Ohjelmistojen testauksen kehittäminen ja parantaminen

Tuula Kyllönen

23.7.2008
Joensuun yliopisto
Tietojenkäsittelytiede
Pro gradu -tutkielma

Tiivistelmä

Ohjelmistojen testauksen tutkiminen, soveltaminen ja kehittäminen ovat viimeisen vuosikymmenen aikana edistyneet merkittävästi. Ohjelmistojen testauksen noin 40-vuotisen historian aikana testaus on saanut sille kuuluvaa huomiota ja arvostusta vasta 1990-luvulta lähtien. Testaus on alettu ymmärtää merkittävänä osana ohjelmistotuotantoa sekä tärkeänä informaation tuottajana. Vaativien aikataulujen ja budjettien vuoksi testaus on usein jäänyt vähälle huomiolle ohjelmistotuotannon eri vaiheissa. Lisäksi mittavat ja raskaat dokumentointistandardit eivät ole soveltuneet kaikkiin ohjelmistotuotantoympäristöihin. Järjestelmien kasvaminen ja monimutkaistuminen sekä ketterien ohjelmistotuotantomenetelmien kehittyminen ja soveltaminen enenevässä määrin asettavat ohjelmistojen testaukselle, testausprosessille ja testauksen hallinnalle uusia vaatimuksia ja haasteita. Testauksen kehittämiseen on olemassa useita erilaisia menetelmiä, joita soveltamalla voidaan organisaatioissa arvioida testauksen nykytilannetta ja kehittämiskohteita. Testaaminen liittyy kiinteästi tuotteen laatuun. Yrityksissä määriteltävän laatustrategiaan kuuluvalla laatuprosessilla on kolme selkeää tavoitetta: parantaa ohjelmistotuotetta, määrittää ohjelmistotuotteen laatu ja parantaa jatkuvasti sekä laatua että itse laatuprosessin kustannustehokkuutta. Nämä tavoitteet edellyttävät laadun varmistamisen ja aktiviteettien parantamisen liittämisen tiiviisti koko ohjelmistontuotteen elinkaareen, tuotteen alusta sen käyttöönottoon, kehittämiseen ja käytöstä luopumiseen.

Tutkielman tavoitteena on tarkastella testauksen kehittymistä, nykytilannetta sekä uusia testaamiskäytäntöjä testauksen kehittämisen ja parantamisen valossa. Tutkielma perustuu kirjallisuuskatsaukseen ja pienimuotoiseen projektikohtaiseen kyselytutkimukseen.

ACM-luokat (ACM Computing Classification System, 1998 version): D.2.4, D.2.5, D.2.8, D.2.9
Avainsanat: Ohjelmistojen testaus, testausprosessi, testauksen kehittäminen

Sisällysluettelo

1 JOHDANTO

Ohjelmistotuotanto on kokenut suuria muutoksia viimeisen vuosikymmenen aikana. Näin ollen myös ohjelmistojen testaaminen ja analysointi ovat olleet tutkimuksen ja kehityksen kohteina. Laatuun ja testaamiseen on alettu kiinnittää käytännössä riittävästi huomiota vasta viime aikoina. Testaaminen on aikaisemmin usein nähty vain yhtenä vaiheena toteutuksen ja toimituksen välillä. Nykyään tiedostetaan, että ohjelmiston laadun varmistaminen on jatkuvien aktiviteettien kokoelma sisällettynä jokaiseen tehtävään vaatimusten keräämisestä toimitetun version evoluutioon.

Yrityksen laatustrategia määrittelee, miten laatu varmennetaan yrityksessä. Laadun varmennus on prosessi ja onnistunut yrityksen sisäinen laatustrategia on edellytys hyvälle laatuprosessille. Yrityskohtaisen laatustrategian suunnittelussa on huomioitava organisaation muut ohjelmistotuotannon menetelmät ja projektikohtaisessa laatusuunnittelussa on lisäksi otettava huomioon sovellusalue ja projektin koko. Laadunvarmennusprosessin eli laatuprosessin on oltava yhteensopiva kehitystyössä käytettävän prosessimallin kanssa.

Perinteisissä ohjelmistotuotannon menetelmissä ohjelmiston testaus sijoittui usein prosessin loppupuolelle. Kun ohjelmistotuotantoprosessi toteutettiin usein muutenkin riittämättömien resurssien puitteissa, oli selvää, että testaamista ei voitu tehdä niin hyvin kuin oli suunniteltu. Ohjelmistojen kasvaessa ja muuttuessa entistä monimutkaisemmiksi testaamisen tutkiminen ja kehittäminen ovat tulleet entistä tärkeämmiksi. Lisäksi ohjelmistotuotannon menetelmien kehittyminen ja muutokset ovat vaatineet uusia lähestymistapoja testaukseen ja analysointiin. Perinteisen ohjelmiston testaamisen edellytyksenä on riittävä ja ajantasainen dokumentaatio; ketterien ohjelmistotuotantomenetelmien käyttö ja arvomaailma taas eivät suosi mittavaa dokumentaatiota. Tästä johtuen ohjelmistojen testaamisen kehittäminen on monessa organisaatiossa jäänyt taka-alalle; vanhat testauskäytännöt ovat epäsopivia ja yrityksissä ei aina ole resursseja uusien testauskäytäntöjen kehittämiseen ja soveltamiseen.

Järjestelmiä kehitetään nykyisin usein ns. inkrementaalisesti: tuotetaan ensin osa toiminnallisuudesta ja laajennetaan järjestelmää myöhemmin. Inkrementaalinen kehitystapa ja toinen kehittämispuolen valtavirtaus, ns. iteratiivinen kehittäminen ovat käsitteellisesti lähellä toisiaan. Iteratiivisessa ja inkrementaalisessa kehitysprojektissa tulisi testata jokainen tuotettu versio kunkin iteraation lopuksi. Näin ollen testaajan työmäärä kasvaa käytännössä jatkuvasti, kunnes se on mahdotonta suorittaa manuaalisesti asetettujen aikataulujen ja henkilöresurssien puitteissa. Testauskäytäntöjä joudutaan usein muuttamaan ja testauksen kehittämistä miettimään koko organisaation tasolla.

Tutkielmassani perehdyn testaamisen ja analysoinnin kehittämiseen ja parantamiseen ohjelmistotuotannossa. Tarkoituksena on kuvata testaamisen kehittymistä, perehtyä testaamiseen tänä päivänä ja ottaa esille muutamia uusia testauskäytäntöjä. Luvussa kaksi kuvaan ensin lyhyesti viime vuosikymmenien testauksen historiaa, sen jälkeen perehdyn ohjelmistojen analysointiin ja testaamiseen yleisellä tasolla tarkastelemalla testaamisen periaatteita, menetelmiä ja prosessia. Luvun kaksi lopussa otan esimerkkeinä esille testauksen nykykäytännöistä testiohjatun kehittämisen (Test-Driven Development, TDD), mallipohjaisen testauksen (Model-Based Testing, MBT), oliopohjaisen testauksen ja tutkivan testauksen (Exploratory Testing, ET).

Luvun kolme tavoitteena on luoda katsaus testaustoiminnan kehittämiseen ja parantamiseen. Ensin tarkastelen ohjelmistojen testaustoiminnan kehittämistä kahdesta eri näkökulmasta: aluksi kuvaan miten Edward Kit (1995) lähestyy testaamista ja testausprosessin kehittämistä teoksessaan 'Software Testing in the Real World'. Sen jälkeen selvitän Bach & al. (2002) näkemyksiä testaamisesta teoksessa 'Lessons Learned in Software Testing'. Sekä Kit että Bach & al. ovat lähestymistavaltaan käytännönläheisiä ja kokemuksiin perustuvia; Kit ottaa lisäksi esille useita standardeja ja määritelmiä. Seuraavaksi keskityn testauksen kehittämiseen ja esittelen muutamia kehittämismalleja. Lopuksi otan esille testauksen parantamisen haasteita ja tulevaisuuden näkymiä.

Luvussa neljä kuvaan lyhyesti työnantajani AtBusiness Oy:n erään projektin testaamisen nykytilannetta sekä erään järjestelmän testaamiseen liittyviä kehittämissuunnitelmia. Aloitin edellä mainitussa projektissa testaus- ja dokumentointitehtävissä maaliskuussa 2008 ja sain työnantajaltani luvan ottaa projektin testauksen kehittämisen käytännön esimerkiksi tekeillä olevaan graduuni. Tässä tutkielmassa käytän sekä projektista että järjestelmästä nimeä Atlas. Lopuksi esitän tutkielmasta yhteenvedon.

2 OHJELMISTOJEN TESTAUS

Kolmisenkymmentä vuotta sitten ilmestyi Glenford J. Myersin teos ‘The Art of Software Testing’, jossa Myers (1979) esitti mullistavan ajatuksensa liittyen testaamiseen; hän määritteli testaamisen ‘prosessiksi, jolla suoritetaan ohjelmaa tarkoituksena löytää virheitä‘. Siihen asti testaus oli määritelty muun muassa seuraavasti: ‘testaus on prosessi, jolla esitetään, että virheitä ei ole’ tai ‘testauksen tarkoitus on näyttää, että ohjelma suorittaa sille määrätyt funktiot oikein’. Nämä olivat Myersin mielestä vastakkaisia käsityksiä sille, miten testaus tulisi ymmärtää. Myersin ja muiden hänen aikalaistensa tutkimukset ja julkaisut aloittivat uuden vaiheen ohjelmistojen testauksessa.

Tänä päivänä ohjelmiston testaus elää jälleen murroskautta. Uusien ohjelmistotuotantomenetelmien, kuten ketterien menetelmien, suosio ja soveltaminen enenevässä määrin sekä testauksen ymmärtäminen tärkeänä osana ohjelmistotuotannon kokonaisuutta, ovat merkittävästi lisänneet testauksen tutkimista sekä uusien testauskäytäntöjen kehittämistä ja soveltamista. Testaus nähdään monipuolisena, laajana ja kehittyvänä alueena. Monissa organisaatioissa testauskäytännöt ja –prosessit ovat kehittämisen kohteina ja uusia menetelmiä otetaan käyttöön testauksessa muiden ohjelmistotuotantomenetelmien ohella.

2.1 Ohjelmistojen testauksen kehittyminen

Ohjelmistotuotannon alkuaikoina testaus käsitettiin Kitin (1995) mukaan virheiden etsimisenä ja poistamisena ohjelmista, ns. debuggaamisena. Tavallisesti testauksen toteutti kehittäjä itse, eikä testauksessa ollut erikseen määriteltyjä resursseja; jos oli, ne otettiin mukaan vasta kehittämisen loppuvaiheessa, kun tuote oli koodattu ja melkein valmis. Seuraavassa kehitysvaiheessa testaus erotettiin virheiden etsimisestä ja korjaamisesta (debugging), mutta pidettiin edelleen kehittämisen jälkeisenä aktiviteettina. Testauksen tarkoitus oli vakuuttaa, että ohjelmisto toimii. Yliopistoissa testausta ei käsitelty juuri ollenkaan; opetusohjelmat käsittivät numeerisia menetelmiä tai algoritmien kehittämistä, mutta eivät ohjelmistotuotantoa tai testausta. Tietokoneet, käyttöjärjestelmät ja tietokannat olivat ensisijaisia huomion kohteita, mutta niistä ei ollut apua testausongelmiin todellisessa maailmassa.

Termiä ohjelmistotuotanto alettiin käyttää 1970-luvulla ja ensimmäinen virallinen testaamisen konferenssi pidettiin Pohjois-Carolinan yliopistossa vuonna 1972, minkä jälkeen ilmestyi sarja julkaisuja. Glenford J. Myers käsitti vuonna 1979, että inhimillisten tavoitteiden itseään toteuttavalla luonteella on suuri vaikutus testaustyöhön. Hän määritteli testauksen ‘ohjelman suoritusprosessiksi, jonka tarkoitus on löytää virheitä’. Hän osoitti, että jos tavoitteemme on osoittaa virheiden poissaolo, me löydämme niitä hyvin vähän; jos taas tavoitteemme on osoittaa niiden olemassaolo, löydämme niitä paljon enemmän. Myersin ja muiden työ 1970-luvulla oli iso askel testausprosessin kehittämisessä, mutta todellisessa elämässä testaus oli edelleen ensimmäisiä asioita, joista oltiin valmiita joustamaan aikataulun tai budjetin hyväksi. Testaus aloitettiin usein liian myöhään, eikä sitä ollut aikaa tehdä riittävällä tasolla ja mitä myöhemmin virhe havaitaan, sitä kalliimmaksi sen korjaus tulee.

Aikaisin 1980-luvulla ohjelmiston laatu ja laatuun liittyvät tekijät nousivat esille. Ohjelmistojen kehittäjät ja testaajat alkoivat keskustella keskenään ohjelmistotuotannosta ja testaamisesta. Muodostettiin ryhmittymiä, jotka loivat monet niistä standardeista, joita meillä on tänäkin päivänä. Nämä standardit, kuten IEEE, ANSI tai ISO ovat muodostumassa liian mittaviksi, jotta ne voitaisiin sisäistää koko julkaistussa laajuudessaan päivittäisessä käytännössä (Kit, 1995).

Testausvälineet pääsivät oikeuksiinsa 1990-luvulla. Nykyään ymmärretään, että välineet eivät ole vain hyödyllisiä, vaan myös välttämättömiä riittävälle testaamiselle. Välineet tai teknologia itsessään eivät kuitenkaan poista itse ongelmaa. Huolimatta valtavasta edistymisestä viimeisen 40 vuoden aikana ohjelmistoprosessi (sisältäen testausprosessin) on useimmissa yrityksissä edelleen alhaisella kypsyystasolla (Kit, 1995). Testauksen kehitys oli merkittävää 1990-luvulla: testauksen opetus itsenäisinä kursseina käynnistyi Suomen yliopistoissa ja testauksen automatisointia aloitettiin (Pohjolainen, 2008).

Testausta alettiin ulkoistamaan yrityksissä 1990-luvulla; testaus nähtiin yhä enemmän asiantuntijuutena, joka vaati tietynlaista osaamista. Uusien ketterien ohjelmistotuotantomenetelmien myötä paine testauksen kehittämiseen ja parantamiseen kasvoi; muun muassa *testiohjattu kehittäminen* (Test Driven Development, TDD) ja *mallipohjainen testaus* (Model Based Testing, MBT) kehittyivät ja niitä alettiin soveltaa. Testauksen opetus yliopistoissa on nykyään vakiintunut hyvälle tasolle (Pohjolainen, 2008).

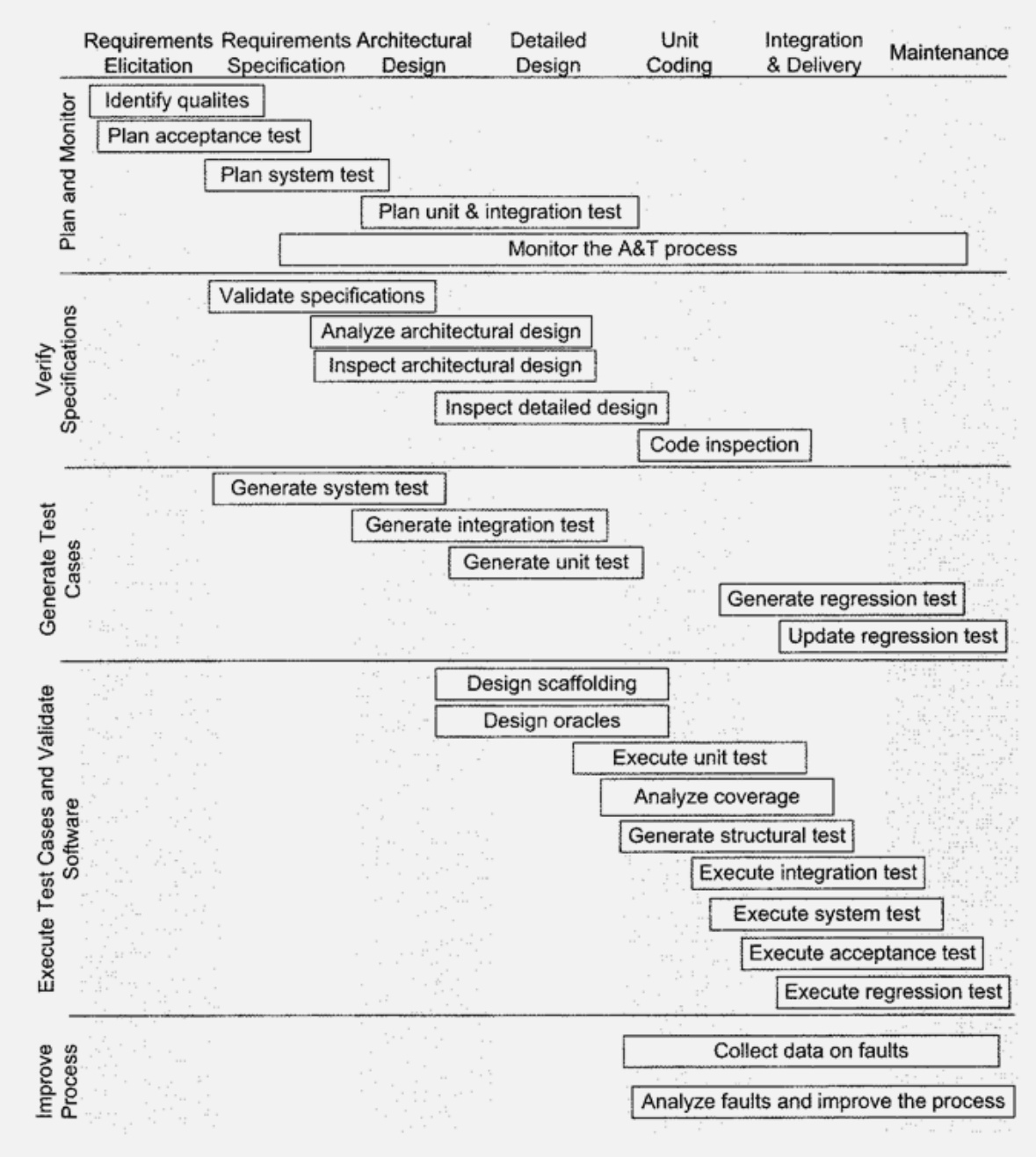
2.2 Ohjelmistojen testaus ja analysointi: periaatteet, menetelmät ja prosessi

Testaaminen ja analyysi kuuluvat tiettyyn kehykseen ja omaavat tiettyjä periaatteita sekä käsittävät joukon aktiviteetteja, jotka ovat sidoksissa koko ohjelmistoprosessiin (Pezze & Young, 2008). Kuva 2.1 esittää testauksen ja analyysin pääaktiviteetit ohjelmiston elinkaaren aikana. Sopivien verifiointiaktiviteettien löytäminen ja soveltaminen riippuu sovellusalasta, kehittämisprosessista, lopputuotteesta ja laatuvaatimuksista.

2.2.1 Testauksen ja analyysin periaatteet

Pezze & Youngin (2008) mukaan testauksen ja analyysin perusperiaatteet muodostavat tieteenalan ytimen: ne muodostavat pohjan metodeille, tekniikoille, metodologioille ja työvälineille. Ne sallivat erilaisten lähestymistapojen ymmärtämisen, vertaamisen, arvioimisen ja laajentamisen sekä muodostavat pysyvän pohjan tietämykselle.

Varhaisessa vaiheessa suoritetuilla analyysi- ja testausaktiviteeteilla on monia hyötyjä: ne kasvattavat koko prosessin kattavaa kontrollia, nopeuttavat virheiden tunnistamista sekä vähentävät virheiden poistamisesta seuraavia kustannuksia. Lisäksi ne tuottavat tietoa jatkuvasti säätöä tarvitsevalle kehittämisprosessille ja nopeuttavat ohjelmiston toimittamista. Toisaalta mikään analyysin ja testauksen määrä ei voi nostaa muiden ohjelmistotuotannon vaiheiden huonoa laatua. Tuotteen laatuattribuutit ovat tuotteelle asetettuja tavoitteita. Ulkoiset laatuattribuutit näkyvät suoraan asiakkaalle; ulkoisia attribuutteja ovat muun muassa luotettavuus, viive, käytettävyys ja läpäisykyky. Sisäiset attribuutit vaikuttavat välillisesti tuotteen laatuun; sisäisiä attribuutteja ovat esimerkiksi ylläpidettävyys, uudelleenkäytettävyys, testattavuus ja seurattavuus.

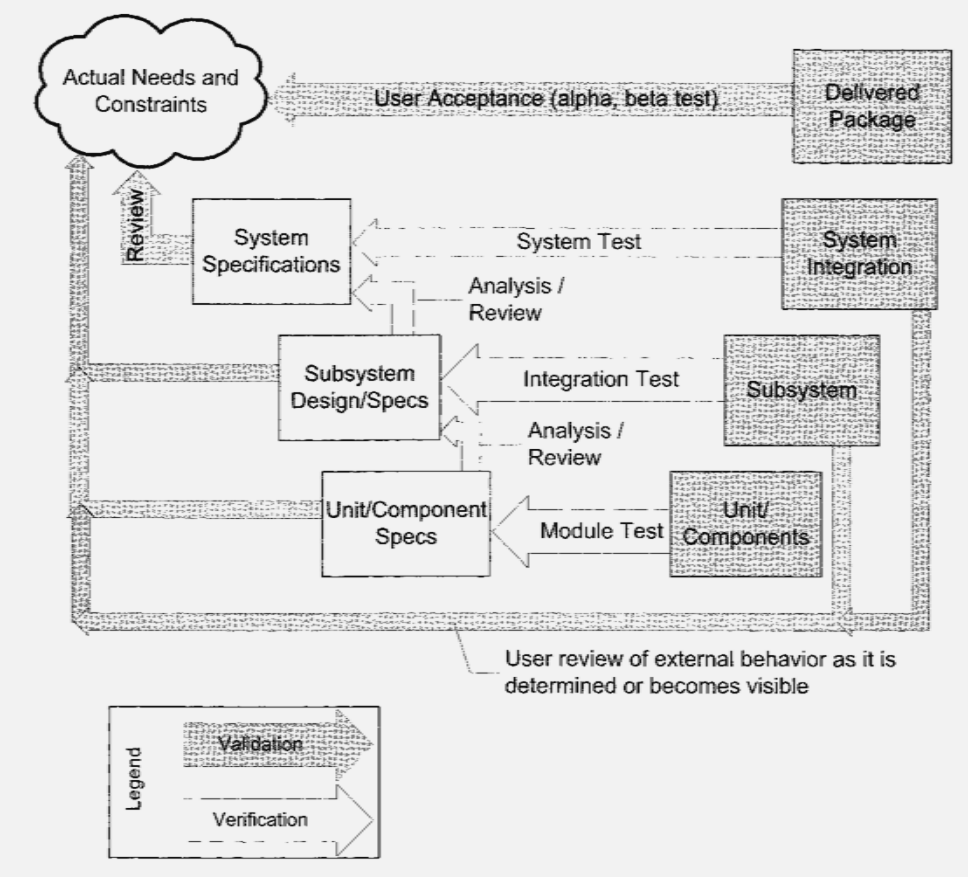


Kuva 2.1 Analyysin ja testauksen pääaktiviteetit ohjelmiston elinkaaren aikana (Pezze & Young, 2008).

