Jani Arovaara

TESTAUSPROSESSIN KEHITTÄMINEN – KESKEISIÄ KYSYMYKSIÄ

Tietojärjestelmätieteen
pro gradu -tutkielma
14.6.2008

Jyväskylän yliopisto
Tietojenkäsittelytieteiden laitos
Jyväskylä

TIIVISTELMÄ

Arovaara, Jani Petteri

Testausprosessin kehittäminen – keskeisiä kysymyksiä / Jani Arovaara

Jyväskylä: Jyväskylän yliopisto, 2008.

120 s.

Tietojärjestelmätieteen pro gradu -tutkielma

Tutkielmassa tarkastellaan ohjelmistojen testausprosessin kehittämistä. Eritoten keskitytään testausprosessin kehittämisen aikana tai kehittämistä suunniteltaessa huomiota vaativiin kysymyksiin. Testausprosessin kehittämisen avuksi voidaan valita jokin useista testausprosessin kehitysmalleista tai kehitystä voidaan tehdä mallia käyttämättä. Myös kehitysmalleihin liittyviä kysymyksiä otetaan tutkielmassa huomioon.

Tutkielman tavoitteena on kehittää testausprosessin kehittämisen ymmärrystä useammalta kannalta. Tavoitteeseen pyritään esittelemällä neljä testausprosessin kehitysmallia. Malleihin liittyen esitetään kysymyksiä, joihin haetaan vastauksia kirjallisuudesta ja muista kehitysmalleista. Yleisiin testausprosessin kehittämistä koskeviin kysymyksiin pyritään etsimään vastauksia esitellyistä kehitysmalleista sekä muusta kirjallisuudesta. Tutkielman aihetta käsitellään lähinnä kirjallisuustutkimuksen pohjalta.

Tutkimuksen tuloksena esitetään keskeisiä testausprosessin kehityksen yhteydessä huomioon otettavia kysymyksiä ja niiden vastauksia. Lisäksi testausprosessin kehitysmallien pohjalta esitetään joitain keskeisiä huomiota vaativia asioita kehitykseen, joka tapahtuu ilman kehitysmallin tukea.

AVAINSANAT: Testausprosessin kehittäminen, prosessin kypsyys, testaaminen, TMM, TMMi, TIM, TPI

ESIPUHE

Tämä pro gradu -tutkielma on tehty vuonna 2008 Jyväskylän yliopiston tietojärjestelmätieteen laitoksen ohjelmistotuotannon linjalle.

Tahdon kiittää vaimoani Jenniä, jonka kannustus ja tuki oli työn aloittamisen ja loppuun saattamisen kannalta korvaamatonta. Kiitokset myös tyttärelleni Seenalle piristyksestä työn lomassa.

Kiitokset kuuluvat ehdottomasti myös ohjaajalleni, FT, professori Markku Sakkiselle joustavuudesta kiireisenä aikana sekä tehokkaasta yhteydenpidosta.

Jyväskylässä 14.6.2008

TERMIT JA LYHENTEET

Ad hoc	Latinaa: ”tähän tarkoitukseen.” Ratkaisuista tai toiminnasta puhuttaessa ad hoc viittaa usein hiukan kaoottiseen tai improvisoituun toimintaan.
ATR	Activities and tasks with responsibilities. TMM:n rakenteen osa, jonka kautta saavutetaan jokin tietty kypsyyden osatavoite, MSG. Suomennettu: aktiviteetit, tehtävät ja velvollisuudet.
CM	Configuration management. Konfiguraation hallinta sisältää muun muassa dokumentaation, ohjelmakoodin, testien sekä ympäristön versionhallinnan. Kehitysmallista riippuen konfiguraation hallinta voi sisältää osan mainituista hallinnan kohteista tai mainittuja enemmänkin.
CMM	Capability Maturity Model. SEI:n kehittämä CMMI:ia edeltänyt asteittainen prosessin kehitys- ja arviointimalli.
CMMI	Capability Maturity Model Integration. SEI:n CMM:n pohjalta kehittämä lähestymistapa prosessien kehittämiseen ja arviointiin.
CMMI-DEV	CMMI for Development. CMMI:n malli tuotekehitysprosessien kehittämiseen ja mittaamiseen. CMMI-DEV sisältää sekä asteittaisen että jatkuvan kehitysmallin.
IEEE	Institute of Electrical and Electronics Engineers. Kansainvälinen, muun muassa standardointia harjoittava organisaatio.

