista ja testattua tuotetta silloin, kun siitä saatavat edut ovat varmasti saatavilla. Mielestäni käyttöön-ottoprosessissa olisi kehitettävää tältä osin. Tuotteita ei voida vain määrätä käytettäväksi jossain projektissa, vaan tämä asia on pystyttävä tavallaan

myymään projektin työntekijöille, jotta he ovat motivoituneet käyttämään uutta välinettä.

Jotta muutosvastaisuutta voitaisiin vähentää, tulee johtajien tiedottaa muutoksen tarpeellisuudesta sekä luoda realistiset ja positiiviset odotukset lopputulokselle. (Forsell 2001, 5) Hankkeen varsinaisella johtajalla onkin prosessissa erittäin oleellinen osa. Davenportin (1993 , viitannut Forsell 2001, 5) mukaan koko johdon tulee sitoutua ja osallistua organisaation muutokseen, mutta viime kädessä muutoksen pitää olla yhden määrätietoisen ja asialleen omistautuneen johtajan vastuulla. Vastuullisella johtajalla täytyy olla riittävästi valtaa päättää muutosta koskevista asioista. Davenport (1993 , viitannut Forsell 2001, 5) jopa väittää, että johtajan tyytymättömyys senhetkiseen tilanteeseen, sekä tyytymättömyydestä johtuva sitoutumattomuus muutokseen on tärkein yksittäinen tekijä onnistumisen kannalta.

Tutkimuksen aikana johdon tuki toimi alussa esimerkillisesti. Johdon tuen avulla pilottiprojekti saatiin käyntiin. Alun jälkeen tukea ei erityisesti ollut, vaikka projektin työntekijät olisivat varmasti tarvinneet johdon tukea vielä hankalissa tilanteissa pilottiprojektin aikanakin. Kaikkien organisaation työntekijöiden on myös tiedostettava olemassa olevan prosessin ongelmat ja muutoksen tarpeellisuus. Tämän asian teroittamiseen olisi varmasti kaivattu johdolta enemmän tukea. Tässä vaiheessa tosin kaivataan jonkinlaisia oppimisprosessejakin yritykseen ja olisi toivottavaa, että näitä asioita suunniteltaisiin jo etukäteen paremmin.

Tutkimuksen tuloksiin vaikuttavien tekijöiden analysoinnin jälkeen päästään varsinaisten tulosten analysointiin. Tutkielman alussa havaittiin, että testauksen hallitsemiseksi työvälineitä löytyy jokaiseen V-mallin vaiheeseen. Tutkielmassa on esitelty Fewsterin ja Grahamin (1999, 7) esittämä malli työvälineiden luokittelemiseksi. Mallia tutkimalla selvisi, että välineillä on paljon päällekkäisiä ominaisuuksia. Riippuu siis työvälineen käyttäjästä ja välineen

toteuttajasta pitkälti, mitä ominaisuutta hän painottaa ja mihin luokkaan välineen sijoittaa. Pilottiprojektissa käytetyn testauksen hallintavälineen kehittäminen jatkuu yrityksessä ja siihen kehitetään uusia ominaisuuksia ja liitetään uusia välineitä. Testauksen hallintaväline vaatii siis jatkossakin tutkimista ja ominaisuuksien mittaamista. On tärkeää tietää, mihin suuntaan väline kehittyy. Tässä vaiheessa tutkitaan kuitenkin välineen ominaisuuksia, joiden avulla prosessin suoritus voidaan automatisoida ja tuloksia raportoida.

Ohjelmistotuotannossa testaaminen vaatii erittäin paljon resursseja sekä aikaa. Tehokkuuden hakeminen onkin yleisin syy automatisoida testaamista; tällöin projekteissa voidaan vähentää manuaalisesti suoritettavan testauksen osuutta. Erityisesti toistettavat testitapaukset tulisi toteuttaa automatisoinnin avulla, jolloin resurssien säästö tulee parhaiten esille. Ajamalla regressiotestejä ohjelman uusissa versioissa voidaan suorittaa useampia testitapauksia vähemmässä ajassa (Fewster & Graham 1999, 9; Giraudon & Tonellan 2003). Nämä näkemykset saivat tapaustutkimuksesta tukea. Työvälineen voidaan katsoa tehostaneen testausprosessia.