Ohjelmisto, joka on yhdenmukainen määrittelynsä kanssa, on luotettava. Luotettavuus voidaan nähdä vastauksena kysymykseen ‘tekeekö ohjelmisto sitä mitä sen on tarkoitus tehdä’. Kun ohjelmisto ei ole luotettava, siinä on vikoja, puutteita tai ohjelmointivirheitä, mitkä aiheuttavat epätoivottua käyttäytymistä tai epäonnistumista. Luotettavuuden ominaisuuksia ovat oikeellisuus, käyttövarmuus, saatavuus, käyttöaika, turvallisuus, riskialttius ja vakaus. Peruskysymyksiä ohjelmistotuotteen laadun varmistamisessa ovat muun muassa: milloin verifiointi tai validointi alkaa tai loppuu, mitä tekniikoita tulisi soveltaa tuotteen kehityksen aikana, että saavutetaan hyväksyttävä laatu hyväksyttävillä kustannuksilla sekä kuinka määritämme julkaistavan tuotteen valmiuden ja kuinka valvomme onnistuneiden julkaisujen laatua.

Ohjelmistotuotteen kaikkien virheiden löytäminen ja korjaaminen on lähes mahdotonta, joten on välttämätöntä määritellä riittävä taso toimitettavan tuotteen toiminnallisuudelle ja laadulle. Tuotteesta voidaan mitata *luotettavuutta* (dependability), *saatavuutta* (availability), *keskimääräistä käytettettävyysaikaa* (Mean Time Between Failures, MTBF) ja *käyttövarmuutta* (reliability). Samoin käyttäjien suorittamilla alpha- ja beta-testeillä voidaan määritellä tuotteen valmius. Alpha-testin suoritus tapahtuu valvotuissa olosuhteissa kun taas beta-testi suoritetaan todellisessa ympäristössä. Onnistuneiden julkaisujen laatu voidaan varmistaa ylläpidettävillä regressiotesteillä. Kehitysympäristö voi tukea esimerkiksi testitapausten nauhoitusta, luokittelua ja automaattista uudelleensuorittamista.

Ohjelmistotuotannossa tuotteen vikojen etsimistä ja korjaamista kutsutaan *verifioinniksi* ja *validoinniksi*. Verifiointi (verification) tarkoittaa työtä, jolla varmistetaan, että ohjelmisto vastaa sille tehtyä määrittelyä (specification) eli ohjelmisto on *luotettava* (dependable). Validointi (validation) tarkoittaa työtä, jolla varmistetaan, että ohjelmisto täyttää asiakkaan sille asettamat tarpeet eli ohjelmisto on *hyödyllinen* (useful). Sopivien validointi- ja verifiointiaktiviteettien valinta riippuu sovellusalueesta, tuotantomenetelmästä, lopputuotteesta ja laatuvaatimuksista. Mikään yksittäinen testi- tai analyysitekniikka ei voi palvella kaikkia tarkoituksia. On suositeltavampaa yhdistää tekniikoita kuin valita yksi ainoa ‘paras’ tekniikka. Tekniikoiden tehokkuus vaihtelee eri virheluokissa ja eri tekniikat soveltuvat eri projektivaiheisiin. Testauksella voi olla erilaisia tavoitteita tai on tehtävä kompromissi kustannusten ja varmistamisen välillä.



Kuva 2.2 Verifioinnin ja validoinnin suhde kehitettävään tuotteeseen (Pezze & Young, 2008).

Verifioinnin ja validoinnin eron voimme ymmärtää asettamalla seuraavat kysymykset:

verifiointi = ‘rakennammeko tuotetta oikein ?’

validointi = ‘rakennammeko oikeaa tuotetta ?’

Verifioinnin ja validoinnin suhdetta kehitettävään tuotteeseen esitetään kuvassa 2.2.

Ohjelmistojen analyysi ja testaus sisältävät kuusi perusperiaatetta riippumatta sovelluksesta ja prosessista. Periaatteet ovat (Pezze & Young, 2008):

Herkkyys (Sensivity)

Toisteisuus (Redundancy)

Rajoittaminen (Restriction)

Ositus (Partition)

Näkyvyys (Visible)

Palaute (Feedback)

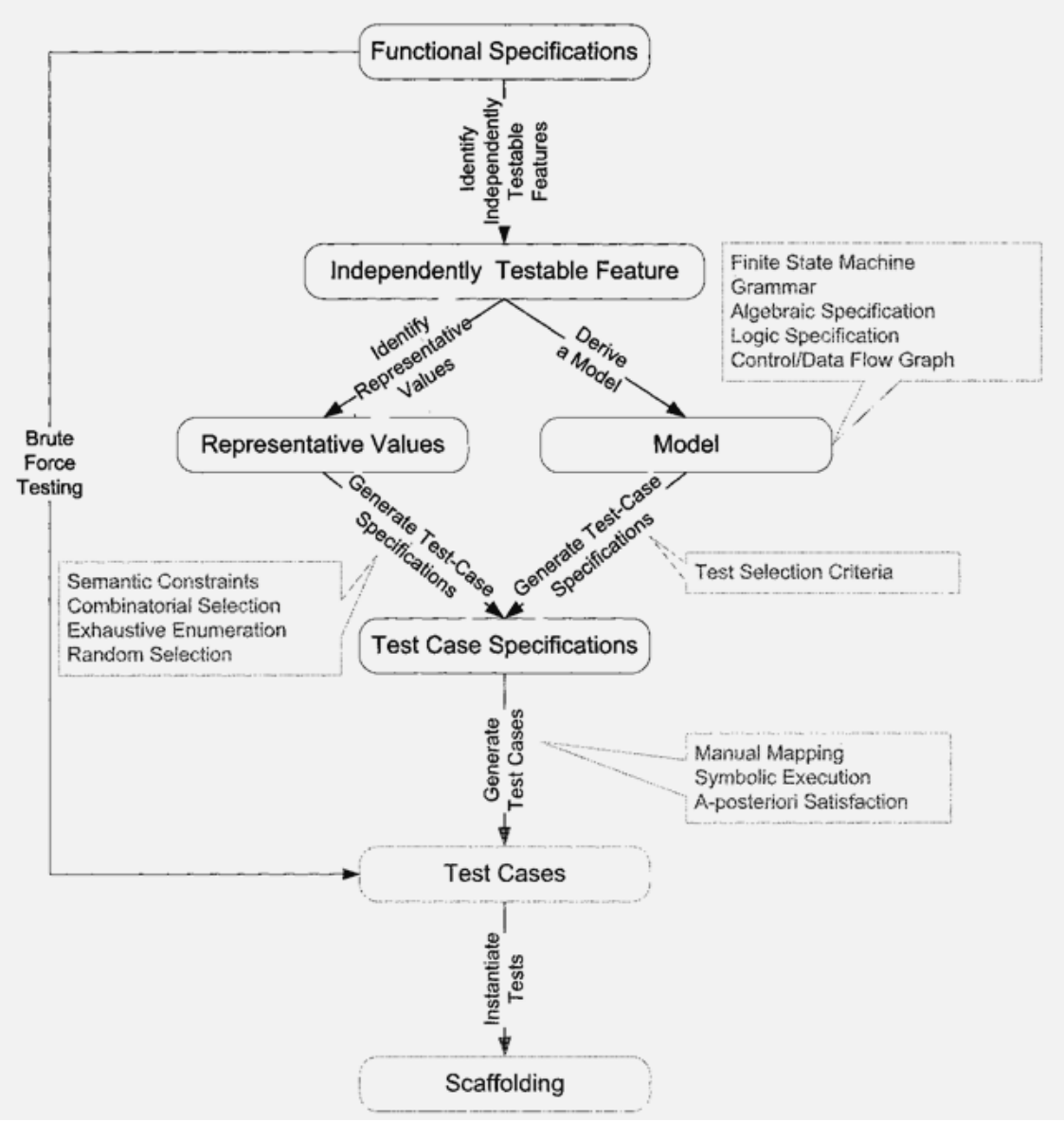
Herkkyys liittyy ohjelmistojen virheherkkyyteen. Ohjelmistossa olevat virheet aiheuttavat virheellisen suorituksen, mutta eivät jokaisella suorituskerralla; mitä pienempi on ohjelmiston virheherkkyys, sitä suuremmalla todennäköisyydellä virhe jää havaitsematta. Herkkyys-periaatteen mukaan kaadutaan mieluummin aina kuin toisinaan. Herkkyys-periaate soveltuu suunnittelu- ja koodausvaiheeseen.

Kahden ohjelmistotuotteen ja/tai dokumentaation osan välillä on *toisteisuutta*, jos ne ovat toisistaan riippuvaisia; kun toisteisuuden tyyppi tiedetään, sitä voidaan testata. Rajoittamisella tarkoitetaan ohjelman yleisen osan korvaamista toisella ominaisuudella, joka on helpommin varmennettava. *Rajoittamisen* periaate on hyödyllinen suunnittelu- ja määrittelyvaiheessa, mutta sitä voidaan harvoin soveltaa valmiin ohjelmistotuotteen yhteydessä. *Ositus* on yleinen insinööritaidon ja ongelmanratkaisun periaate. Ohjelmistotuotanto soveltaa ositusta lähes kaikilla tasoilla, määrittelystä ylläpitoon. Osittamisella tarkoitetaan monimutkaisten ongelmien osittamista pienemmiksi osaongelmiksi, jolloin niiden ratkaisut ovat helpommin verifioitavissa.

Näkyvyys tarkoittaa kykyä mitata kehitystä tai tilaa tavoitteita vasten; näkyvyys-periaatteen mukaan ohjelmisto on sitä näkyvämpi mitä paremmin sen toiminta tunnetaan. Näkyvä ohjelmisto on helpompi verifioida kuin ei-näkyvä. *Palaute* on klassinen periaate, jota sovelletaan analyysissä ja testauksessa. Palaute-periaatteen mukaan aiemmin kerätty tieto vastaavien ohjelmistojen verifioinnista ja validoinnista tulee olla käytettävissä ja hyvä palautemekanismi tulee olla rakennettuna suoraan prosessiin.

2.2.2 Menetelmät

Avainongelma testaamisessa Pezze & Youngin (2008) mukaan on usein testitapausten valinta ja arviointi. Ohjelmisto on testattava systemaattisesti ja riittävästi. Systemaattinen testaus tarkoittaa, että testaukselle on määritelty prosessi tai prosesseja, joita noudatetaan kaikessa testauksessa. Riittävä testaus täyttää sille asetetut ehdot, jotka määritellään yrityksen testausstrategiassa. Testitapausmäärittely sisältää ne vaatimukset, jotka määrittelyn mukaisten testitapausten on täytettävä.



Kuva 2.3 Toiminnallisen testauksen systemaattisen lähestymistavan päävaiheet (Pezze & Young, 2008).

Toiminnallinen testaus eli mustalaatikkotestaus perustuu ohjelmiston toiminnalliseen määrittelyyn. Ohjelmiston toiminnallinen määrittely on tärkein informaatio testitapausten suunnittelussa. Toiminnallinen testaus on perustestausta. Se aloitetaan vaatimusmäärittelyn aikana suunnittelemalla testitapausmäärittelyjä ja jatketaan jokaisessa suunnitteluvaiheessa. Toiminnallisen testauksen tekniikoita voidaan soveltaa jokaisella ohjelmiston kuvaustasolla. Testauksen testitapausten suunnittelun

perustana on ositus. Ohjelmiston toimintatavat ositetaan yhtenäisiksi luokiksi; jokaiselle luokalle tehdään testitapausmäärittely ja tuotetaan sopivat testitapaukset. Toiminnallinen testaus voidaan automatisoida, jos määrittelyt ovat toteutettu jollakin formaalilla tavalla, esimerkiksi laajennetulla tilakoneella. Usein määrittelyt ovat epäformaalissa muodossa ja testitapausten suunnittelu on manuaalista. Olennainen osa toiminnallisen testauksen suunnittelussa on testitapausten karsinta. Systemaattisen ja toistettavan toiminnallisen testauksen prosessi on kuvattu kuvassa 2.3.

Kombinaatiotestauksessa toiminnallisen määrittelyn riippumattomat osat jäsennetään ominaisuuksiksi tai attribuuteiksi. Siten niiden arvoja voidaan määritellä uudelleen ja niiden kombinaatiot voidaan laskea. Toiminnallinen määrittely on usein toteutettu ei-formaalilla menetelmällä kuten luonnollisella kielellä. Kombinaatiotestaustekniikoita ovat luokittelutestaus, parittainen kombinaatiotestaus ja luettelopohjainen testaus; edellä mainituista luokittelutestaus on yleisin.

Rakenteinen testaus on ohjelman rakenteeseen perustuvaa testausta, jota kutsutaan myös lasilaatikkotestaukseksi. Ohjelmiston rakennetta itsessään käytetään informaationa, kun valitaan testitapauksia ja määritellään testitapausten perusteellisuus. Ohjelmiston (t. ohjelman) metodi voidaan kuvata verkkona, jota kutsutaan ohjausvuokaavioksi. Vuokaavion avulla määritellään riittävyysehto, joka kertoo onko ohjelmaa testattu tarpeeksi. Rakenteinen testaus täydentää toiminnallista testausta lisäämällä testitapauksia, joita ei voi yksilöidä vain määrittelyjen avulla.

Tietovuotestaus perustuu huomioon, että virhe esiintyy kohdassa, missä väärin alustettua muuttujan arvoa käytetään ja tietovuotestauksessa käytetään tietovuokattavuuksia. Tietovuokattavuudet ovat tehokkaita, mutta vaativat kehittyneitä työkaluja.

Määrittelyt ilmaistaan usein malleilla. *Mallipohjaisessa testauksessa* käytetään tai johdetaan määrittelystä odotetun käyttäytymisen malleja, jotka tuottavat testitapausmäärittelyjä. Testitapausmäärittelyt voivat paljastaa epäjohdonmukaisuuksia ohjelmiston käyttäytymisen ja mallin välillä. Määrittelyssä käytettävät formaalit mallit voivat olla merkittävä apu testitapausten suunnittelussa. Mallipohjaista testausta käsitellään lisää alikohdassa 2.3.2.

Oliopohjaisessa testauksessa on huomioitava, että olioparadigma sisältää proseduraalisen paradigman ja olioparadigma on proseduraalista laajempi. Oliopohjausta testausta käsitellään enemmän alikohdassa 2.3.3.

Testitapausjoukkojen arvioinnin ja suunnittelun apuna voidaan käyttää ohjelmiston potentiaalisten virheiden mallia. *Virheperusteinen testaus* käyttää virhemallia suoraan olettamalla mahdolliset virheet testauksen aikana sekä luomalla ja arvioimalla testisarjoja, jotka perustuvat sen tehokkuuteen löytää oletettuja virheitä. Virheperusteisen testauksen perusajatus on valita testitapauksia, jotka testaamalla erottavat ohjelmiston vaihtoehtoisesta ohjelmistosta, joka sisältää oletettuja virheitä. Tämä toteutetaan yleensä muokkaamalla ohjelmistoa testauksen aikana.

Testauksen suunnittelu on luovaa, mutta *testauksen suoritus* tulisi sitä vastoin olla mekaanista ja automatisoitua, samoin testien tulosten analysointi. Hyvin tehty testitapausmäärittely antaa joko yksikäsitteisen testitapauksen tai niin selkeän testitapausjoukon, että testitapauksen valinta on suoraviivaista tai automaattista. Testitapaussuunnittelun ja testitapausten tekemisen välillä ei ole selkää rajaa; testitapaussuunnittelu sisältää arviointikykyä ja luovuutta kun taas testitapausten tekeminen pitäisi olla mekaaninen toimenpide.

Ohjelmiston kehityksen aikana tarvitaan *testiympäristö*, koska valmiina olevia osia on voitava testata. Testiympäristön elementtejä ovat testiajurit, testauskehys, tyngät ja oraakkelit. Testiajurit käynnistävät testejä ja välittävät parametreja ja testauskehys korvaa osan asennusympäristöstä. Tynkiä käytetään korvaamaan ohjelmiston niitä osia, joita ei vielä testata. Oraakkeli tarjoaa automaattisen tulkinnan testin läpimenolle.

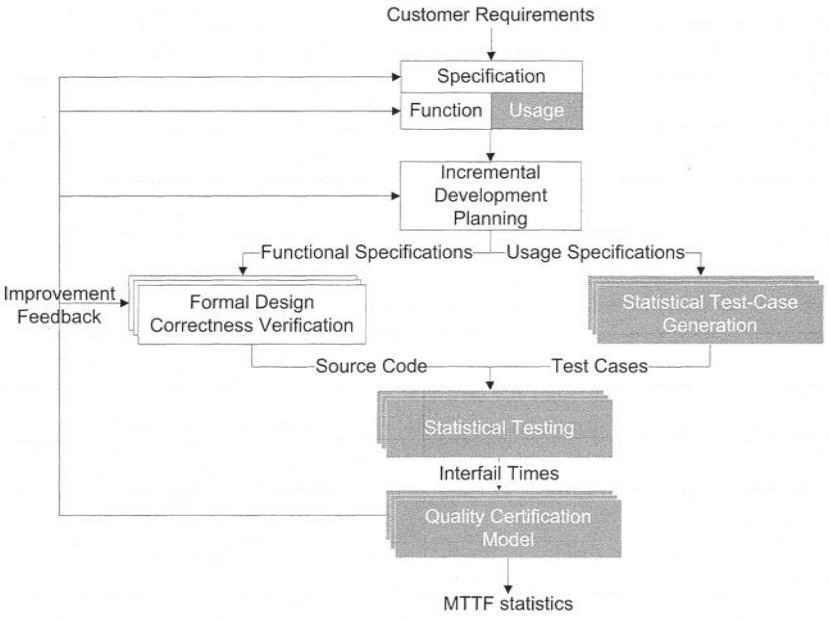
Ohjelmiston tarkastukset ovat manuaalisia, yhteisiä tarkistuksia, joita voidaan soveltaa mille tahansa ohjelmiston osalle määrittelydokumenteista lähdekoodiin ja testaussuunnitelmiin. Tarkastus täydentää testausta auttamalla tarkistamaan monia ominaisuuksia, jotka ovat vaikeita tai mahdottomia verifioida dynaamisesti. Vaikka tarkastuksen tavoite on löytää ja poistaa virheitä, sosiaaliset ja koulutukselliset vaikutukset ovat lähes yhtä tärkeitä.

Ohjelmiston määrittelyjä ja ohjelmakoodia voidaan analysoida automaattisesti. *Automaattinen analysointi* ei pysty osoittamaan, että määrittelyt ja koodi ovat toiminnallisesti oikein, mutta se voi estää joitakin yleisiä virheitä ja tuottaa lisätietoa, joka on hyödyllistä tarkastuksissa ja testaamisessa.

2.2.3 Prosessi

Pezze & Young (2008) esittävät, että laadunvarmennusprosessin eli *laatuprosessin* täytyy olla suunniteltu, seurattu ja systemaattinen kuten kaikki prosessit. Laadun suunnittelu on yksi osa projektisuunnittelua ja laatuprosessit on yhdistettävä läheisesti muihin kehittämisprosesseihin. Laadun suunnittelu alkaa projektin alussa ja sitä kehitetään inkrementaalisesti yhtä aikaa projektisuunnitelman kanssa koko projektin elinkaaren ajan. Suunnitelma sisältää riskianalyysin ja varasuunnitelman.

Laatuprosessin täytyy olla yhteensopiva käytettävän kehitysmallin kanssa ja hyvä laatuprosessi on yhtenevä ohjelmistoprosessin kanssa. Perinteisessä vesiputousmallissa käytetään V-mallia tai sovellettua V-mallia, esimerkiksi Cleanroom-mallia, jonka IBM esitteli 1980-luvun lopussa. Cleanroom-mallissa laatuprosessi yhdistetään V&V –aktiviteetteihin ja siinä painotetaan ensimmäisten vaiheiden analyysiä. Cleanroom-prosessi käsittää kaksi yhdessä toimivaa ryhmää: kehittämistiimi ja laatutiimi sekä viisi pääaktiviteettia: määrittely, suunnittelu, tekninen suunnittelu ja verifiointi, laatusertifiointi ja palautteiden anto.



Kuva 2.5 Cleanroom –prosessimalli (Pezze & Young, 2008).

Kuva 2.5 esittää Cleanroom-mallia: vaaleat laatikot esittävät kehittämistiimin toimintoja ja tummat laatikot laatutiimin toimintoja. Esimerkiksi määrittelyvaiheessa (Specification) kehittämistiimi määrittelee järjestelmän vaaditun käyttäytymisen ja laatutiimi määrittelee järjestelmän käytön skenaariot, joita myöhemmin käytetään testijoukkojen johtamiseen.

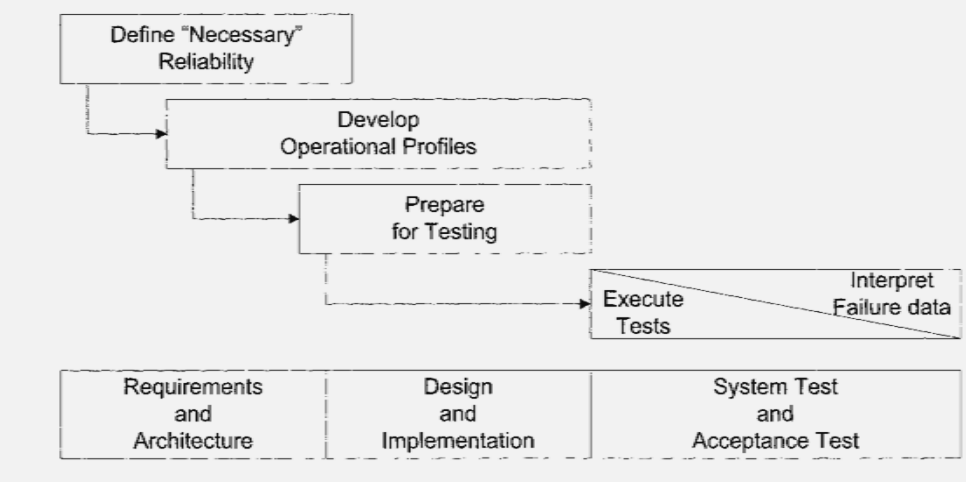
Koska ohjelmistoyrityksen sisällä on paljon samankaltaisia ja samalla sovellusalueella toimivia projekteja, ne voivat kehittää hyödyllisiä, organisaatio-sidonnaisia strategioita. Testaus- ja analyysistrategiat yhdistävät eri projektien yhteiset ominaisuudet ja tuottavat suosituksia, joilla ylläpidetään laatusuunnitelmien yhtenäisyyttä. Ne tarjoavat yleisen kehyksen eri projektien laatusuunnitelmille, täyttävät organisatoriset laatustandardit, edistävät projektien yhtenäisyyttä ja tekevät yksittäisten projektien laatusuunnitelmat tehokkaiksi sekä tehdä että toteuttaa. *Laatustrategia* on jokaisen organisaation yksilöllinen omaisuus, jolla kuvataan joukko ratkaisuja ongelmiin, jotka ovat ominaisia juuri kyseessä olevalle organisaatiolle.

