Ossi Savolainen

Ohjelmistotestaus: Testausprosessin luonti ja kehittäminen

Tietojärjestelmätieteen
Kandidaatin tutkielma
3.3.2005

Jyväskylän yliopisto
Tietojenkäsittelytieteen laitos
Jyväskylä

Tiivistelmä

Savolainen, Ossi Johannes
Ohjelmistotestaus: Testausprosessin laatiminen ja kehittäminen
Jyväskylä: Jyväskylän yliopisto, 2005-03-3
Tietojärjestelmätieteen Kandidaatin tutkielma

Tutkielman tarkoituksena on lähestyä perinteisen testausprosessin kautta täydellisempää ja kehittyneempää testausprosessia, jossa on pyritty huomioimaan ohjelmistotestauksen laatuun, kustannuksiin ja testausaikaan vaikuttavat positiiviset tekijät. Kehittyneemmässä testausprosessissa on myös eliminoitu perinteisiä ohjelmistotestaukseen liittyviä ongelmakohtia.

Tutkielmassa perinteisten testausprosessin menetelmien kautta on löydetty ohjelmistotestauksen kannalta kriittisiä osa-alueita, joihin ohjelmistotestausprosessia laadittaessa tai kehitettäessä tulisi kiinnittää huomiota. Tutkielmassa on myös pyritty keskittymään menetelmiin joilla sijoitettuun pääomaan nähden saadaan suurimmat laadulliset hyödyt. Tällaisia asioita ovat muun muassa testauksen staattisten menetelmien älykäs käyttö, testauksen riittävyyden tehokas arviointi, vaatimuksien jäljitettävyys ja testauksen mitattavuus.

Testauksen laadukkuuteen, kustannuksiin ja käytettyyn aikaan vaikuttavat monet tekijät. Kaikki nämä tekijät eivät välttämättä liity ohjelmistotestauksessa käytettyihin menetelmiin. Tällaisia asioita ovat esimerkiksi johtaminen, työntekijöiden motivaatio ja henkilöstön sitoutuminen. Nämä kaikki ovat testauksen onnistumisen kannalta kriittisiä tekijöitä, mutta eivät kuulu tämän tutkielman aihepiiriin.

Ajan kuluessa myös prosessi ”kuluu” ja kaipaa päivittämistä ja ylläpitämistä. Tutkielma ei kata ohjelmistotestausprosessin ylläpitoa kehittämisprosessin jälkeen.

Sisällysluettelo

Termit

Termi	Selitys
Ad hoc- testaus	Arvaukseen perustuva testauksen ja virheiden jäljittämisen menetelmä.
Ajuri	Työkalu jolla simuloidaan testattavan komponentin palveluiden kutsuja.
Automaattinen testaus	Tietokoneen suorittama (ja mahdollisesti myös analysoima) testaus.
Dynaaminen testaus	Ohjelmiston testaaminen ohjelmakoodi suorittamalla
EVO-malli	Ohjelmistotuotantoprosessia kuvaava elinkaarimalli, jossa ohjelmistoa kehitetään pienehköinä peräkkäisinä inkrementteinä
GUI	Graphical User Interface, graafinen käyttöliittymä
Harmaalaatikkotestaus	Musta- ja valkolaatikkotestauksen välimuoto, jossa käytetään hyväksi tietoa ohjelman toteutusperiaatteista.
Hyväksymistestaus	Loppukäyttäjän suorittama testaus jossa arvioidaan tuotteen kykyä vastata asiakasvaatimuksiin.
Integrointitestaus	Testaus jonka tarkoituksena on testata moduulien välisten rajapintojen toimintaa.
Järjestelmätestaus	Testaus jonka tarkoituksena on testata koko järjestelmän toimintaa.
Katselmointi	Menetelmä jolla todetaan jokin ohjelmistotuotantoprosessin vaihe päättyneeksi vaihetuotoksia tarkastelemalla.
Kelpoistaminen	Menetelmä jolla tutkitaan tuotteen sopivuus käyttötarkoitukseensa.
Manuaalinen testaus	Testaus jonka suorittaa ihminen (testaaja)
Moduulitestaus	Testausvaihe, jossa kohteena on vain yksittäinen moduuli tai komponentti.
Mustalaatikkotestaus	Testausmenetelmä joka perustuu testattavan kokonaisuuden ulkoiseen toimintaan (syötteisiin ja tulosteisiin).

Regressiotestaus	Testausmenetelmä jolla todennetaan ohjelmistoon tehdyn korjauksen toimivuus ja varmennetaan, että uusia vikoja ei ole syntynyt muuttumattomiin ohjelmiston osiin korjauksen myötä.
Staattinen testaus	Ohjelmiston testaaminen ilman suorittamista.
Tarkistuslista	Staattisen testauksen apuväline, jolla katselmointeja voidaan kohdistaa keskittymään olennaisiin asioihin tarkasteltavassa vaihetuotteessa.
Testausraportti	Testauksesta ja sen tuloksista tehtävä vaihetuotos.
Testaussuunnitelma	Suunnitelma jonka pohjalta ohjelmistotestaus voidaan viedä organisoidusti läpi.
Todentaminen	Menetelmä jolla varmistetaan, että tuote vastaa vaatimuksia ja määritelmiä.
Toimintotestaus	Testaus jossa uudet tai muuttuneet ominaisuudet testataan käyttäjän näkökulmasta.
Tynkämoduuli	Työkalu, jolla simuloidaan testattavan komponentin tarvitsemia muita komponentteja.
Valkolaatikkotestaus	Testausmenetelmä jossa huomioidaan testattavan kokonaisuuden sisäinen toiminta ja rakenne. Tunnetaan myös nimellä lasilaatikkotestaus
Vesiputousmalli	Perinteinen ohjelmistotuotantoprosessia kuvaava elinkaarimalli

1. Johdanto

Tutkielma antaa yleiskuvan ohjelmistotestauksesta, laadunvarmistuksesta ja ohjelmistotestausprosessista. Ohjelmistotestauksen eri käsitteet, vaiheet, vaihetuotokset, menetelmät, tekniikat ja prosessit tullaan käymään läpi loogisessa järjestyksessä. Tutkielman tarkoituksena on löytää menetelmät ja keinot, joilla ohjelmistotestauksesta saadaan suurimmat hyödyt laadunvarmistuksen näkökulmasta, mahdollisimman kustannustehokkaasti. Tutkielma tulee vastaamaan muun muassa seuraaviin kysymyksiin:

- Mitä on ohjelmiston laatu?
- Mitä on ohjelmistotestaus?
- Miten ohjelmistotestaus vaikuttaa ohjelmiston laatuun?
- Mihin ohjelmistotestaus sijoittuu ja mihin se voidaan sijoittaa ohjelmistotuotantoprosessissa?
- Mitä tekniikoita ohjelmistotestaukseen on olemassa?
- Mitkä ovat ohjelmistotestausprosessin vaiheet ja vaihetuotokset?
- Mitkä ovat kustannustehokkaita menetelmiä suorittaa ohjelmistotestausta?
- Mitkä ovat laadun kannalta tärkeimmät ohjelmistotestauksen menetelmät ja vaiheet?
- Kuinka ohjelmistotestausta voidaan nopeuttaa?
- Mitä ovat ohjelmistotuotanto- ja testausprosessit?
- Mihin prosesseja tarvitaan?
- Kuinka testausprosessia voidaan uudistaa ja kehittää?

Ohjelmistotestaus on eräs tärkeimmistä laadunvarmistuksen toimenpiteistä ohjelmistotuotantoprosessissa. Testauksen päämääränä on saavuttaa ohjelmistotuotteen hyvä ja riittävä laatu. Tutkielmassa aihepiiriin syventyminen aloitetaan tutustumalla laadun käsitteeseen, jotta voitaisiin paremmin ymmärtää testausprosessiratkaisujen syyt ja seuraukset.

Laadun määritelmän jälkeen ohjelmistotestaus esitellään yleisellä tasolla. Samalla määritellään ohjelmistotestaukseen kuuluvat toimenpiteet. Tämä on tärkeää, koska usein

esimerkiksi *staattiset testausmenetelmät*, kuten katselmoinnit, katsotaan kuuluvan varsinaisen testauksen määritelmän ulkopuolelle.

Ennen testausprosessin kehittämiseen siirtymistä määritellään prosessin käsite ja mitä prosessilla tarkoitetaan. Perinteinen testausprosessi, jossa testausta suoritetaan ohjelmistotuotantoprosessin lopussa on hyvä lähtökohta prosessiuudistukselle. Siksi perinteisen testausprosessin kautta lähdetään pohtimaan prosessiuudistusta ja sen tarjoamia hyötyjä ja mahdollisuuksia. Tutkielman lopussa esitellään kehittynyt ohjelmistotestauksen prosessimalli, jossa hyödynnetään havaintoja perinteisen testausprosessin heikkouksista ja muutamia eri teorioita ohjelmistotestausprosessin kehittämiseksi. Tarkoituksena on löytää ohjelmistotestausprosessin alueet joita kehittämällä saadaan suurimmat laadulliset hyödyt käytettyihin resursseihin nähden.

2. Ohjelmistotestaus

Ohjelmiston hyvä laatu on ohjelmistotestauksen tärkein tavoite. Siksi ohjelmistotestausta on syytä lähteä tarkastelemaan käsitteen *laatu* kautta.

2.1 Laadun määritelmä

Ohjelmistotestaus on yksi ohjelmiston laadunvarmistuksen tärkeimmistä toimenpiteistä. Ohjelmistotestauksen käsitettä määriteltäessä on syytä tutustua ensin laadun käsitteeseen, koska tuotteen riittävä laatu on ohjelmistotestauksen päätarkoitus. Lewisin (2000, 3) mukaan laadulle on olemassa kaksi yleisesti hyväksyttyä määritelmää:

- Kuinka hyvin ohjelmisto toteuttaa vaatimukset?
- Tekeekö ohjelmisto sen mitä asiakas tarvitsee?

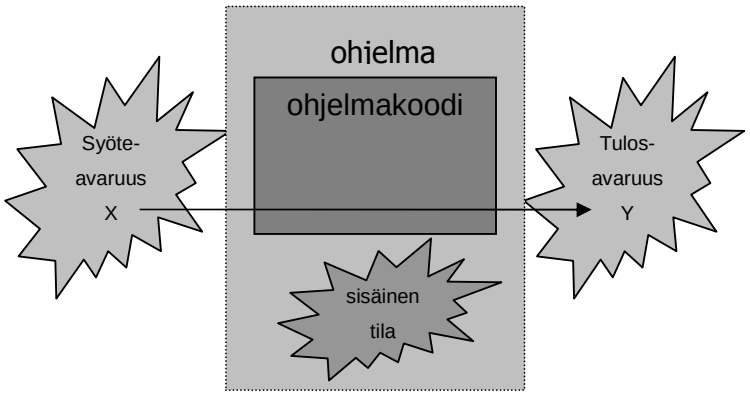
Näihin kysymyksiin liittyvät *todentamisen* ja *kelpoistamisen* käsitteet, jotka ovat ohjelmistotestauksen päätehtäviä. Todentamisella (objektiivinen laatukomponentti) varmistetaan, että tuote vastaa vaatimuksia ja määritelmiä. Kelpoistamisessa (subjektiivinen laatukomponentti) taas tutkitaan tuotteen sopivuus käyttötarkoitukseensa. Riittävän laadun saavuttamiseksi ohjelmistotuotantoprosessissa on suoritettava useita erilaisia toimenpiteitä. Tärkein laadunvarmistuksen toimenpide on ohjelmistotestaus, joka laajassa merkityksessään sisältää kaikki todentamisen ja kelpoistamisen toimenpiteet.

2.2 Ohjelmistotestauksen määritelmä

Haikalan ja Märijärven (2000, 266) mukaan, yleiskielessä testauksella tarkoitetaan lähes mitä tahansa kokeilemista. Erään toisen määritelmän mukaan (Pol, Teunissen, Veenendaal, 1995) testaus voidaan määritellä seuraavasti:

> Testing is a process of planning, preparation, and measuring aimed at establishing the characteristics of an information system and demonstrating the difference between the actual an the required status.

Ohjelmistojen testauksen yhteydessä testaus voidaan kuitenkin määritellä tarkemmin suunnitelmalliseksi virheiden etsimiseksi. Käytännössä tämä tarkoittaa, että ohjelmistotestaus on suunniteltu jollakin järkevällä menetelmällä, jonka pohjalta ohjelmistotestaus voidaan suorittaa. Jotta testaus ylipäätään voidaan suunnitella on testauksen pohjaksi oltava spesifikaatioita, joista selviää testattavan ohjelman syöteavaruus, tulosavaruus ja tietenkin oikea lopputulos. Näiden tietojen pohjalta voidaan rakentaa testitapaus. Yksinkertaisimmillaan testausta voidaan kuvata kuten Haikala ja Märijärvi ovat kuvanneet (kuva 1). Tehdään testitapaus, jossa valitaan syöte X, joka mahdollisesti aiheuttaa ohjelmassa sisäisen tilan muutoksen ja tuloksen Y. Näin spesifikaatioiden perusteella laaditun testitapauksen perusteella voidaan varmentaa toimiiko ohjelma oikein.



Kuva 1: Ohjelmiston testaus mukaillen Haikalan ja Märijärven (2000) mallia

Käytännössä testaaminen ja ohjelmiston laadun varmistaminen ei ole näin itsestään selvä ja helppo tehtävä. Yksittäisten spesifikaatioiden perusteella laadittujen testitapausten suorittaminen implementointityön päätteeksi ei riitä varmistamaan ohjelmiston riittävää laatua. Lewis (2000) toteaa kirjassaan, että testauksen suorittaminen ohjelmistotuotantoprosessin loppupäässä, tekee mahdottomaksi vaikuttaa enää tuotteen laatuun positiivisesti. Siksi on tärkeää ymmärtää, että ohjelmistotestaus on muutakin kuin vain laadittujen testitapausten suorittamista. Ohjelmistotestaukseen

kuuluvat muutkin todentamisen ja kelpoistamisen tehtävät. Todentamisen ja kelpoistamisen menetelmiin kuuluvat myös kaikkien ohjelmistotuotantoprosessin vaihetuotosten oikeellisuuden todentaminen esimerkiksi katselmointien, analyysien ja testauksen suorittamisen kautta.