Kirjallisuuden pohjalta tehdystä tarkastelusta tosin ilmenee, että säästöt testausbudjetissa ja testausajan lyhenemisessä ovat vaikeita saavuttaa, koska juuri näiden tavoitteiden saavuttaminen edellyttää aikaa ja riittävän monta testauskierrosta. Testauksen automatisoinnin suunnittelu ja ylläpito vie selkeästi enemmän aikaa kuin vastaava manuaalinen suunnittelu ja ylläpito, joten testejä on ajettava useaan kertaan, ennen kuin säästöä voidaan saavuttaa. Giraudon ja Tonellan (2003) mukaan suurin osa automatisoinnin tavoitteista saavutetaankin pitkällä aikavälillä.

Fewsterin ja Grahamin (1999, 9) mukaan testauksen automatisointi voi parantaa ohjelmistojen laatua lisäämällä testaajien aikaa perehtyä testaamaan toimintoja, joita ei voida automatisoida. Myös kohdeyrityksen pilottiprojektissa testauksen automatisoinnin parissa työskennelleiltä testaajilta jäi aikaa testien

suorittamisen ohella osallistua manuaalisen testauksen suorittamiseen. Eli samalla määrällä resursseja voidaan lisätä testauksen määrää ja kattavuutta testauksenhallinnan automatisoinnin avulla. Lisäksi tietokoneet voivat ajaa testitapauksia esim. öisin sekä viikonloppuisin, jolloin testaajien olisi hankalampi testata vastaavia testitapauksia. On kuitenkin muistettava, että testauksenhallinnan automatisointijärjestelmän ja -ympäristön luominen on useissa tapauksissa monin verroin resursseja vaativa toimenpide verrattuna manuaaliseen testaamiseen. Pitkällä aikavälillä testauksenhallinnan automatisointi on kuitenkin usein järkevää.

Kuten aikaisemmin jo todettiin, testauksen hallinnan automatisoinnin avulla voidaan päästä parempaa testiympäristön, dokumenttien sekä virheiden hallintaan. Edellä mainittuihin tavoitteisiin voidaan päästä juuri organisoimalla dokumentteja sekä mittaamalla erilaisia asioita. Virheidenhallinnan avulla puolestaan päästään käsiksi virheen elinkaareen, jolloin raportteihin voidaan kerätä tietoja tietyn virheen muuttuneista tiloista, luokitteluista ja priorisoinnista. Avainasia on se, kuinka seurattavaksi testausprosessi voidaan tehdä testien suorituksen aikana sekä ennen ja jälkeen varsinaisia testiajokertoja. Tavoitteena on päästä reaaliaikaiseen seurantaan, jolloin projektin todellinen tilannetieto olisi koko ajan saatavilla. Näiden seikkojen paraneminen automatisoidussa prosessissa on nähtävillä testauksen raportoinnin ja analysoinnin tehostumisena pilottiprojektissa.

Tehokkuus on suorassa vaikutuksessa rahaan, joka kuluu prosessia suoritettaessa, ja siten se onkin yksi pääsyy sille, miksi testausprosessia halutaan automatisoida (Fewster & Graham 1999). Tehokkuutta kehitettäessä ja mitattaessa pitäisi kuitenkin pyrkiä kiinnittämään huomiota välineen muihinkin ominaisuuksiin, jotta esim. välineen luotettavuus ei kärsi tehokkuuden kustannuksella (Harsu 2003). Tutkimuksessa pyrittiin seuraamaan luotettavuutta analysoitaessa tuotteen tehokkuutta, sillä

tarkoituksena oli saada järjestelmästä mahdollisimman tehokas luotettavuudesta tinkimättä.

Pilottiprojektissa pyrittiin arvioimaan prosessin ja välineen kypsyyttä luotettavuuden näkökulmasta. Operaatioiden epäonnistumiset jaksottuivat selkeästi testauksen ensimmäisiin testiajoihin. Testaus kannattaakin aloittaa lyhyillä testeillä ja pienellä testitapausten määrällä. Kun ohjelmisto näyttää toimivan oikein, voidaan testitapausten määrää lisätä tarpeen mukaan (Fewster & Graham 1999). Myös käytettävyydestä kerätyt tiedot osoittavat, että väline koetaan hankalammaksi käyttää alussa. Käytettävyyden osalta huomioitavaa on, että tuotteen käyttö vaatii asiantuntijan opastusta ja testaajat eivät tunteneet itseään kovin varmoiksi pilottiprojektin aikana. Muutoin käytettävyydestä annettiin melko hyvät pisteet ja voidaankin todeta, ettei väline enää lopussa tuntunut käyttäjistä tarpeettoman monimutkaiselta tai hankalalta käyttää. Fewsterin ja Grahamin (1999, 10) mukaan testauksen hallintajärjestelmän käyttö vaatii testaajalta teknistä osaamista. Hänen on hallittava tarkat tekniset tiedot järjestelmän käytöstä. Jokaisen järjestelmää käyttävän tulisi aina käydä koulutus järjestelmän käytöstä.