Laatustrategiat voidaan luokitella yrityksen ominaisuuksien mukaan seuraavasti (Pezze & Young, 2008):

1) Rakenne ja koko: suurien organisaatioiden kehitys- ja laaturyhmät ovat tyypillisesti erotettu toisistaan, vaikka testaushenkilöstö olisikin osa kehitystiimejä. Pienemmissä organisaatioissa on yleisempää, että yksi henkilö hoitaa useaa roolia
2) Kokonaisprosessi: laatustrategian on noudatettava kokonaisprosessia; jos organisaatiossa käytetään XP-menetelmää, ’testaa ensin’ ja pariohjelmointi ovat luonnollisia lähestymisen elementtejä, ja raskaat dokumenttipainotteiset lähestymistavat ovat vaikeita soveltaa
3) Sovellusalue: kohdealue vaikuttaa sekä tiettyihin laatutavoitteisiin että joissakin tapauksissa tiettyihin vaiheisiin ja dokumentaatioon

Spiraaliin ohjelmistokehitysmalliin kehitettiin 1990-luvun alussa testauksen lähestymistapa SRET (Software Reliability Engineered Testing, SRET). SRET (kuva 2.6) nimeää kaksi testauksen pää-tyyppiä: kehitystestaus ja varmentamistestaus. SRET–lähestymistapa sisältää seitsemän pääaskelta. SRET-mallin seitsemän ydinaskelta ovat:

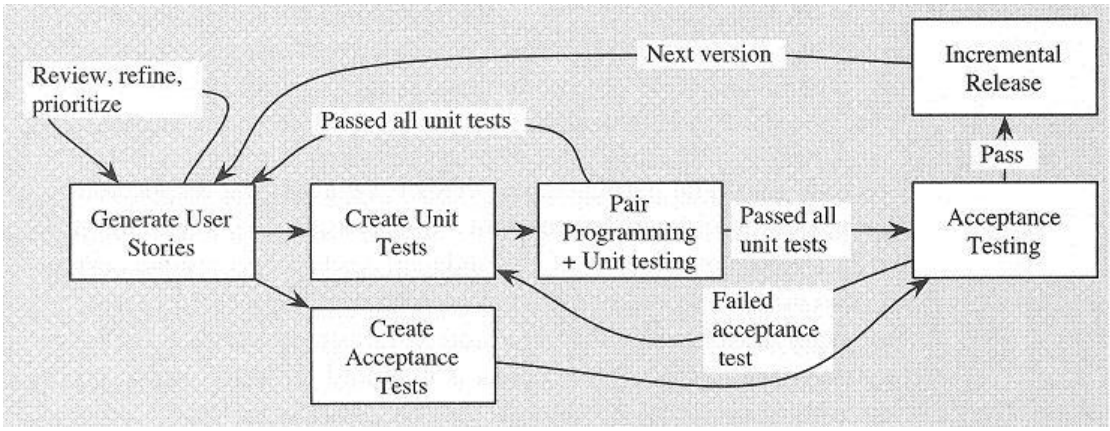
1) määrittele ”tarpeellinen” luotettavuus
2) kehitä toiminnalliset profiilit
3) valmistele testaus
4) suorita testit / tulkitse virheellinen aineisto
5) vaatimukset ja arkkitehtuuri
6) suunnittelu ja toteutus
7) järjestelmätestaus ja hyväksymistestaus



Kuva 2.6 Ohjelmiston luotettavuuden muuntuva testaus **(**Software Reliability Engineered Testing, SRET) (Pezze & Young, 2008).

Kaksi ensimmäistä, nopean päätöksen askelta päättävät mitkä systeemit vaativat erillistä testausta ja millaista testausta tarvitaan missäkin systeemissä. Loput viisi ydinaskelta suoritetaan rinnakkain jokaisessa kehitysspiraalin kierteessä.

XP-mallissa yksikkötesti yhdistetään alijärjestelmä- ja järjestelmätestaukseen; testin suunnittelu ja suoritus paketoidaan jokaiseen inkrementaaliseen kehitysvaiheeseen. Asiakkaan läsnäolo XP-projektissa, käyttäjätarinat ja kehitystyön tekeminen pariohjelmointina ovat laatuprosessin kannalta tärkeitä. Pariohjelmointi on katselmointitekniikka, testitapaukset perustuvat käyttäjätarinoihin ja asiakas on vastuussa jokaisen syklin lopussa tehtävästä hyväksymistestauksesta. Laatuprosessi XP-mallille on kuvattu kuvassa 2.7.



Kuva 2.7 XP:n (Extreme Programming, XP) laatuprosessi (Pezze & Young, 2008).

Yrityksen sisäinen laatustrategia määrittelee miten laatu varmennetaan yrityksessä. Laatustrategia määrittelee projekteille yhteiset ominaisuudet, antaa suositukset laatusuunnitelmien yhtenäisyydelle ja tarjoaa yleisen kehyksen laatusuunnittelulle. Laatustrategia kuvataan yrityksen laatukäsikirjana, jota parannetaan säännöllisillä laatuauditoinneilla.

Analyysi- ja testisuunnitelman pitäisi vastata seuraaviin kysymyksiin(Pezze & Young, 2008):

1) mitä laatutoimintoja suoritetaan
2) mitkä ovat riippuvuudet laatutoimintojen kesken ja laatu- ja kehittämistoimintojen välillä
3) mitä resursseja tarvitaan ja millä tavalla ne allokoidaan
4) miten prosessia ja kehitettävää tuotetta valvotaan, jotta ylläpidetään riittävä laatuarviointi ja laatu- ja aikatauluongelmien aikainen varoitus

Riskianalyysi kuuluu jokaiseen projektiin pakollisena osana. Riskejä ei voi eliminoida, mutta ne voidaan määritellä ja niitä voidaan valvoa ja hallita. Laatusuunnitelman riskianalyysi koskee ensisijaisesti henkilöstö-, teknologia- ja aikatauluriskejä.

Perinteinen V-malli jakaa testauksen neljään vaiheeseen: *yksikkötestaus, integrointitestaus, järjestelmä-, ja hyväksymistestaus*. Integrointivaiheessa integroitavat yksiköt tulee olla testattu niin hyvin, että yksiköiden sisäisiä ongelmia ei ole. Kaikki ongelmat integrointitestauksessa johtuvat yksiköiden keskinäisestä yhteensopimattomuudesta. Perinteisten integrointitestausmenetelmien (top-down, bottom-up, sandwitch) rinnalle on kehitetty uudempia menetelmiä kuten säietestaus ja kriittisen yksikön testaus.

Järjestelmä-, hyväksymis- ja regressiotestaus liittyvät kaikki ohjelmistojärjestelmän käyttäytymiseen, mutta niillä on kaikilla eri tarkoitus. Järjestelmätestaus tarkistaa ohjelmiston ja sen määrittelyn yhteneväisyyden (verifiointi), hyväksymistestaus ohjelmiston soveltuvuuden asiakkaalle (validointi) ja regressiotestaus tarkistaa uudelleen aikaisempien tuoteversioiden testit.

Käytettävyydeltään hyvä ohjelmistotuote on nopeasti opittavissa, mahdollistaa käyttäjän tehokkaan työskentelyn ja on miellyttävä käyttää. Käytettävyys sisältää puolueettomia kriteereitä kuten vaadit-

tava aika tai operaatioiden lukumäärä tehtävän suorittamiseen. Testauksen ja analyysin kannalta on hyödyllistä erottaa vain käytettävyyteen liittyvät ominaisuudet muista ohjelmiston laatuun liittyvistä näkökulmista kuten luotettavuus, suorituskyky ja turvallisuus. Käytettävyyden verifiointi- ja validointiprosessi sisältää seuraavat vaiheet (Pezze & Young, 2008):

1) määrittelyjen tarkastaminen käytettävyyden tarkistuslistojen avulla
2) aikaisten protyyppien testaus loppukäyttäjien avulla
3) inkrementaalisten julkaisujen testaus
4) järjestelmä- ja hyväksymistestaus

Testauksen automatisointi voi parantaa joidenkin laatutoimintojen tehokkuutta. Vaikka suurempi automatisointiaste ei voi korvata järkevää, hyvin organisoitua laatuprosessia, prosessin suunnittelussa ja parantamisessa on tärkeää huomioida automatisointimahdollisuudet. Automatisoinnin roolia voidaan testauksen ja analyysin toiminnoissa tarkastella kolmen ulottuvuuden avulla: aktiviteetin arvo ja sen nykyinen kustannus, määrä, jonka aktiviteetti vaatii tai tulee halvemmaksi automatisoinnilla ja hankittavan tai asennettavan välineen tuen kustannukset.

Kypsä ohjelmistoprosessi sisältää dokumentointistandardit kaikille ohjelmistoprosessin aktiviteeteille, myös testaus- ja analyysiaktiviteeteille. Dokumentaatiolle suoritetaan tarkastuksia tarkoituksena verifioida edistymistä aikataulua ja laatutavoitteita vasten sekä tunnistaa ongelmia tukemalla prosessin näkyvyyttä, valvontaa ja toistettavuutta. *Analyysin ja testauksen dokumentointi* on merkittävä osa ohjelmistonkehitysprosessia sisältäen laatuprosessin. Täydelliset ja hyvin rakennetut dokumentit lisäävät testipakettien uudelleenkäyttöä. Dokumentit ovat välttämättömiä tietämyksen ylläpidossa ja yhtenäiset dokumentit tarjoavat perustan prosessin valvonnalle ja arvioinnille. Dokumentaatio sisältää lisäksi yhteenvedon ja nykyisen tiedon, jotka muodostavat pohjan prosessin parantamiselle.

Dokumentit jaetaan kolmeen kategoriaan: suunnittelu, määrittely ja raportointi. Suunnitteludokumentit kuvaavat laatuprosessin organisaation, sisältävät osaston tai yrityksen strategian ja suunnitelman ja suunnitelmat yksittäisille projekteille. Määrittelydokumentit kuvaavat testipaketit ja testitapaukset. Täydelliset analyysi- ja testimäärittelydokumentit sisältävät testaussuunnittelun määrittelyn, testitapausten määrittelyn, tarkistuslistan ja analyysiproseduurin määrittelyn. Raportointidokumentit käsittävät yksityiskohtaiset analyysi- ja testitulokset sekä yhteenvedon niistä (Pezze & Young, 2008).

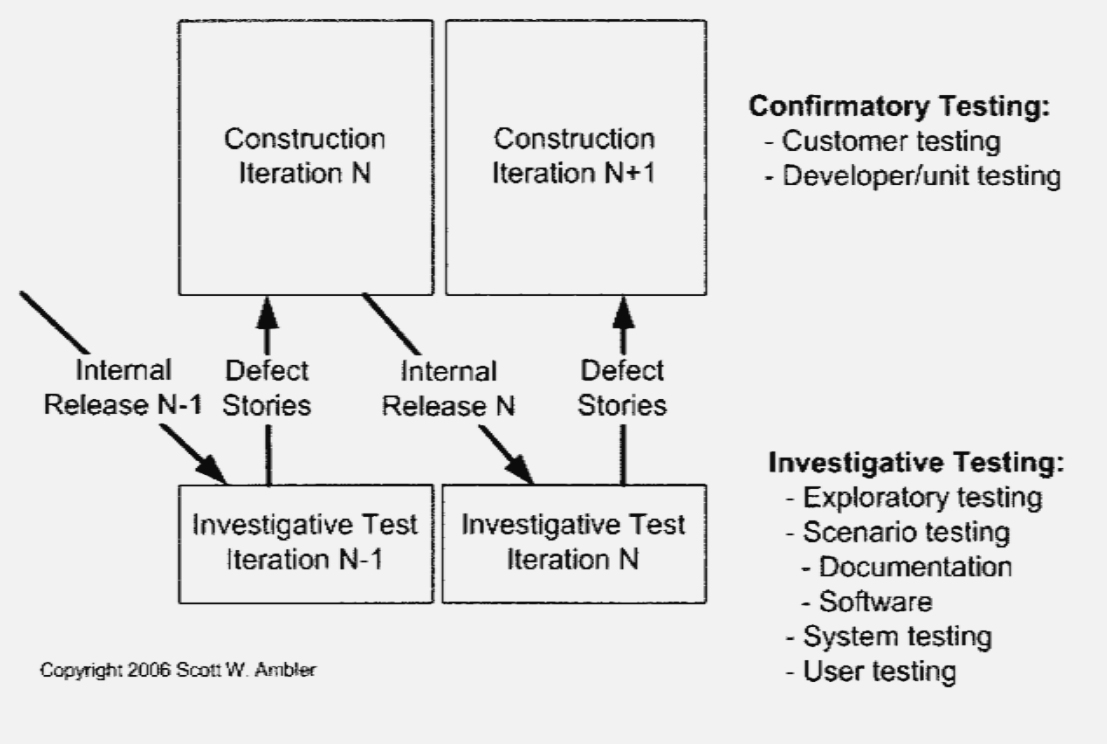
2.3 Testauksen uusia käytäntöjä

Testauskäytännöt ovat kehittyneet huomattavasti viime vuosina. Joissakin tapauksissa testaus liittyy niin kiinteästi kehittämisprosessiin, että on kysymyksessä uusi ajattelutapa ja kehittämismenetelmä. Mallipohjaiset menetelmät perustuvat testitapausten generointiin mallien avulla. Ketterien menetelmien testausmenetelmiä ovat muun muassa tutkiva testaus ja automatisoidut yksikkö- ja hyväksymistestit.

2.3.1 Testiohjattu kehittäminen

Testiohjatun kehittämisen (Test Driven Development, TDD) tavoite on Amblerin (2007) mukaan tuottaa määrittely ilman validointia ja puhdasta, toimivaa koodia. Testi suunnitellaan ennen toteutusta ja näin se voidaan ajatella ohjelmointitekniikaksi, joka korostaa yksikkötestausta. Koska testiohjattu kehittäminen ei koske vain testausta, vaan määrittelyä, suunnittelua ja testausta, se on uusi ajattelutapa ohjelmiston kehittämisestä.

Testiohjattu kehittäminen kääntää perinteisen ohjelmiston kehittämisen ylösalaisin: sen sijaan, että kirjoitetaan toiminnallinen koodi ensin ja siihen sopiva testi jälkeenpäin, testi kirjoitetaan ensin ja vasta sen jälkeen toiminnallinen koodi. Testiohjattu kehittäminen tehdään kahden säännön mukaan: uutta koodia ei kirjoiteta ennen kuin automatisoitu testi epäonnistuu ja kaikki löydetyt duplikaatit on poistettu. Se on ensisijaisesti suunnittelutekniikka, mutta sen sivutuotteena saadaan varmuus perusteellisesti suoritetusta yksikkötestauksesta. Ohjelmistotuotteen kokonaisvaltaiseen testaamiseen tarvitaan todennäköisesti lisäksi muita testaustekniikoita, esimerkiksi tutkivaa testausta ja hyväksymistestausta (kuva 2.8).



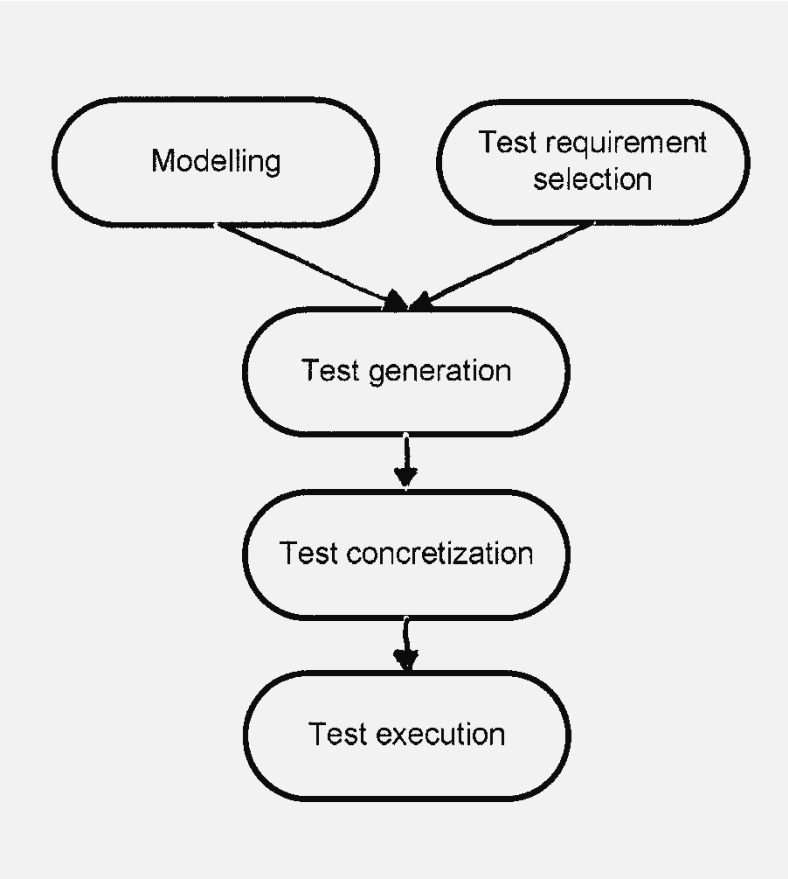
Kuva 2.8 Ketterä testaus (Ambler, 2007).

Testiohjattu kehittäminen on tullut suosituksi menettelyksi ketterissä menetelmissä, niin ohjelmistokehityksen ja -koodauksen kuin tietokannan kehityksessä. Testiohjattu kehittäminen ei korvaa perinteistä testausta. Sen sivutuotteena saadaan testit, jotka ovat dokumentoivia ja määrittelevät toimivan koodin. Ne toimivat perustana dokumentoinnille ja määrittelylle.

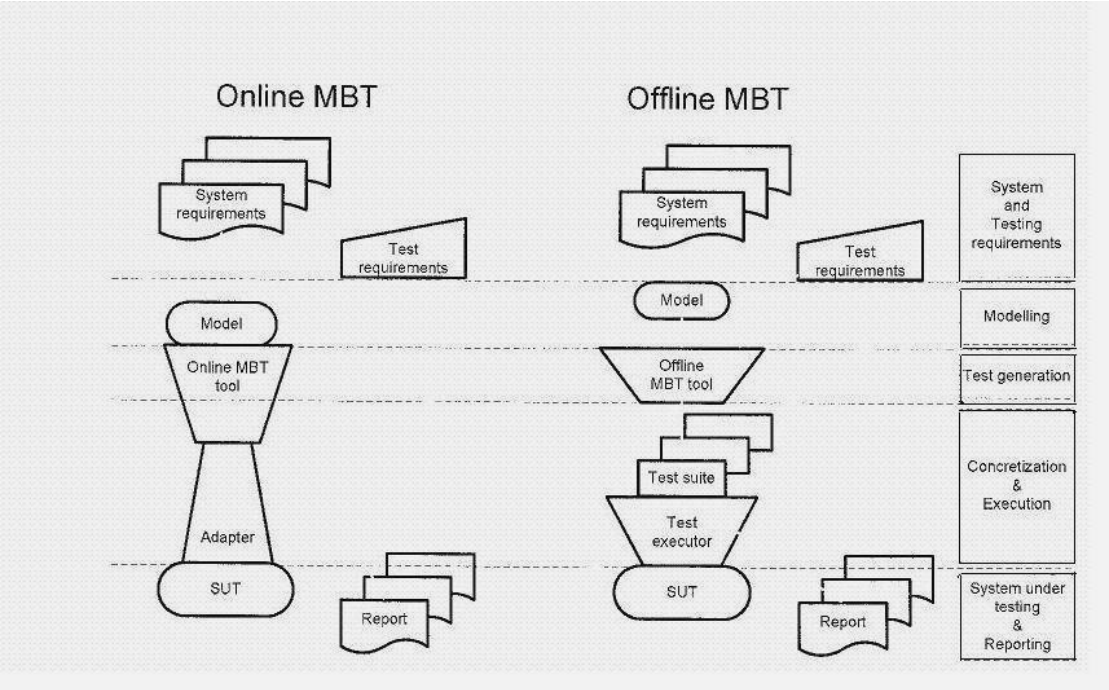
2.3.2 Mallipohjainen testaus

Puolitaival (2008) esittää, että *mallipohjainen testaus* (Model-based Testing, MBT) on ohjelmistotestausta, jossa testitapaukset generoidaan kokonaan tai osittain mallin avulla, joka kuvaa testattavan järjestelmän (yleensä toiminnallisia) ominaisuuksia. Kuvassa 2.9 esitetään mallipohjaisen testauksen perusperiaate.

Mallipohjaisen testauksen hyötyjä ovat muun muassa testipaketin helppo ylläpito, automaattinen testaussuunnittelu ja parempi testauksen laatu. Mallipohjainen testiautomaatioratkaisu koostuu kolmesta komponentista: mallinnusformalismista, analyysi- ja suorituslogiikasta ja liitynnöistä testattavaan järjestelmään. Mallipohjaisessa testauksessa on kaksi päälähestymistapaa: Online MTB ja Offline MTB. Ne eroavat toisistaan testiajon ajankohdan suhteen kuten kuvassa 2.10 näkyy.