ISO	International Organization for Standardization. Kansainvälinen standardointiorganisaatio.
ISTQB	International Software Testing Qualifications Board. Kansainvälinen järjestö, joka ylläpitää testaajien koulutus- ja sertifiointiohjelmaa sekä testausalan termistöä.
PDCA	Plan-Do-Check-Act. W. Edwards Demingin esittämä, Walter Shewhartin ajatuksiin perustuva ongelmanratkaisumalli.
Refaktorointi	Kirjallisen materiaalin, kuten ohjelmakoodin muokkaaminen on refaktorointia, kun se tehdään esimerkiksi luettavuuden tai rakenteen kehittämistarkoituksessa siten, että alkuperäinen merkitys säilyy muuttumattomana.
SEI	Software Engineering Institute. USA:n valtion ja etenkin puolustusvoimien rahoittama Carnegie Mellon Universityn (CMU) yksikkö.
SPICE	Software Process Improvement and Capability dEtermination, virallisesti ISO/IEC 15504. Standardi prosessien kehittämiseen ja niiden kyvykkyyden arviointiin.
Spiraalimalli	Barry Boehmin vuonna 1988 esittämä spiraalimalli on ohjelmistotuotantoprosessia kuvaava inkrementaalinen malli, jossa suunnitellaan, luodaan ja arvioidaan tuotteeksi kehittyviä prototyyppejä.
TAMAR	TMMi Assessment Method Application Requirements. Kehikko, jonka vaatimusten mukaan on mahdollista kehittää TMMi:n mukaisia arviointimalleja.

TDD	Test Driven Development. Ketterään ohjelmistokehitykseen liittyvä testausmetodologia, jossa luodaan suunnitellut toiminnot täysin kattavat automaattiset testitapaukset ennen ohjelmakoodin kirjoittamista.
TIM	Test Improvement Model. Ruotsissa Linkopingin yliopistolla ja Saab Combitechillä kehitetty jatkuva testausprosessin kehitysmalli, joka sisältää myös arvioinnin.
TMap	Test Management approach. Sogeti-yrityksen kehittämä rakenteinen testausmenetelmä.
TMM	Testing Maturity Model. Illinois Institute of Technologyn CMM:n tueksi kehittämä asteittainen testausprosessin kehitysmalli.
TMM-AM	TMM Assessment Model. TMM:lle luotu laajennos, joka lisää siihen arviointimallin.
TMMi	Testing Maturity Model Integration. TMMi Foundationin CMMI:n tueksi kehittämä testausprosessin kehitysmalli. Malli on vielä kehityksen alla ja on toistaiseksi vain asteittainen eikä sisällä arviointia (ks. TAMAR).
TMMi Foundation	TMMi:ia ja TAMAR:ia kehittävä kansainvälinen organisaatio.
TPI	Test Process Improvement. Sogeti-yrityksen kehittämä TMap:iin tukeutuva jatkuva testausprosessin kehitysmalli, joka soveltuu myös arviointiin.