2.3 Testauksen historia

Ohjelmistotestausta on ollut olemassa yhtä kauan kuin on ollut ohjelmistotuotantoakin, jonka historia voidaan Heikkilän (2003) mukaan katsoa alkaneeksi 1950-luvulla. Dustinin, Rashkan ja Paulin (1999, 5) mukaan ensimmäiset ohjelmistot oli rakennettu lähinnä tieteellisiin tarkoituksiin tai ne olivat puolustusministeriön tuottamia ohjelmistoja. Testiskenaariot rakennettiin paperille ja testaus oli lähinnä monimutkaisten algoritmien oikeellisuuden todistamista ja testaamista ns. tietovirtakaavioiden mukaan (lasilaatikkotestaus). Testaukseen ei tarvinnut ylipäätään kiinnittää suurta huomiota koska rajallinen määrä testaustoimenpiteitä pystyi tehokkaasti kattamaan koko järjestelmän. Ohjelmistotestaus suoritettiin lähes poikkeuksetta viimeisenä ohjelmistotuotantoprosessin vaiheena. Ohjelmistotestauksen suorittaminen ohjelmistotuotantoprosessin lopussa on valitettavan yleistä vielä tänäkin päivänä.

Ohjelmistotestauksen uusi aikakausi alkoi kaupallisen ohjelmistokehityksen ja niin sanottujen PC-tietokoneiden (Personal Computer) markkinoille tulon myötä. Online-järjestelmät, asiakas-palvelin järjestelmät ja graafiset käyttöliittymät (GUI, Graphical User Interface) asettivat testaukselle kokonaan uusia haasteita. Esimerkiksi graafisten käyttöliittymien testauksessa merkkipohjaiset järjestelmät olivat muuttuneet graafisiksi ja testattavana olivat syötteiden sijaan graafiset komponentit, kuten painonapit, tekstikentät ja ikonit. Testattavana oli lukematon määrä erilaisia kombinaatioita. Asiakas-palvelin systeemeissä taas ei enää testattukaan yhtä ainoaa sovellusta yhdessä ainoassa järjestelmässä, vaan kolmen komponentin yhteistoimintaa: verkko, asiakas ja palvelin (Dustin , Rashka ja Paul 1999, 5).

Vuosien kuluessa paitsi itse testaus ja testaustyökalut ovat kehittyneet, ovat myös testausmenetelmät ja –prosessit kehittyneet. Ohjelmistojen on pystyttävä käsittelemään valtavia käyttäjämääriä, pöytäkoneet ovat muuttuneet tehokkaammiksi ja tuotantoprosessit ovat suuruusluokkansa vuoksi erittäin vaikeasti hallittavissa.

Lukemattomia erilaisia lähestymistapoja ohjelmistotestauksen haasteisiin on esitelty vuosien varrella. Useimpia esitellyistä testausmenetelmistä tarvitaan, koska jokainen testattava sovellus eroaa toisesta mm. rakenteen, suunnittelun, käytetyn teknologian ja käyttötarkoituksen mukaan. Tämän vuoksi on laadunvarmistuksen näkökulmasta erittäin tärkeää, että käytetään ohjelmistoon sopivia testausmenetelmiä.

2.4 Ohjelmistotestaus osana ohjelmistotuotantoprosessia

Yleensä ohjelmistotestaus sijoitetaan ohjelmistotuotantoprosessin loppupäähän. Tämä ei sinällään kerro lukijalle mitään testauksen sijoittumisesta ennen kuin tiedetään millainen ohjelmistotuotantoprosessin rakenne on. Seuraavassa kuvataan ohjelmistotuotannon perusrakennetta yleisimpien mallien perusteella.

2.4.1 Testauksen sijoittuminen ohjelmistotuotantoprosessiin

Testaus on osa ohjelmistotuotantoprosessia. Prosessilla tarkoitetaan joukkoa toisiinsa liittyviä toimintoja, jotka muuttavat syötteitä tuotoksiksi (ISO/IEC 15504-9, 1998). Näin tapahtuu myös ohjelmistotuotantoprosessissa. Siinä jokainen vaihe omine syötteineen ja tuotoksineen seuraa toista kunnes ohjelmisto on valmis.

Ohjelmistotestaus sijoittuu kaikissa ohjelmistotuotantoprosessin kuvauksissa lähes poikkeuksetta ohjelmistotuotantoprosessin loppupäähän. Valitettavasti tämä antaa hieman harhaanjohtavan kuvan siitä, kuinka ohjelmistotestausta tulisi suorittaa. Kuten myöhemmissä luvuissa tulemme toteamaan, testauksen tulisi olla mukana koko ohjelmistotuotantoprosessin ajan aina määrittelyvaiheesta prosessin loppuun saakka, jotta ohjelmiston riittävä laatu voidaan varmistaa.

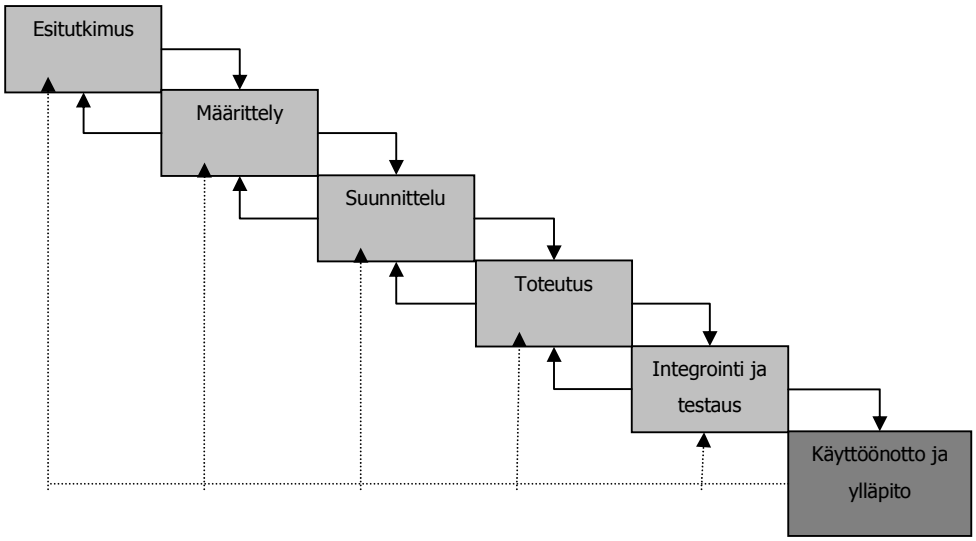
Ohjelmistokehitysprosessi on jaettu useampiin pienempiin osaprosesseihin eri teorioissa hieman eri tavoin. Ohjelmistotuotantoprosessin perusvaiheina voidaan pitää seuraavia vaiheita (Haikala & Märijärvi, 2000, 23):

1. Määrittely
2. Suunnittelu
3. Toteutus
4. Testaus.
5. Käyttöönotto ja ylläpito

Tämän perinteisen jaon mukaan testaus on ohjelmistotuotantoprosessin viimeinen vaihe, jossa tuotettu ohjelmisto testataan kun implementointityö on saatu valmiiksi. Tulevissa kappaleissa tullaan esittämään useita syitä, miksi testauksen sijoittaminen viimeiseksi tuotantoprosessin vaiheeksi aiheuttaa sekä kustannuksia, että ongelmia tuotteen laadun suhteen.

2.4.2 Ohjelmistotuotantoprosessin elinkaarimallit

Elinkaarimallilla tarkoitetaan kuvausta jolla ohjelmistotuotantoprosessia kuvataan. Ohjelmistotuotantoprosessin eri vaiheiden nimet voivat elinkaarimalleissa erota toisistaan, mutta perusrakenne on yleensä aina sama. Perinteisin ohjelmistotuotantoprosessia mallintava kuvaus on vesiputousmalli (Kuva 2, Haikala & Märijärvi 2000, 24). Vesiputousmallia on käytetty useiden elinkaarimallien perustana.



Kuva 2: Perinteinen vesiputousmalli ohjelmistotuotantoprosessin vaiheista

Vesiputousmallin lisäksi on esitetty lukuisia muita elinkaarimalleja kuvaamaan ohjelmistotuotantoprosessia ja sen eri vaiheita. Yleisimpiä kuvauksia lienevät inkrementaalista lähestymistapaa kuvaava ns. EVO-malli, ohjelmistosta rakennettaviin prototyyppeihin perustuva protoilumalli ja spiraalimalli. EVO-malli lienee käytännössä yleisin käytössä olevista malleista (Haikala & Märijärvi, 2000, 30). Tällaisen mallin soveltamisesta esimerkkinä voisi toimia esimerkiksi muutosalttiin ohjelmiston tuottaminen, jota kiivaassa tahdissa kehitetään asiakkaan tarpeisiin, siten että uutta ohjelmistoversiota aloitetaan laatimaan ehkäpä jo samaan aikaan kun edellinen versio on vielä tuotannossa. Ohjelmiston uusi versio on esimerkiksi täydentynyt ominaisuuksilla, joita edelliseen julkaisuun ei ole ehditty tehdä tai sitten asiakas on halunnut jonkun itselle tärkeän ominaisuuden pikaisesti ohjelmistoonsa.

Testauksen sijoittumisen kannalta ei ole merkityksellistä mitä elinkaarimallia käytetään, koska kaikissa eri elinkaarimallien kuvauksissa testaus yleisesti (ja lähes poikkeuksetta) sijoitetaan ohjelmistotuotantoprosessin loppupäähän. Testausmenetelmien ja testaustyökalujen kannalta käytetyllä elinkaarimallilla on paljonkin merkitystä. Esimerkiksi ohjelmistotestausta automatisoidessa sen vaatimukset on otettava huomioon jo ohjelmistoa suunniteltaessa. Riippumatta käytetystä elinkaarimallista testaus tulisi pyrkiä ottamaan mukaan ohjelmistotuotantoprosessiin mahdollisimman varhaisessa vaiheessa.. Muun muassa testauksen automatisoinnin kannalta on erittäin tärkeää, että

automatisoinnin vaatimukset huomioidaan jo ohjelmistoa suunniteltaessa. Usein testauksen automatisointi aiheuttaa toimenpiteitä liittyen muun muassa ohjelmakoodin rakenteeseen ja käytettäviin teknologioihin. Tärkeintä testauksen tuomisessa osaksi jokaista ohjelmistotuotantoprosessin vaihetta, on se että, mitä varhaisemmassa vaiheessa ohjelmistotuotantoprosessia virheet löydetään sitä pienempiä ovat ohjelmistosta löytyneiden virheiden korjauskustannukset (McGregor & Korson).

2.4.3 Testauksen merkitys ohjelmistotuotannossa

Testauksen tarkoituksesta on sanottu, että sen tarkoitus on löytää ohjelmistosta virheitä (Haikala & Märijärvi. 2000, 28). Erään toisen määritelmän mukaan testauksen tarkoitus on paljastaa maksimimäärä painotettuja virheitä annetussa ajassa (McGregor & Korson). Valitettavan usein testauksella kuitenkin käsitetään, että sen tarkoitus olisi osoittaa että ohjelmisto tai sovellus olisi toimiva tai jopa virheetön. Muun muassa tästä syystä joissakin organisaatioissa yhä edelleen ohjelmistotestauksen suorittaa jopa ohjelman implementoija alusta loppuun itse. Ongelmana on se, että jos implementoija suorittaa testauksen itse on hän helposti taipuvainen testaamaan siten, että ohjelma varmasti toimii.

Testauksen merkitystä ei useinkaan ymmärretä niin merkittävänä kuin se todellisuudessa on. Tämä voi johtua siitä että useinkaan ei tiedetä mitä kaikkea ohjelmistotestaus pitää sisällään. Testaus voidaan jakaa kolmeen kategoriaan, joissa jokaisessa testaus saa aivan uuden merkityksen.

1. Virheiden etsintä käytetystä tuotantoprosessista
2. Virheiden etsintä ei-suoritettavista (vaihe-)tuotoksista, kuten malleista, suunnitelmista, määritelmistä ja dokumenteista
3. Virheiden etsintä suoritettavasta ohjelmakoodista.

Yleisesti testauksella ymmärretään vain kohta kolme (3). Testauksella usein käsitetään vain itse testauksen suoritus eli *dynaaminen testaus.* Kuitenkin laadun kannalta paljon tärkeämpää roolia näyttelevät, sekä käytetty prosessi, että *staattiset testausmenetelmät*, joissa paneudutaan ei-suoritettavista tuotoksista löytyviin virheisiin. Esimerkkinä

staattisesta testauksesta voidaan mainita vaikkapa katselmointi. Ohjelmistotuotannossa usein katselmoidaan vaihetuotokset, joita syntyy jokaisen ohjelmistotuotantoprosessin vaiheen aikana. Usein katselmoinnin tarkoituksena on tarkastaa vaihetuotos jotta voidaan siirtyä ohjelmistotuotantoprosessin seuraavaan vaiheeseen.

2.5 Ohjelmistotestauksen tekniikat

Ohjelmistotestauksen menetelmät tarkoittavat toisiinsa liittyviä toimintoja, jotka yhdessä muodostavat testituotoksen (Lewis 2000, 22). Näitä testauksen menetelmiä on lukemattomia ja niistä voidaan muodostaa suuri määrä eri kategorioita. Näistä menetelmistä kuvataan seuraavissa kappaleissa yleisimmät.

2.5.1 Ad hoc-testaus

Ehkä yleisimmin käytetty testausmenetelmä, joka perustuu inspiraatioon, arvaukseen ja luovaan ajatteluun testauksen suunnittelun ja virheiden löytämisen välineenä. Menetelmä on hyödyllinen, jos taustalla on esimerkiksi kokemuksia ja tietoutta testattavasta tuotteesta tai jos menetelmää käytetään formaaleja testausmenetelmiä täydentävänä menetelmänä. Menetelmä on kuitenkin yksinään käytettynä erittäin epäluotettava laadunvarmistuksen väline.