Tutkimuksessa ei paneuduttu testauksen hallintavälineen tekniseen puoleen erityisemmin, mutta sen merkitys tuli työn yhteydessä esille väistämättä, sillä järjestelmän käyttö ilman tehokasta ja toimivaa ympäristöä on mahdotonta. Toimiva järjestelmä auttaa ymmärtämään koko välineen merkityksen omassa työssä. Järjestelmävikojen nopea havaitseminen ja korjaaminen ovat edellytyksiä, jotta välinettä ei hylättäisi ja unohdettaisi (Ehrlich 1987). Järjestelmän toimimattomuus vaikuttaa pahimmillaan niin, että palataan takaisin vanhoihin toimintamalleihin. Ylläpidon onnistuminen työssään ja vikojen vähäinen määrä vaikuttivat positiivisesti tämän tutkimuksen tuloksiin. Välinettä voidaan siis pitää ylläpidettävänä teknisen ylläpitäjän näkökulmasta, vaikka tehokkuudesta saatujen tulosten valossa voidaan todeta, että testaajien työajasta merkittävä osa kuluu ympäristön ja testien ylläpitoon. Automatisoitujen testitapausten ja

ympäristön ylläpito vaatii resursseja. Testattavan ohjelmiston muuttamisen yhteydessä saatetaan joutua muuttamaan myös suuri osa testitapauksista. (Fewster & Graham 1999, 10; Chan 2000, 713)

Testauksen tekniset viat ja ongelmat aiheuttavat usein vaikeuksia. Testauksen hallinnan automatisointijärjestelmä on itsessään hyvin monimutkainen ja monenlaista toiminnallisuutta sisältävä kokonaisuus, joten sekään tuskin on täysin immuuni virheille. Pilottiprojektin aikana ei kuitenkaan ilmaantunut paljon virheitä, joten järjestelmää voidaan pitää jo nyt hyvin vikasietoisena. Kaatumisen välttämissuhteeksi saatiin 94 %, joten järjestelmä pystyy välttämään kaatumisen 94 % todennäköisyydellä virheen ilmaantuessa. Lukua voidaan pitää hyvänä ja se varmasti paranee pilottiprojektin yhteydessä ilmaantuneiden virheiden korjaamisen jälkeen.

Uutena ominaisuutena testauksen kuormitustestissä ilmeni järjestelmän optimaalisen tehokkuuden vaihtelu tilanteesta riippuen. Kirjallisuudessa ei mainittu mitään järjestelmän toiminnan vaihtelemisesta tilanteesta riippuen, mutta kuormitustestausten tulosten perusteella voidaan todeta, että järjestelmästä saatava maksimaalinen hyöty tehokkuuden suhteen riippuu testien määrästä sekä tavasta, jolla järjestelmä konfiguroidaan suorittamaan testiajo. Tällä saattaa olla merkitystä etenkin, jos samassa testilaboratoriossa ajetaan useiden projektien testejä yhtä aikaa, jolloin järjestelmä kuormittuu.

7 YHTEENVETO

Tutkielmassa on tarkasteltu testausprosessia ja testauksen hallintaa automatisoinnin näkökulmasta. Tutkielman tavoitteena oli kartoittaa kirjallisuuden pohjalta testausprosessiin ja sen hallintaan liittyviä seikkoja, jotka on osattava ottaa huomioon testausprosessia ja sen hallintaa automatisoitaessa ja kehitettäessä. Edelleen tavoitteena oli tapaustutkimuksen keinoin arvioida erään testauksen hallintavälineen laatua prosessissa sekä analysoida saatuja tuloksia kirjallisuuden pohjalta. Tutkielman tutkimusongelmana oli selvittää, mitkä ovat testausprosessin automatisoinnin menestystekijät ja ongelma-alueet. Tutkimuksen lähestymistapana toimi käytännönläheinen deduktiivinen päättely. Päättely nojasi aikaisempaan teoriaan, jota tutkielman alussa on käyty läpi. Päättelyn keinoin verrattiin tehtyä tutkimusta olemassa olevaan teoriaan ja haettiin kohdeyrityksestä kerätyille tutkimustuloksille selityksiä.