V-malli	Ohjelmiston kehitysmalli, joka jakautuu karkeasti suunnitteluun, toteutukseen ja testaukseen. Se esitetään V-kirjaimena, jossa kyljet ovat suunnittelu sekä testaus ja kärkenä on toteutus.
V2M2	Verification and Validation Maturity Model. Nimi, jolla TMMi tunnettiin aiemmin kehityksensä aikana.
Vesiputousmalli	Winston Roycen vuonna 1970 kehittämä vesiputousmalli on ohjelmistotuotantoprosessia kuvaava malli. Se esitetään yleisesti varsin yksinkertaistettuna ja täysin lineaarisena.
Virheestys, virheestin	Englanniksi debugging, debugger. Termi tarkoittaa olemassaolevien virheiden syiden paikantamista ja korjaamista. Toiminnalle ei ole Suomen kielessä ollut hyvin kuvaavaa ja käytettäväksi soveltuvaa termiä. Termit ”virheiden poisto” sekä ”perkaus” sopivat merkitykseen varsin heikosti. Virheestys ajatellaan virheiden ”metsästykseksi” (vrt. sorsastus), jossa suuri osa toiminnasta on nimenomaan virheen syyn paikantamista.

SISÄLLYSLUETTELO

1 JOHDANTO

Ohjelmistotuotantoyrityksissä pyritään jatkuvasti kehittämään prosesseja tehokkaammiksi ja laadukkaammiksi. Testaus on tärkeä, mutta usein aliarvostettu osa ohjelmistotuotantoyrityksen toimintaa. Testausta kehittämällä voidaan kuitenkin saada nostettua koko tuotantoprosessin tehokkuutta ja lopputulosten laatua selkeästi. Testausprosessin kehitystä voidaan suunnitella erilaisten kehitysmallien pohjalta suhteellisen helposti, mutta muutetun tai uuden testausprosessin käyttöönotossa saattaa tulla vastaan ongelmia. Näitä ongelmia voidaan lievittää ja poistaa ottamalla käyttöönotossa ja prosessin kehityksessä paremmin huomioon sekä muutoksen aiheuttamat että koko organisaation tarpeet.

Tavoitteena on löytää kehitysmallien avulla tekijöitä, jotka testausprosessia kehitettäessä vaikuttavat organisaatioon, sen testausprosessia ympäröiviin prosesseihin, työkaluihin tai tuotteisiin. Tutkimuksessa pyritään painottamaan tekijöitä, joiden vaikutukset yltävät selkeästi henkilöstöön tai itse organisaation toimintaan.

Testausprosessin kehitysmalleja on laaja ja varsin vaihteleva valikoima. Näistä valitseminen voikin olla huomattava urakka, sillä se vaatii mahdollisesti useaan kehitysmalliin tutustumista. Lisäksi kehitysmalleista on mahdollista nostaa esiin erilaisia kysymyksiä kyseisen mallin, tai kilpailevien mallien toimintaan ja soveltuvuuteen liittyen. Millaisia kysymyksiä tai aiheita testausprosessin kehityksessä sitten tulee vastaan ja miten näitä voidaan ratkaista? Onko kehitysmallin käytölle perusteita? Miten kehitysmallin valitsemista voidaan helpottaa?

Tutkimuksessa pyritään löytämään ratkaisuja edellä esitettyihin kysymyksiin kirjallisuustutkimuksen keinoin. Tarkasteltavaksi valitaan neljä tunnettua tai perustellusti lupaavalta vaikuttavaa testausprosessin kehitysmallia:

- TMM, joka pitkään oli lähes de facto standardi testausprosessin kehitysmalliksi.
- TIM, jonka rakenne tekee siitä tietyissä tapauksissa TMM:ia houkuttelevamman, mutta aiheuttaa myös joitain rajoituksia.
- TPI, joka on tämän hetken suosituimpia testausprosessin kehitysmalleja ja erittäin mukautuvainen organisaation tarpeisiin.
- TMMi, joka on vielä kehityksen alla, mutta TMM:iin pohjautuvana ja hyvin tutkittuna mallina vaikuttaa erittäin lupaavalta.