2.5.2 Staattiset ja dynaamiset testausmenetelmät

Staattisilla testausmenetelmillä voidaan parantaa ohjelmiston laatua jo varhaisessa vaiheessa ohjelmistotuotantoprosessia. Staattisen testauksen menetelmien suurin etu on virheiden löytäminen vaihetuotoksista mahdollisesti jo ennen suoritettavan ohjelmakoodin kirjoittamista. Nämä staattiset testausmenetelmät ovat erittäin kustannustehokas menetelmä, jossa ohjelmistovirheitä voidaan löytää ilman varsinaisen dynaamisen testauksen suorittamista. Siksi kustannuksien ja laadun kannalta on järkevää panostaa juuri staattisiin testausmenetelmiin. Staattisista testausmenetelmistä yleisimpiä lienevät ns. koodikatselmoinnit ja määrittely/suunnittelu dokumenttien katselmoinnit. Katselmointeja on kuitenkin mahdollista suorittaa kaikille

ohjelmistotuotantoprosessin vaihetuotoksille. On erittäin suositeltavaa, että testausvaiheiden tuotokset katselmoidaan myös, jotta voidaan varmistua testauksen suunnittelun, suorittamisen ja raportoinnin laadusta. Simo Silander (1999) on jakanut katselmoinnin merkityksen ja pääkohdat seuraavasti:

- Katselmuksissa tarkastetaan vaihetuotteet
- Katselmus auttaa tuottamaan laadukasta jälkeä
- Katselmus jakaa vastuuta
- Katselmus ei ole suunnittelua varten
- Katselmuksissa opitaan (…myös virheistä)
- Kirjoitusvirheisiin ei pidä puuttua

Määrittely, suunnittelu- ja implementointivaiheiden katselmoinneissa löydetyt virheet ovat vielä edullisesti korjattavia ja siksi katselmointeja voidaan pitää kustannustehokkaana menetelmänä ohjelmiston laadun parantamiseksi. Haikala ja Märijärvi (2000, 257) toteavatkin kirjassaan: ”Tarkastusten tiedetään yleisesti olevan esimerkiksi testausta tehokkaampi virheiden etsintäkeino”. Tämä perustuu siihen, että mahdolliset virheet löydetään aikaisemmin ja niiden korjaaminen on halvempaa ohjelmistotuotantoprosessin alkupäässä. Myös Dustin, Rashka ja Paul (1999, 8) esittävät kirjassaan laskelman kuinka virheiden estäminen ja löytäminen käyttäen muun muassa katselmointeja, on tehokkaampaa ja halvempaa kuin virheiden korjaaminen.

Error Removal Cost Multiplies Over System development Life Cycle

Phase	**Cost**
Definition	$1
High-Level Design	$2
Low-Level Design	$5
Code	$10
Unit Test	$15
Integration Test	$22
System Test	$50
Post-Delivery	$100+

Taulukosta nähdään selvästi, että myöhään löydettyjen virheiden korjaaminen on huomattavasti kalliimpaa kuin ohjelmistotuotantoprosessin alkupäässä löydettyjen virheiden.. Tässä yhteydessä täytyy kuitenkin muistaa, että staattiset testausmenetelmät eivät sulje pois dynaamisen testauksen tarvetta. Staattisia testausmenetelmiä ei voida pitää ainoana laadunvarmistuksen toimenpiteenä ohjelmistotuotantoprosessissa.

Dynaamisilla testausmenetelmillä tarkoitetaan varsinaisen ajettavan ohjelmakoodin testausta suunniteltujen ja valittujen testitapausten mukaan, joiden perusteella ohjelman toimintaa arvioidaan. Testitapaukset on laadittu käyttäen määrittely-, suunnittelu- ja implementointivaiheen tuotoksia syötteenä. Dynaaminen testaus testisuunnitelman ja testitapausten mukaan jatkaa laadunvarmistusta siitä mihin katselmoinnit jäävät. Katselmointeja voi ja on suositeltavaa käyttää myös dynaamisen testauksen eri vaiheiden vaihetuotosten todentamiseen.

Dynaaminen testaus tarkoittaa testauksen suorittamista ajettavalle ohjelmakoodille. Dynaaminen testaus voidaan suorittaa sekä *manuaalisesti*, että *automatisoidusti*. Manuaalisessa testauksessa testauksen suorittaa testaaja. Testaaja suorittaa suunnitellut testitapaukset suunnitellussa järjestyksessä ja vertaa saatuja tuloksia odotettuihin. Automatisoiduissa testauksessa testauksen suorittaa ja testauksen tulokset mahdollisesti analysoi tietokone. Dynaamisella testauksella voidaan teoriassa saavuttaa täysin virheetön ohjelmisto. Tämä vaatisi kuitenkin kaikkien mahdollisten syötteiden ja niiden kombinaatioiden testaamista. Käytännössä näin suuri testauskattavuus ei koskaan toteudu, ellei kyseessä ole todella vaatimaton ohjelmisto.

2.5.3 Musta-, harmaa-, ja valkolaatikkotestaus

Mustalaatikkotestauksessa testitilanteet on luotu ohjelman tai järjestelmän toiminnan perusteella. Käytännössä tämä tarkoittaa sitä, että testaajan tulee tietää testattavan ohjelman syötteet ja tulosteet, mutta ei tarvitse olla tietoinen toiminnasta jonka ohjelma suorittaa sisällään. Mustalaatikkotestauksessa ohjelman sisäinen rakenne ja sisäinen toiminta ei ole tarkastelun kohteena vaan se, että sovellus tai järjestelmä toimii oikein.

Mustalaatikkotestauksen suuri etu on se, että testauksella todennetaan se mitä sovelluksen tai systeemin todella tulee tehdä. Mustalaatikkotestausta käytetään yleensä testausmenetelmänä testausprosessin loppupäässä toiminto- ja järjestelmätestauksessa.

Valkolaatikkotestauksella tarkoitetaan testausta, jossa testaus etenee loogisia polkuja pitkin. Testaajan tulee tuntea ohjelman sisäinen toiminnallisuus ja tilat välittämättä ohjelmalle asetetuista vaatimuksista. Esimerkkejä valkolaatikkotestausmenetelmistä ovat: ehtokattavuus ja lausekattavuus. Lewisin mukaa valkolaatikkotestauksen heikkous on se että se ei todenna vaatimusten ja määrittelyjen oikeellisuutta, vaan keskittyy ainoastaan komponentin sisäiseen logiikkaan.

Harmaalaatikkotestauksella tarkoitetaan musta- ja valkolaatikkotestauksen välimuotoa, jossa käytetään hyväksi tietoa ohjelman toteutusperiaatteista. Esimerkkinä harmaalaatikkotestauksesta on tilanne, jossa testaaja on tutustunut vaatimuksiin ja määrittelyihin ja kommunikoi implementoijan kanssa selvittääkseen komponentin sisäisen toiminnan. Tällaisesta menettelytavasta voidaan saada hyötyä muun muassa tilanteessa jossa selvitystyön tuloksena testaajalle paljastuu, että samaa toiminnallisuutta on hyödynnetty eri puolilla testattavaa sovellusta. Koska tilanne on testaajan tiedossa toiminnallisuutta ei tarvitse siten testata erikseen jokaisessa paikassa, vaan riittää että toiminnallisuus on testattu kerran jossakin osassa ohjelmistoa.

2.6 Ohjelmistotestauksen vaiheet ja tuotokset

Perinteisesti ohjelmistotestaus on jaettu vaiheisiin, joissa testataan ohjelmistoa eri vaihetuotteita hyväksikäyttäen. Erilaisia testausvaiheita on olemassa useita riippuen käytettävästä testausprosessista, mutta seuraavassa esittelen yleisimmät käytetyt ja yleisesti hyväksytyt testausvaiheet.

2.6.1 Moduulitestaus eli yksikkötestaus

Moduulitestauksessa testattavana on vain yksittäinen moduuli tai komponentti (Haikala & Märijärvi, 2000, 271). Testauksessa moduulin toimintaa verrataan moduulisuunnitteluvaiheen tuotoksiin esimerkiksi tekniseen määrittelydokumenttiin.

Moduulitestaukselle tunnusomaista on se, että moduulin rajapintoja simuloimaan joudutaan usein laatimaan niin sanottuja tynkämoduuleita (stub) ja ajureita (driver). Näillä tarkoitetaan puuttuvia ohjelmistokomponentteja simuloivia osia. Ajureilla simuloidaan testattavan komponentin palveluiden kutsuja ja tynkämoduuleilla taas simuloidaan testattavan komponentin tarvitsemia muita komponentteja. Moduulitestauksen vaihetuotteena saadaan yleensä moduulitestaussuunnitelma ja moduulitestausraportti.

Moduulitestaukselle asettaa omat vaatimuksensa ja haasteensa *oliopohjaisten ohjelmien* yleistyminen. *Proseduraalisissa ohjelmissa,* joissa ohjelman rakenne koostuu pää- ja aliohjelmista, aliohjelma on pienin testattava yksikkö, kun taas oliopohjaisissa ohjelmissa pienintä testattavaa yksikköä on huomattavasti vaikeampi rajata (Partanen 2003, 14). Koska oliopohjaiset komponentit ovat yleensä uudelleenkäytettäviä komponentteja on niiden testaukseen myös kiinnitettävä huomattavasti enemmän huomiota.

2.6.2 Integrointitestaus

Integrointitestauksessa moduulit yhdistetään toimivaksi kokonaisuudeksi valitun strategian mukaan. Testauksen tarkoituksena on testata moduulien välisien rajapintojen toimintaa, siten että voidaan varmistaa moduulien integroituvuus ja parametrien välitys (Lewis 2000, 75). Integrointitestaus tulee aloittaa vasta siinä vaiheessa kun kaikki integroitavat moduulit täyttävät moduulitestauksen hyväksymiskriteerit..

Integrointitestauksen testitapausten suunnittelu Lewisin (2000, 75): mukaan sisältää seuraavat osatekijät:

- rajapintojen tunnistaminen

- rajapintojen yhteensovittaminen
- integrointitestauksen olosuhteiden luominen
- integrointitestauksen olosuhteiden arviointi

Testituloksia verrataan yleensä tekniseen määrittelydokumentaatioon. Integrointitestauksessa ympärillä olevia luokkia simuloimaan voidaan joutua rakentamaan tynkämoduuleita ja ajureita samaan tapaan kuin moduulitestauksessa.

2.6.3 Toimintotestaus

Toimintotestauksen tarkoituksena on testata uudet tai muuttuneet ominaisuudet käyttäjän näkökulmasta (Silander 1999, 18). Toimintotestauksessa yksittäiseen ominaisuuteen liittyvät komponentit on integroitu toimivaksi kokonaisuudeksi. Ominaisuudet testataan vaatimusmäärittelyjä vasten. Joissakin testauksen prosessimalleissa toimintotestaus nähdään omana testausvaiheenaan integrointitestauksen ja järjestelmätestauksen välissä. Joissakin testausprosesseissa toimintotestauksella käsitetään järjestelmätestauksen toiminnallinen osa.

2.6.4 Järjestelmätestaus

Järjestelmätestauksen tarkoituksena on testata koko järjestelmän toimintaa. Järjestelmätestaus yleensä jakautuu toiminnalliseen (functional) ja ei-toiminnalliseen (non-functional) testaukseen. Funktionaalisen testauksen tarkoitus on testata tuote määrittelyjä ja vaatimuksia vasten toiminnallisuuden suhteen. Ei-toiminnallisella testauksella taas tarkoitetaan muun muassa suorituskyvyn (performance), luotettavuuden (reliability), yhteensopivuuden (compatibility), vikatilanteista toipumisen (recovery) ja asennuksen (installation) testausta. Järjestelmä- eli systeemitestauksen osa-alueita on lähteestä riippuen useita ja niiden valinta riippuu testattavasta sovelluksesta.

2.6.5 Hyväksymistestaus

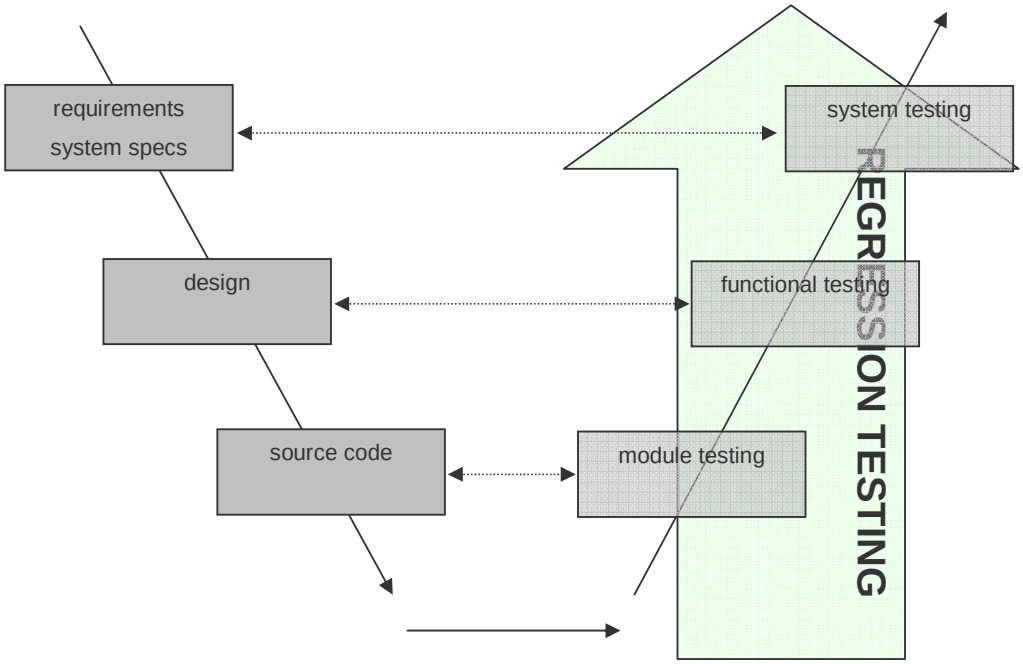
Hyväksymistestauksella tarkoitetaan loppukäyttäjän suorittamaa testausta, jossa arvioidaan tuotteen kykyä vastata sille annettuihin asiakasvaatimuksiin. Hyväksymistestaus ei ole mukana kaikissa testausta kuvaavissa esityksissä ja malleissa. Siksi testausvaihetta ei käsitellä tarkemmin myöskään tässä.

2.7 Ohjelmistotestauksen mallintaminen

Ohjelmistotuotannossa ohjelmistotestaus yleensä irrotetaan omaksi erilliseksi prosessikseen. Testaus vaatii oman tarkoin märitellyn prosessin muun muassa seuraavista syistä (Juha Taina, 2004):

- Huolellinen testaus vie 50 % tai enemmän käytettävissä olevista resursseista
- Vain äärimmäisen harvoin laadukas ohjelmisto voidaan toteuttaa ilman systemaattista testausta

Niin kuin kaikissa muissakin prosessikuvauksissa, myös testausprosesseista on olemassa lukemattomia variaatioita, jotka kuitenkin noudattava pääasiassa samaa pohjamallia. Käytetty kuvaustapa on yleisesti ns. V-malli (kuva 3), jossa määrittely ja suunnittelu ovat vasemmassa haarassa ja testaus oikeassa haarassa.