Tässä tutkielmassa on pyritty tuomaan esiin testauksen hallinta -työvälineen prosessia tukevat ominaisuudet. Testauksen hallinta –työvälineen avustavan ohjelmiston eri osilla voi hallinnoida testiprosessia organisoimalla testauksen dokumentteja (Giraudo & Tonella 2003) ja pitämällä kirjaa erilaisista metriikoista (Hetzel 1988). Näin testausprosessista ja sen kehittymisestä voidaan saada ajantasaista tietoa ja koko prosessi on paremmin uudelleenkäytettävissä ja ylläpidettävissä. Testausprosessin ja sen hallinnan kehittäminen ovat ajankohtaisia asioita monessa yrityksessä kuten tapaustutkimuksen kohdeyrityksessäkin.

Tutkimuksen keskeisenä tuloksena selvisi, että testauksen automatisointi on vaikeaa ja automatisoinnin yhteydessä esiintyy paljon erilaisia ongelmia, mutta automatisoinnin avulla testausprosessia voidaan tehostaa, mikäli toistoja testien ajossa tehdään useita kertoja. Ongelmallisena pidettiin välineen käytön aloitusta, jolloin testien ajaminen ei alussa ottanut onnistuakseen ja todettiin,

että testaus kannattaakin aloittaa lyhyillä testeillä ja pienellä testitapausten määrällä. Käyttäjät tarvitsevat myös koulutusta välineen käytöstä.

Tapaustutkimus antoi kiinnostavaa tietoa ja vahvisti muualla tehtyjä arvioita oletetun järjestelmän toiminnasta. Kuitenkin lopullinen tarkka arvio testauksen hallintavälineen eduista ja ongelmista edellyttäisi useiden järjestelmien tutkimista. Erilaisten ratkaisujen tutkimiseen ja laajojen kokonaisuuksien hallitsemiseen ei riitä pelkkä päättely yhden järjestelmän tiedoista, vaan siihen tarvittaisiin lisää aineistoa ja työkaluja tietojen analysointiin. Tämä olisi vaatinut enemmän resursseja, kuin tässä työssä oli käytettävissä.

Ohjelmistoteollisuus on verraten nuorta, ja voidaan yleisesti olettaa, että prosessien ja välineiden laadun seurannan tärkeys tulee nousemaan tulevaisuudessa. Tällä on vaikutuksensa ohjelmistokehitysprosessien mitattavan materiaalin sekä laatumallien kehittämisen tarpeisiin, ja sitä kautta kehitys edesauttaa mittaamisen yleistymistä. Ohjelmistomittojen käyttöönoton keskeinen kysymys on myös käyttökelpoisten työkalujen saatavuus ja integroitavuus ohjelmistoprosessiin ja sen välineisiin. Vaikka mittadataa pystyttäisiinkin tuottamaan, raakadatan jalostamiseen ja analysointiin on oltava työkaluja. Pelkästään tämän tutkielman toteuttamiseen tarvittiin useita manuaalisesti suoritettuja tietokantahakuja, lokitiedostojen ja raporttien läpikäyntejä sekä tuntikirjausten analysointeja.

Olennainen jatkotutkimuksen kohde aihealueelta on mielestäni syventyä testaustyökalujen tehokkaaseen käyttöön. Tutkimuksen avulla voitaisiin pohtia esim., millaisia metriikoita testauksen hallintavälineen avulla voidaan automaattisesti kerätä ja millaista apua näiden metriikoiden käytöstä projektille on. Toinen jatkotutkimusaihe alueelta voisi liittyä käyttöönottoprosessiin, joka kirjallisuuden mukaan tuntuu hyvin usein organisaatioissa epäonnistuvan. Millainen käyttöönottoprosessin ylipäätään pitäisi olla, jotta se tukisi parhaalla mahdollisella tavalla sekä järjestelmän käyttöönottoa että toimintojen

oppimisprosessia? Varsin usein käyttöönoton yhteydessä esiintyy esim. muutosvastarintaa vaikka organisaatio määrittelisikin itsensä oppivaksi organisaatioksi. Tilanne tuntuu mielestäni ristiriitaiselta.