Malleja on tarkasteltavana erittäin rajallinen määrä. Jo tarkastelluista malleista erotetaan joitain piirteitä, joiden perusteella mallin valintaa pystytään helpottamaan. Selkeimpiä piirteitä ovat mallin tyyppi (jatkuva/asteittainen), raskaus, joustavuus sekä dokumentaation kattavuus.

Tutkimuksessa esitetään ensin lyhyesti testauksen historiaa. Esityksessä pyritään tuomaan esille tärkeitä testauksen tuntemuksen kehitysaskeleita. Historiasta edetään testausprosessin kehittämisen motivaatioon ja yleiseen kuvaukseen, jonka jälkeen kuvataan valitut kehitysmallit. Luku 5 keskittyy kuvaamaan testausprosessin kehityksessä huomioitavia asioita ja etsimään näille ratkaisuja. Yhteenvedossa kerrotaan yleisemmin tutkimuksen tuloksista ja rajoitteista.

2 TESTAUKSEN HISTORIAA

Testaaminen on kehittynyt vähitellen laitteiston, ohjelmointiteknologioiden ja testausta tukevien ohjelmistojen kehityksen ratkaistua testaamista aiemmin estäneitä ja vaikeuttaneita käytännön ongelmia. Kun testien ajamisen ja ohjelmien analysoinnin vaatiman suoritusajan hinta on laskenut, on pystytty kehittämään tehokkaampia testausmenetelmiä. Ohjelmakoodin testaamisen vähittäisen kehittymisen rinnalla ovat myös testaamisen suunnittelu ja testausprosessit kokonaisuutena kehittyneet.

Tässä luvussa käsitellään lyhyesti testaamisen kehittymistä 1950-luvulta nykypäivään. Lyhyt aikajana testauksen kehityksestä, sen ”kasvun vaiheista”, on esitetty taulukossa 1 (Gelperin, Hetzel 1988). On syytä huomioida, että esitetty aikajana kuvaa lähinnä testauksen teorian kehittymistä, eikä niinkään käytäntöä. Moni yritys toimii 2000-luvulla edelleen aikajanan tuhoamiseen suuntautuneen kauden tasolla.

Taulukko 1: Testauksen kasvun vaiheet & testauksen mallit (Gelperin, Hetzel 1988) yhdistettynä.

- 1956	Virheestykseen suuntautunut kausi
	Tavoitteena löytää esiintyneet virheet
1957 - 1978	Esittelyyn suuntautunut kausi
	Tavoitteena varmistaa, että tuote toimii määritellysti
1979 - 1982	Tuhoamiseen suuntautunut kausi
	Tavoitteena löytää virheitä toteutuksesta
1983 - 1987	Arviointiin suuntautunut kausi
	Tavoitteena löytää virheitä vaatimuksista, suunnitelmista ja toteutuksesta
1988 -	Ennaltaehkäisyyn suuntautunut kausi
	Tavoitteena estää virheiden päätyminen vaatimuksiin, suunnitelmiin ja toteutukseen

2.1 Testaus 1950-luvulla

1950-luvulla tietokoneet olivat harvinaisia ja kalliita laitteita. Niiden käyttöön kului rahaa helposti enemmän, kuin niitä käyttävien henkilöiden palkkoihin. Korkeista kustannuksista johtuen ohjelmakoodin testaaminen pyrittiin mahdollisuuksien mukaan toteuttamaan ilman, että sitä suoritettiin tietokoneella. Testejä ajettiin siis ajatusharjoituksina ja ohjelmakoodin tarkasti mahdollisesti joku muukin, kuin koodin kirjoittaja. (Boehm 2006)