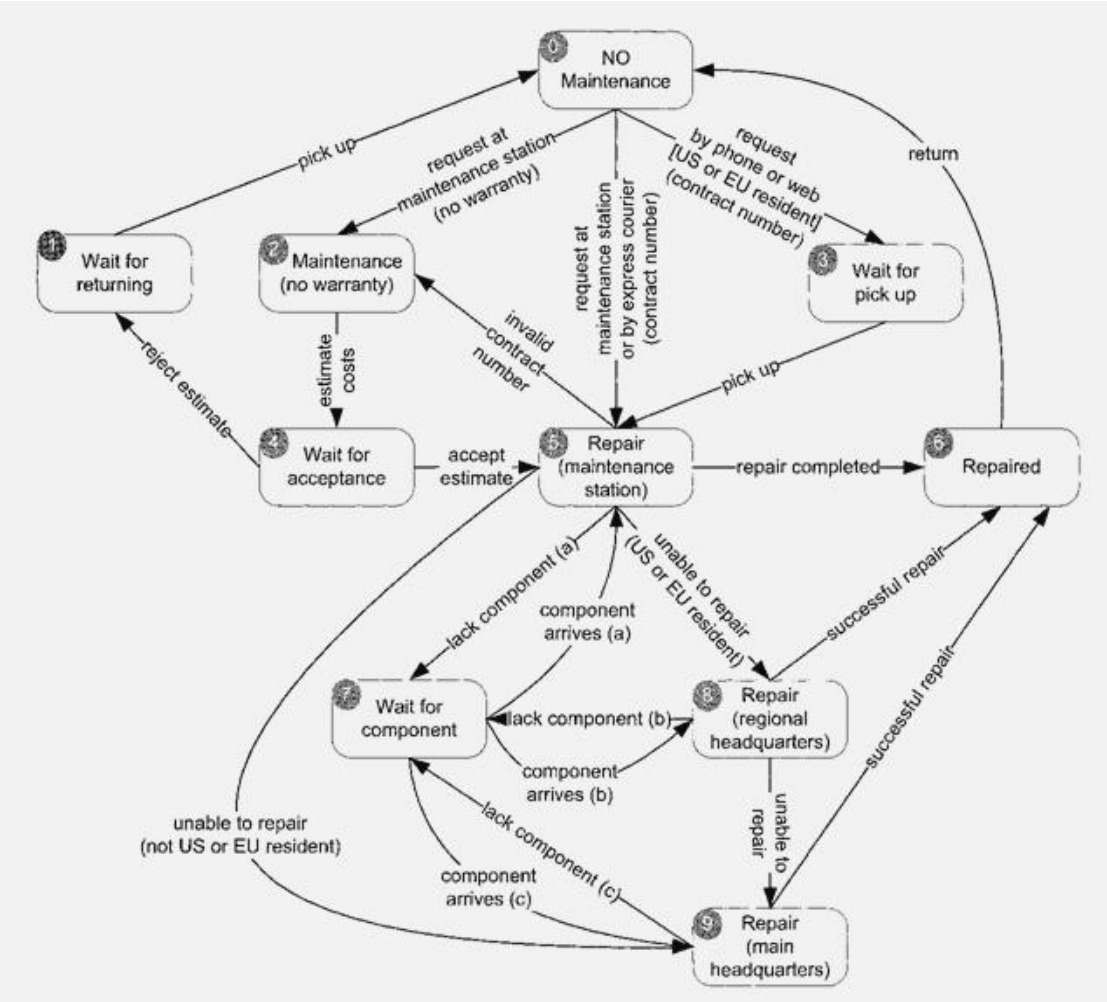
Kuva 2.9 Mallipohjainen testaus (Puolitaival, 2008).



Kuva 2.10 Online MBT ja Offline MBT (Puolitaival, 2008).

Testimallia käytetään mallipohjaisessa testiautomaatiossa kuvaamaan testattava kohde ja sen käyttäytyminen siltä osin kun se on testaamisen kannalta olennaista. Testimallin tulee sisältää tiedon järjestelmän erilaisista mahdollisista tiloista ja niistä ehdoista tai lainalaisuuksista, jotka säätelevät tilasta toiseen siirtymistä. Mallin voi rakentaa käyttämällä erilaisia kuvaustapoja; käytännössä olisi edullista, jos mallinnusformalismi olisi jokin ennestään tuttu tai jo olemassa olevia määrityksiä voitaisiin käyttää. Tuotteen määrittelyiden ja testimallien välillä on kuitenkin merkittävä ero: arkkitehtuurimalli kuvaa tuotteen sisäistä rakennetta ja sen jakoa osakokonaisuuksiin, kun taas testimalli kuvaa tuotteen ja sen ympäristön välistä rajapintaa ja tuotteen liittymää todellisuuteen. Tästä johtuen testiaineiston generoiminen suoraan määrittelytiedoista ei ole kovin suositeltavaa. Testimalli sisältää tietouden siitä kuinka, ja millaisilla lainalaisuuksilla, testattavan järjestelmän tulisi toimia ulkoapäin katsottuna (Siegberg, 2005).

Formaalit mallit tarjoavat tarpeeksi informaatiota testitapausten generointiin; formaaleja malleja ovat esimerkiksi äärelliset automaatit, päätöstaulut ja kieliopit; puoliformaalit mallit vaativat ihmisen arvostelukykyä ennen testitapausten generointia; puoliformaaleja malleja ovat esimerkiksi luokkakaaviot, tietovuokaaviot ja laajennetut käyttötapaukset. Äärellisiä automaatteja käytetään usein määrittelemään järjestelmän ja ympäristön yhteistyön peräkkäisiä tapahtumia. Äärelliset automaatit muodossa tai toisessa ovat yleisiä reaktiivisissa järjestelmissä tai valvontajärjestelmissä, joissa tilojen määrä on usein rajoittamaton. Kuvassa 2.11 on esimerkki äärellisestä automaatista. Esimerkissä on kuvattu erään järjestelmän tuotteen huolto-toiminnallisuuden eri tilat ja niiden vaihtuminen. Määrittelyt esitetään usein päätösrakenteina, kuten joukkona syötearvojen ehtoja ja vastaavat toiminnot tai tulokset. Päätösrakennetta voidaan käyttää testitapausten valinnassa ja ne voivat paljastaa ristiriitaisuuksia koodin ja päätösrakenteen välillä (Pezze & Young, 2008).



Kuva 2.11 Esimerkki äärellisestä automaatista (Pezze & Young, 2008).

Suurien testiaineistomäärien tuottaminen on Pezze & Youngin (2008) mukaan mallipohjaisen testiautomaation suurin hyöty. Mallipohjaiset testiautomaatioratkaisut ovat vaativia analysoinnin ja suorituslogiikan suhteen. Analysaatio- ja suorituslogiikan tehtävänä on muun muassa analysoida testimalli ja todeta sen mahdollistamat testitapahtumat ja –aineisto sekä tuottaa mallin avulla syöttöaineistoa testattavalle järjestelmälle. Lisäksi logiikan on hoidettava toteutuneiden ja suoritettujen testien ja niiden lopputulosten kirjanpito sekä seurattava testikattavuutta ja verrata saavutettua kattavuutta testiavaruuteen.

Mallipohjainen testiautomaatioratkaisu liitetään reaalimaailmaan testausrajapinnan avulla. Testaus voidaan suorittaa joko reaaliajassa tai eräajona jonkin ohjelmointikielen kautta. Vaikka rajapinta on järjestelmäkohtainen, sen tulisi olla testaamisen kannalta merkityksellinen. Mallipohjaisessa testiautomaatiossa on mahdollista toteuttaa järjestelmän logiikkaa lähellä olevia rajapintoja kun taas perinteinen käyttöliittymäpohjainen testiautomaation rajapinta rajoittuu sovelluksen käyttöliittymään ja sen kontrolloimiin toiminnallisuuksiin. Mallipohjaisen testiautomaation kytkentä testattavaan järjestelmään toteuttaa muun muassa seuraavat toiminnot; testityökalun tuottamien testiaineistojen muuntaminen järjestelmän ymmärtämään muotoon, testien vaatimien syötteiden välittäminen testityökalulta järjestelmälle, järjestelmän vastausten muuntaminen testityökalun ymmärtämään muotoon ja järjestelmän vastauksien välittäminen testityökalulle.

Testauksen suunnittelussa on huomioitava, että mitä aikaisemmassa vaiheessa uusi menetelmä otetaan huomioon ja mitä tarkemmin tavoitteet asetetaan, sitä paremmin ne pystytään saavuttamaan. Lisäksi testausautomaatiotyökalut vaativat henkilöstöltä erilaisia taitoja ohjelmoinnista järjestelmän tuntemukseen, toisaalta automatisoimalla testauksen suoritus- ja raportointitöitä mallipohjainen testaus vapauttaa testaushenkilöstöä korkeamman tason suunnittelu- ja analyysitehtäviin. Mallipohjaisen testiautomaation käyttöönotto merkitsee organisaatiolle panostuksia eri osa-alueilla, esimerkiksi teknologiassa, osaamisen kehittämisessä ja laadunvarmennuksessa (Siegberg, 2005).

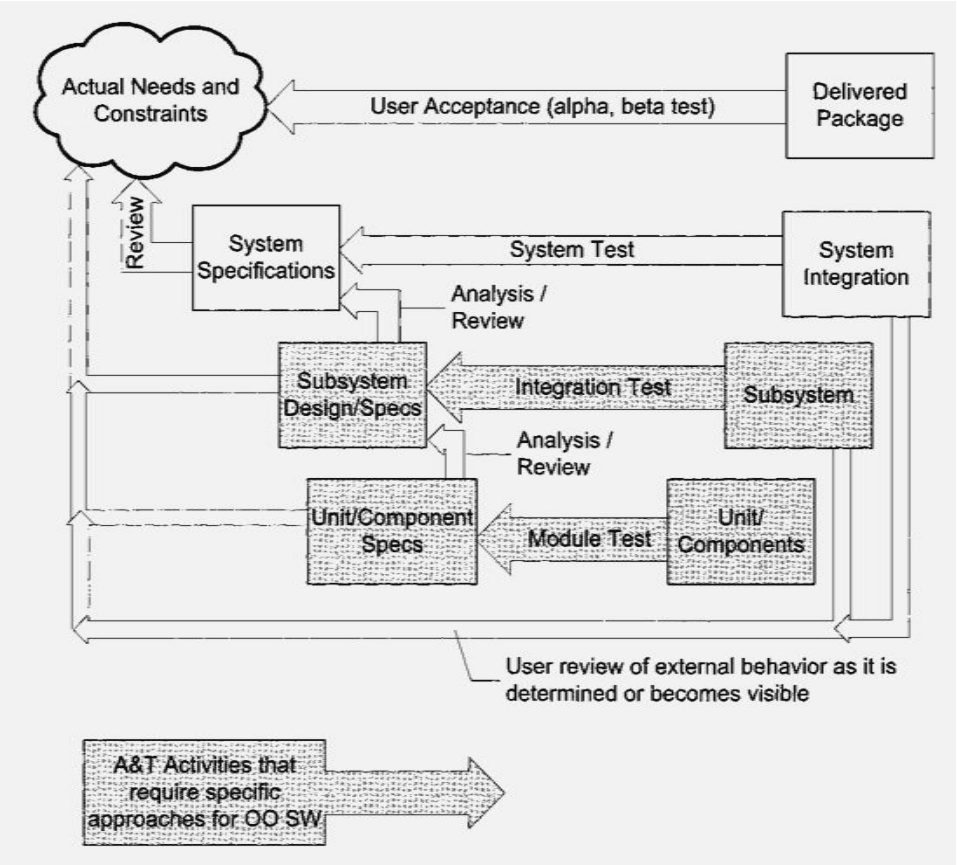
2.3.3 Oliopohjainen testaus

Oliopohjaisen ohjelmiston systemaattinen testaus on Pezze & Youngin (2008) mukaan pohjimmaltaan samanlaista kuin proseduraalisen ohjelmiston systemaattinen testaus: testaus aloitetaan toiminnallisella testauksella, joka perustuu määrittelyihin tai odotettuun käytökseen, testaukseen lisätään ohjelmiston rakenteeseen perustuvia testitapauksia ja testausta jatketaan yksikkötestauksesta ja vähitellen kasvavasta integrointitestauksesta kohti järjestelmätestausta. Silti proseduraalisen ja oliopohjaisen ohjelmiston erot riittävät erilaisten tekniikkojen lähestymistapoihin. Metodit ovat oliopohjaisessa ohjelmistossa lyhyempiä kuin proseduurit proseduraalisessa ohjelmistossa, mutta toisaalta metodien välillä on enemmän sidontaa.

Oliopohjaiselle ohjelmistolle sopivat kaikki proseduraalisen ohjelmiston testaus- ja analyysitekniikat, koska olioparadigma sisältää proseduraalisen paradigman. Testauksen suunnittelussa on huomioitava oliopohjaisen ohjelmiston seuraavat erityispiirteet (Pezze & Young, 2008):

1) Tilariippuva käytös: jokaisella oliolla on tila ja se on huomioitava testauksessa
2) Tiedon piilotus: jokainen metodi tulee voida testata niin, että vastaavan olion rakenne on näkyvillä
3) Perintä: luokalla voi olla perinnän kautta suuri määrä perittyjä ominaisuuksia
4) Polymorfismi ja dynaaminen sidonta: kaikki mahdolliset sidonnat on testattava ja mahdollisesti sidontojen kombinaatiot
5) Abstraktit luokat ja rajapinnat: abstrakteja luokkia ei voi suoraan testata, mutta ne on välttämätöntä testata edes puutteellisella tietämyksellä
6) Poikkeuskäsittely: kaikki poikkeukset ja poikkeuskäsittelijät tulee testata
7) Rinnakkaisuus: rinnakkaisuuden haasteita ovat resurssien käytön kilpailutilanteet, skedulointi ja deadlock-tilanteet

Erityispiirteiden vaikutuksia testaukseen ja analyysiin on kuvattu kuvassa 2.12.



Kuva 2.12 Oliopohjaisen suunnittelun ja koodauksen vaikutus analyysiin ja testaukseen (Pezze & Young, 2008).

2.3.4 Tutkiva testaus

Ketterien ohjelmistotuotantomenetelmien myötä ohjelmiston *tutkiva testaus* (Exploratory Testing) on saavuttanut suosiota. Tutkiva testaus on vastakohta käsikirjoitetulle testaukselle. Se voidaan määritellä seuraavasti (Bach, 2002): **t**utkiva testaus on samanaikaisesti oppimista, testien suunnittelua ja testien suorittamista.

Kanerin (2008a) mukaan testaus on informaation etsintää. Erilaiset tavoitteet ja päämäärät vaativat erilaisia testausmenetelmiä ja tuottavat erilaisia testejä, erilaisia testidokumentteja ja erilaisia testituloksia. Ohjelmistojen testauksen suunnittelun ja suorittamisen tavoitteena on saada hyödyllistä tietoa tuotteen laadusta.

Tutkiva testaus on tapa testata ohjelmistotuotetta. Se ei ole testaustekniikka vaan lähestymistapa testaukseen ja kaikki testaaminen sisältää jossain määrin tutkivaa testausta. Se painottaa yksittäisen testaajan yksilöllistä vapautta ja vastuuta: työn arvon kasvattaminen jatkuvasti tapahtuu testaamalla oppimisen, testauksen suunnittelun, testauksen suorittamisen ja testitulosten tulkinnan avulla sekä muiden projektissa samanaikaisesti tapahtuvien tukitoimintojen avulla.

Oppimista on kaikki se, mikä voi opastaa seuraavissa ongelmissa: mitä testata, miten testata ja kuinka tunnistaa ongelman. Suunnittelulla tarkoitetaan luomista ja muokkaamista, ei käsikirjoitta-

mista. Suorittaminen on testin tekeminen ja tulosten kerääminen; suorittaminen voi olla automatisoitua tai manuaalista. Tulkinta on sitä, mitä me opimme tuotteesta ja tuotteen testaamisesta.

Tutkivaa testausta sovelletaan Itkosen ja Rautiaisen (2005) mukaan teollisuudessa muun muassa seuraavista syistä: testitapauksia on hankala suunnitella monimutkaiselle toiminnallisuudelle ja testauksen tarve loppukäyttäjän näkökulmasta. Testauksen hyötyjä ovat testauksen monipuolisuus ja mahdollisuus hahmottaa nopeasti kokonaiskäsitys järjestelmän laadusta. Puutteena on testauksen kattavuuden hallinta. Itkosen ja Rautiaisen (2005) mukaan voidaan johtaa viisi ominaisuutta, jotka kuvaavat tutkivaa testausta:

1) testausta ei ole suunniteltu etukäteen yksityiskohtaisina testiskripteinä tai testitapauksina; sen sijaan, tutkiva testaus on tutkimusta yleisen näkemyksen mukaan ilman erityistä askel-askeleelta –käskyjä

2) tutkivaa testausta opastaa aikaisemmin suoritettujen testien tulokset ja niiden avulla saavutettu tietämys; tutkiva testaaja käyttää mitä tahansa saatavilla olevaa tietoa testauksen kohteesta, esimerkiksi määrittelydokumentteja, käyttäjän opasta tai jopa markkinointiesitettä

3) pääpaino tutkivassa testauksessa on virheiden löytämisessä tutkimisen avulla kattavan testitapausjoukon tuottamisen sijaan

4) tutkiva testaus on samanaikaisesti oppimista testattavasta järjestelmästä, testauksen suunnittelua ja testauksen suorittamista

5) testauksen tehokkuus perustuu testaajan tietämykseen, taitoihin ja kokemukseen

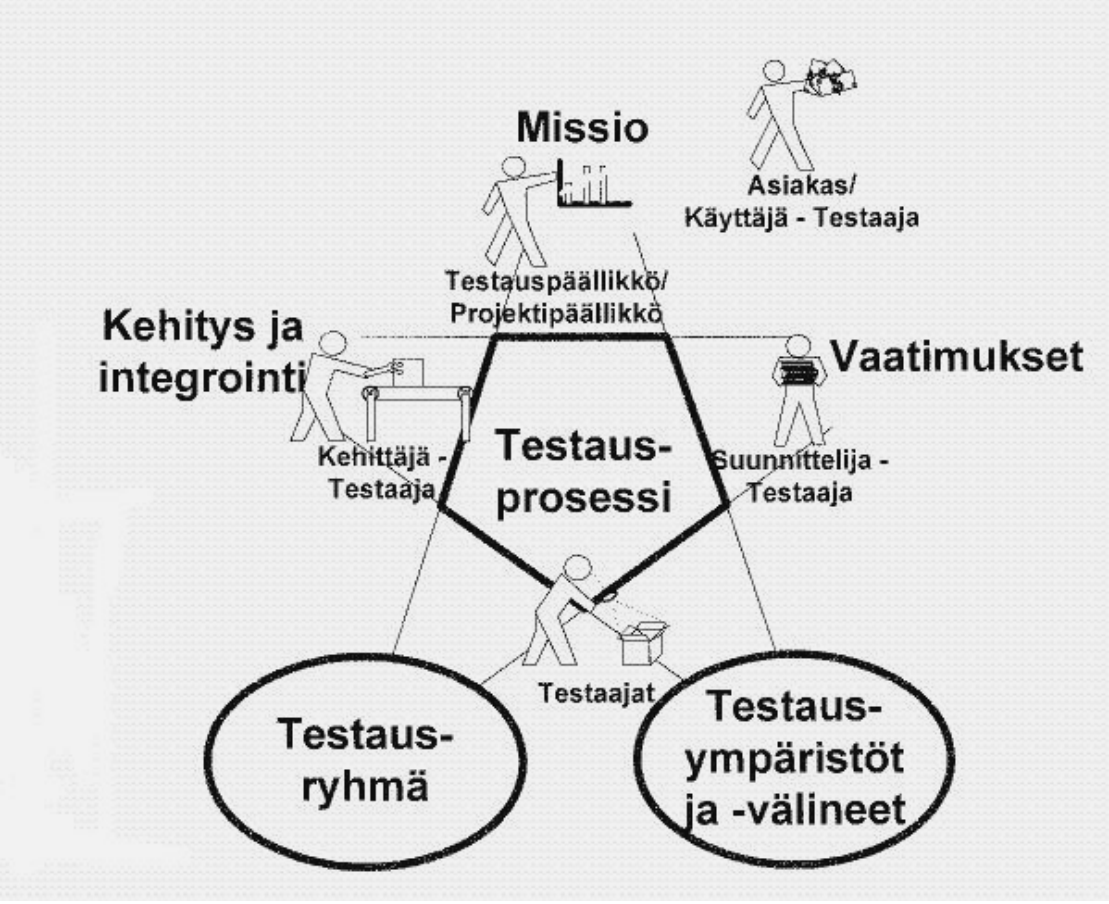
Tutkiva testaus soveltuu tilanteisiin, joissa vaaditaan nopeaa palautetta tuotteesta tai tuotteen nopeaa oppimista tai joissa ei ole tarpeeksi aikaa systemaattisen testauksen lähestymistavalle. Tutkiva testaus on myös hyvä tapa tutkia tiettyjen riskien tilaa ja se tarjoaa monipuolisuutta käsikirjoitettuihin testeihin. Lisäksi virheraportteihin perustuvaa regressiotestausta voidaan suorittaa tutkivana testauksena ja se soveltuu hyvin testauksen suorittamiseen käyttäjän näkökulmasta.

Tutkivan testauksen hyötyjä ovat tehokkuus, oppiminen, valmistelevien dokumenttien määrän minimointi, testauksen suoritus ilman täydellisiä määrittelydokumentteja ja nopea palaute. Puutteena voidaan nähdä vaikeus seurata yksittäisten testaajien ja koko testauksen edistymistä ja testauksen kattavuutta. Tutkivan testauksen avulla ei voida myöskään estää virheitä.

3 OHJELMISTOJEN TESTAUKSEN KEHITTÄMINEN

Ohjelmistojen testauksen kehittäminen on kehitysprosessi organisaatiossa. Nykytilan kartoitus, tavoitetilan asettaminen ja kehittämisaktiviteettien määrittäminen vaativat määrätietoista suunnittelua, sitoutumista, toteutusta ja ohjausta. Tavoitteiden asettamisessa kehittäminen tulee rajata: on päätettävä onko kysymyksessä toiminnan kehittäminen yleensä vai tietyn rajatun alueen, esimerkiksi testausprosessin kehittäminen. Prosessinkehitys koostuu tehtäväsarjoista, jotka koostuvat syklisesti, kun taas koko testaustoiminnan kehittäminen on huomattavasti laajempi käsite.

Testauksen tutkimuksessa voidaan nykyään erottaa useita koulukuntia ja näin ollen myös testauksen kehittämiseen on olemassa useita näkökulmia. Testauksen kehittämistä voi lähestyä esimerkiksi prosessinäkökulmasta, kontekstinäkökulmasta tai testaamisen hallinnan ja johtamisen näkökulmasta. Lisäksi testauksella on monia sidosryhmiä: määrittelemällä mitä testaus on ja mitä se ei ole voidaan saada hyvin erilaisia vastauksia riippuen miltä sidosryhmältä kysytään (kuva 3.1).



Kuva 3.1 Testauksen tilannemalli (Pyhäjärvi & Pöyhönen, 2006).

Testausta on hyvä lähestyä siitä näkökulmasta, että se ei itsessään tuota mitään, vaan auttaa muita tuottamaan parempaa tehokkaammin ja halvemmalla. Kehittämällä testausta vaikutetaan siis muuhunkin kuin itse testaukseen. Testausta voi kehittää sisällön, osaamisen, hallinnan ja tekemistapojen osalta. On erilaisia asiakkaita ja eri asiakkailla erilaisia tavoitteita. Testauksen kehittäminen helpottaa tavoitteiden saavutettavuutta ja kustannusten hallintaa. Testauksen ja etenkin testausprosessin kehittämiseen on kehitetty useita malleja ja menetelmiä.

3.1 Näkökulmia ohjelmistojen testauksen kehittämiseen

Uusien sovellusten komponenttipohjaisuus, hajautus sekä monimutkaisuus lisäävät testauksen haasteellisuutta. Lisäksi integroinnin haasteet, puutteelliset määrittelydokumentit sekä monitasoinen toiminnallisuus hankaloittavat testausta. Testauksen muodostuessa yhä tärkeämmäksi ohjelmistotuotannon alueeksi, sen kehittämiseen ja parantamiseen on alettu kiinnittää yhä enemmän huomiota.