Jatkotutkimista mielestäni kaipaisi myös tehokkuuden ja suorituskyvyn tutkiminen. Perinteisesti suorituskykyanalyysiä on tehty mittaamalla olemassa olevia järjestelmiä niiden testaus- ja käyttövaiheissa kuten tässäkin tutkielmassa. Vaikka mittaamalla saadaankin luotettavia tuloksia, menetelmä on kuitenkin melko uhkarohkea, sillä mittauksella voidaan ainoastaan jälkikäteen todeta suorituskyky riittäväksi tai riittämättömäksi. Suorituskyky tulisi ottaa huomioon jo järjestelmän suunnitteluvaiheessa. Suunnittelussa tulisi eliminoida epäsuotuisat ratkaisut jo ennen niiden toteutusta. Lisätutkimusta kaivattaisiin siis mielestäni järjestelmien suorituskykysuunnittelusta.

LÄHDELUETTELO

Adrion R. W., Branstad M. A. & Cherniavsky C. A. 1982. Validation, Verification, and Testing of Computer Software. ACM Computing Surveys 14(2), 159-152.

Ahonen J. & Junttila T. 2003. A Case Study on Quality-Affecting Project Management Problems in Software Engineering. University of Jyväskylä. Information Technology Research Institute.

Allen C. D. 1995. Succeeding as a clandestine change agent. Communications of the ACM. 38(5), 81-88.

ANSI/IEEE 1998. IEEE Standard 829-1998: IEEE standard for software test documentation [online]. IEEE Computer Society [viitattu 3. 4.2004]. Saatavilla www-osoitteessa <http://ieeexplore.ieee.org/xpl/tocresult.jsp?isNumber=16010>

Bach J. 1999. Test Automation Snake Oil [online]. Satisfice inc. [viitattu 25.2.2004]. Saatavilla www-osoitteessa <URL:http://www.satisfice.com/articles/test_automation_snake_oil.pdf>.

Bai X., Tsai W. T., Paul R., Techeng S. & Li B. 2001. Distributed end-to-end testing management. Proceedings of the Fifth IEEE International Enterprise Distributed Object Computing Conference, Seattle, September 4-7. IEEE Computer Society, 140 – 151.

Banker R.D. Datar S. M., Kemerer C. F., Zweig D. 1993. Software Complexity and Maintenance Costs. Communications of the ACM 36 (11), 81-94.

Bassin K.A., Kratschmer T., Santhanam P. 1998. Evaluating Software Development Objectively. IEEE Software, 15(6), 66-74.

Boehm B. 1979. Guidelines for verifying and validating software requirements and design specifications. Euro IFIP 79, Nort-Holland Publishing Company IFIP, 711-719.

Boehm B., Brown J. R., and Lipow M. 1976. Quantitative evaluation of software quality. Proceedings of the 2nd international conference on Software engineering, 592-605.

British Computer Society SIGIST. 1998a. Software testing – Part 1: Standard for Vocabulary, BS 7925-1:1998 [online]. BCS SIGIST Standards Working Party [viitattu 3. 4.2004]. Saatavilla www-osoitteessa <http://www.testingstandards.co.uk/bs_7925-1_online.htm>.

British Computer Society SIGIST. 1998b. Software testing – Part 2: Standard for Software component testing, BS7925-2:1998[online]. BCS SIGIST Standards Working Party [viitattu 3. 4.2004]. Saatavilla www-osoitteessa <http://www.testingstandards.co.uk/bs_7925-2_online.htm>.

British Computer Society SIGIST. 1999. Reliability Testing [online]. BCS SIGIST Standards Working Party [viitattu 3. 1.2004]. Saatavilla www-osoitteessa <http://www.testingstandards.co.uk/ReliabilityTesting23_4_99.htm>.

Brocklehurst S. & Littlewood B. 1992. New ways to Get Accurate Reliability Measures. IEEE Software, 9(4), 34-42.

Brooke J. 1986. SUS – A usability scale. [online] User information Architecture A/D Group [viitattu 2. 10.2004]. Saatavilla www-osoitteessa <http://www.cee.hw.ac.uk/~ph/sus.html>.

Chan A. H. 1995. The Benefits of Stress Testing. IEEE Transactions on Components, Packaging and Manufacturing Technology 18(1), 23-29.

Chan L. 2000. Platform-Independent And Tool-Neutral Test Descriptions For Automated Software Testing. Proceedings of the 22nd international

conference on Software engineering, Limerick, June 2000. New York, NY, USA, ACM Press, 713–715.

Chung C. S., Jung H. W. & Kim S. G. 2004. Measuring Software Product Quality: A Survey of ISO/IEC 9126. IEEE Software 21(5), 88–92.