Kuva 3: V-malli mukaillen ISEB kurssimateriaalia

V-mallin tarkoituksena on kuvata ohjelmistotuotantoprosessin vaiheiden välisiä riippuvuuksia siten, että määrittely- ja suunnitteluvaiheet ovat kiinteästi yhdistetty niitä vastaaviin testausvaiheisiin. Seuraavassa käsitellään jokainen määrittely- ja suunnitteluvaihe ja niitä vastaava testausvaihe.

- Määrittely – Järjestelmätestaus: Järjestelmätestauksen tarkoituksena on varmistaa järjestelmän toimivuus vaatimusmäärittelyjä vasten.
- Suunnittelu – Toimintotestaus: Toimintotestauksen tarkoitus on testata ohjelmiston toiminnallisuus käyttäen hyväksi määrittely ja suunnitteluvaiheen dokumentteja
- Lähdekoodi – Moduulitestaus: Moduulitestauksessa testitapaukset suunnitellaan ohjelmakoodia vasten.

Käytännössä nämä riippuvuudet eivät ole näin yksinkertaisia vaan ohjelmistotestausta suunniteltaessa joudutaan käyttämään useita eri lähteitä määrittely-, suunnittelu- ja implementointivaiheista. V-malli antaa kuitenkin yksinkertaistetun yleiskuvan ohjelmistotuotantoprosessin vaiheiden välisistä riippuvuuksista.

3. Perinteinen ohjelmistotestausprosessi

Tässä luvussa on tarkoitus esitellä perinteinen ohjelmistotestausprosessi, joka on keskittynyt pääasiassa dynaamiseen testaukseen ohjelmiston laadunvarmistamisen keinona. Myöhemmissä luvuissa tätä testausprosessia tullaan tarkentamaan ja kehittämään, kohti tehokkaampaa ja laadukkaampaa testausprosessia.

Perinteinen tai primitiivinen testausprosessi voidaan käsittää siten, että testaus käsitetään pääasiassa dynaamisena testauksen suorittamisena ajettavalle ohjelmakoodille. Tämä testaus suoritetaan vain hieman ennen kuin ohjelmisto siirretään tuotantoon ja käyttöön. Tällaisessa primitiivisessä testauksessa monet asiat perustuvat intuitioon ja oletuksiin, käyttäen Ad hoc- testausmenetelmää. Vaikka tämäntyyppisen testausprosessin heikkoudet ovat yleisesti tiedossa, silti primitiivisiä testausmenetelmiä käytetään paljon vielä tänäkin päivänä, jopa ainoana laadunvarmistuksen menetelmänä.

3.1 Testauksen suunnittelu

Testauksen suunnittelun tarkoituksena on nimensä mukaisesti laatia suunnitelma, jonka perusteella testaus voidaan viedä organisoidusti läpi. Lewis (2000, 119) toteaa kirjassaan: ”If a test plan is comprehensive and carefully thought out, test execution and analysis should proceed smoothly”.

Jokaisesta ohjelmistotestauksen vaiheesta tulisi tehdä oma testaussuunnitelma, jossa käydään läpi suoritettava testaus kokonaisuutena. Testaussuunnitelmasta tulee käydä ilmi Haikalan ja Märijärven (2000, 281) mukaan muun muassa seuraavat asiat:

- Mitä testejä suoritetaan?
- Milloin testit suoritetaan?
- Miten testaus järjestetään?
- Millaisia lopputuloksia odotetaan?

Haikalan ja Märijärven testaussuunnitelman sisällöstä kuitenkin puuttuu eräs tärkeimmistä asioista, jonka testaussuunnitelman kuuluisi pitää sisällään ja joka tekee

heidän esittämästään testaussuunnitelmasta vajanaisen. Se on testauksen lopettamiskriteerien asettaminen. Testaussuunnitelmassa täytyy olla selkeät kriteerit sille milloin testaus on saavuttanut riittävän pisteensä. Tähän aiheeseen palataan myöhemmissä kappaleissa. Haikalan ja Märijärven teos yksinkertaistaa testauksen suunnittelua myös muilta osin. Haikala ja Märijärvi (2000, 281) väittävät kirjassaan, että moduulitestausta suunniteltaessa testaussuunnitelman korvaavat yrityksen laatukäsikirjasta löytyvät ohjeistukset. Tämä väite on ristiriidassa usean muun lähteen kanssa. Testattavat komponentit tai moduulit voivat olla hyvinkin erilaisia ja yleispätevän ohjeistuksen laatiminen laatukäsikirjassa näiden testaamiselle lienee mahdoton urakka. Kaikille testausvaiheille tulisi rakentaa oma testisuunnitelmansa, koska eri testausvaiheissa testaus suoritetaan eri vaihetuotoksia vasten. Näiden asioiden perusteella voimme pitää Haikalan ja Märijärven esimerkkiä testaussuunnittelusta esimerkkinä liian yksinkertaistetusta mallista suorittaa laadukasta ohjelmistotestausta.

Konkreettisemmin ja tarkemmin testaussuunnitelman sisällön määrittelevät Dustin, Rashka & Paul (1999, 192-197). Heidän mukaansa testaussuunnitelman tulisi sisältää Haikalan ja Märijärven luettelemien asioiden lisäksi seuraavat asiat:

- testaushenkilöiden roolit ja vastuut
- projektin aikataulu
- testauksen riskit
- testauksen hyväksyttävä kattavuus.

Nämä ovat niin sanotut perusvaatimukset testaussuunnitelman sisällölle. Kuitenkin testaussuunnitelma voi pitää sisällään testausvaiheesta ja testattavasta tuotteesta riippuen paljon muutakin informaatiota. Testaussuunnitelma voi ja sen tuleekin sisältää kaikki oleellinen informaatio suoritettavaan testausvaiheeseen ja testaukseen liittyen.

Primitiivisessä testausprosessissa testaus suoritetaan yleensä lyhyessä ajassa ja erittäin vähällä määrällä testausresursseja. Tämä ei useinkaan mahdollista kunnollisten testaussuunnitelmien tekemistä. Usein myös testitapaukset eivät primitiivisessä testausprosessissa ole kovinkaan geneerisiä tai uudelleenkäytettäviä. Testausta suunniteltaessa on kuitenkin hyvä pitää mielessä, että hyväkään testaussuunnitelma ei

voi paikata huonon ohjelmistotuotantoprosessin puutteita vaan loputtomat testaamisen ja korjauksen kierrokset heikentävät tuotteen laatua ja lisäävät kustannuksia.

3.2 Testauksen suorittaminen

Testauksen suoritusvaiheessa aikaisemmin suunnitellut testitapaukset suoritetaan suunnitelman mukaisessa järjestyksessä ja testausympäristössä. Testitapauksia suoritettaessa tulee huomioida, että kaikkia tarkasteltavia asioita ei useinkaan voida tai kannata kirjoittaa testitapaukseen. Testausta suoritettaessa testaajan ammattitaito korostuu, jotta hän osaa havainnoida mahdollisia ongelmia myös muualta kuin juuri testattavalta osa-alueelta.

Yksittäisen testitapauksen suoritus voi olla manuaalista tai automatisoitua. Testauksen automatisointi voi kuitenkin helposti olla isompi operaatio kuin itse testauksen manuaalinen suorittaminen. Testauksen automatisointia ei kannata lähteä viemään läpi lyhytnäköisesti suurten kustannussäästöjen toivossa. Karkeasti ottaen voidaan sanoa, että moduulitestaus on helpoimmin automatisoitavissa oleva testausvaihe.

3.3 Testauksen raportointi

Testausraporttiin tulisi raportoida kaikki löytyneet virheet. Samoin virheet tulisi analysoida ja purkaa auki siten, että lukijalle kerrotaan muun muassa: virheen kuvaus, virheen vakavuus, milloin virhe löydettiin, miten se olisi voitu löytää aikaisemmin, milloin virhe ilmeni ja miten virhe olisi voitu estää. Primitiivisessä testausprosessissa luonnollisesti testausraportin sisältöön ei kuulu muuta kuin testauksen status.

3.4 Ohjelmistotestauksen riittävyys

Primitiivisessä ohjelmistotestausprosessissa testauksen riittävyys perustuu usein olettamuksiin. Tällainen olettamus testattavan tuotteen laadukkuudesta voi olla vaikkapa se, että virheitä ei ole testauksesta huolimatta löydetty hetkeen (Koomen And

Pol 1999, 17). Ohjelmistotestauksen riittävyyden arviointi on erittäin vaativa tehtävä ja usein primitiivisissä testausprojekteissa näiden lopetuskriteerien miettimiseen ei käytetä tarpeeksi aikaa. Kehittyneemmistä ohjelmistotestauksen lopettamiskriteereistä lisää myöhemmissä luvuissa.

3.5 Ohjelmistotestauksen kustannukset primitiivisessä testausprosessissa

Primitiivisen testausprosessin kustannukset yleensä pyritään pitämään kurissa käyttämällä testaukseen mahdollisimman vähän aikaa ja henkilöresursseja. Vaikka testaus olisikin suunniteltu hyvin, usein staattisten testausmenetelmien puuttuminen aiheuttaa testauksen suorittamisvaiheessa kierteen jossa korjaus ja testaus kierrokset toistuvat vuoronperään. Tämä on kallista ja aiheuttaa erittäin paljon regressiotestauksen tarvetta, jota ei lyhyessä ajassa voida suorittaa. Regressiotestauksella tarkoitetaan testausta, jolla todennetaan korjauksen toimivuus ja varmistetaan, että testattavan ohjelmiston muuttumattomiin osiin ei ole korjauksen myötä luotu uusia vikoja (Lewis 2000, 283). Tästä voidaan päätellä, että tuotteeseen jää dynaamisesta testauksesta huolimatta paljon virheitä, jotka ovat generoituneet muun muassa löydettyjä virheitä korjattaessa. Kuten aikaisemmin todettu lopputuotteeseen asti päässeet virheet ovat juuri kalleimpia virheitä korjata.

Primitiivisen testausprosessin tarkoituksena on usein pitää kurissa testauksen kustannukset, jotka näyttelevät merkittävää osaa koko ohjelmistotuotantoprosessin kustannuksista. Joidenkin arvioiden mukaan testauksen resurssitarve voi olla jopa 50 % ohjelmistoprojektin resursseista. Vaikka toisin luullaan, primitiivisen testausprosessin käyttö kuitenkin päinvastoin lisää kustannuksia. Niin kuin useaan kertaan on jo todettu, myöhään löydeltyjen virheiden korjaaminen luultavasti tulee kalliimmaksi kuin kunnolliseen ja kehittyneeseen testausprosessiin panostaminen. Testauksen kustannuksia arvioitaessa on myös muistettava, että suuret määrät löydettyjä virheitä ja isot resurssitarpeet usein kertovat koko ohjelmistotuotantoprosessin laadun heikkoudesta. Kuitenkin usein suuret kustannukset mielletään testauksen kustannuksiksi

joista päästään helposti eroon karsimalla testausta. Tämä on kuitenkin lyhytnäköistä koska ohjelmistotestauksen kustannuksiin tulisi laskea mukaan myös löytymättömien virheiden aiheuttamat kustannukset.

4. Testausprosessin uudistaminen ja kehittäminen

Ennen syventymistä testausprosessin uudistamiseen tarkemmin on hyvä määritellä *testausprosessin kehittäminen* käsitteenä. Koomen ja Pol (1999, 19) määrittelevät testausprosessin kehittämisen seuraavasti:

> Optimizing the quality, costs, and lead time of the test process, in relation to the total information services.

4.1 Uudistamisen aika: Milloin uudistetaan?

Testausprosessin uudistamisen syyt ovat osittain samoja kuin mitkä ylipäätään ajavat laatimaan testausprosessia. Testausprosessin uudistuksella yleensä haetaan parannusta ohjelmistotuotteen laatuun, testauksen kestoon ja testauksesta aiheutuviin kustannuksiin.

Testausprosessin uudistamisessa tulee huomioida, että useinkaan pelkästään testausprosessin uudistaminen ei ole ratkaisu koko ohjelmistotuotantoprosessin ja ohjelmistotuotteen laadun ongelmiin. Myös ohjelmistotuotantoprosessin muut osa-alueet tulisi olla samalla tasolla testausprosessin kanssa. Tässä valossa tarkasteltuna on loogisempaa aloittaa prosessiuudistus ohjelmistotuotantoprosessin alkupäästä. Pelkästään laadukas testaus ei voi kompensoida muuten huonolaatuisen ohjelmistotuotantoprosessin ongelmia.

4.1.1 Laatujärjestelmät prosessiuudistuksen laukaisijana

Ohjelmistotuotteen laadunvarmistukseen liittyy olennaisena osana laatujärjestelmät, jotka on laadittu turvaamaan ohjelmistotuotteen laadukkuus. Monesti juuri laatujärjestelmän tavoitteet toimivat lähtölaukauksena myös testausprosessin kehittämiseen ja parantamiseen. Usein kuitenkin laatujärjestelmät määräävät ainoastaan sen, mitkä prosessit täytyy olla määritelty, ei sitä kuinka ne toteutetaan (Heikkilä, 2003). Näiden laatujärjestelmien tarkoituksena on antaa raamit toiminnalle ja

mahdollistaa prosessien analysointi ja kehittäminen, myös testausprosessin osalta. Laatujärjestelmään liittyvät olennaisena osana kehittämismallit, joissa usein on olemassa erilaisia tasoja. Jokaisella tasolla vaaditaan yrityksen toiminnalta enemmän kypsyyttä kuin edellisellä. Testaukseen olennaisesti liittyvä malli on CMMi (Capability Maturity Model Integration), joka määrittelee organisaation prosessien kypsyystason. Usein tietyn kypsyystason vaatimukset liittyvät ohjelmistoalan eri standardeihin ja niiden saavuttamiseen. Esimerkkinä käytetyssä CMMi mallissa on viisi tasoa, jotka jokainen asettavat omat vaatimuksensa yrityksen prosessien laadukkuudelle ja kypsyydelle. Testausprosessin ja testausmenetelmien kehittäminen onkin usein laatujärjestelmän aikaansaama tapahtuma.

4.1.2 Nykyisen testausprosessin tila ja ohjelmiston laatu uudistustarpeen aiheuttajana

Toinen laukaiseva tekijä testausprosessin uudistamiselle voi olla, että tekeminen ja käytäntö ovat ajautuneet kauas käytetystä prosessista ja sen ohjeista. Tässä tilanteessa nykyinen prosessi on menettänyt merkityksensä ja käytäntö on ohjannut tekemistä, tosin ei välttämättä parempaan suuntaan. Tällaisessa tapauksessa testaukselle on rakennettava uudet raamit, jonka puitteissa toimia ja joka sopii paremmin nykyisen tekemisen tarpeisiin. Prosessia muutettaessa ei ole kuitenkaan syytä välttämättä pyrkiä noudattamaan käytännön ohjaamaa mallia vaan todella pyrkiä kehittämään parhaat menetelmät ohjelmistotuotteen laadun parantamiseksi asiakkaan näkökulmasta.