Ohjelmistotuotanto oli erittäin uusi ala. Kustannusten lisäksi eräs toinen syy kuvatunlaiseen toimintaan olikin, että valtaosa kauden ohjelmistosuunnittelijoista oli alunperin laitteistoinsinöörejä ja matemaatikoita. Ala lainasi työntekijöiden lisäksi myös suuren osan kehitysmalleista muilta insinöörialoilta, joilla kehitystyö on perinteisesti ollut huomattavasti nykyistä ohjelmistotuotantoa muodollisempaa. Näillä aloilla ei tuotosten testaaminen ja muokkaaminen käytännön syistä johtuen ole ollut yhtä joustavaa kuin ohjelmakoodin testaaminen on. Testaaminen 1950-luvulla keskittyi lähinnä ohjelmakoodin virheestykseen virhetilanteissa. (Boehm 2006)

2.2 Testaus 1960-luvulla

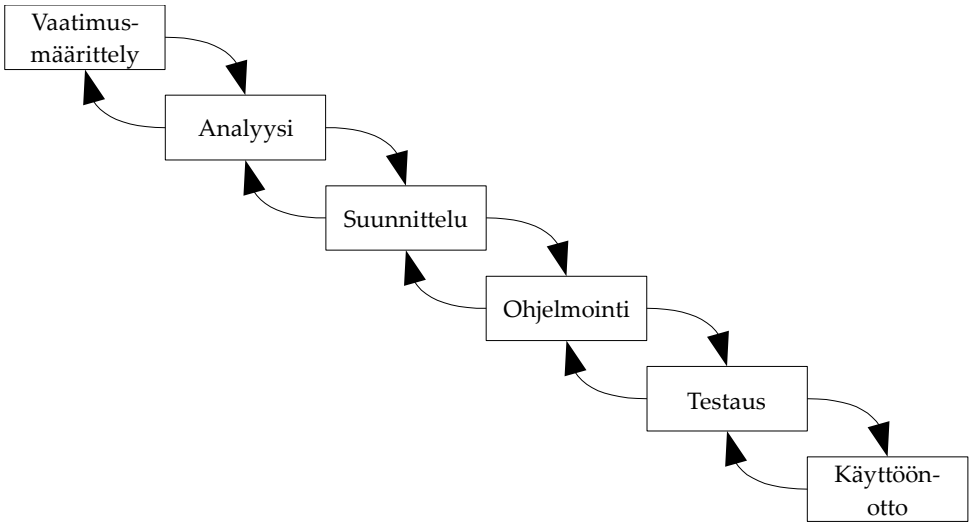
Vähitellen ohjelmistotuotannon parissa työskenteleville selvisi, että ohjelmakoodin muokkaaminen on paljon helpompaa kuin fyysisten tuotteiden kehittäminen ja korjaaminen. 1960-luvulla laitteisto kehittyi ja ohjelmistojen kysyntä alkoi kasvaa huomattavasti, eikä työvoimaa pystytty hankkimaan enää pelkästään insinöörejä ja matemaatikoita palkkaamalla. Ohjelmistotuotannon pariin alettiin palkata ihmisiä, joille tiukasti muodollinen toiminta oli vierasta. (Boehm 2006)

Tapahtuneiden muutosten ansiosta vallalle pääsi eräänlainen koodaa ja korjaa -mentaliteetti. Testaus oli kuitenkin vielä varsin alkeellista ja koostui edelleen lähinnä ongelmatilanteiden virheestyksestä sekä valmiin ohjelmakoodin suorittamisesta. Testauksen tavoitteena oli varmistaa, että ohjelma toimii ja siitä löytyy määritellyt toiminnot (Gelperin, Hetzel 1988).