Dromey, G. 1996. Cornering the Chimera, IEEE Software, 13(1), 33-43.

Daskalantonakis M.K. 1992. A practical view of software measurement and implementation experiences within Motorola. Software Engineering 18(11), 998 – 1010.

Davenport T. H. 1993. Process Innovation – Reengineering Work through Information Technology. Boston, Harvard Business School Press.

Davis A. M. 1988. A comparison of techniques for the specification of external system behavior. Communications of the ACM 31(9), 1098-1115.

Desai H. D. 1994. Test Case Management System. Proceedings of the Global Telecommunications Conference, San Francisco, Nov 28 - Dec 2. Atlanta: The Global Bridge, IEEE, 1581–1585.

Dromey R. 1995. A model for software product quality. IEEE Transactions on Software Engineering, 21(2):146-162.

Ehrlich S. 1987. Strategies for encouraging successful adoption of office communication systems. ACM Transactions on Office Information Systems 5(4), 340-357.

Elbert. M., Mpagazehe. C. & Weyant, T. 1994. Stress testing and reliability. Proceedings of the Conference Record IEEE, Orlando, March 29-31. 357 – 362.

Everett W., Keene S. & Nikora A. 1998. Applying Software Reliability Engineering in the 1990s. IEEE Transactions on reliability, 47(3), 372-378.

Fenton N. 1994. Software measurement: a necessary scientific basis. Software Engineering 20(3), 199–206.

Fenton N.E. & Neil M. 2000. Software Metrics: Roadmap. Proceedings of the conference on The future of Software engineering Limerick, Ireland, June 4-11. New York: ACM Press, 357-370.

Fenton N.E. & Pfleeger S.L. 1996. Software metrics: a rigorous and practical approach. Boston: International Thompson Computer Press.

Fewster M. & Graham D. 1999. Software Test Automation: Effective use of test execution tools. London: Addison-Wesley.

Forsell M. 2000. OT-prosessin kehittäminen kohti uudelleenkäyttöä. Pisko-projektin raportti, Jyväskylän yliopisto, Tietotekniikan tutkimusinstituutti.

Forsell M. 2001. Uudelleenkäytön organisointi osa 3 – muutosprosessi. Pisko-projektin raportti, Jyväskylän yliopisto, Tietotekniikan tutkimusinstituutti.

Giraudo G. & Tonella P. 2003. Designing and Conducting an Empirical Study on Test Management Automation. Empirical Software Engineering 8(1), 59-81.

Grady, R. and Caswell, D. 1987. Software Metrics: Establishing a Company-Wide Program, Prentice Hall.

Grady R. 1993. Practical Results from Measuring Software Quality. Communications of the ACM 36(11), 62-68.

Gueheneuc Y. & Khosravi K. 2004. A Quality Model for Design Patterns. [online]. [viitattu 28.1.2005]. Saatavilla www-osoitteessa <http:// http://www.yann-gael.gueheneuc.net/Work/Tutoring/Documents /041021+Kashayar+Khosravi+Technical+Report.doc.pdf>.

Hall P. A. V., May H. R. & Zhu H. 1997. Software Unit Test Coverage and Adequacy. ACM Computing Surveys 29(4), 366-427.

Harsu M. 2003. Metriikat [online]. Tampereen teknillinen yliopisto [viitattu 25.2.2004]. Saatavilla www-osoitteessa <http://www.cs.tut.fi/kurssit /8102030/kalvot/metriikat.pdf>.

Hayes L. 1995. Extracts from The Automated Testing Handbook. Teoksessa Fewster M. & Graham D, Software Test Automation, Adisson-Wesley. New York: ACM Press, 493–517.

Hendrickson E. 1999: Making the Right Choice [online]. The Software Testing & Quality Engineering Magazine [viitattu 25.2.2004]. Saatavilla www-osoitteessa <URL: http://www.qualitytree.com/feature/mtrc.pdf>.

Hetzel B. 1988. A Complete Guide to Software Testing: Second Edition. New York: A Wiley-QED Publication.

Humphrey W. S. 1989. Managing the Software Process. Addison-Wesley, Reading, Massachusetts.

Hynynen P. 2000. Kun tiedät tavoitteen, olet jo matkalla sinne. Jyväskylän yliopisto. Terveystieteiden laitos. Lisensiaatin tutkielma.

IEEE 610. 1990. IEEE Standard Computer Dictionary – A Compilation of IEEE Standard Computer Glossaries, Institute of Electrical and Electronic Engineers IEEE.