Kolmas tekijä voi olla laadulliset ongelmat tuotteessa, jotka aiheuttavat paineita nykyisen testausprosessin kehittämiselle. Kuitenkin suuret laadulliset ongelmat ohjelmistotuotteessa ovat harvoin ainoastaan huonon testausprosessin aiheuttamia.

Laadullisia hyötyjä tavoitellessa on kuitenkin syytä selvittä mistä tuotteen huono laatu johtuu. Koomen & Pol (1999, 21) esittävät erilaisia syitä tuotteen huonolle laadulle. Kyse voi olla siitä yksinkertaisesta syystä, että testaajat eivät ole tehneet hyvää työtä tai eivät ole tarpeeksi ammattitaitoisia tai siitä syystä että organisaatio on ottanut tietoisen riskin tekemällä testausta minimi resurssein mahdollisimman lyhyessä ajassa.

Nykypäivänä ohjelmistoyritysten tärkeä kilpailutekijä on ehtiä markkinoille mahdollisimman nopeasti ja näin ollen myös ohjelmistotestaukselle voi jäädä kovin vähän aikaa (Koomen & Pol 1999, 18). Usein ohjelmistotuotantoprosessin alkupään toimintojen venyessä määräajasta jolloin projekti on lopetettava joudutaan silti pitämään kiinni. Tämä aiheuttaa sen, että testausta tehdään ainoastaan se aika kuin on mahdollista. Molemmat tavat vaativat testausprosessilta erilaisia toimia ongelman korjaamiseksi. Esimerkiksi testitapausten tarkalla priorisoinnilla testauksen suunnitteluvaiheessa voidaan varmistaa, että ohjelmiston toiminnan kannalta kriittisin toiminnallisuus on testattu tehokkaasti riippumatta käytettävästä ajasta.

4.1.3 Nykyisen testausprosessin arviointi

Jotta uudistyötä kannattaa tai on ylipäätään mahdollista lähteä tekemään, tulee tietää nykyisen testausprosessin tila. Nykyisen testausprosessin laadukkuutta ja tehokkuutta voidaan arvioida muun muassa tutkimalla organisaatiota ja sen seuraavia ominaisuuksia (Perry 2000, 13):

- Testauksen suunnittelu
- Testauksen laadun valvonta
- Johdon tuki testaukselle
- Testausprosessi
- Testityökalut
- Testauksen koulutus
- Testauksen tehokkuus
- Käyttäjän tyytyväisyys testaukseen

Nämä ovat ominaisuuksia joita Perry pitää testauksen kannalta kriittisinä ominaisuuksina.

On olemassa myös muita esimerkkejä siitä mitkä asiat ovat kriittisiä ja tärkeitä testausprosessin laadun ja tehokkuuden kannalta. Quality Assurance Institute (QAI) on tutkinut maailmanluokan testausorganisaation ominaisuuksia. He ovat löytäneet

kahdeksan asiaa, jotka ovat maailmanluokan testausorganisaation tunnusmerkkejä. Tunnusmerkit ovat:

- Testauksen suunnittelu
- Testaajien kouluttaminen
- Johdon tuki testaukselle
- Käyttäjän tyytyväisyys testaukseen
- Testausprosessien käyttö
- Tehokkaiden testausmenetelmien käyttö
- Testityökalujen käyttö
- Laadunvalvonta testausprosessissa

Vertaamalla omaa testausprosessiaan ja sen ominaisuuksia näihin kahdeksaan tunnusmerkkiin on mahdollista saada käsitys oman testausprosessin kypsyydestä.

Kun päätös testausprosessin uudistamisesta on tehty, on syytä selvittää prosessiuudistuksen vaatima työmäärä ja resurssit. Perry (2000, 8-9) esittelee kirjassaan kolmivaiheisen tavan arvioida testausprosessin uudistamisen työmäärää:

- Vaihe 1: Nykyisen testauksen ja testaajien kyvykkyyden arviointi
- Vaihe 2: Aseta parannustavoitteet, sekä prosessille, että testaajille
- Vaihe 3: Laadi suunnitelma tavoitteisiin pääsemiseksi

Vaiheessa yksi on tunnistetaan nykyisen testausprosessin ja testaajien vahvuudet ja heikkoudet aikaisemmin lueteltuja ominaisuuksia tutkimalla. Vaiheessa kaksi asetetaan tavoitteet testausprosessin parantamiselle ja kehittämiselle. Tämän jälkeen koko uudistusprosessi suunnitellaan tarkasti.

4.2 Mitä kannattaa uudistaa?

Testausprosessin uudistamiskohteet tai uudistamisen laajuus riippuu tietenkin sen hetkisen prosessin kypsyydestä ja laadusta. Yleisesti voidaan sanoa, että nykyisestä testausprosessista kannattaa pyrkiä löytämään heikoimmat kohdat ja uudistaa tekemistä osa kerrallaan, siten että ohjelmistotestauksen heikoimmat osat uudistetaan ensin. Koko

käytetyn prosessin uudistaminen kerralla ei välttämättä tuo toivottua tulosta ja voi osoittautua liian isoksi urakaksi, sekä ajallisesti että resurssimielessä. On syytä pitää mielessä, että suurin laadullinen ja kustannuksellinen hyöty saadaan *estäviä testaustoimenpiteitä* kehittämällä. Tämä tarkoittaa virheiden syntymisen estämistä käyttämällä staattisia testausmenetelmiä. Näitä menetelmiä käyttäen virheet voidaan löytää aiemmin jo ohjelmistotuotantoprosessin alkuvaiheessa. Olisi kuitenkin liian rohkeaa väittää, että staattiset testausmenetelmät olisivat kaivattu ”hopealuoti” testauksen ongelmien ratkaisemiseksi. Voidaan kuitenkin sanoa, että useassa organisaatiossa staattisia testausmenetelmiä ei käytetä siinä mittakaavassa kuin laadullisessa ja kustannuksellisessa mielessä olisi järkevää.

Myös dynaamisia testausmenetelmiä voidaan kehittää. Testitapausten suunnittelua voidaan tehostaa ja testauksen suorittamisen avuksi voidaan ottaa apuun esimerkiksi testauksen automatisointityökaluja. Testauksen laajamittainen automatisointi vaatii kuitenkin koko ohjelmistotuotantoprosessin tuen. Ohjelmisto tulee alusta asti suunnitella testauksen automatisointia silmälläpitäen.

Myös testaushenkilöiden kompetenssia voidaan pitää kriittisenä osa-alueena testauksen laadukkuuden kannalta. Usein dynaamisen testauksen heikon laadun taustalla voi olla se, että testauksessa käytetään henkilöitä, jotka sillä hetkellä sattuvat olemaan saatavilla. Testauskompetenssia voidaan kehittää muun muassa koulutuksella.

4.3 Uudistamisprosessin vaiheet

Testausprosessin uudistaminen voidaan jakaa karkeasti neljään vaiheeseen (Kooman ja Pol, 1999):

- Parannuskohteiden kartoittaminen
- Nykyisen tilan havainnointi
- Tavoitetilan asettaminen
- Muutoksien toteuttaminen

Testausprosessi uudistus tulee aloittaa määrittelemällä ja kartoittamalla parannuskohteet ja testausprosessin nykytila. Näissä kahdessa ensimmäisessä vaiheessa on mietittävä se mitä ohjelmistotestauksen haluttaisiin olevan ja kuinka testauksen tulisi parantua. Tulisiko testauksen olla esimerkiksi nopeampaa tai halvempaa? Näiden vaiheiden jälkeen on mahdollista määritellä tavoitetila johon prosessiuudistuksella pyritään. Tässä vaiheessa on syytä muistaa, että parhaaseen tulokseen ei välttämättä päästä tekemällä liian suuria muutoksia kerralla. Toisaalta joidenkin muutosten tekeminen testausprosessissa saattaa vaatia muutoksia toisaalla ohjelmistotuotantoprosessissa. Esimerkkinä tästä voi olla vaikkapa testauksen automatisoinnin laajamittainen käyttöönotto. Viimeisenä vaiheena prosessiuudistuksessa on varsinaisten muutoksien toteuttaminen suunnitelman mukaan.

4.4 Kehittyneen testausprosessin hyödyt

Jotta testausprosessiuudistus olisi mielekästä on tehtävistä muutoksista saatava, joitakin konkreettisia hyötyjä. Nämä tavoiteltavat hyödyt liittyvät useimmiten kustannuksiin, laatuun tai aikaan.

4.4.1 Kustannushyödyt

Testauksen kehittämisellä ja parantamisella tähdätään usein taloudellisiin säästöihin, jotka aiheutuvat testattavan tuotteen laadun paranemisesta tai testausmenetelmien tehostumisesta siten, että tavoiteltavaan laatuun päästään pienemmin resurssein.

Koomen ja Pol (1999, 21) esittävät mielenkiintoisen ajatuksen siitä kuinka laadukas testausprosessi aiheuttaa huomattavia lisäkustannuksia, jos muut ohjelmistotuotantoprosessin vaiheet eivät ole samalla kypsyystasolla. Tehostunut testausprosessi löytää enemmän vikoja ja osoittaa ohjelmiston heikon laadun, tämä taas aiheuttaa enemmän korjauksia ja mahdollisia generoituneita uusia vikoja. Näin ollen kokonaiskustannus testauksen ja korjauksen kierroksien myötä kasvaa paljon, mutta tuotteen laatu paranee hitaasti. Juuri siksi on tärkeää, että ohjelmistotuotantoprosessin osa-alueet ovat suurin piirtein samalla kypsyystasolla.

4.4.2 Laadulliset hyödyt

Testausprosessin kehittämisessä pyritään usein laadullisiin hyötyihin. Testausprosessia kehittämällä on tarkoituksena varmistaa tuotteen parempi laatu. Testausprosessin tulisi testausmenetelmistä riippumatta pystyä varmistamaan, että ohjelmisto toimii asiakkaan vaatimusten ja spesifikaatioiden mukaisesti.

4.4.3 Ajalliset hyödyt

Ajalliset hyödyt ovat eräs yleisimmistä tavoitteista testausprosessia kehitettäessä. Tässäkin tulee huomioida testauksen nykytila. Jos testaus vie paljon aikaa voi syynä olla se, että testausorganisaatio ei ole tarpeeksi tehokas tai vaihtoehtoisesti tuote tulee keskeneräisenä ja huonolaatuisena testaukseen jolloin tuotteen laatua on myöhäistä parantaa (Koomen & Pol 1999, 21). Jos vika on testausorganisaatiossa voi syynä olla esimerkiksi testaajien motivaatio-ongelmat, testauskompetenssin puute tai testausmenetelmien heikkous. Jos vika taas on ohjelmistotuotantoprosessin alkupäässä eli ohjelmistotuotteen laadussa voidaan testauksen suorittamiseen käytettyä aikaa vähentää muun muassa tehostamalla staattisia testaus menetelmiä, jolloin virheet löydetään jo varhaisessa vaiheessa.

4.5 Testausprosessiuudistuksen riskit

Testausprosessin uudistamisessa on olemassa aina riskinsä. Voi olla, että uusi menetelmä ei ole tehokas, tai on mahdollista, että testaajat eivät sitoudu uuden ohjelmistotestausprosessin käyttöön. Nämä kaikki tekijät on huomioitava testausprosessia suunniteltaessa.

Testausprosessiuudistuksen riskiksi voi helposti muodostua myös testausta suorittavat ihmiset. Laadukkaan testausprosessin aikaansaamiseksi on testauksen suorittajien oltava ammattitaitoisia ja psykologisesti testaukseen sopivia. Psykologinen sopivuus

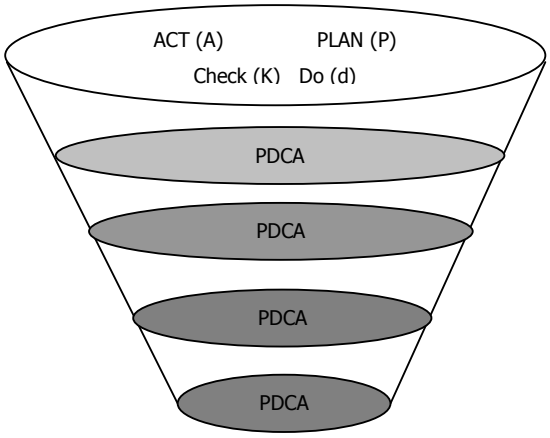
testaukseen tarkoittaa Myersin (1979) mukaan sitä, että testaaja yrittää osoittaa toteen ohjelmiston riittämättömän laadun, eikä sitä että järjestelmä on hyvä ja laadukas.

4.6 Testausprosessiuudistuksen menetelmät

Testausprosessin uudistamiseen on olemassa lukemattomia teorioita. Tässä esitellään kuitenkin kolme teoriaa, joita on käytetty hyväksi laadittaessa seuraavassa luvussa esiteltävää kehittyneempää testausprosessia.

4.6.1 PDCA-malli

Testauksen parantamisen yhtenä lähtökohtana on käytetty Lewisin (2000, 36-38) esittelemää PDCA mallia, jossa Deming käyttää laatuympyrää (kuva 5), periaatteita ja staattisia tekniikoita. Malli perustuu suunnitteluun (plan), tekemiseen (do), tarkistamiseen (check) ja toimintaan (act). Tässä mallissa kehitystyö noudattaa kiertävää kehitystä tai vesiputousmallia (kuva 4.) Mallissa testaus on jatkuva prosessi, joka alkaa erittäin varhaisessa vaiheessa ohjelmistotuotantoprosessia ja on mukana loppuun saakka. Testaus on siis kiinteästi integroitu sovelluskehitykseen. Pohjimmiltaan Lewisin esittämä malli perustuu perinteiseen vesiputousmalliin ja siitä kehitettyyn spiraalimalliin.