Aluksi kehittämisehdotukset perustuivat olemassa oleviin testauskäytäntöihin ja -menetelmiin, toisin sanoen 1980-luvulla tai jopa aikaisemmin kehitettyihin testauksen ratkaisuihin. Kehittämisen kohteena nähtiin usein testausprosessi ja prosessinäkökulmaa tutkittiin etenkin 1990-luvulla.

Testausprosessi on sidottu myös käytettävään ohjelmiston kehittämismenetelmään sekä käyttökontekstiin. Tästä johtuen perusmallia joudutaan usein muokkaamaan tiettyyn organisaatioympäristöön, ohjelmistoprosessiin ja sovellusalueeseen sopivaksi. Minimaalinen testauskäytännön kehys on kehitetty pienille tai keskisuurille, mutta voimakkaassa kasvuvaiheessa oleville yrityksille.

3.1.1 Prosessinäkökulma

Testaus perustuu Kitin (1995) kriittisiin valintoihin, jotka voidaan asettaa vastaamalla kysymyksiin mitä, milloin ja miten. Koska kaikkea on mahdoton testata, on varmistuttava siitä, että testataan oikeita asioita. Yksi peruste valita on riskiin pohjautuvat testivalinnat: mitä riskialttiimpi ohjelmiston osa on, sitä varmemmin se tulee testata. Toinen peruste testivalinnalle on käyttötiheys: mitä enemmän kyseistä osiota käytetään, sitä suurempi on virheiden esiintymismäärä. On järkevää painottaa myös niitä ohjelmiston osia, joissa löytyy suurella todennäköisyydellä virheitä. Näitä ovat esimerkiksi ratkaisut, joita on tehty resurssien vähyyden vuoksi. Milloin -kysymykseen voidaan vastata yksinkertaisesti: mahdollisimman aikaisin. Näin estetään virheen siirtyminen tasolta toiselle, mikä säästää kustannuksia.

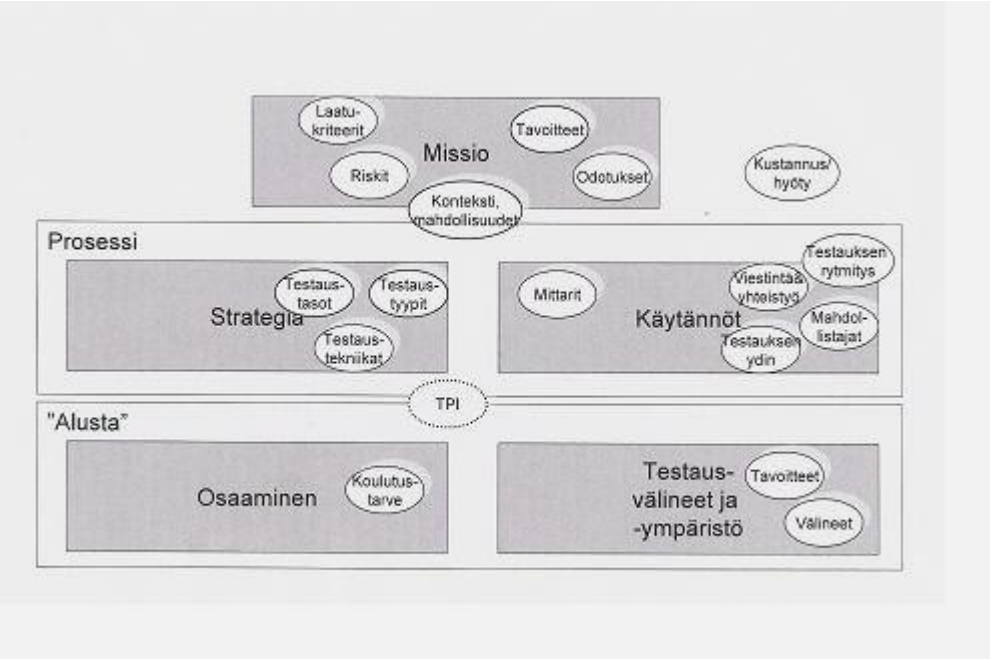
Testauksen kehykseen sisältyvät kriittiset kohdat ovat: suunnittelu, ohjelmistotuotannon kypsyystasot, kokoonpanon hallinta, standardit, muodolliset dokumentit, testausmateriaali ja -dokumentit, mittaaminen ja välineet. Suunnittelu sisältää sekä verifiointi- että validointisuunnittelun. Mitä alhaisempi kypsyystaso ohjelmistolla on, sitä enemmän testauksessa kuluu aikaa muuhun kuin testauksen tekemiseen. Kokoonpanon eli konfiguroinnin hallinnalla tarkoitetaan koko ohjelmiston kaikkien osien hallintaa. Standardit, muodolliset dokumentit ja testausdokumentit liittyvät dokumentaatioon. Mittaamisen avulla saadaan selville kehityksen suunta ja oikein valituilla työkaluilla ja välineillä saadaan nostettua testauksen tehokkuutta.

Laatuprosessin valvonta on sitä tehokkaampaa mitä näkyvämpi prosessi on; prosessin eteneminen ja toimintojen kehitys, tulosten ja aikataulun seuranta sekä poikkeamien tunnistaminen laatusuunni-

telmassa kuuluvat laatuprosessin valvontaan. Monet usein toistuvat virheet johtuvat prosessin ja kehityksen puutteista. Prosessin ja ympäristön parantamistoimenpiteet johtavat usein parempaan tuotteeseen. Prosessin heikkojen kohtien tunnistaminen voi olla kuitenkin vaikeaa ja prosessianalyysin tulokset yllättäviä. Root Case Analysis (RCA) on tekniikka, jolla tunnistetaan ja poistetaan prosessivirheitä. Se koostuu neljästä pääaskeleesta: mitä, milloin, miksi ja miten. Ne ovat merkittävimmät virheluokat ja niiden perusteella voidaan selvittää alkuperäiset syyt. Ensimmäisen askeleen tavoite on tunnistaa merkittävien virheiden luokka. Virheet luokitellaan vakavuuden ja tyypin perusteella; virheiden vakavuus luonnehtii virheen vaikutusta tuotteeseen. Vaikka eri menetelmät käyttävät hieman erilaisia asteikkoja ja termejä, kaikki niistä tunnistavat muutamia vakiotasoja (Pezze & Young, 2008).

RCA-tekniikan lähestymistapa virheiden luokitteluun ei käytä etukäteen määriteltyä luokkien joukkoa. RCA-tekniikan tavoite ei ole verrata erilaisia virheluokkia tai analysoida ja eliminoida kaikkia mahdollisia virheitä vaan tunnistaa muutama tärkeimmistä virheluokista ja poistaa niiden syyt. Onnistunut RCA-sovellus poistaa jatkuvasti tärkeimpien virheiden syitä mistä johtuen virheet menettävät vähitellen merkitystään. Lisäksi virheluokkien tunnistamisen tarkkuus riippuu projektista ja prosessista (Pezze & Young, 2008).

Testausprosessin suunnittelussa on huomioitava suuri määrä erilaisia sidosryhmiä ja niiden tarpeita. Kuvassa 3.2 voidaan nähdä mistä testausprosessi koostuu (Pyhäjärvi & Pöyhönen, 2006).



Kuva 3.2 Mistä testausprosessi koostuu (Pyhäjärvi & Pöyhönen, 2006).

3.1.2 Kokemukseen ja käyttökontekstiin perustuva näkökulma

Testaus voidaan nähdä Bach & al. (2002) mukaan virheiden asianajona: miten ja missä muodossa löydetyt virheet esitetään, vaikuttaa ratkaisevasti niiden korjaamiseen. Testaus yhdistää tekniikoita, joiden painotus on testaajissa (who), kattavuudessa (what), potentiaalisissa ongelmissa (why), aktiviteeteissa (how) ja arvioinnissa (how to tell). Testaustekniikoiden luokittelumenetelmänä voidaan

ajatella viisinkertaista testausmenetelmää, johon sisältyvät kaikki edellä mainitut osa-alueet: kaikki testaus sisältää kaikkia viittä osa-aluetta.

Testaajan rooli ohjelmistotuotannossa on monipuolinen ja tärkeä. Koska testaamisen tarkoituksena on saada tietoa ja tähän tietoon perustuvat monet projektista tehtävät kriittiset päätökset, testaajaa voidaan ajatella projektin valoina ja kartalle merkitsijänä. Testaus on myös palvelurooli. Testaaja palvelee monia asiakkaita: projektipäällikköä, ohjelmoijia, teknistä suunnittelijaa, teknistä tukea, markkinointia, ylintä johtoa ja osakkeenomistajia sekä käyttäjää. Hyvät testaajat ajattelevat teknisesti, luovasti, kriittisesti ja käytännöllisesti.

Testauksen automatisointia tulee käyttää vasta, kun se edistää testaamisen tehtävää. Testaus kannattaa suunnitella ensin, ennen kuin päättää automatisoinnista. Muuten on vaarana automatisoida osia, jotka ovat helppoja automatisoida, mutta heikkoja virheen löytämisessä. Automatisointistrategia on perustuttava kontekstiin eli testausvaatimuksiin, ohjelmistotuotteen arkkitehtuuriin ja testaushenkilöiden taitoihin. Automatisoinnissa on myös muistettava, että automatisointi ei ole testausstrategia; testauksen automatisointi on ohjelmistokehitysprosessi ja merkittävä sijoitus. Lisäksi testauksen automatisointiprojektit vaativat taitoja ohjelmoinnissa, testauksessa ja projektin johtamisessa. Testausstrategia on valintojen joukko ja yhteys testaamisen ja tehtävän välillä. Testausstrategian suunnittelussa on otettava huomioon seuraavat kolme näkökulmaa (Bach & al., 2002):

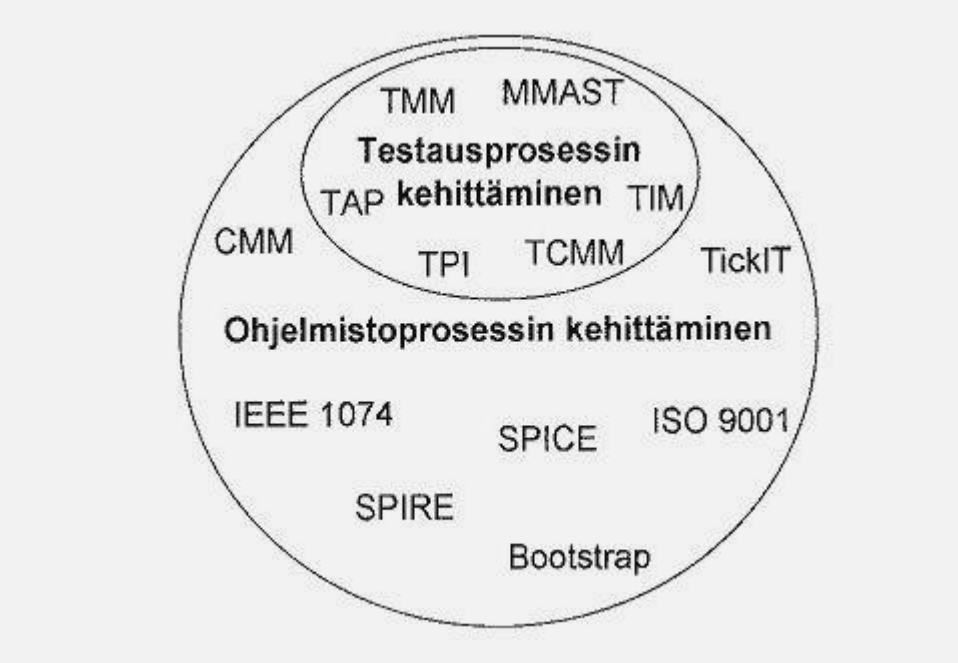
1) testaaminen on kallista; strategiaan ei tule sisällyttää aktiviteetteja ilman, että ne osoittavat tarpeeksi merkityksellisiä riskejä

2) strategiaan ei tule sisällyttää aktiviteetteja ilman, että ne merkitsevät jollekin jotakin

3) jotkut strategiat ovat paljon helpompia sanoa kuin tehdä; on päätettävä, kuinka paljon aiotaan tehdä

3.2 Testauksen kehittämisen menetelmiä

Testausprosessin kehittäminen on osa ohjelmistoprosessin kehittämistä. Testauksen ja testausprosessin kehittämiseen on kehitetty useita erilaisia menetelmiä (kuva 3.3). Testauksen parantamiseen kehitettiin 1990-luvun puolivälissä Hollannissa ja Belgiassa *testauksen hallinnan lähestymismalli* (Test Management Approach, TMap) ja *testausprosessin parantamismalli* (Test Process Improvement, TPI). TMap on tarkoitettu testausprosessin parantamiseen ja testauksen hallinnan parantamiseen, TPI testausprosessin parantamiseen. *Testauksen kypsyystasomalli* (Testing Maturity Model, TMM) on CMM:n pohjalta tehty malli, joka mittaa testauksen kypsyyttä organisaatioissa. *Minimaalinen testauskäytännön kehys* (Minimal Test Practice Framework, MTPF) on malli, joka on tarkoitettu kasvaville pienille ja keskisuurille yrityksille.



Kuva 3.3 Testausprosessin kehittäminen osana ohjelmistoprosessin kehittämistä (Toroi, 2006).

3.2.1 Testausprosessin mallinnus

Testausprosessin mallintamiseen on monta mahdollista syytä ja mallinnustapaa. Yleinen vaiheistus noudattelee Pyhäjärven & Pöyhösen (2006) seuraavaa kaavaa:

1) rajataan prosessi
2) tunnistetaan prosessin elementit
3) hahmotetaan elementtien suhteet
4) tuotetaan kuvaukset
5) otetaan malli käyttöön

Prosessin rajaus suoritetaan tiedonkeruulla esimerkiksi sidosryhmiä haastattelemalla ja hyväksyttämällä sidosryhmillä mallin osat. Prosessista luodaan kokonaiskuva ja erilliset prosessit synkronoidaan. Koska testaus on harvoin yhtenäinen kehitysprojektin riippumaton osa, rajauksessa on oltava näkyvissä tärkeimmät testauksen ulkoiset riippuvuudet ja ylätason vaiheistus. Rajauksen avulla selvitetään miten eri testaustasot ja –tyypit sijoittuvat kokonaisuuteen.

Prosessien elementtien tunnistamisessa pohjana ovat yleiset mallinnuksen tekniikat ja kuvaustavat sekä prosessikehykset. Ne ovat hyödyllisiä niissä olevan valmiin syntaksin ja semantiikan sekä työkalutuen ansiosta. Mallien rajoitukset on kuitenkin hyvä ottaa huomioon.

Elementtien suhteiden hahmottamisessa voidaan erottaa seuraavia suhteita: koostuminen, ryhmittelyt, erikoistaminen ja yleistäminen, instantiointi, riippuvuudet, herätteiden kulku ja tietovirrat. Suhteet kuvataan kukin omalla tavallaan mallista riippuen.

Mallinnustyökalujen hyvällä tuella abstraktioille on mallin hallinnalle suuri merkitys. Kuvausten tuottamisessa on keskeistä kuvausten yhtenäisyys; kuvausten graafiset elementit, ulkoasu ja tyyli sekä kuvausten tulkinta ovat yhtenäisiä. Osaprosessit tunnistetaan ja kuvataan erikseen, jolloin saadaan kuvatuksi prosessihierarkia. Prosessihierarkiassa on samalla tasolla olevia osaprosesseja, jotka kytketään yhteen kokonaisprosessiin. Jos osaprosessien väliset riippuvuudet ovat vähäisiä ja selkeästi määriteltyjä, mahdollisuudet delegointiin ja hallittuun rinnakkaisuuteen ovat paremmat.

Viimeinen vaihe testausprosessin mallintamisessa on mallin käyttöönotto. Malli tulee olla sekä johdon että tekijöiden hyväksymä ja mallia tulee käyttää kaikessa viestinnässä, arvioinnissa ja kehittämisessä.

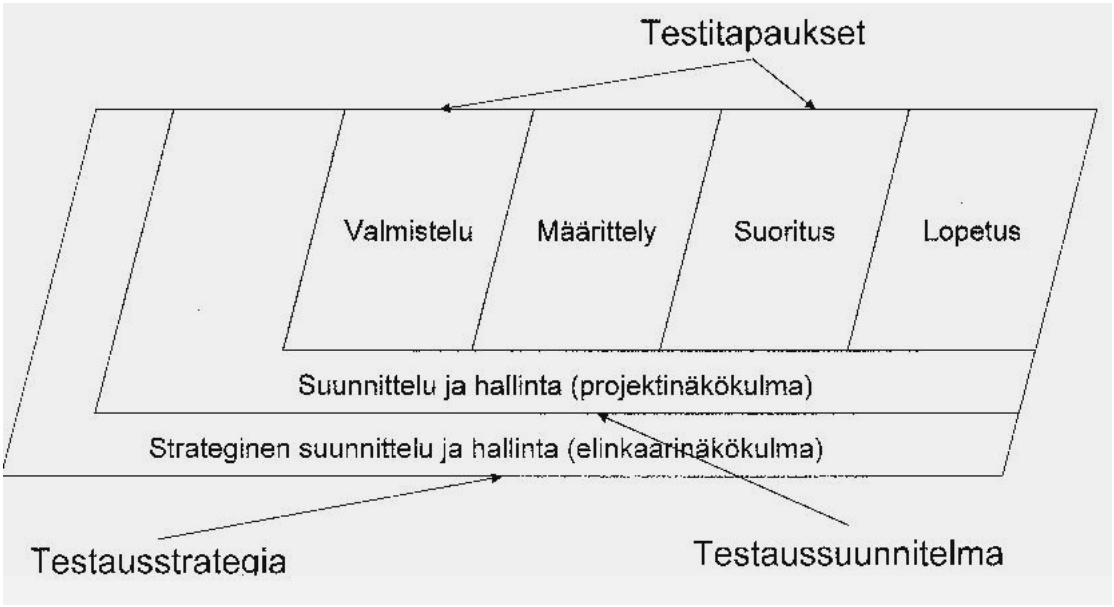
3.2.2 Testauksen kypsyysmalli TMM

Testauksen kypsyysmalli (Testing Maturity Model, TMM) kehitettiin Toroin (2006) mukaan vuonna 1996 täydentämään koko ohjelmistoprosessin kehittämiseen tarkoitettua CMM-mallia (Capability Maturity Model) ja sen tarkoituksena oli keskittyä testaukseen liittyviin asioihin. TMM-mallissa on 5 eri kypsyystasoa, joilla testausprosessi voi olla. Kypsyystasoja ovat: alkutaso, vaiheen määrittely, integrointi, hallinnointi ja mittaaminen sekä optimointi, virheiden ehkäisy ja laadunvalvonta.

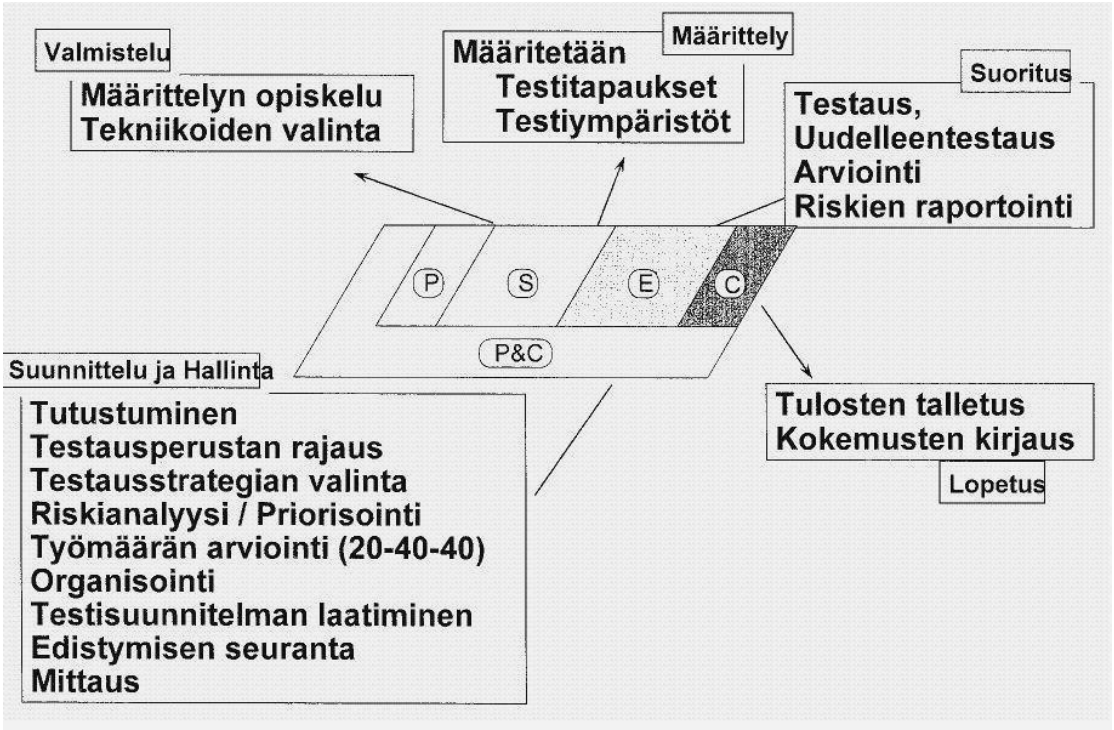
Alkutasolla testaus on kaoottinen prosessi, jolla ei ole tavoitteita. *Vaiheen määrittely* –tasolla testausvaihe määritellään ohjelmistokehitysprosessin osaksi ja erotetaan virheiden etsimisestä. Testaus suunnitellaan ja tavallisimmat työkalut ja testausmenetelmät otetaan käyttöön. *Integrointitasolla* testausprosessi integroidaan mahdollisimman aikaisessa vaiheessa osaksi ohjelmiston elinkaarta ja ohjelmistoon rakennetaan laatua kehitysprosessin alusta lähtien. Testauksen vastuuhenkilöt määritetään, testauskoulutusta järjestetään ja testausprosessia valvotaan ja mitataan. *Hallinnoinnin ja mittaamisen* –tasolla otetaan käyttöön organisaation laajuinen tarkastusmenettely sekä testausprosessin mittauskäytännöt ja arvioidaan ohjelmiston laatua laatukriteerien avulla. *Optimointi sekä virheiden ennaltaehkäisy* –taso mahdollistaa organisaation tietyn tason luotettavuuden julkistamisen. Tasolla panostetaan virheiden ennaltaehkäisyyn ja virheiden syiden analysointiin sekä testausprosessin optimointiin ja laadun valvontaan. Jokaiseen kypsyystasoon liittyy omat tavoitteet ja alitavoitteet.