ISO/IEC. 2000a. ISO/IEC 9126-2 standard: Software Engineering - Product Quality Part 2 - External Metrics. Standards Council of Canada (SCC).

ISO/IEC. 2000b. ISO/IEC 9126-4 standard: Software Engineering - Product Quality - Part 4: Quality In Use Metrics. Standards Council of Canada (SCC).

ISO/IEC. 2002. ISO/IEC 9126-3 standard: Software engineering - Software product quality requirements and evaluation - Part 3: Quality metrics – Metrics reference model and guide. Standards Council of Canada (SCC).

Järvinen P. 1999. PC Tietosanakirja 99. Jyväskylä: Teknolit Oy.

Järvinen P. & Järvinen A. 2000. Tutkimustyön metodeista. Tampere: Opinpajan kirja.

Kaner C. 1997. Improving the Maintainability of Automated Test Suites[online]. Paper Presented at Quality Week [viitattu 25.2.2004]. Saatavilla www-osoitteessa <URL:http://www.testing.com/writings/classic/mistakes.html>.

Kaner C. 2001. Developing the Right Test Documentation [online]. Pacific Northwest Software Quality Conference [viitattu 25.2.2004]. Saatavilla www-osoitteessa <http://www.kaner.com/pdfs/test_docs_pnsqc.pdf>.

Kauppinen R. & Taina J. 2003. RITA Environment for Testing Framework-based Software Product Lines. Proceedings of the Eighth Symposium on Programming Languages and Software Tools, Kuopio, Finland, June 2003, University of Kuopio, 58-69.

Kautto T. 1996. Ohjelmistotestaus ja siinä käytettävät työkalut [online]. University of Jyväskylä [viitattu 1. 2.2004]. Saatavilla www-osoitteessa <http://www.mit.jyu.fi/opiskelu/seminaarit/ohjelmistotekniikka/testaus>.

Kettunen J. & Simons M. 2001. Toiminnanohjausjärjestelmän käyttöönotto pk-yrityksessä. Vantaa, VTT Julkaisuja. ISBN 951-38-5881-1.

Korel B. & Al-Yami A. 1998. Automated regression test generation. Proceedings of ACM SIGSOFT international symposium on Software testing and

analysis, Clearwater Beach, March 1998. New York, USA, ACM Press, 143–152.

Lanning D.L. & Khoshgoftaar T.M. 1994. Modeling the Relationship Between Source Code Complexity and Maintenance Difficulty. IEEE Computer 27(9), 35-40.

Marick B. 1997. How to Misuse Code Coverage[online]. Testing Foundations [viitattu 6.5.2004]. Saatavilla www-osoitteessa <http://testing.com/writings/coverage.pdf>.

Marick B. 1998. When Should a Test Be Automated [online]. Testing Foundations [viitattu 4.5.2004]. Saatavilla www-osoitteessa <URL: http://www.testing.com/writings>.

Marick B. 2000. New Models for Test Development [online]. Testing Foundations [viitattu 26.11.2003]. Saatavilla www-osoitteessa <http://www.testing.com/writings >.

McCall, J. A., Richards, P. K., and Walters, G. F. 1976. Factors in software quality, AD/A-049-014/015/055, National Technical Information Service, Springfield, VA.

Myers G. J.1979. The Art of Software Testing. USA: John Wiley & Sons Inc.

Mynatt B. 1990. Software Engineering With Student Project Guidance. USA: Prentice-Hall.

Nagappan N. 2004. Toward a Software Testing and Reliability Early Warning Metric Suite. Proceedings of the 26th International Conference on Software Engineering, May. IEEE Computer Society, 60-62.

Ogasawara H., Yamada A. & Kojo M. 1996. Experiences of software quality management using metrics through the life-cycle. Proceedings of the 18th International Conference, March 25-30. IEEE, 179 - 188.

Ortega M., Pérez M. and Rojas T. 2003. Construction of a Systemic Quality Model for evaluating a Software Product. Software Quality Journal, 11(3), July 2003, 219-242. Kluwer Academic.

Paaso E., Mattila M. & Sivonen J. 2004. Mittaaminen [online]. Menetelmä - opetuksen valtakunnallinen tietovaranto [viitattu 4.10.2004]. Saatavilla www-osoitteessa <http://www.fsd.uta.fi/menetelmaopetus/mittaaminen/mittaaminen.html>.

Pettichord B. 2001. Success with Test Automation. Quality Week [online]. [viitattu 21.4.2004]. Saatavilla www-osoitteessa <http://www.io.com /~wazmo/succpap.htm>.