Kuva 4: Lewisin PDCA malli

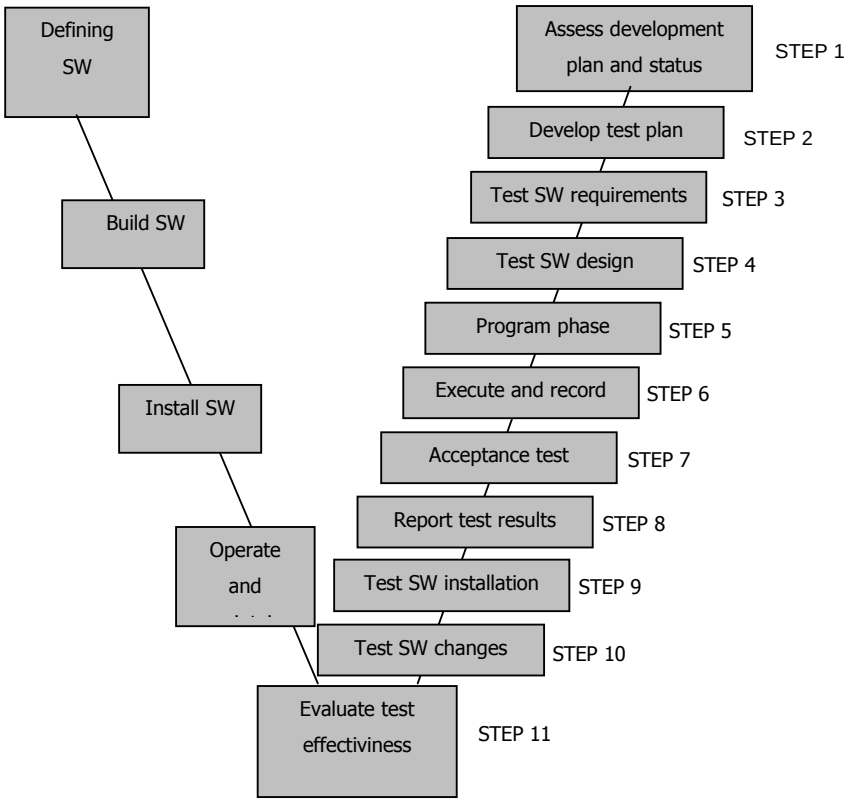
Mallissaan Lewis käyttää erilaisia testausmenetelmiä verifioidakseen vaatimusmäärittely-, suunnittelu- ja implementointivaiheet. Tässä mallissa korostuu teknisten katselmointien merkitys laadunvarmistuksessa määrittelystä implementointiin.

4.6.2 Quality Assurance Instituten 11-kohdan malli

Toisena mallina on käytetty Quality Assurance Instituten henkilöstön kokemuksiin perustuvaa 11-askeleen mallia (Perry, 2000, 171), jossa siinäkin on pohjana elinkaarimalli, jossa testaus on jatkuvana osana ohjelmistokehitystä. 11-askeleen mallin ajatuksena on ohjelmistotestauksen kustannusten vähentäminen. Kustannusten vähentäminen ei mallin mukaan suinkaan tarkoita ohjelmistotestauksen vähentämistä vaan jopa päinvastoin. Perryn mukaan ohjelmistotestauksen kustannukset sisältävät itse testauksen kustannuksien lisäksi testauksessa löytymättömien vikojen aiheuttamat kustannukset.

11-askeleen mallissa ohjelmistotestausta suoritetaan ohjelmistotuotantoprosessin kanssa samaan aikaan. Tämä tarkoittaa sitä että testauksen suunnittelu aloitetaan samaan aikaan kehitystyön kanssa.

Kuva 5. kuvaa 11-askeleen mallin eri vaiheet. Vaiheissa 1-5 todentamista käytetään oikeellisuuden arviointimenetelmänä. Vaiheesta 5 eteenpäin kelpoistamista käytetään oikeellisuuden todentamiseen..



Kuva 5: 11-askeleen testausprosessi mukaillen Perryn (2000) mallia

4.6.3 TPI-malli

Kehittynyttä testausprosessia laatiessa on myös käytetty hyväksi Koomanin ja Polin (1999) *Test Process Improvement*-mallia (TPI), joka käytännönläheisesti ja esimerkkien kautta esittelee mallin testausprosessin kehittämiseksi ja parantamiseksi. TPI-mallin vahvuudeksi todetaankin se, että toisin kuin muut testausprosessin kehittämiseen tarkoitetut mallit kuten CMM, SPICE ja Bootstrap, TPI-malli on ainoa joka tarjoaa yksityiskohtaisen askel-askeleelta etenevän mallin kohti täydellisempää testausprosessia. Lukijan kannalta hyvää TPI-mallissa on konkreettiset parannusehdotukset testauksen eri vaiheisiin.

TPI-mallissa analysoidaan nykyinen testausprosessi, tarkoituksena löytää sen heikkoudet ja vahvuudet käyttäen *Testauksen Kypsyys Matriisia* (Test Maturity Matrix). Tämän perusteella laaditaan tarkistuspisteet ja kehittämisehdotukset (taulukko).

Esimerkki testauksen kypsyys matriisista mukaillen Koomanin ja Polin (1999) materiaalia

SCALE/KEY AREAS	***CONTROLLED***	***EFFICIENT***	***OPTIMIZING***
TEST STRATEGY	**X**		
TEST TOOLS			***X***
ETC.			

5. Kehittynyt testausprosessi

Aikaisemmin esitellyn perinteisen testausprosessin pohjalta esitellään kehittyneempi testausprosessi, jossa on huomioitu paremmin ja tehokkaammin testauksen laatuun ja kustannuksiin vaikuttavia tekijöitä. Kappaleessa käsitellään kehittyneen testausprosessin lähtökohdat ja kuinka testaus tuodaan osaksi koko ohjelmistotuotantoprosessia.

5.1 Kehittyneen testausprosessin lähtökohdat

Perinteistä testausprosessia on pyritty muuttamaan ja kehittämään painottamalla staattisten testausmenetelmien merkitystä. Seuraavissa kappaleissa selvitetään kehittyneen testausprosessin lähtökohdat ja kehityskohteet verrattuna perinteiseen testausprosessiin.

5.1.1 Mitä kehitetään?

Testausprosessin kehittämisessä pääpaino kannattaa kohdistaa asioihin, joista saadaan suurin laadullinen hyöty verrattuna käytettävissä oleviin resursseihin. Kirjallisuuslähteiden perusteella staattisia testausmenetelmiä kehittämällä saadaan suurimpia laadullisia ja kustannuksellisia hyötyjä, koska näin mahdolliset virheet löydetään jo varhaisessa vaiheessa ohjelmistotuotantoprosessia. Aikaisemmin esiteltyä primitiivistä testausprosessia on pyritty parantamaan muun muassa siten, että perinteiseen prosessiin verrattuna testaus on tuotu mukaan ohjelmistotuotannon elinkaaren jokaiseen vaiheeseen, vaihetuotteiden staattisen testauksen kautta.

Dynaamisia testausmenetelmiä kehitettäessä laadun radikaali parantaminen ja keinojen löytäminen ei ole niin yksinkertaista. Primitiiviseen testausprosessiin verrattuna, kehittyneessä testausprosessissa korostuu testauksen suunnitelmallisuus, mittaaminen ja testauksen lopettamisen kriteerit.

Laadukkaankaan ohjelmistotestausprosessin pohjalta ei voida tehdä tehokasta, laadukasta ohjelmistotestausta lyhyessä ajassa, jos testausprojektin henkilöstö ei ole

motivoitunutta ja sitoutunutta. Joissakin tapauksissa suurimmat laadulliset ja taloudelliset parannukset ohjelmistotestaukseen saadaan henkilöstön hyvinvointia kehittämällä. Vaikka työntekijöiden motivointiin ja sitoutumiseen liittyvät asiat vaikuttavatkin suuresti lopputuotteen laatuun, ei näitä asioita käsitellä tämän tutkielman piirissä.

5.1.2 Käytetyt teoriat

Kehittynyttä ohjelmistotestausprosessia laatiessa on käytetty erittäin suurta määrää lähteitä. Alla kuitenkin lueteltuina muutamia, jotka ovat eniten vaikuttaneet esiteltävän ohjelmistotestausprosessin sisältöön.

Testauksen kustannuksien karsimisessa on pyritty löytämään parhaita menetelmiä Perryn (2000) 11-askeleen mallista. Perryn mallin johtoajatuksena on testauksen kustannusten vähentäminen.

Testauksen staattisien menetelmien kehittämisessä on käytetty hyväksi muun muassa Lewisin (2000) katselmointeihin liittyviä ohjeistuksia ja neuvoja.

Testauksen eri vaiheiden ja menetelmien kehittämisessä ja parantamisessa on käytetty hyväksi TPI-mallia.

Vaikka monet testausprosessin kehittämiseen tarkoitetut teoriat ovat sidoksissa johonkin tiettyyn ohjelmistokehityksen elinkaarimalliin ei tutkielmassa esitetty testausmalli ota kantaa käytettyyn elinkaarimalliin.

5.2 Testaus osana ohjelmistotuotantoprosessin määrittely, suunnittelu ja implementointi vaiheita

Niin kuin aikaisemmin todettiin on eräs tärkeimmistä kehittyneen testausprosessin piirteistä testauksen tuominen osaksi määrittely-, suunnittelu- ja implementointivaihetta.

Seuraavissa kappaleissa käydään läpi kuinka testaus saadaan integroitua osaksi ohjelmistotuotantoprosessia.

5.2.1 Vaatimusmäärittelyn ja suunnitteluvaiheen todentaminen

Vaatimusmäärittely ja suunnitteluvaiheessa voidaan vastoin yleistä käsitystä myös tehdä testausta. Käytössä ovat staattiset testausmenetelmät, joiden avulla näiden vaiheiden vaihetuotoksia voidaan testata.

Vaatimusmäärittely on vaihe, josta koko ohjelmistotuotantoprosessi vesiputousmallia noudattaen alkaa. Vaatimusmäärittelyn tulee olla tehty oikein ja täydellisesti, jotta laadukas suunnitteluvaihe on mahdollinen. Seuraavassa kuitenkin esimerkkejä huonosta vaatimusmäärittelystä (Lewis 2000, 61):

- vain osa toiminnallisuudesta määritelty
- suorituskykyä ei ole huomioitu
- epätarkat vaatimukset
- tietoturvaa ei ole huomioitu
- rajapintoja ei ole dokumentoitu
- virheelliset ja liikasanaiset vaatimukset
- vaatimukset liian rajoittavia
- ristiriitaiset vaatimukset

Näitä virheitä varten käytetään staattisia testausmenetelmiä, kuten *katselmointeja*. Katselmoinnit ovat yleisesti käytetty tekniikka, jossa dokumentaatiota tutkitaan tarkistuslistoja käyttäen. Tarkistuslistojen avulla vaihetuotteen tarkastelua voidaan kohdentaa tärkeisiin ja relevantteihin seikkoihin, jotka tulee löytyä katselmoitavasta materiaalista riittävän selkeästi esitettynä.

Katselmoinnin tarkoituksena on todeta jonkun ohjelmistotuotantoprosessin vaiheen päättyminen. Yleisesti katselmointeja käytetään suunnittelu- ja implementointivaiheiden yhteydessä.

Katselmoinnin voi kutsua koolle kuka tahansa. Muutamaa päivää, yleensä viikkoa aikaisemmin osallistujille lähetetään katselmoitava materiaali tutustumista varten. Kommentit ottaa vastaan katselmoitavan tuotoksen tekijä. Katselmointien tulee olla intensiivisyydestään johtuen lyhyitä (Lewis 2000, 572). Katselmointia johtaa esimerkiksi projektipäällikkö, joka johtaa katselmoinnin etenemistä. Katselmoinnissa katselmoitava tuotos käydään kohta kohdalta läpi, käytettävän tarkastuslistan mukaan. Katselmoinnin yhteydessä pyritään löytämään osa-alueet joissa on selviä virheitä tai joita voidaan parantaa. Lopuksi päätetään katselmoitavan tuotoksen hyväksymisestä tai hylkäämisestä. Katselmoinneissa löydetyistä puutteista, virheistä ja korjauksista tehdään raportti, jota voidaan hyödyntää muun muassa tarkastuslistoja päivitettäessä.

Vaatimusmäärittelyn yhteydessä tulee vaatimuksista rakentaa ns. *vaatimusten jäljitys matriisi (Requirement Traceability Matrix)*. Tämän tarkoituksena on kulkea koko matka vaatimusmäärittelystä testaukseen asti ja varmistaa vaatimuksien toteutumien ja kaikkien vaatimusten testaaminen (Lewis 2000, 63). Alla olevassa taulukossa on ohjelmistotuotteelle asetetut vaatimukset esitelty vasemmalla ja testausvaiheet on lueteltu ylärivillä. Kun vaatimus on testattu jossakin testausvaiheessa sen kohdalle on merkitty ’X’-kirjain.

Vaatimusten jäljitys matriisi testaukselle

	MT	FT	ST
REQ 1	X	X	X
REQ 2	X	X	X
REQ 3	X	X	X

Myös Perryn 11-askeleen testausmallissa (1999) käsitellään vaatimusten testaamista. Perry pitää vaatimusten testaamista erittäin tärkeänä, koska vaatimusmäärittelyvaiheessa tehdään kaikkein kriittisimmät järjestelmään liittyvät päätökset. 11-askeleen mallissa vaiheessa kolme (3) varmistetaankin, että

- vaatimukset on oikein kirjoitettu vaatimusmäärittelydokumentaatioon.

- vaatimukset käsittelevät käyttäjän tarpeita.
- kustannus/hyöty tutkimus vaatimuksille on tehty.
- asianmukaista prosessia on noudatettu kehitettäessä liiketoimintaratkaisua.
- paras mahdollinen vaihtoehto on valittu kaikista madollisista ratkaisumalleista.

Vaatimuksien testaamisen ja testauksen suunnittelun yhteydessä tulee huomioida sekä toiminnallisten, että ei-toiminnallisten vaatimusten todentaminen. Toiminnallinen testitapaus voi kattaa esimerkiksi yhden vaaditun toiminnon testaamisen ja ei-toiminnallinen testitapaus voi testata esimerkiksi ohjelmiston suorituskykyä tai käytettävyyttä.

Suunnitteluvaiheen tuotoksia voidaan myös katselmoida vaatimusmäärittelyvaiheessa kuvatulla tavalla. Perry (1999) käyttää suunnitteluvaiheen testaamiseen myös tarkastuslistoja (*Checklists*). Tarkastuslistoissa on aikaisemmin kuvatun lisäksi lueteltu kysymyksiä suunnitteluvaiheesta. Näiden kysymysten vastausten perusteella voidaan päätellä suunnitteluvaiheen taso. Näitä tarkastuslistoja voidaan käyttää joko katselmoinnin apuna tai yleisesti jonkin ohjelmistotuotantoprosessin vaiheen laadun arvioinnissa. Tarkastuslistoja käytettäessä tulee varmistaa että implementointi, testaus ja muut tahot ovat selvillä tarkastuslistojen käytöstä, jotta tarkasteltavista asioista vaihetuotoksissa ollaan tietoisia (Koomen & Pol 1999, 108).