3.2.3 TMap-menetelmä

TMap-menetelmä (Testing Management Approach) sisältää neljä testauksen olennaista asiaa, joita ovat: testauksen elinkaari, ympäristö, organisaatio ja tekniikat. Organisaation tärkeys voidaan liittää siihen tosiasiaan, että informaatioteknologian merkitys liiketoiminnalle kasvaa jatkuvasti. Testaukseen liittyvät monet näkökulmat asettavat omia vaatimuksia testausprosessin tehokkaalle suorittamiselle: TMap tarjoaa lukuisia tekniikoita testausprosessin suorituksen parantamiseen. TMap on myös rakenteeltaan joustava ja sitä voidaan soveltaa sekä uusien järjestelmien kehittämiseen että olemassa olevien järjestelmien ylläpitoon (Sogeti, 2008). Kuvassa 3.4 on Pyhäjärven ja Pöyhösen (2006) näkemys TMap-mallin elinkaaresta.



Kuva 3.4 TMap-mallin elinkaari (Pyhäjärvi & Pöyhönen, 2006).



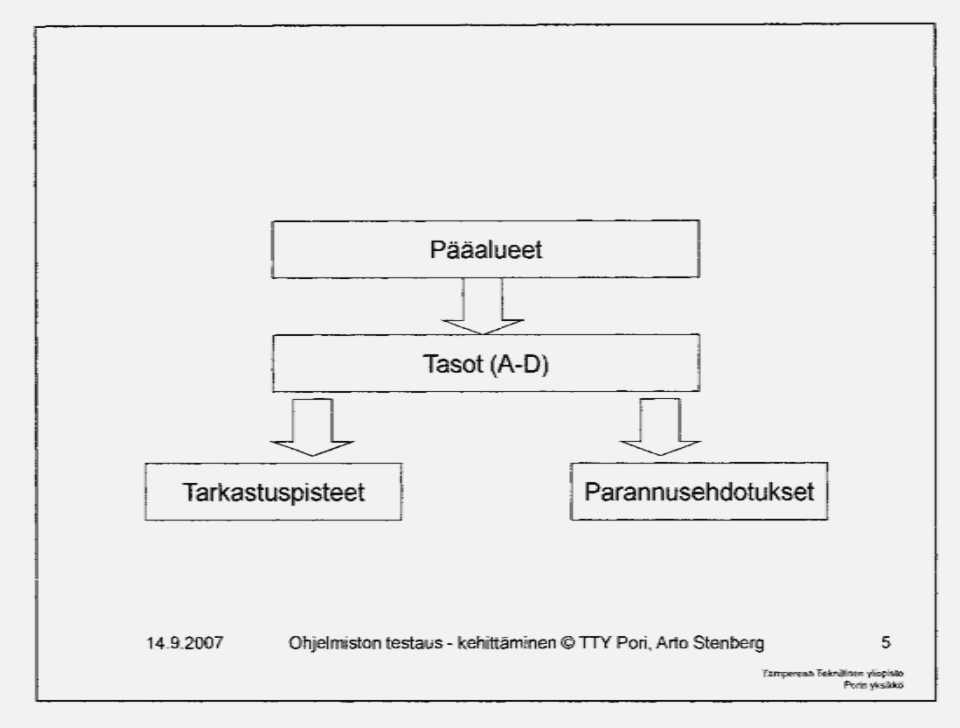
Kuva 3.5 TMap-mallin vaiheet (Pyhäjärvi & Pöyhönen, 2006).

Hyväksymistestaus ja järjestelmätestaus nähdään omina itsenäisinä prosesseina, jotka on organisoitava: niillä on oma testisuunnitelma, oma budjetti ja usein myös oma testausympäristö. Niitä voidaan suorittaa rinnakkain ohjelmiston kehittämisen kanssa. TMap elinkaarimallia käytetään sekä testisuunnitelman luomisessa ja muiden testausprosessin aktiviteettien suorituksessa. Elinkaarimalli on yleinen malli, jota voidaan soveltaa kaikilla testaustasoilla ja testaustyypeillä. TMap elinkaarimallissa testausaktiviteetit jaetaan seitsemään vaiheeseen, joita ovat suunnittelu, hallinta, infrastruk-

tuurin luominen ja ylläpito, valmistelu, määrittely, suoritus ja lopetus. Jokainen vaihe jakautuu lukuisiin aktiviteetteihin (Sogeti, 2008). Kuvassa 3.5 Pyhäjärvi ja Pöyhönen (2006) esittävät TMap-mallin vaiheita.

3.2.4 TPI -menetelmä

Testausprosessin parantamisen toimenpiteiden suunnittelussa ja toteutuksessa TPI (Test Process Improvement) -menetelmä on hyvä perusmenetelmä ja sitä voidaan käyttää, etenkin sovellettuna, monenlaisissa ympäristöissä (kuva 3.6). TPI-menetelmä on kehitetty vuonna 1997 ja se on menetelmä olemassa olevan testausprosessin arviointiin tai täysin uuden testausprosessin kehittämiseen. Sitä käytetään analysoimaan testausprosessin ja testausympäristön tämänhetkinen tila ja tunnistamaan testausprosessin vahvat ja heikot puolet (Boyer, 2000).



Kuva 3.6 TPI-malli (Stenberg, 2007).

Testausprosessia tarkastellaan Toroin (2006) monesta eri näkökulmasta, joita kutsutaan *avainalueiksi* **(**key areas). Avainalueita on yhteensä 20 kappaletta. Avainalueita tutkimalla saadaan jokaiselle niistä luokiteltua *kypsyystaso* (maturity level). Kypsyystasot ilmaistaan kirjaimilla A, B, C ja D; A on heikoin ja D paras. Lisäksi on olemassa niin kutsuttu nollataso, joka on tasoa A alhaisempi taso.

Testauksen kypsyysmatriisi (test maturity matrix) esittää (kuvassa 3.7) avainalueet ja niiden kypsyystasot matriisirakenteena. Matriisissa käytetään asteikkoa 1-13: 1-5 on kontrolloitu, 6-10 on tehokas ja 11-13 on optimoitu. Jokaiselle kypsyystasolle on määritelty *tarkistuspisteitä* (checkpoints). Ne ovat tiettyyn avainalueeseen kohdistuvia vaatimuksia. Tarkistuspisteitä on yli 200 ja ne ovat luonteeltaan kumulatiivisia. Vaatimus kypsyystasolle B sisältää kypsyystason B tarkistuspisteiden läpäisyn lisäksi kypsyystason A tarkistuspisteiden läpäisyn. Lyhyen ja pitkän aikavälin *parannusehdotukset* **(**improvement suggestions) ovat ehdotuksia tietyn kypsyystason saavuttamiseen.

Niistä voidaan mainita esimerkiksi testauksen automatisointi. Tietyt avainalueet ja niiden kypsyystasot ovat toisistaan riippuvaisia.

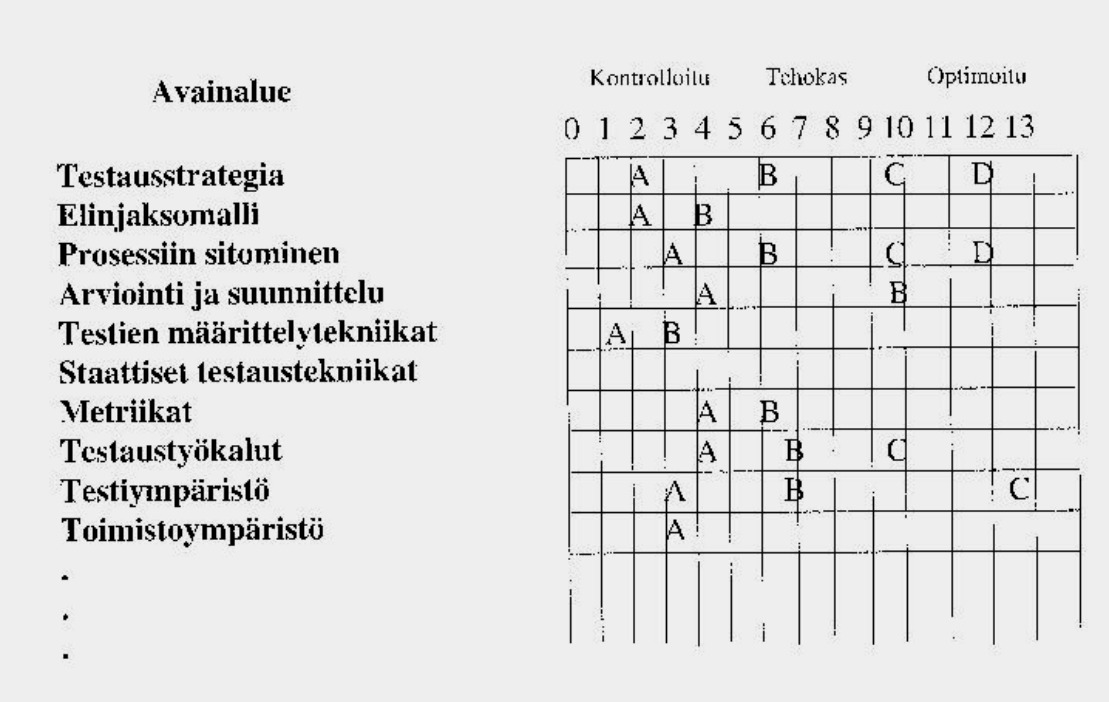
TPI-malli käsittää 20 avainaluetta. *Testausstrategiana* on, että tärkeimmät virheet löydetään mahdollisimman aikaisin ja mahdollisimman pienillä kustannuksilla. *Elinkaarimallilla* tarkoitetaan testausprosessin sisällä olevien vaiheiden määrittelyä ja jokaisessa vaiheessa suoritettavien toimintojen tavoitteet, syötteet, prosessi, tulokset ja riippuvuudet. Elinkaarimallin avulla testausprosessin ennustettavuus ja kontrolloitavuus paranee. *Testauksen aloitusajankohta* tulisi aloittaa mahdollisimman aikaisin, jotta virheet saadaan poistettua mahdollisimman halvalla. *Työmääräarviointi ja suunnittelu* osoittavat, mitkä toiminnot pitää aloittaa milloinkin ja kuinka paljon resursseja tarvitaan. *Testitapausten määrittelytekniikat* ovat standardoitu tapa määritellä testitapauksia. Soveltamalla näitä tekniikoita saadaan parempi käsitys testitapausten laadusta ja kattavuudesta sekä lisätään testitapausten uudelleen käyttöä.

Staattiset testausmenetelmät tarkoittavat ohjelman tarkastamista staattisesti ajamatta sitä ja arvioimista testausprosessia mittareiden avulla. *Mittarit* ovat mitattavia havaintoja tuotteen tai prosessin ominaisuuksista. Mittareiden avulla voidaan seurata testausprosessin kehittymistä ja järjestelmän laatua. *Testaustyökalut* ovat automatisoituja testauksen apuvälineitä. Työkalujen avulla voidaan saavuttaa merkittäviä hyötyjä, kuten lyhyempi läpimenoaika ja manuaalisten virheiden väheneminen. *Testiympäristö* sisältää mm. seuraavat osiot: menetelmät, laitteistot, ohjelmistot, tietokannat ja kommunikoinnin tavat. Testiympäristöön liittyviä tärkeitä piirteitä ovat vastuiden määrittäminen, hallinto, saatavuus ja pääsy testiympäristöön, ympäristön joustavuus ja testiympäristön vastaavuus reaalimaailmaan. Tehokas testaus vaatii hyvän *työskentely-ympäristön.*

Testauksessa mukana olevien henkilöiden *sitouttaminen ja motivointi* ovat tärkeitä edellytyksiä sujuvan testausprosessin läpiviemiseksi. Testaajien lisäksi projektipäälliköt ja johto tulee sitouttaa testaukseen. *Testaustoiminnot ja koulutus:* Testaustiimissä on tärkeää olla mukana oikeanlaisia henkilöitä. Tarvitaan eri osa-alueiden asiantuntemusta sekä kohdealueen tietämystä ja osaamista voidaan hankkia kouluttautumisella. Jokaisessa testausprosessissa käytetään tietynlaisia *työskentelytapoja*, joiden tulisi olla sekä riittävän yleisiä että yksityiskohtaisia. *Kommunikoinnin* tulee tapahtua sekä sisäisesti testaajien kesken että ulkoisesti ohjelmistokehittäjien, asiakkaiden ja käyttäjienkin kesken. *Raportoinnin* avulla voidaan tuotteen laadun taso viestiä asiakkaille.

Virheidenhallinnassa pitää pystyä jäljittämään virheiden elinkaari. Testauksessa käytettävät dokumentit, koodit ja tietokannat eli testauksen materiaalin hallinta pitää olla ylläpidettävissä, uudelleenkäytettävissä ja hallittavissa. On myös tärkeää, että edellisten vaiheiden tuotokset esimerkiksi toiminnalliset määrittelyt ja toteutus ovat hallinnassa eikä versioiden vaihdokset aiheuta sekaannusta testauksessa. Testausprosessinhallinnassa seuraavat askeleet ovat olennaisia: suunnittele, toteuta, tarkasta ja toimi. *Katselmoinnilla* tarkoitetaan välituotteiden, kuten vaatimusmäärittelyn sekä toiminnallisten määrittelyjen tarkastamista. Katselmoinnin hyöty on siinä, että virheet löydetään kehitysprosessin aikana paljon aikaisemmassa vaiheessa kuin testauksessa. Tällöin korjauskustannukset jäävät alhaisemmiksi. Katselmointi on myös helpompi suorittaa kuin testaus, koska ohjelmien ajamista tai testiympäristön pystytystä ei tarvita. Testaajan suorittamat yksikkö- ja integrointitestit ovat *alemman eli matalan tason testejä*. Matalan tason testaus on tehokasta, koska se vaatii vain vähän kommunikointia. Usein virheen löytäjä on virheen tekijä ja samalla myös sen korjaaja.

Suurimpia hyötyjä TPI-menetelmän käytöstä ovat muun muassa hyvä käsitys testauksen asemasta laadun varmistuksessa, kannustin prosessin parantamiseen ja keino asettaa testausprosessi organisaation muiden valvonta- ja hallintajärjestelmien rinnalle. TPI-malliin tutustuminen voi antaa käsityksen, että sen käyttö johtaa hyvään analyysiin testausprosessin sekä nykytilasta että tavoitetilasta. Näin ei välttämättä ole. TPI-malli on työkalu ja apuväline testausprosessin kehittämisessä ja kommunikaation parantamisessa organisaatiossa. Testausprosessin kehittäminen vaatii osaamista erityisesti testauksen, johtamisen ja muutoksenhallinnan alueilta (Boyer, 2000).



Kuva 3.7 Esimerkki TPI-matriisista (Pyhäjärvi & Pöyhönen, 2006).

3.2.5 Minimaalinen testauskäytännön kehys

Pienille organisaatioille olemassa olevat testauksen parantamiseen liittyvät menetelmät ovat usein liian laajoja ja liian suuria investointeja vaativia. Karlström & al. (2005) esittävät mallin, jota he kutsuvat nimellä *Minimaalinen testauskäytännön kehys* (Minimal Test Practice Framework, MTPF). Se tarjoaa pienille ja kasvaville yrityksille mallin, joka mahdollistaa inkrementaalisen lähestymistavan sopiviin käytäntöihin. Minimaalisen testauskäytännön prosessi yrittää minimoida muutosvastarintaa maksimoimalla koko organisaation laajuisen osallistumisen parantamishankkeisiin ja varmistamalla, että muutokset tehdään pienin askelin.

Minimaalinen testauskäytännön kehys, MTBF, on jaettu viiteen kategoriaan ja kolmeen tasoon (kuva 3.8). Viisi kategoriaa vastaavat testausta ja testausorganisaatiota seuraavasti (Karlström & al., 2005):

1) ongelma- ja kokemusraportointi: systemaattinen virheiden raportointi- ja jäljitysmenetelmä, johon kerätään myös kokemukseen perustuvaan tietoa projektista
2) roolit ja organisatoriset asiat: testaustoimintojen organisaatiorakenne ja roolit, jotka on

määritelty organisaatiossa testaustarkoituksiin
3) verifiointi ja validointi: tuotteen verifiointi- ja validointitoiminnot
4) testauksen hallinta: testausympäristön hallinta
5) testauksen suunnittelu: testien suunnittelu

Kolme tasoa vastaavat yrityksen kasvua. Ensimmäinen taso on tarkoitettu 1-10 hengen yrityksille, toinen taso 11-20 hengen yrityksille ja kolmas taso yli 30 hengen yrityksille. Menetelmän perusajatuksena on parantamiskäytäntöjen lisääminen yrityksen kasvaessa ja uusien asioiden tullessa esille.

Phase 3 (30+)	Maintain & Evolve System	Define Teams	Perform Inspections	Risk Manage ment	Coordinate Software Quality Assurance
Phase 2 (~20)	Create System	Define Roles	Perform Walkthroughs	Test Cases	
Phase 1 (~10)	Define Syntax	Define Responsibility	Use Checklists	Basic Administ ration of Test Environ ment	Test Plan
Category	**Problem & Experience Reporting**	**Roles & Organisation**	**Verification & Validation**	**Test Administ ration**	**Test Planning**

Kuva 3.8 MTPF-mallin yleiskuvaus (Karlström & al., 2005).

3.3 Testauksen automatisointi

Automatisoinnilla voidaan Pezze & Youngin (2008) mukaan parantaa joidenkin laatuaktiviteettien tehokkuutta. Vaikka automatisointiasteen nostaminen ei korvaa rationaalista, hyvin organisoitua laatuprosessia, on huomioitava automatisoinnin mahdollisuudet prosessin parantamisessa. Testaus- ja analyysiaktiviteettien automatisoinnin rooli voidaan nähdä kolmen ulottuvuuden avulla: aktiviteetin arvo ja sen nykyiset kustannukset, aktiviteetin laajuus ja automatisoinnin avulla saavutettavat kustannushyödyt siinä sekä tarjolla olevien automatisointityökalujen kustannukset.

3.3.1 Automatisoinnin perusteita

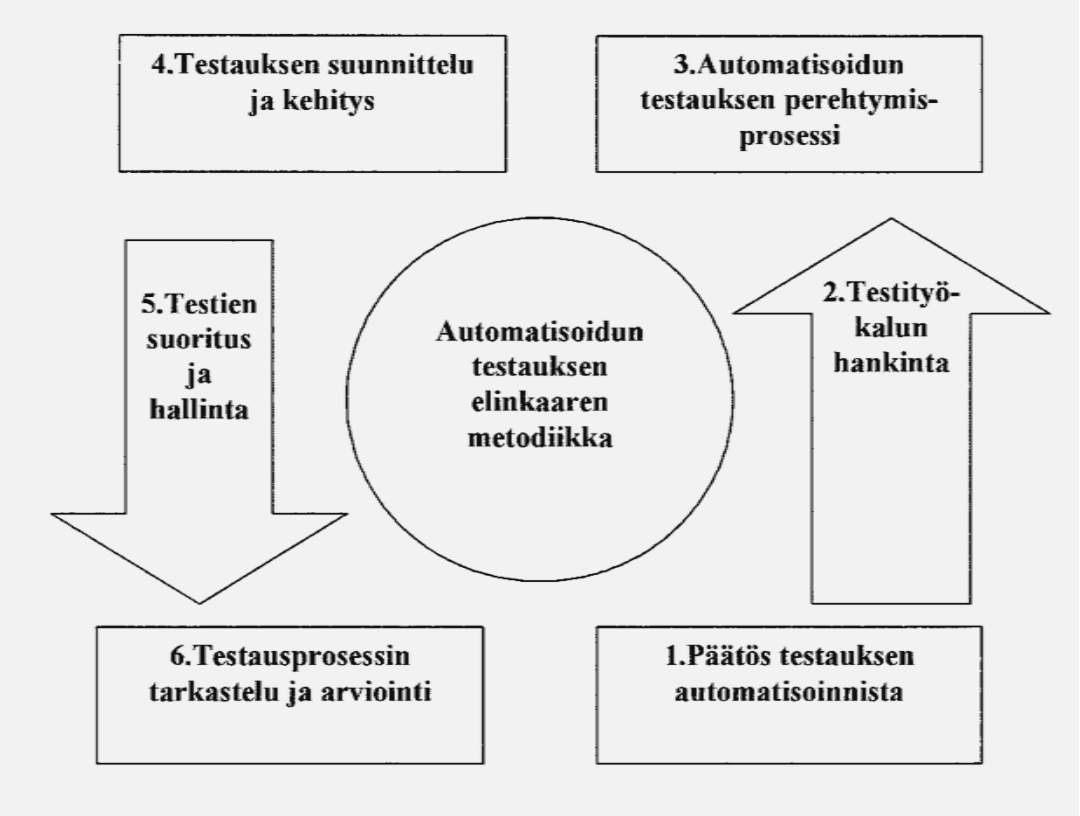
Osa testauksen ja analysoinnin toiminnoista perustuu niin vahvasti automatisointiin, että päätös tekniikan valinnasta johtaa automaattisesti automatisointityökalujen valintaan. Ottamalla käyttöön esimerkiksi rakenteellisen kattavuusarvioinnin ohjelmantestauksessa, on käytettävä kattavuuden mittaamisen välineitä. Toisaalta, vaikka ohjelmiston tarkastus on periaatteessa manuaalinen toiminto, informaation organisointiin ja esittämiseen soveltuvia välineitä on kehitetty paljon. Automatisoinnin hankaluus ja kustannukset vaihtelevat paljon; automatisoinnin hyödyt on huolellisesti punnittava suhteessa kustannuksiin, mukaan lukien koulutukset ja integraatio. Välineiden monimut-

kaisuus ja kustannukset liittyvät usein laajuuteen ja virheettömyyteen: toisinaan yleistä välinettä on vain marginaalisesti hankalampi soveltaa kuin yksinkertaista, tiettyyn projektiin tarkoitettua välinettä, toisinaan taas on huomattavasti kustannustehokkaampaa luoda yksinkertaisia projektiin erikoistuneita välineitä. Ihmiset ovat hitaita ja virhealttiita toistettavien rutiinitehtävien suhteen; vastaavasti yksinkertaiset, toistettavat tehtävät ovat usein suoraviivaisesti automatisoitavissa, kun taas arviointi- ja luovuusongelmat jäävät automatisoinnin ulkopuolelle. Ihminen on hyvä tunnistamaan oleelliset suorituksen ennusteet, jotka vastaavat testitapausmäärittelyjä, mutta huono ja tehoton tuottamaan suuria määriä testitapauksia tai tunnistamaan virheelliset tulokset regressiotestin tuottamasta tulosten suuresta määrästä. Automatisoimalla tehtävän toistettavat osuudet vähennetään kustannuksia ja parannetaan virheettömyyttä ja tarkkuutta (Pezze & Young, 2008).