Pfleeger S.L., Fenton N. & Page S. 1994. Evaluating software engineering standards. Computer 27(9), 71-79.

Pohjolainen P. 2002. Software Testing Tools [online]. University of Kuopio [viitattu 13.2.2004]. Saatavilla www-osoitteessa <http://www.cs.uku.fi/research /Teho/julkaisut.html>.

Pohjolainen P. 2003. Testauksen automatisointi ja sen työkalut [online]. University of Kuopio [viitattu 13.2.2004]. Saatavilla www-osoitteessa <http://www.cs.uku.fi /research/Teho/testingday/automatisointi.pdf>.

Pyhäjärvi M. 2005. Testauskirja [online]. Maaret Pyhäjärven kotisivut [viitattu 28.2.2005]. Saatavilla www-osoitteessa <http://www.testauskirja.com/maaret_fi.htm>

Rathburg K. 1993. Managing automatic test systems in the 90's. Proceedings of the IEEE Systems Readiness Technology Conference, San Antonio, 20-23 Sept. 1993. IEEE, 19 – 25.

Smith M. 2000. Test Process Management [online]. Test Management Solutions Ltd [viitattu 25.2.2004]. Saatavilla www-osoitteessa <http:// www.sigist.org.uk/newsletterarchive/tester-a.shtm>.

Taina J. 2002. Software Testing and Validation [online]. University of Helsinki [viitattu 18.3.2004]. Saatavilla www-osoitteessa <http:// www.cs.helsinki.fi/u/taina/ohte/s-2002/luennot/testaus.pdf>.

Weller E.F. 1994. Using metrics to manage software projects. Computer 27(9), 27-33.

Winston E.R. 1999. IS consultants and the change agent role. ACM SIGCPR Computer Personnel. 20(4), 55-74.

Ylén J.P. 2004. Epälineaaristen riippuvuuksien kuvaaminen [online]. Helsinki University of Technology [viitattu 6.2.2005]. Saatavilla www-osoitteessa http://www.control.hut.fi/Kurssit/AS%2D74.400/materiaali.html.

Zambelich K. 1998. Totally Data-Driven Automated Testing [online]. Automated Testing Specialists [viitattu 18.4.2004]. Saatavilla www-osoitteessa <URL: http:// sqa-test.com/w_paper1.html>.

LIITE 1: SUS – SYSTEM USABILITY SCALE QUESTIONNAIRE (BROOKE 1986)

	Vahvasti eri mieltä				**Vahvasti samaa mieltä**
1. Olen sitä mieltä, että voisin käyttää tätä tuotetta säännöllisesti.	1	2	3	4	5
2. Tuote on mielestäni tarpeettoman monimutkainen.	1	2	3	4	5
3. Tuotetta on mielestäni helppo käyttää.	1	2	3	4	5
4. Mielestäni tuotteen käyttö vaatii asiantuntijan opastusta.	1	2	3	4	5
5. Mielestäni tuotteen eri toiminnot ovat liitetty toisiinsa onnistuneesti.	1	2	3	4	5
6. Mielestäni tuotteessa oli liikaa epäjohdonmukaisuuksia.	1	2	3	4	5
7. Uskoisin, että useimmat oppivat käyttämään tuotetta hyvin nopeasti.	1	2	3	4	5
8. Mielestäni tuotetta on hyvin hankala käyttää.	1	2	3	4	5
9. Tunsin oloni hyvin varmaksi tuotetta käyttäessäni.	1	2	3	4	5
10. Mielestäni pitää opetella paljon uusia asioita, jotta voi käyttää tuotetta.	1	2	3	4	5

LIITE 2: OSALLISTUJIEN OMINAISPIIRTEET

ISO 9126 Part 4 (ISO/IEC 2000b, 43)

Ikä: Henkilön kronologinen ikä.

Koulutus: Tutkinto sekä aika vuosina valmistumisesta. (Esim. KTM, 3 vuotta).

Rooli: Kuvaava nimi työroolista, joksi henkilöä tuotetta käyttäessä nimitetään.

Ammatillinen kokemus: Kuvaile käyttäjän taustaa, kuten kuinka paljon kokemusta käyttäjällä on vastaavista ympäristöistä tai käyttöjärjestelmistä tai tuotteen kohdealueesta.

Tuotekokemus: Viittaus kokemukseen vastaavanlaisten tuotteiden käytöstä ja käytön kestosta.