5.2.2 Implementoinnin todentaminen

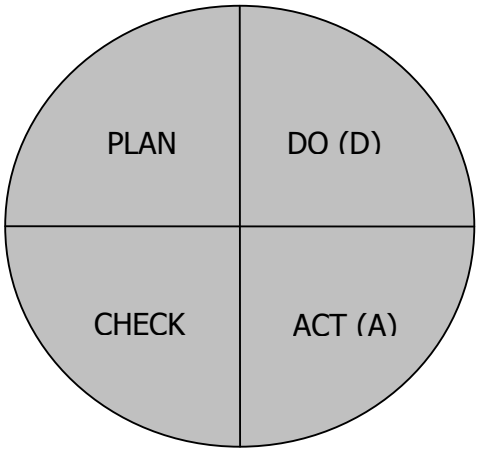
Moduulitestausvaiheessa on päätarkoituksena varmistaa, että suunnitteluvaiheen dokumenttien perusteella on ohjelmakoodi oikein implementoitu. Ohjelmakoodin todentamiseen voidaan käyttää useita erilaisia menetelmiä ja tekniikoita. Olisi liian yksinkertaista ajatella, että ohjelmisto on täysin vapaa virheistä, jos ohjelmakoodi on testattu 100% kattavuudella. Kuitenkin harvoin tai koskaan tähän on resursseja ja silti ohjelmisto ei välttämättä toimi asiakkaan haluamalla tavalla. On tärkeää, että testauksen piiriin ei tuoda kaikkien mahdollisten teorioiden menetelmiä, vaan valitaan juuri omaan tarkoitukseen sopivia menetelmiä. Kuitenkin on olemassa yleisiä menetelmiä joilla ohjelmiston laatua voidaan parantaa huomattavasti.

Sekä staattiset, että dynaamiset testausmenetelmät sopivat hyvin implementointivaiheen testaukseen. Staattisista tekniikoista joita voidaan soveltaa implementoinnin todentamiseen voidaan mainita esimerkiksi *katselmoinnit*, joissa voidaan määrättyä menetelmää käyttäen käydä ohjelmakoodi läpi ja varmistaa sen olevan suunnitteludokumenttien mukainen. Katselmoinneista ja katselmointimenetelmistä palataan myöhemmin.

5.3 Testauksen elinkaaren kehittäminen

5.3.1 Testauksen suunnitteluvaiheen kehittäminen

Demingin PDCA mallin mukaan (kuva 7) spiraalimallinmukaisessa ohjelmistokehityksessä PLAN-vaihe suoritetaan ensimmäisenä. Vaikka emme kehitetyssä testausprosessissamme otakaan kantaa käytettyyn ohjelmistotuotantoprosessiin voimme silti ottaa käyttöön parhaat puolet Lewisin (2000) esittelemästä testausprosessista.



Kuva 6: Demingin PDCA-malli

PLAN-vaiheen osia ovat Lewisin (2000) mukaan:

- informaation kerääminen

- testauksen suunnittelu

Informaation keräämisen tarkoituksena on relevantin informaation kerääminen ohjelmistotuotantoprosessin tarpeisiin haastattelemalla asiakasta. Riippuu kuitenkin paljolti toimintakentästä onko tehtävälle tuotteelle olemassa varsinaista tai suoranaista asiakasta. Voihan olla että ollaan kehittämässä *pakettiohjelmistoa*, jota myydään esimerkiksi suoraan kaupanhyllyltä. Riippumatta siitä onko käytettävissä haastateltavaa asiakasta on informaatio projektille saatava jotakin kautta. Seuraavat asiat tulisi Lewisiä (2000, 107-115) mukaillen ehdottomasti huomioida informaatiota kasatessa:

- liiketoiminta-tason vaatimusten tunnistaminen
- riskianalyysin suorittaminen

Testauksen suunnitteluvaihe on todennäköisesti tärkein testaukseen liittyvää vaihe. Testaussuunnitelman tulisi olla dokumentti, jota pystytään muuttamaan tehtävän ohjelmiston muuttuessa. Koska nykyisin monet ohjelmistoprojektit ovat muutosherkkiä ja uusia vaatimuksia asetetaan kesken projektin on tärkeää, että testaus pystyy muuttumaan tehtävän tuotteen muuttuessa. Tämä asettaa testaussuunnitelmallekin omat haasteensa.

Testaussuunnitelma on johdon näkökulmasta tärkein dokumentti, koska se helpottaa testausprojektin johtamista. Muun muassa seuraavat asiat testaussuunnitelman tulisi Lewisin (2000, 119) mukaan tarjota:

- Omaa hyvät mahdollisuudet löytää suurin osa virheistä
- Tarjoaa hyvän koodikattavuuden
- On joustava
- On helposti suoritettava, toistettava ja automatisoitava
- Määrittelee testattavat testityypit
- Helppotajuisesti dokumentoi odotetut testaustulokset
- Helppotajuisesti määrittelee testauksen päämäärät
- Kertoo testauksen strategian
- Määrittelee testauksen lopettamiskriteerit (exit criteria)
- Ei ole tarpeeton tai ylisanainen

- Tunnistaa riskit
- Dokumentoi testauksen vaatimukset
- Määrittelee testausvaiheen tuotokset

Lewisin mallista kuitenkin puuttuu joitakin asioita, joita muissa kirjallisuuslähteissä korostetaan erittäin tärkeänä osana testaussuunnitelmaa. Nämä asiat ovat testattavat vaatimukset ja testitapausten priorisointi.

Testaussuunnitelma tulisi laatia hyvin varhaisessa vaiheessa ohjelmistotuotantoprosessia, siten että testaussuunnitelma on tehty niin sanotun *Master Test Planin* pohjalta. Master Test Planissa on yleiskuva jokaisen testausvaiheen testauksesta ja myös testaussuunnitelmista.

Kehittyneemmässä testausprosessissa testauksen suunnitteluvaihe pitää sisällään paljon toimintoja ja tehtäviä. Lewis (2000, 120) esittelee kirjassaan erittäin käyttökelpoisen prosessimallin testauksen suunnitteluun, jossa eritellään kaikki tarvittavat tehtävät, jotka vaiheessa tulee suorittaa (liite 1).

5.3.2 Testitapausten suunnittelu

Testitapausten suunnittelun lopputuloksena syntyvät testitapaukset, jotka tulee hyväksyä ja katselmoida kuten muukin formaali tuotos. Testitapaukset voivat olla joko manuaalisesti suoritettavia testejä tai automatisoituja tietokoneen suorittamia (ja joskus myös analysoimia) testitapauksia.

Testitapausten suunnittelussa käytetty testausmenetelmä korostuu. Testausmenetelmiä on olemassa lukemattomia liittyen eri testausvaiheisiin.

5.3.3 Testien suorittaminen

Testauksen varsinaisessa suorittamisessa on koko testausta ajatellen vähiten mahdollisuuksia parantaa tuotteen laatua. Jos testaus määrittely ja suunnittelu vaiheissa on tehty hyvin, vähentää se testauksen tarvetta suoritettavan ohjelmakoodin osalta.

Käytännössä tämä tarkoittaa, että jos kaikki vaihetuotteet varsinaiseen dynaamiseen testaukseen tultaessa ovat laadukkaita, on todennäköisyys virheettömään ja vaatimukset toteuttavaan ohjelmakoodiin myös suuremmat. Kuitenkin jos jostakin syytä aikaisempia vaihetuotoksia ei ole tehty tai testattu riittävän hyvin on testauksen suorituksen oltava todella kattavaa (Perry 1999, 403)

Dynaamisessa testauksessa suunnitellut testitapaukset ajetaan ennalta suunnitellussa järjestyksessä. Tärkeintä on, että testitapauksien suoritus priorisoidaan siten, että lopetettiinpa testaus missä vaiheessa tahansa on ajettu paras mahdollinen otos testauksia. Testitapaukset voidaan priorisoida muun muassa riskien perusteella. Riskiperustaisessa priorisoinnissa testataan ensimmäisinä kriittiset toiminnallisuudet tai alueet jotka ovat eniten virhealttiita.

Yhtä tärkeää kuin on dokumentoida suoritetut testaukset ja löydetyt virheet on tehtyjen korjausten dokumentointi. Kuitenkin dokumentoinnin määrä riippuu täysin organisaatiosta ja kenties käytetyistä standardeista.

Testauksen suorittamisen aikana on tärkeää pitää yllä tilastotietoa testauksista. Käytettyjä tilastoja voivat olla esimerkiksi seuraavat tilastot (Lewis 2000, 170):

- testitapausten tämänhetkinen tilanne
- Virheväli Analyysi (Defect Gap Analysis)
- virheiden vakavuus
- testien lopettamisen mittaaminen

Tilastojen tarkoituksena on kerätä informaatiota muun muassa ajetuista testitapauksista ja löydetyistä virheistä. Virheväli analyysi kertoo kuinka suuri ero on löydettyjen ja korjattujen virheiden välillä. Virheiden vakavuutta tutkimalla saadaan selville testattavan tuotteen laadullisia piirteitä. Ja testien lopettamisen mittaamisella voidaan tarkastella löydettyjä virheistä aikajanalla. Tilastoa tutkimalla voidaan havaita missä vaiheessa testaus tällä hetkellä on.

5.3.4 Testauksen riittävyys

Testauksen riittävyyden arviointi on erittäin tärkeä kehittämiskohde kustannusnäkökulmasta. Tehtävän tuotteen käyttötarkoitus ja kohde vaikuttavat paljon testauksen riittävyyden kriteereihin. Seuraavissa kappaleissa on käsitelty kustannuksia, löydettyjen virheiden määrää ja riskejä testauksen riittävyyden perusteena.

5.3.4.1 Kustannukset testauksen riittävyyden perusteena

Testauksen kustannukset voidaan Perryn (1999, 162) mukaan ajatella seuraavasti:

Testauksen kustannukset = testaustyön kustannukset + löytymättömien virheiden kustannukset

Kuitenkin testauksen kustannuksiksi ajatellaan usein vain kustannukset, jotka aiheutuvat suoranaisesta testauksesta. Kuitenkin jos ohjelmistojen kehittäjät tietäisivät löytämättömistä virheistä aiheutuvat kustannukset tarkemmin testaukseen mitä luultavimmin uhrattaisiin huomattavasti enemmän resursseja. Tätä ei voida kuitenkaan tehdä ennen kuin on pystytty arvioimaan virheen poistamisen todelliset kustannukset. Kustannuksien laskemiseen voidaan käyttää esimerkiksi seuraavaa käytännönläheistä tapaa:

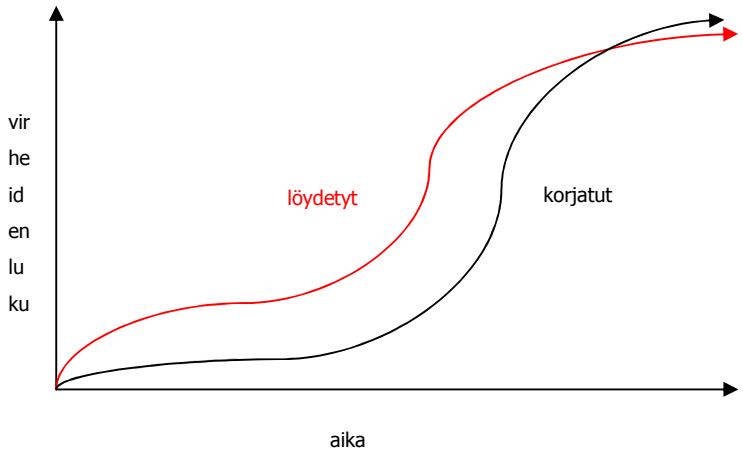
1. Jokainen virhe raportoidaan
2. Jokaisen virheen löytämisen paikka ohjelmiston elinkaaressa raportoidaan
3. Lasketaan korjaamiseen käytetyt resurssit (dokumentoinnin muutokset koodimuutokset ja niin edelleen) siten että:
    a. suunnitteluvaiheessa löydetyt kustannukset ovat todelliset kustannukset *x*
    b. systeemitestauksessa löydetyt virheet ovat kustannukseltaan luokkaa *10x*
    c. tuotannossa löydettyjen virheiden kustannukset ovat luokkaa *100x*

Näin voidaan laskea suhteellisen helposti yhden virheen kustannus. Näitä tuloksia vasten on suhteellisen helppo laskea kuinka kattavaa testauksen tulisi olla, jotta saavutettaisiin paras mahdollinen suhde kustannusten ja laadun välillä. Tuloksista voidaan päätellä myös ohjelmistotuotantoprosessin heikoimmat osa-alueet.

Nyrkkisääntönä voidaan pitää että virheiden löytämisen ja korjaamisen kustannukset tulee jäädä pienemmiksi kuin virheen löytymisen kustannukset tuotteen ollessa jo käytössä Koomen ja Pol (1999, 10).

5.3.4.2 Löydettyjen virheiden määrä testauksen riittävyyden perusteena

Ohjelmistotestauksen riittävyyden arviointi on hankala toimenpide. Joskus testauksen lopettamisen ajankohdan sanelee aikataulu ja joskus taloudelliset resurssit. Yleensä se on kompromissi tuotteessa olevien vikojen aiheuttamien kustannusten ja markkinoilta myöhästymisen aiheuttaman tuoton menetyksen välillä (Haikala & Märijärvi, 2000, 275). Testaukselle tulisi aina asettaa hyväksymiskriteeri. Tällainen voi olla vaikkapa löydettyjen virheiden määrän tasaantuminen ja kun virhekäyrä tasaantuu voidaan testaus lopettaa (kuva 2.). Tämä on kuitenkin suhteellisen huono kriteeri koska ajankohtaa jolloin virhekäyrä tasaantuu ei voi tarkasti ennustaa.



Kuva 8: Virhekäyrä Haikalan ja Märijärven (2000) mukaan

Testauksen päättämiseen voi myös vaikuttaa saavutettu testauksen kattavuus. Testauksen kattavuuden arviointiin on olemassa useita menetelmiä. Kattavuusmitoilla on tarkoitus varmistaa kuinka suuren osan testit kattavat ohjelman toiminnasta. Valkolaatikkotestaukseen liittyvistä kattavuusmitoista voidaan esimerkkeinä mainita vaikkapa lausekattavuus, päätöskattavuus ja ehtokattavuus. Eräs tunnetuimmista ohjelmistotestauksen tutkijoista Boris Beizer pitääkin koodikattavuutta tärkeänä osana laadukasta ohjelmistotestausta. Tämän lisäksi testauksen tulisi olla myös

vaatimuskattavaa, siten että kaikki toiminnalliset ja ei-toiminnalliset vaatimukset tulevat testattua.