Yksi testausstrategian tärkeä rooli on määritellä organisaatiossa käytössä olevien laatuprosessin avainelementtien työkalut ja välineet. Yksityiskohtaisesti määritelty laatuprosessi ja siihen sopivat välineet eivät sovi keskenään erilaisille ohjelmistoprojekteille; testausstrategia voi suositella erilaisia välineitä muun muassa sovellusalueen, ohjelmointikielen tai koon mukaan ryhmitellyille projekteille. Yleisen tason laatustrategiat viittaavat testauksen suunnittelu- ja suoritusvälineisiin sekä laatudokumenttien tuottamiseen, mittaamistietojen keräämiseen ja regressiotestitapausten hallintaan liittyviin välineisiin. Tietyn projektin laatusuunnitelma perii laatustrategian määrittelemät ja suosittelemat välineet sekä määrittelee ne välineet, jotka valitaan projektikohtaisesti. Organisaatiostandardin mukaisten ja projektikohtaisten välinevalintojen lisäksi suunnitelmaan on sisällyttävä automatisointiin liittyvät kustannukset kuten koulutus, toteutetut toiminnot ja potentiaaliset riskit (Pezze & Young, 2008).

Automatisoinnilla saavutettavia hyötyjä ovat: *testin toistettavuus* (repeatability), *laajennettavuus* (leverage) ja *kertymä* (accumulation). Testin toistettavuudella tarkoitetaan testin toistamista monta kertaa samanlaisena; laajennettavuudella tarkoitetaan mahdollisuutta suorittaa testejä, jotka ovat manuaalisesti mahdottomia suorittaa; kertymä-hyöty saavutetaan, kun testitapauskanta suunnitellaan niin hyvin ylläpidettäväksi, että ohjelmiston piirteiden määrän lisääntyessä testien määrä ei lisäänny eli mahdollisuus suoriutua ohjelmamuutoksista vähemmällä määrällä testitapauksia kuin manuaalisessa testauksessa (Dustin & al., 1999).

Testauksen automatisointi on itsenäinen ohjelmiston kehitysprosessi. Testauksen automatisointiprojektit epäonnistuvat yhtä todennäköisesti kuin ohjelmistojen kehitysprosessit ja niihin on panostettava yhtä paljon kuin ohjelmistojen kehitysprosesseihin. Automatisointi tarkoittaa yhden tai useamman seuraavan aktiviteetin automatisoimista: testauksen ehtojen tunnistaminen, testitapausten suunnittelu, testien rakentaminen, testien suorittaminen ja saatujen tulosten vertailu odotettuihin tuloksiin. Testauksen automatisointia varten perustettavassa ryhmässä on hyvä olla eri alojen asiantuntemusta, mutta erityisen tärkeää on kokemus testauksesta, ohjelmoinnista ja automatisoinnista. Yrityksen johdon on myös sitouduttava hankkeeseen koko toteutusajaksi (Dustin & al., 1999).



Kuva 3.9 Automatisoidun testauksen elinkaaren vaiheet (Dustin & al., 1999).

3.3.2 Automatisoidun testauksen elinkaari

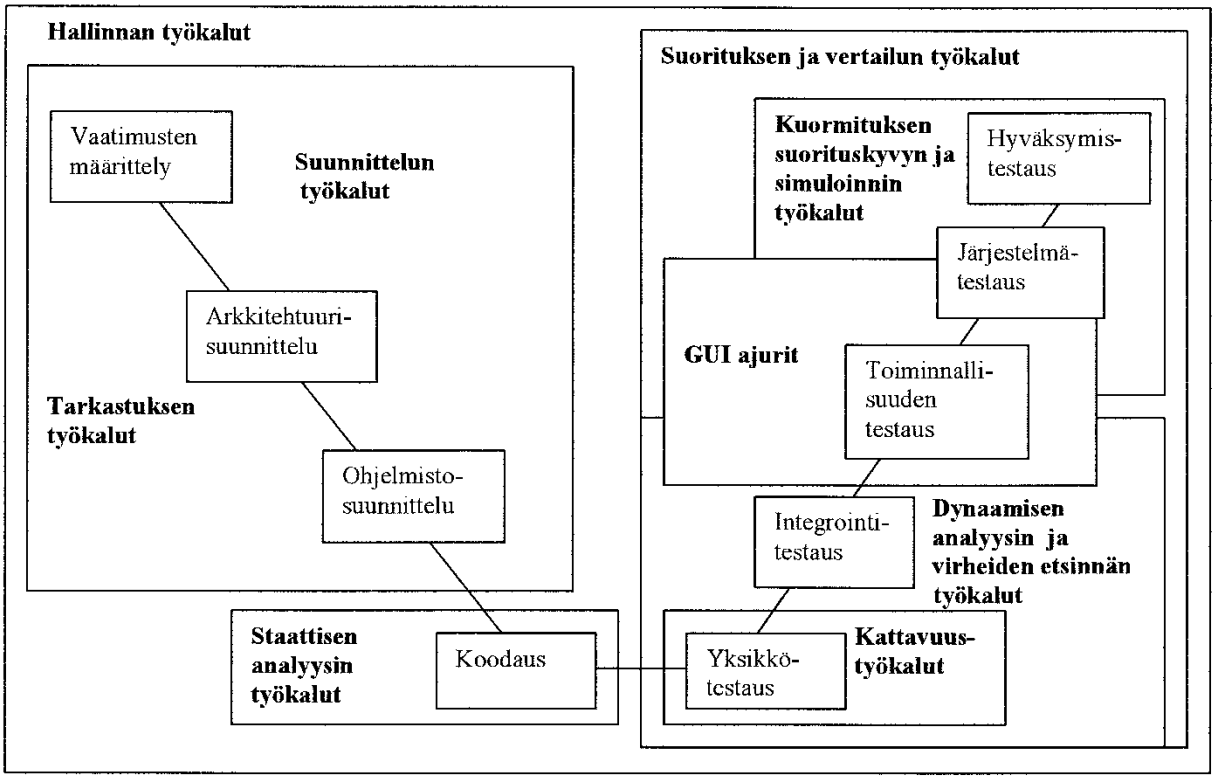
Automatisoidun testauksen elinkaareen sisältyy Dustin & al. mukaan (1999) kuusi eri vaihetta: päätös automatisoinnista, testityökalun hankinta, automatisoidun testauksen perehtymisprosessi, testauksen suunnittelu ja kehitys, testien suorittaminen ja hallinta sekä testausprosessin tarkastelu ja arviointi (kuva 3.9). Päätöstä automatisoida edeltää testauksen nykytilan analysointi, joka sisältää automatisointiin liittyvien väärien odotusten kartoituksen ja selvityksen, automatisoinnin etujen ymmärtämisen, johdon tuen saamisen sekä työkalujen arvioinnin. Analysoinnin tuloksena tehdään päätös testauksen automatisoinnista tai luovutaan siitä. Työkalun valinnassa on huomioitava, että työkalu on yhteensopiva mahdollisimman monen organisaatiossa käytössä olevan menetelmien ja alustojen kanssa. Työkalujen piirteitä voidaan arvioida ja painottaa eri tavoin; piirteitä ovat muun muassa helppokäyttöisyys, työkalun muunnettavuus, tuetut alustat, monikäyttömahdollisuus, virheen jäljitys, työkalun toimivuus, tulostuskapasiteetti, suorituskyky, yhteensopivuus eri versioiden työkalujen kanssa sekä toimivuus testauksen suunnittelun ja hallinnan työkalujen kanssa.

Automatisoidun testauksen kehitysprosessissa analysoidaan nykyinen testauskäytäntö ja etsitään kehityskohteet sekä käydään läpi valitun työkalun yhteensopivuutta ja kelvollisuutta. Testauksen suunnittelu ja kehitys sisältävät automatisointia sisältävän testausohjelman elinkaaren mukaiset vaiheet, kuten testauksen vaatimusten määrittelyn, analyysin, suunnittelun ja toteutuksen. Testien suorituksen ja hallinnan vaiheessa voidaan käyttää erilaisia mittareita, joiden avulla testausprosessin laatua on mahdollista valvoa ja tehdä päätös testauksen jatkamisesta tai lopettamisesta. Testauspro-

sessin viimeisessä vaiheessa arvioidaan kuinka hyvin alkuperäiset suunnitelmat toteutuivat ja päätetään mitä parannuksia voidaan tehdä.

3.3.3 Automatisoidun testauksen välineet

Testauksen apuvälineet voidaan ryhmitellä usealla tavalla; seuraavassa ryhmittely on tehty käyttökohteen mukaan: suunnittelu, graafinen käyttöliittymä, kuormitus ja suorituskyky, hallinta, toteutus, arviointi sekä staattinen analyysi. Apuvälineessä on usein yhdistelmä useisiin käyttökohteisiin sopivia piirteitä. Kuvassa 3.10 esitetään apuvälineiden sijainti perinteisessä ohjelmistokehityksen elinkaaressa (Pohjolainen, 2003).



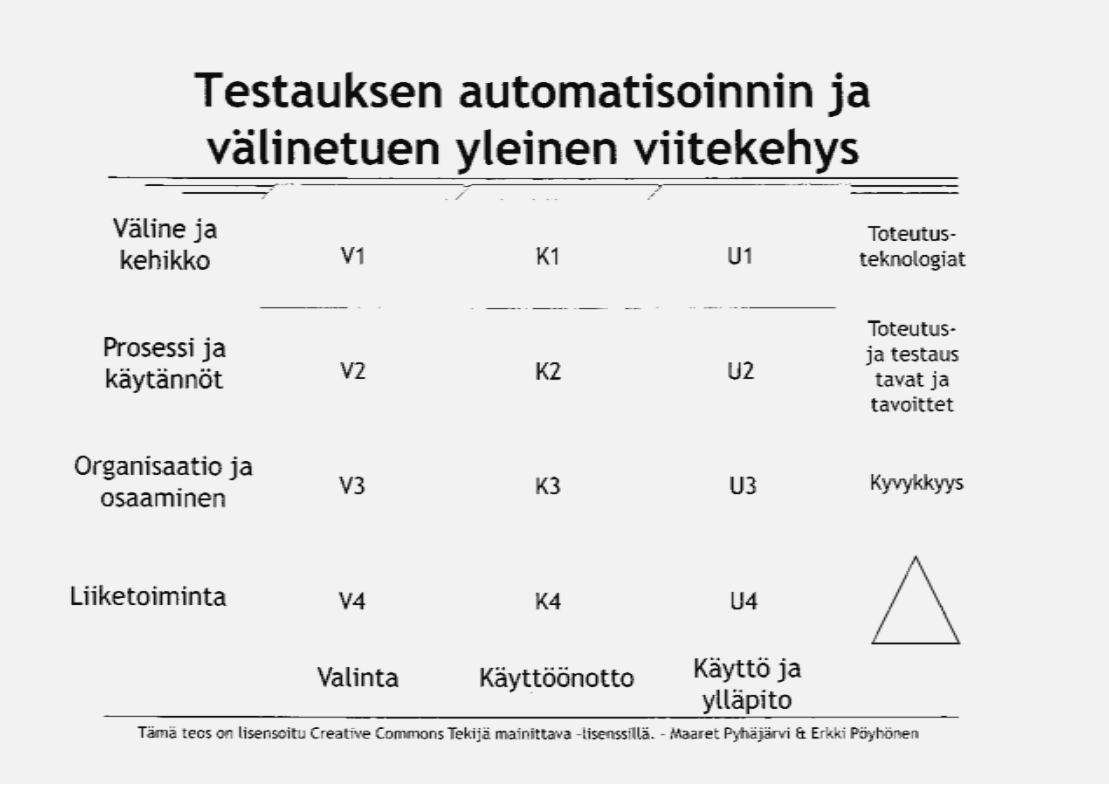
Kuva 3.10 Apuvälineiden sijainti perinteisen ohjelmistokehityksen elinkaaressa (Pohjolainen, 2003).

Testausvälineitä on Pyhäjärvi & Pöyhösen mukaan (2004) kahta perustyyppiä: testauksen tukemisen välineet ja testiautomaatiovälineet. Testauksen tukemisen välineitä ovat välineet, joiden käyttö voi tukea sekä käsin suoritettavia että automatisoituja testejä ja joiden käyttöön ei liity oletusta käsin testaamiselta välttymisestä; testiautomaatiovälineitä ovat välineet, joilla suoritetaan testausta tietokoneavusteisesti ja joiden käyttöön liittyy oletus käsin testaamiselta välttymisestä.

Automatisoidun testauksen hyötyjä ajan ja budjetin suhteen ovat muun muassa kehityksen ja testauksen läpimenoaikojen lyheneminen, kehityksen ja testauksen kustannusten pieneneminen, testikierroksissa tarvittavien työmäärien hallinta. Lisäksi automatisointi lisää näkyvyyttä etenemiseen ja kykyä muuttaa suunnitelmia markkinatarpeisiin vastaamiseksi. Testausmahdollisuudet kasvavat esimerkiksi nopeilla palautteilla koonneista ja julkaisuista, yhdistelmien, yhtäaikaisuuden ja uusin-

tatestauksen mahdollisuuksilla, löytyneiden virheiden toistamiskyvyllä, riskipohjaisella suuntaamisella, testien suorituksen valvonnan tarpeettomuudella ja mahdollisuudella keskittyä motivoivaan työhön rutiinin asemesta. Hyötyjä tulee verrata kustannuksiin sekä lisäksi yrityksessä on kartoitettava testausautomaation valmiudet, joita ovat testauskyvykkyys ja välinevalmius.

Välineen valinta, käyttöönotto sekä käyttö ja ylläpito sisältävät seuraavat tavoitteet: välineen valinnan tavoite on onnistumisen mahdollisuuksien luominen, käyttöönoton tavoite on eteneminen hallituissa ja näkyvissä palasissa ja käytön ja ylläpidon tavoite on jatkuvan hyödyn saavuttaminen ja osoittaminen. Testauksen automatisoinnin ja välinetuen yleinen viitekehys on kuvattu kuvassa 3.11.

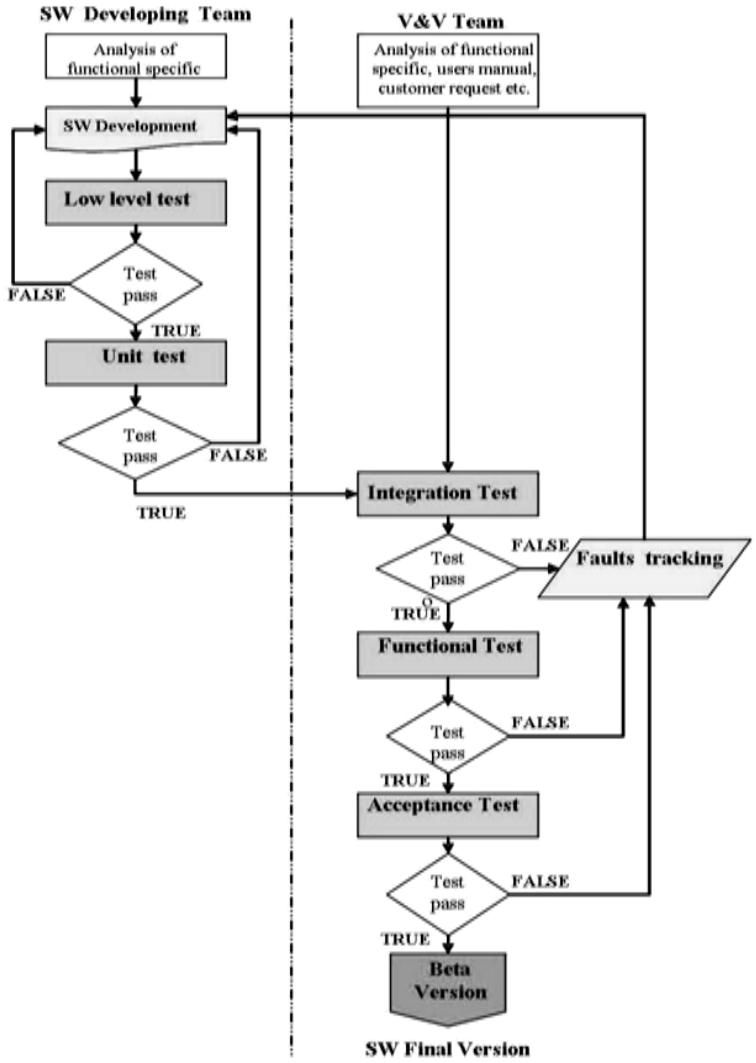


Kuva 3.11 Testauksen automatisoinnin ja välinetuen yleinen viitekehys (Pyhäjärvi & Pöyhönen, 2004).

3.3.4 Automatisointi prosessin osissa

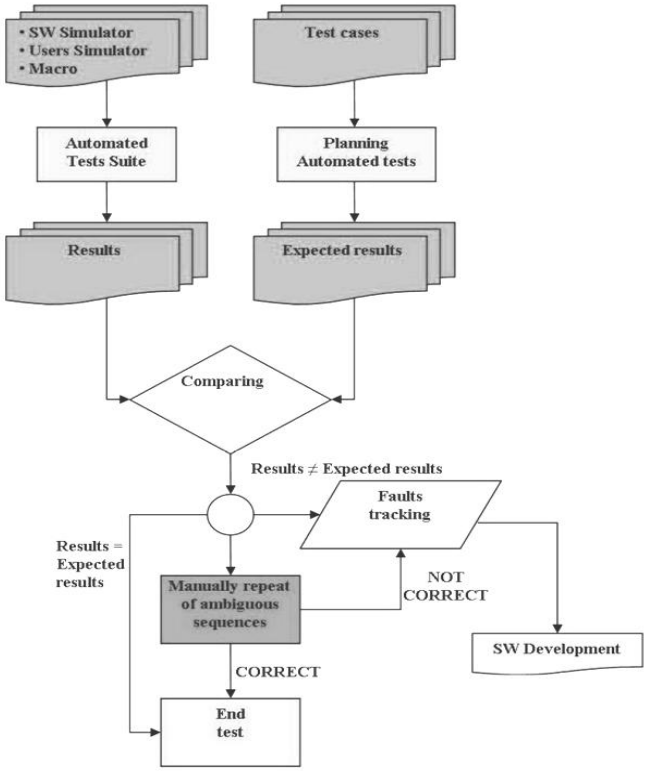
Testaussuunnittelun tarkasteleminen osissa on Bacioccola & al. (2008) mukaan edellytys ohjelmiston korkean laatutason saavuttamiselle kohtuullisilla kustannuksilla. Tämä johtaa sopivan kattavuuden hankkimiseen tuotteen toiminnallisuudelle testausajan vähentämiseksi. Automatisoitu testaus on yksi menetelmistä, joilla voidaan vähentää tuottamisen kustannuksia kasvattamalla ohjelmiston luotettavuutta. Tuotanto jaetaan kolmeen suurempaan vaiheeseen: suorituksen määrittelyt (vaatimusmäärittelyt ja arkkitehtuuri), projektointi ja kehittäminen (suunnittelu ja toteutus) sekä testaus (järjestelmätestaus ja hyväksymistestaus). Testaus jaetaan kahteen vaiheeseen: ensimmäinen on alemman tason testaus (alpha-testaus tai projektitestaus) ja toinen vaihe korkean tason testaus (toiminnallinen testaus). Ensimmäisen vaiheen suorittajina ovat kehittäjät kun taas toisen vaiheen suorittavat ulkopuoliset resurssit. Ohjelmiston kehittämisen ja testauksen kaikki vaiheet jaetaan kahden

ryhmän välillä: kehittämistiimin ja verifiointi- ja validointitiimin (V&V-tiimi) kesken. Kuva 3.12 esittää ohjelmiston kehitys- ja testausprosessia.



Kuva 3.12 Ohjelmiston kehitys- testausprosessi (Bacioccola & al., 2008).

Erilaiset ohjelmiston automatisoidun testauksen lähestymistavat voidaan erotella satunnaisiin tekniikoihin, staattisiin tekniikoihin ja dynaamisiin tekniikoihin. Satunnaisissa tekniikoissa testidata tuotetaan satunnaisesti, staattisissa tekniikoissa ohjelmaa ei suoriteta ja dynaamisissa tekniikoissa ohjelma suoritetaan. Bacioccola & al. (2008) esittämä automatisoitu ohjelmistotestaus voidaan luokitella dynaamiseksi tekniikaksi: automatisoinnin avulla voidaan suorittaa sekä suorituskykytestausta, joka on verrattavissa käyttäjien toimintaan että nopea ja helppo regressiotestaus. Ennen kuin kehitetään automaattinen testisarja on määriteltävä testitapaukset, järjestelmän rakenne, alku- ja loppuehdot, systeemiparametrit, toiminnan ehdot sekä syötteet ja tulosteet. Testitapausten, testisuunnitelman ja vastaavan ohjelmiston kehittämisen jälkeen automatisoidut testit ovat suoritettavissa. Automatisoitu testausprosessi on esitetty kuvassa 3.13.



Kuva 3.13 Automatisoitu testausprosessi (Bacioccola & al., 2008).

3.4 Testauksen kehittämisen haasteita ja tulevaisuuden näkymiä

Verifiointi- ja validointiaktiviteetit ovat Dumke & Farooqin (2007) mukaan monipuolisia ja niitä sovelletaan ohjelmistotuotannon eri vaiheissa. Verifiointi- ja validointiprosessi koostuu tarkastus-, arviointi- ja ohjelmiston testausprosesseista. Verrattuna moniin ohjelmiston kehitysprosessin kehittämiseen liittyviin arviointi- ja parantamislähestymistapoihin, verifioinnin ja validoinnin parantamiseen tarkoitettuja lähestymistapoja on vähän. Tutkimuksen ja käytännön välillä on edelleen syvä kuilu ja ratkaisuun tarvitaan sekä tutkijan suurempaa käytännönläheisyyttä ja tasapuolisuutta että käyttäjän avoimuutta uusille tekniikoille ja käytännöille. Tulevaisuuden verifiointi- ja validointiprosessien haasteita (Dumke & Farooq, 2007) ovat

1) tehokkaiden virheiden estämis- ja löytämistapojen tuottaminen
2) soveltaminen uusiin kehittämisparadigmoihin
3) kehittämisen ja testauksen kustannusten ja aikamäärien minimointi
4) skaalautuvaisuus sekä pieniin että suuriin järjestelmiin
5) laadun tuottaminen tuotteeseen odotusten mukaisesti

Testaus on Kanerin (2008b) mukaan kokemusperäinen, tekninen tutkimus, jonka suorituksen tarkoituksena on saada laatuun liittyvää tietoa ohjelmistotuotteesta tuotteen osallisille. Kokemusperäisellä eli empiirisellä tarkoitetaan testien eli kokeiden tekemistä, teknisellä teknisten välineiden käyttöä,

tutkimuksella organisoitua ja perusteellista etsintää ja laatuun liittyvällä tiedolla erilaisia informatiivisia päämääriä kuten virheitä ja niiden korjausta, laadun määrittelemistä ja yhteensopivuuden tarkistamista muiden tuotteiden kanssa.

Testauksen prosessien ja menetelmien tehostamisessa on viisi oleellista näkökohtaa, jotka tulisi huomioida. Testaus on merkittävä osa kokonaiskustannuksista ja siten tärkeä kehittämisen kohde, toiseksi näkyvyys tuotoksen laatuun on kriittinen projektille ja näkyvyys saavutetaan testauksen avulla, kolmanneksi toteutettavat järjestelmät ja projektit ovat yhä suurempia ja monimutkaisempia. Oleellisia näkökohtia ovat edelleen ihmisten välinen viestintä ja yhteistyö tietotyössä sekä prosessien kehittämisen ymmärtäminen luonteeltaan organisaatiomuutoksena. Automaatio tulee olemaan tulevaisuudessa oleellinen osa monipuolista testausta. Automatisoinnin hyödyllisyys on arvioitava huolellisesti ja muistettava, että manuaalinen testaaminen ja automatisoitu testaaminen ovat hyvin erilaisia prosesseja eivätkä kaksi tapaa suorittaa sama prosessi. Esimerkkejä kehittämistoimenpiteistä ovat muun muassa testitapausarkkitehtuurin kehittäminen ja testitapausten uudelleenkäytön parantaminen, automaatio, testausosaaminen, mittarien kehittäminen, viestinnän ja yhteistyön kehittäminen sekä virheraportoinnin käytännöt. Arviointimallit ovat hyödyllisiä kehittämisen lähtökohtia, mutta omat mallit auttavat konkretisoimaan oman organisaation ongelmat; testauksen yhtenäistäminen organisaationlaajuisesti vaatii osallistumista ja projektinomaista toteutusta (Pyhäjärvi & Pöyhönen, 2008).

Testauksen kehittämismalleissa ei oteta kantaa yhteentoimivuustestaukseen, avointen rajapintojen testaukseen, määrityksen mukaiseen testaukseen eikä sertifiointiin. Regressiotestausta on tutkittu jo pitkään, mutta silti regressiotestauksen teoria ja käytäntö ovat edelleen kaukana toisistaan. Yleensä regressiotestausta käsittelevässä tutkimuksessa on käytetty pieniä tai keskisuuria ohjelmia ja tutkittu testitapausten valintaa niille. Kuitenkin käytännössä järjestelmät ovat laajoja ja monimutkaisia. Monimutkaisten järjestelmien testauksessa tarvitaan eritasoisia testitapauksia. Testausprosessia tulee kehittää myös niin, että testausprosessin ja palvelujenhallinnan (Information Technology Infrastructure Library, ITIL) väliset yhteydet on selkeästi määritelty (Toroi, 2006).

On yleisesti oletettu, että uudet ja kehittyneet ohjelmistotuotantomenetelmien myötä virheet vähenevät ja näin ollen testauksen tarve vähenee. Tähän mennessä näin ei ole ollut ja on oletettavaa, että niin kauan kun ohjelmistoja tekevät ihmiset virheitä syntyy ja näin myös testausta tarvitaan. Tulevaisuudessa on selvästi nähtävissä erilaisten apuvälineiden kasvava rooli sekä mallipohjaisen testauksen kasvu. Apuvälineiden avulla tullaan automatisoimaan tiettyjä työvaiheita sekä tekemään asioita, joita ei ole edes mahdollista tehdä manuaalisesti. Kokonaan automaattiseksi testaus ei kuitenkaan muutu. Hyvä testaaja pystyy asettumaan loppukäyttäjän asemaan ja tarkastamaan hänen vaatimustensa toteutumisen paremmin kuin kone koskaan pystyy (Katara, 2005).

4 TESTAUKSEN KEHITTÄMINEN ATLAS-PROJEKTISSA

Työnantajani on AtBusiness Oy ja työskentelen tällä hetkellä Atlas-projektissa testaus-, dokumentointi-, ja määrittelytehtävissä. Atlas-projekti on muutamia vuosia sitten perustettu projekti, jonka tehtävänä on ollut rakentaa ja toimittaa asiakkaalle laivojen buukkausjärjestelmä. Atlas-järjestelmän perustoiminnallisuus on otettu käyttöön noin vuosi sitten minkä jälkeen toiminnallisuutta on lisätty osa kerrallaan. Järjestelmä on toteutettu AtBusiness Solution Framework –alustalle.

4.1 Projektin esittely

AtBusiness Oy on ohjelmisto- ja asiantuntijayritys, joka on erikoistunut vaativien ja liiketoiminta-kriittisten tietojärjestelmien projektitoimituksiin. Tyypillisesti järjestelmät ovat asiakkaan liiketoiminnan kilpailuetua parantavia, yksilöllisiä ratkaisuja. AtBusiness Oy sovittaa asiakkaidensa ratkaisut joko tarjoamiensa tuoteratkaisujen pohjalta tai toteuttaa ratkaisut avaimet käteen -periaatteella asiakkaan vaatimusten mukaisesti. Tuoteratkaisut perustuvat joko omiin tuotteisiin tai kumppanien, kuten Microsoftin, tuotteisiin. AtBusiness Oy toteuttaa ratkaisuja keskeisiin tietojärjestelmäarkkitehtuureihin kuten J2EE ja Microsoft .NET. AtBusiness Oy toimii kahdessa maassa ja neljässä eri toimipisteessä. Päätoimipiste on Helsingissä ja muut toimipisteet sijaitsevat Kuopiossa, Lappeenrannassa ja Pietarissa. Osaamisalueena ovat liiketoimintakriittiset järjestelmät kuten asiakkuudenhallinta ja myynnin ohjaus, toiminnan ja tuotannon ohjaus, strateginen henkilöstön ohjaus, tiedon jalostaminen ja raportointi sekä näihin liittyvät integraatiot. Asiakkuudet sijoittuvat muun muassa kaupan, teollisuuden ja palvelujen, julkisen sektorin, telecomin ja finanssin alueille (AtBusiness Oy, 2008).

AtBusiness Solution Framework on joustava ja avoimiin teknologiastandardeihin nojaava sovelluskehitysympäristö, jonka avulla voidaan tuottaa kustannustehokkaasti tietotekniikkasovelluksia liiketoiminnan tarpeisiin. Tyypillisiä kohteita ovat muun muassa CRM, ERP, SCM –ratkaisut. AtBusiness Solution Framework koostuu kahdesta toiminnallisesta yksiköstä:

- määrittely- ja protoiluvaiheen mallinnustyökalusta, AtBusiness Visual Customiser
- ajonaikaisesta sovellusympäristöstä, AtBusiness Application Framework

AtBusiness Solutions Framework sisältää monipuoliset työkalut, mallit ja valmiskomponentit teknisten ratkaisujen tuottamiseen kaikille järjestelmäarkkitehtuurin tasoille. Periaatteena on visuaalisesti tuotettujen mallien ja komponenttien mahdollisimman laaja uudelleenkäyttö kaikilla monikerrosarkkitehtuurin tasoilla (AtBusiness Oy, 2006).

AtBusiness Oy toimitti vuonna 2007 yhdelle Euroopan suurimmista rahdin kuljettamiseen erikoistuneista linjaliikennevarustamoista AtBusiness Solution Framework –teknologia-alustalle toteutetun Atlas –järjestelmän. Atlas –järjestelmällä kirjataan rahtilaivojen rahtitilaukset, hoidetaan check-in, toteutetaan asianmukainen dokumentointi, luodaan manifestit, hinnoitellaan ja laskutetaan. Lisäksi kokonaisuuteen kuuluu extranet, jossa asiakkaat voivat luoda ja muuttaa omia varauksiaan sekä tulostaa esimerkiksi merirahtikirjat. Järjestelmä on integroitu asiakkaan taloushallinnon järjestelmiin, kumppaneiden järjestelmiin sekä viranomaisten järjestelmiin (Puruskainen, 2008).

4.2 Testauksen nykytilanne

Atlas–järjestelmä on inkrementaalisesti kehitetty järjestelmä; asiakkaalle on aluksi toimitettu tietty toiminnallisuus, minkä jälkeen toiminnallisuutta on lisätty lisäämällä jo olemassa olevaan järjestelmään uusia kokonaisuuksia. Tämä on asettanut testaukselle kasvavia vaatimuksia: ennen jokaista toimitusta uusien toiminnallisuuksien lisäksi on testattava koko toiminnallisuus ja varmistuttava siitä, että lisäykset eivät ole vaikuttaneet olemassa olevaan toiminnallisuuteen haitallisesti. Lisäksi uusien toimintojen myötä jokin vanha toiminto on voinut muuttua tai poistua.

Tämä on yleinen haaste inkrementaalisesti ja nopeasti toimittujen järjestelmien testaukselle. Kehittämisen alkuvaiheen testaussuunnitelmaa ja testitapauksia ei ole aikaa ylläpitää toiminnallisuuden muuttuessa ja lisääntyessä sekä testausmäärän kasvaessa koko ajan. Atlas-järjestelmän testaukselle haasteita ovat lisäksi asettaneet projektin henkilöiden vaihtuvuus ja ajan puute muun muassa koulutuksessa, perehdytyksessä ja testauksen suunnittelussa.

Lähetin tutkielmaani liittyen mielipidekyselyn projektin jäsenille, sekä koko- että osa-aikaisesti projektissa työskenteleville. Kysely on kuvattu liitteessä 1. Lähetin kyselyn 21 henkilölle, joista 9 henkilöä vastasi. Kyselyssä oli kolme kysymystä, joihin kaikkiin oli mahdollista antaa 0-n mielipidettä. Mielipiteitä oli kaikissa vastauksissa yhteensä 87 kappaletta: ensimmäisen kysymyksen vastauksissa 42 kappaletta, toisen kysymyksen vastauksissa 13 kappaletta ja kolmannen kysymyksen vastauksissa 32 kappaletta. Sijoitin kaikki mielipiteet matriisiin, jonka vaakatasolla nähdään kyselyn kolme mielipideryhmää ja pystytasolla testauksen seitsemän osa-aluetta, jotka määrittelin tässä kyselyssä soveltamalla sekä TMap- että TPI-mallin testauksen osa-alueiden jaottelua. Kyselyn tulokset ovat nähtävissä liitteessä 2.

Kyselyn tuloksista voi päätellä, että vastanneiden keskuudessa eniten ongelmia nähtiin testaustekniikoiden alueella; esimerkiksi testausmenetelmät koettiin riittämättömiksi ja testauksen eri vaiheisiin kaivattiin erilaisia menetelmiä. Toinen suuri ongelma vastausten perusteella oli testauksen hallinta ja siihen liittyvät asiat kuten aikataulutus, resurssien hallinta ja testauksen suunnittelu. Näihin kahteen tuli myös eniten mielipiteitä kolmannen eli parannusehdotuksia tiedustelevan kysymyksen kohdalla. Muita esille tulleita ongelma-alueita olivat testausorganisaatio, koulutus ja perehdytys sekä testausympäristö. Toisessa kysymyksessä pyydettiin nimeämään nykyisen testauskäytännön hyviä puolia: osa-alue testaustekniikat sai eniten mielipiteitä, toiseksi eniten sai testausorganisaatio. Testausmenetelmät nähtiin osittain joustavina ja nopeina ja asiakkaan sitoutuminen ja asiantuntemus suurena hyötynä. Yhteenvetona kyselystä voi todeta, että nykyisessä testauskäytännössä on paljon parannettavaa, mutta hyviäkin puolia löytyy.

4.3 Testauksen kehittämissuunnitelmia

Atlas-projektin testausta on tarkoitus kehittää loppuvuoden 2008 aikana. Mahdollisia kehittämisen kohteita löytyy useampia ja näin ollen kohteiden asettaminen tärkeysjärjestykseen on välttämätöntä. Kehittämiskohteiden valintaan vaikuttavat muut kesän ja syksyn 2008 aikana tarkentuvat järjestelmän ylläpito- ja kehittämissuunnitelmat siitä, mitä, miten ja milloin on tarkoitus toteuttaa.

Ottaen huomioon mielipidekyselyn tulokset tärkeimmät Atlas-projektin testauksen kehittämiskohteet ovat testausmenetelmät ja testauksen hallinta. Testausmenetelmiin liittyen projektissa on kesän aikana tarkoitus kartoittaa testauksen automatisoinnin mahdollisuudet etenkin regressiotestauksen ja testitietokannan luomisen osalta. Hallinnan menetelmien parantaminen on aloitettu jo keväällä tarkempien aikataulutusten, resurssoinnin ja vastuiden jakamisen suhteen. Kolmantena tärkeänä testauksen kehittämisen kohteena on järjestelmän ja tuotteen koulutus ja perehdytys sekä tietämyksen jatkuva ylläpito. Testauksen kehittäminen ja parantaminen liittyy olennaisena osana koko projektin hallinnan parantamissuunnitelmiin sekä koko organisaatiotason testausstrategian kehittämissuunnitelmiin (Puruskainen, 2008).

5 YHTEENVETO

Ohjelmistojen testaus on noussut vasta viime vuosikymmenen aikana omaksi tieteenalakseen, jota tutkitaan, sovelletaan ja kehitetään siinä missä muitakin ohjelmistotuotannon alueita. Erilaiset kehitysympäristöt, sovellusalueet ja järjestelmien kehittämismenetelmät painottavat testauksen osa-alueita eri tavalla; usein hyvä testauskäytäntö koostuu useista toisiaan tukevista menetelmistä.

Sopivien validointi- ja verifiointiaktiviteettien valinta riippuu sovellusalueesta, tuotantomenetelmästä, lopputuotteesta ja laatuvaatimuksista. Mikään yksittäinen testi- tai analyysitekniikka ei voi palvella kaikkia tarkoituksia. Joillakin sovellusalueilla voi olla vaatimuksena esimerkiksi standardoidut ja tarkasti määritellyt dokumentit kun taas toisella sovellusalueella testauksen automatisointi voi olla ehdoton edellytys. Esimerkiksi perinteisen V-mallin testausmenetelmät soveltuvat huonosti inkrementaalisesti tai iteroidusti kehitettävien järjestelmien testaukseen ja oliopohjaiset järjestelmät vaativat testaukselta eri asioita kuin proseduraalisesti kehitettävien järjestelmien testaus.

Testauksella on suuri merkitys ohjelmistotuotannossa ja se on merkittävä osa ohjelmiston kehitystä. Tästä johtuen testaus on kiinnostava alue, kun haetaan parantamisen kohteita. Testaus koostuu monesta osa-alueesta, joiden kypsyystasoa voidaan analysoida ja näin arvioida testauksen parantamiskohteita. Testauksen ja testausprosessin parantamiseen on kehitetty useita menetelmiä. Kehittämällä testausta vaikutetaan muuhunkin kuin itse testaukseen. Ennen kuin testausta voidaan kehittää, on kartoitettava testauksen nykytilanne ja määriteltävä tavoitteet. Testausta voidaan kehittää sisällön, osaamisen, hallinnan ja tekemistapojen osalta. Testausprosessin kehittämisen tulisi olla jatkuvaa toimintaa ja liittyä koko ohjelmistotuotantoprosessin kehittämiseen, koska testauksen kehittäminen helpottaa tavoitteiden saavutettavuutta ja kustannusten hallintaa.

Viitteet

Ambler S. (2007) *Introduction to Test Driven Design (TDD*). Ambysoft Inc. http://www.agiledata.org/essays/tdd.html (25.5.2008).

AtBusiness Oy (2006). *052006Framework.* Pdf-dokumentti, AtBusiness Oy.

AtBusiness Oy (2008). Yrityksen kotisivu. http://www.atbusiness.com/default.aspx (14.6.2008).

Bach J. (2002): *Exploratory Testing Explained.* Satisfice Inc. http://www.satisfice.com/articles/et-article.pdf (1.6.2008).

Bach J., Kaner C., Pettichord B. (2002) *Lessons Learned in Software Testing*. John Wiley & Sons, Inc. New York.

Bacioccola, A., Catelani, M., Ciani, L., Scarano, V. (2008) A Novel Approach To Automated Testing To Increase Software Reliability. *I2MTC 2008 IEEE International Instrumentation and Measurement Technology Conference.* Proceedings, 1499 – 1502.

Boyer, K. (2000). Test Process Improvement: A practical step-by-step guide a structured testing. *ACM SIGSOFT Software Engineering Notes*, 25(3), 59-60.

Dumke, R., Farooq, A. (2007). Reasearch Directions in Verification & Validation Process Improvement. *ACM SIGSOFT Software Engineering Notes*, 32(4), 1-4.

Dustin, E., Rashka, J., Paul, J. (1999) *Automated Software Testing. Introduction, Management and Performance.* Addison-Wesley.

Itkonen J., Rautiainen K. (2005) Exploratory Testing: A Multiple Case Study. *Empirical Software Engineering. International Symposium on. IEEE Conference Proceeding 17-18 Nov. 2005,* 84-93.

Kaner C. (2008a) *A Tutorial in Exploratory Testing. Exploratory Testing.* Quest, http://www.kaner.com/pdfs/QAIExploring.pdf (25.5.2008).

Kaner C. (2008b) *The Ongoing Revolution in Software Testing*. Center for Software Testing Education & Research. http://www.testingeducation.org/articles/testingRevolution2007.pdf (31.5.2008).

Karlström D.,Norden S.,Runeson P. (2005) A minimal test practice framework for emerging software organizations. *Software Testing, Verification and Reliability*, 15, 145-166.

Katara M. (2005) Testauksen työvälineet nyt ja tulevaisuudessa – akateeminen näkemys. *Systeemityö* 1, 15-17.

Kit E. (1995): *Software Testing in the Real World.* Addison-Wesley Publishing Company. United States of America.

Myers G. (1979): *The Art of Software Testing.* John Wiley & Sons, Inc. United States of America.

Pezze M., Young M. (2008) *Software Testing and Analysis: Process, Principles and Techniques.* John Wiley & Sons, Inc. United States of America.

Pohjolainen P. (2003) *Ohjelmistojen testauksen automatisointi.* Pro gradu -tutkielma. Kuopion yliopisto. http://www.cs.uku.fi/research/Teho/PenttiPohjolainen_Gradu.pdf (31.5.2008).

Pohjolainen P. (2008) Testauksen historia Suomessa 1950-luvulta lähtien. *Testauspäivät TTY:llä 25.-26.3.2008,* http://www.cs.tut.fi/tapahtumat/testaus08/Pentti.pdf (26.5.2008).

Puolitaival O-P. (2008) Model-Based Testing Tools. *Testauspäivät TTY:llä 25.-26.3.2008.* http://www.cs.tut.fi/tapahtumat/testaus08/Olli-Pekka.pdf (31.5.2008).

Puruskainen P-M. Business Director, AtBusiness Oy (2008). Haastattelu, 1.7.2008.

Pyhäjärvi M., Pöyhönen, E. (2004) *Testauksen välineet ja testauksen automatisointi*. http://www.cs.jyu.fi/~kolli/testaus2006/materiaali/8_TestausvalineetJaTestauksenAutomatisointi_v0_1.ppt (1.6.2008).

Pyhäjärvi M., Pöyhönen E. (2006) Testaustutoriaali. *Testauspäivät TTY:llä 14.-15.8.2006.* http://www.cs.tut.fi/tapahtumat/testaus06/maaret_eki.pdf (1.6.2008).

Pyhäjärvi M., Pöyhönen E. (2008) Testauksen kehittäminen. *Testauspäivät TTY:llä 25.-26.3.2008*. http://www.cs.tut.fi/tapahtumat/testaus08/ (1.6.2008).

Siegberg R. (2005) Mallipohjainen testiautomaatio: perusteet ja huomioita käyttöönotosta. *Systeemityö* 1, 18-21.

Sogeti (2008). Kotisivu. http://eng.tmap.net/Home/index.jsp (29.6.2008).

Stenberg A. (2007) *Ohjelmistojen testaus 2007 – testauksen kehittäminen*. Luentomoniste. Tampereen teknillinen yliopisto, Porin yksikkö.
http://www.pori.tut.fi/~stenberg/index_files/kehittaminen.pdf (31.5.2008).

Toroi T. (2006) *Testausprosessin kehittäminen prosessimallien avulla*. Tutkimus Avointa-tutkimushankkeessa. Kuopion yliopisto.
http://www.uku.fi/tike/his/avointa/julkaisut/ProsessinKehitys.pdf (31.5.2008).

Liite 1

Hei kaikki,

Yhtenä tavoitteenamme Atlas jälkitoimituksissa on tuotteen laadun parantaminen, näin ollen myös testauskäytännön kehittäminen ja parantaminen. Alla pari kysymystä, joihin toivoisin että ottaisit kantaa ja kertoisit mielipiteesi, vaikka vain muutamalla sanalla.

(same in English)

Hi everyone,

One of our goals in Atlas next deliveries is improving the quality of the product, it means improving the testing practice as well. There is a couple of questions below, which I hope you can answer and tell me if you have any thoughts on the subject, even by a few words.

Kiitos,
Thank you,

Tuula

ps.
Tämä liittyy myös tekeillä olevaan graduuni ..
This is also related to my master's thesis I'm working on ..

Atlas – testaamisen kehittäminen
Mielipidekysely

1. *Onko nykyisessä testaamiskäytännössä mielestäsi puutteita tai ongelmia; jos, niin millaisia?*
2. *Nykyisen testaamiskäytännön hyvät puolet ?*
3. *Miten testaamiskäytäntöä ja –prosessia voidaan jatkossa kehittää (parannusehdotuksia) ?*

(same in English)

Atlas - Testing Improvement
Inquiry

1. *Is there any problems or shortcomings in current testing practice; if, what kind of ?*
2. *Good points of current testing practice ?*
3. *How can we improve the testing practice and the testing process in the future (any suggestions) ?*

Atlas-mielipidekyselyn 05/2008 tulokset

Mielipidekysely lähetettiin 21 henkilölle ja kyselyyn vastasi 9 henkilöä. Kyselyssä oli kolme kysymystä ja kuhunkin oli mahdollista antaa 0-n mielipidettä. Mielipiteitä annettiin yhteensä 87 kpl ja ne jakautuivat seuraavasti:

1. kysymys: 42 mielipidettä
2. kysymys: 13 mielipidettä
3. kysymys: 32 mielipidettä

Vastaukset analysoitiin ja jaoteltiin sovellutun mallin mukaan, jossa testaus jakautuu seitsemään osa-alueeseen:

1 Testaustekniikat
Testauksen eri vaiheet, testausmenetelmät
2 Testausorganisaatio
Testaustiimin jäsenet, vastuualueet
3 Koulutus, perehdytys
Tietämys ja sen kehittäminen tuotteesta, sovelluksesta, välineistä
4 Testauksen hallinta
Aikataulutus, työmääräarviointi, testauksen suunnittelu
5 Testausympäristö
Testikanta, vastinajat
6 Virheiden hallinta
Virheiden raportointi, versionhallinta
7 Viestintä ja kommunikaatio
Projektin keskinäinen, asiakkaan ja toimittajan välinen

Kyselyn tulokset ovat alla olevassa matriisissa:
vaakatasolla kysymykset 1,2,3
pystytasolla testauksen osa-alueet 1-7

	Ongelmat	Hyvät puolet	Parannusehdotukset	Yhteensä
1 Testaustekniikat	13	9	11	33
2 Testausorganisaatio	8	2	6	16
3 Koulutus, perehdytys	7	0	3	10
4 Testauksen hallinta	11	1	9	21
5 Testausympäristö	3	0	1	4
6 Virheiden hallinta	0	1	0	1
7 Viestintä ja kommunikaatio	0	0	2	2
Yhteensä	42	13	32	87