5.3.4.3 Testauksen riittävyyden arviointi riskiperustaisesti

Aikaisemmissa luvuissa mainittujen testauksen riittävyyden arviointiin tarkoitettujen menetelmien lisäksi voidaan riittävyyden arvioinnin perusteena käyttää riskiä. Riskillä tarkoitetaan ISEB koulutusmateriaalin mukaan muun muassa seuraavia asioita:

- Riskiä olla havaitsematta tärkeitä vikoja
- Testaamattoman tuotteen tai alitestatun tuotteen julkaisemisen riskiä
- Riskiä menettää uskottavuus markkinoilla
- Ylitestauksen riski tai tehottoman testauksen riski

Näiden riskien perusteella täytyy päättää mitä testataan ensin, mitä testataan eniten ja milloin on testattu riittävästi. Riskiä voidaan näin käyttää aikatauluttamiseen priorisoimalla testauksia. Testauksien priorisoinnissa tulisikin pitää mielessä, että priorisointi tulisi tehdä siten, että lopetettiinpa testaus milloin hyvänsä on siihen mennessä tehty paras mahdollinen otos testejä (ISEB Foundation Certificate in Software Testing koulutusmateriaali, 2002)

5.3.5 Testauksen raportointi

Koomen & Pol (1999, 143) pitävät testauksen raportointia ohjelmistotestauksen tärkeimpänä tuotoksena, siksi että raportoinnin kautta saadaan tietoa tehtävän tuotteen laadusta.

Testauksen raportointi tulee tehdä jokaisesta testauksen suorituksen vaiheesta. Ajetuista testitapauksista tulee kirjoittaa testausraportti, johon kirjataan kaikki testauksen suoritukseen olennaisesti liittyvät asiat. Kun koko testausvaihe tai testauksen osakokonaisuus on saatu valmiiksi tulee testauksesta kirjoittaa dokumentti jossa testausta käsitellään kokonaisuutena.

Testauksesta tulee myös jäädä täydellinen dokumentaatio löydetyistä virheistä. Tämä auttaa myöhemmissä testauksissa löytämään virhealttiit alueet.

5.3.6 Testauksen mittarit

Mittareita käytetään ohjelmistotestausprosessissa prosessin tai testattavan tuotteen eri piirteiden mittaamiseen. Mittausdataa käytetään projektin hallintaan ja johtamiseen ja eri tuotteiden väliseen vertailuun (Kooman & Pol 1999, 109). Mitattavia asioita ohjelmistotestausprosessista voivat olla esimerkiksi Projektin eteneminen, tuotteen laadun mittaaminen ja löydettyjen virheiden määrä.

Mittareiden käyttöön on olemassa kaksi erilaista lähestymistapaa: *top-down* ja *bottom-up*. Ensimmäisessä lähestymistavassa ylimmän johdon toivomukset ja vaatimukset sanelevat käytettävien mittareiden valinnan. Tässä menetelmässä mittarit valitaan sen perusteella kuinka hyvin mittaristo vastaa ylimmän johdon kysymyksiin ja vaatimuksiin. Jälkimmäisessä bottom-up lähestymistavassa itse tuote, työtavat ja ihmiset ovat mittauksen perustana. Tyypillisimmillään mitataan syötteitä, tulosteita ja testauksen tuloksia.

Koomen & Pol suosittelevat, että ohjelmistotestauksen mittaaminen aloitetaan pienessä mittakaavassa, kuten mittaamalla käytettyjä tunteja. Mittaamisessa kannattaa välttää mittaamasta yksilöitä, koska mittausvirheen mahdollisuus kasvaa yksilöä mitatessa liian suureksi. Hyväksi havaitut mittarit joihin yrityksen kaikki jäsenet ovat sitoutuneet, tulee viedä osaksi käytettyjä menetelmiä esimerkiksi templaattien kautta.

5.3.7 Testauksen työkalut

Ohjelmistotestauksen suorittamisessa voidaan käyttää lukemattomia erilaisia testaustyökaluja. Testaustyökaluja ovat esimerkiksi testi data generaattorit, tallenna ja toista työkalut, debuggerit, load & stress generaattorit. Prosessimielessä ei kuitenkaan ole syytä ottaa kantaa testaustyökalujen käyttöön. Kuitenkin testaustyökaluja ja niiden käyttöä tulee hyödyntää jos mahdollista. Kannattaa muistaa, että testityökalujen ei

välttämättä tule tai tarvitse olla kaupallisia tuotteita. Työkalut voivat olla implementoijien tekemiä tai ilmaisia tuotteita.

Ohjelmistotestauksen automatisointi on liian iso kokonaisuus käsiteltäväksi tässä. Kuitenkin voidaan todeta, että testaustyön automatisoinnilla voidaan saada uskomattoman suuria kustannussäästöjä, mutta ei suinkaan ilmaiseksi. Testauksen automatisointi on erittäin laaja ja haastava tehtävä, joka ei välttämättä sovi muuttuvaan ja dynaamiseen ympäristöön. Lisää testauksen automatisoinnista ohjelmistotestausprosessin osana voi lukea muun muassa Dustinin, Rashkan ja Paulin teoksesta *Automated Software Testing*.

6. Yhteenveto

Perinteisen testausprosessin avulla on pystytty löytämään useita ohjelmistotestauksen kriittisiä osa-alueita. Kirjallisuuslähteiden pohjalta näille kriittisille osa-alueille on pyritty löytämään tehokkaampia ja parempia menetelmiä suorittaa ohjelmistotestausta. Yhteenvedossa lyhyesti kuvataan testauksen kriittiset osa-alueet ja menetelmät joilla ohjelmistotestauksen suoritusaikaa voidaan lyhentää ja testauksesta itsessään saadaan laadukkaampaa ja kustannustehokkaampaa.

6.1 Testausmenetelmien kehittämisestä

Perinteinen primitiivinen testausprosessi keskittyy kokeilemiseen ilman suunnitelmallisuutta. Kehittyneemmätkin testausprosessit pyrkivät keskittymään liikaa varsinaiseen testaustapahtumaan, jolla tarkoitetaan suoritettavan ohjelmakoodin testausta. Testausprosessiuudistuksessa suurimmat laadulliset ja kustannukselliset hyödyt saadaan testauksen tuomisella osaksi koko ohjelmistotuotantoprosessia. Näin mahdolliset ohjelmistovirheet löydetään aikaisemmin ja niiden korjaaminen on huomattavasti edullisempaa. Tämä tarkoittaa staattisten testausmenetelmien käyttöönottoa, joilla kaikki ohjelmistotuotantoprosessin vaihetuotokset testataan käyttämällä hyväksi katselmointeja estävinä toimenpiteinä virheiden syntymiselle. Staattisen testauksen apuvälineenä voidaan käyttää tarkistuslistoja, joilla katselmointeja voidaan kohdistaa keskittymään olennaisiin asioihin katselmoitavassa vaihetuotteessa.

Dynaamisen testauksen menetelmien kehittäminen on huomattavasti hankalampaa jos testausprosessi on edes auttavasti kontrolloitu ja suunnitelmallinen. Testauksen suoritus tapahtuu liian myöhään ohjelmistotuotantoprosessia, jotta sillä pystyttäisiin tehokkaasti vaikuttamaan testattavan tuotteen laatuun. Koko dynaamisen testauksen tärkein osa on vaatimusten testaaminen. Vaatimukset ovat tuotetun ohjelmiston perusta ja vaatimukset tulisi pyrkiä todentamaan jokaisessa testausvaiheessa. Vaatimusten todentaminen suoritetaan luonnollisesti kyseessä olevan testausvaiheen tasoisesti. Testauksien jäljitettävyys läpi ohjelmistotuotantoprosessin on mahdollista muun muassa käyttämällä vaatimus matriisia. Vaatimus matriisin avulla voidaan varmistaa, että vaatimukset tulevat toteutettua ja testattua (Lewis, 2000).

Dynaamisen testauksen kehityksen yhteydessä voidaan vähillä kustannuksilla ja resursseilla kehittää myös testauksen suunnittelu- ja raportointikäytäntöjä. Testaus tulisi suunnitella erikseen jokaiselle testauksen vaiheelle. Tämä tarkoittaa testauksen lähestymistavan ja menetelmien valitsemista ja suunnittelemista kaikille testausvaiheille erikseen. Testausta suunniteltaessa on tärkeää, että testauksen lopettamiselle asetetaan kriteerit ja suunnitellut testitapaukset priorisoidaan. Testauksen riittävyyden perusteena voivat olla kustannukset, löydettyjen virheiden määrä tai riskit. Testauksen riittävyyden arviointi tulee tehdä huolellisesti, koska arvion perusteella tuote päästetään asiakkaalle joko riittävän laadukkaana tai huonoimmassa tapauksessa lähdes käyttökelvottomana.

Koomenin ja Polin (1999) mukaan raportointi on eräs tärkeimmistä testaustoimenpiteistä, koska sen kautta saadaan konkreettista tietoa tuotteen laadusta. Raportoinnin tärkeys korostuu varsinkin jos raportoitua dataa voidaan hyväksikäyttää mittaristojen avulla. Kaikista testauksen vaiheista tulee tehdä testausraportti, johon on kirjattu kaikki testauksen suoritukseen olennaisesti liittyvät asiat.

6.2 Ohjelmistotestauksen vaiheista

Moduulitestausta on mahdollista kehittää valkolaatikkotestauksen keinoin. Moduulitestauksen laadukkuutta voidaan mitata esimerkiksi kattavuusmittoja käyttämällä. Esimerkkejä mitattavista testauksen kattavuuksista ovat esimerkiksi lausekattavuus ja ehtokattavuus. Myös staattiset testausmenetelmät sopivat moduulitestauksen laadunvarmistukseen. Vaihetuotteiden, mukaan lukien lähdekoodin, katselmoinnit ovat tyypillisiä moduulitestauksen staattisia testausmenetelmiä.

Toiminto- ja järjestelmätestauksessa voidaan saada huomattavia laadullisia parannuksia kustannustehokkaasti kehittämällä vaatimusten jäljitettävyyttä ja suunnittelemalla testaukset vaatimuskattavasti.

Ohjelmistovirheiden korjaaminen voi generoida uusia vikoja joiden estäminen ja löytäminen on erittäin tärkeää. Korjauksen jälkeen itse korjauksen testaamisen lisäksi

testaus tulee ulottaa myös muille alueille joihin korjaus on mahdollisesti voinut vaikuttaa. Kyseessä olevan testausvaiheen testitapauksista onkin hyvä laatia regressiotestaus kokonaisuus, joka ajetaan tietyin väliajoin korjauksien välillä, jotta voidaan varmistua ohjelmiston toimivuudesta.

6.3 Testausprosessin uudistamisesta

Testauksen prosessiuudistus ei ole ihmelääke ohjelmiston laadullisten ongelmien ratkaisemiseksi. Ohjelmistotuotantoprosessin vaiheiden tulisi olla samalla tasolla, jotta suurin mahdollinen hyöty saataisiin laadukkaista prosesseista.

Testauksen prosessiuudistusta suunniteltaessa on ehdottomasti mietittävä mitä hyötyjä tehtävän muutoksen kautta halutaan. Prosessiuudistuksen menetelmät eroavat paljon riippuen siitä onko haluttuina hyötyinä esimerkiksi kustannussäästöt, läpimenoaikojen pieneneminen vai tuotteen laadullinen parantaminen

Testauksen kustannuksia laskettaessa on erittäin tärkeää muistaa, että ohjelmistotestauksen kustannukset eivät muodostu pelkästään testauksen kustannuksista vaan myös löytymättömien virheiden aiheuttamista kustannuksista (Perry, 2000). Testauksen kustannuksia vähennettäessä, edellä mainitun ajattelutavan mukaan tarkoitetaan usein, jopa varsinaisen ohjelmistotestauksen lisäämistä.

Testauksen kustannuksia ei tule karsia testausta tai testausresursseja karsimalla. Vaikka ohjelmistotestausprosessi olisi miten laadukas hyvänsä on erittäin tärkeää, että testauksen suorittaa testauksen menetelmiin, tekniikoihin ja ajattelumalleihin perehtynyt henkilö. Ohjelmistotestauksesta karsittaessa kustannussyistä on hyvä muistaa, että testauksen kustannuksiin tulee laskea myös löytymättömien virheiden aiheuttamat kustannukset. .

Lähdeluettelo

Beizer B, 1990. Software Testing Techniques, 2nd edition. International Thomson Publishing.

Haikala I, Märijärvi J, 2000. Ohjelmistotekniikka. Satku – Kauppakaari Oyj.

Dustin, Rashka, Paul 1999 Automated Software Testing: Introduction, Management and Performance. Addison-Wesley.

Dustin E 1999. Lessons in Test Automation: A Managers Guide to Avoiding Pitfalls when Automating Testing. Software Testing & Quality Engineering Magazine, Sep/Oct 1999, s. 16-22

Heikkilä H., 2003 Ohjelmistotuotanto kurssimateriaali kevät 2003. Jyväskylän yliopisto: Informaatioteknologian tiedekunta: Tietojenkäsittelytieteen laitos

ISEB Foundation Certificate in Software Testing koulutusmateriaali, 2002. Grove Consultants

John D. McGregor, Timothy D. Korson. Integrated Object-Oriented Testing and Development Process. Communications of ACM

Koomen T. and Pol M., 1999. Test Process Improvement: A practical step-by-step guide to structured testing. Addison-Wesley

Lewis W, 2000. Software Testing and Continuous Quality Improvement. Auerbach.

Marick B, 1998. When Should a Test Be Automated. Proceedings of the 11th International Software Quality Week Conference, Software Research Institute http://www.testing.com/writings/automate.pdf

Myers G.J. 1979. The Art of Software Testing. New York NY: Wiley-Interscience

Partanen Anu, 2003. Olio-ohjelmien Testaus. Pro Gradu-tutkielma

Pol M., Teunissen R., Veenendaal E., 1995. Testing according to Tmap. Tutein Nolthenius, ‘s-Hertogenbosch

Pressman R, 1997. Software Engineering: A Practitioner’s Approach. McGraw-Hill
Silander S., 1999. DX-Aapinen: Johdatus DX 200 –ohjelmistotyöhön. Edita Oy

Taina J., 2004. Ohjelmistojen testaus, Syksy 2004 – luentomateriaali

Liite 1: Lewisin prosessimalli ohjelmistotestauksen suunnitteluun