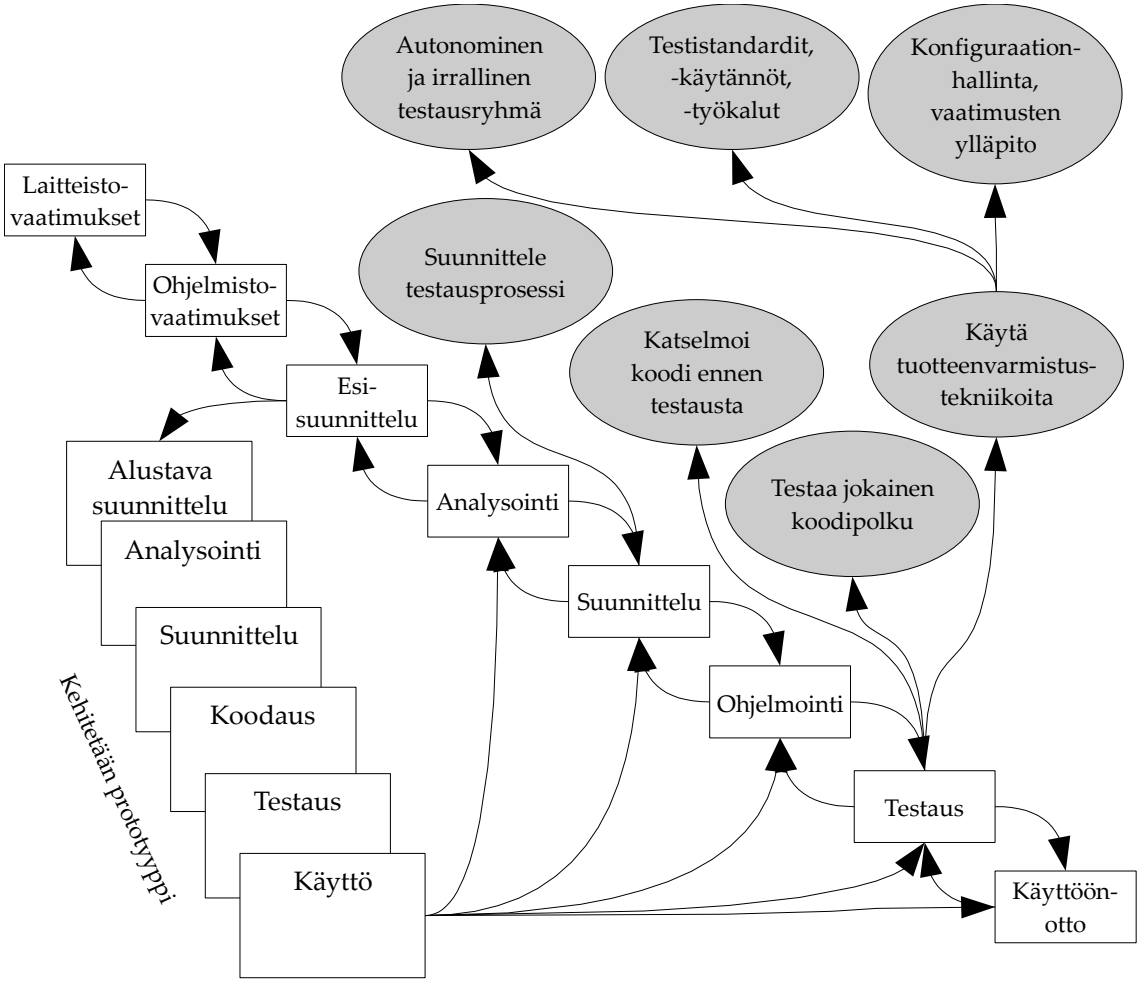
2.3 Testaus 1970-luvulla

Ohjelmistotuotantoalan opittua aiemmista virheistään, syntyi 1970-luvulla vastareaktiona paremmin organisoituja tuotantoprosesseja. Uusissa prosesseissa koodin kirjoittamista edelsi suunnittelu sekä vaatimusten määrittely. Selkeästä vaiheistuksesta kehittyi ohjelmistotuotannon vesiputousmalli (Royce 1970) ja sen yksinkertaisemmat variaatiot (esim. Boehm 1976, 1227), jotka ovat käytännössä nykyaikaisten tuotantoprosessimallien pohjana.

Käytännössä Roycen esittämistä vesiputousmalleista on kirjallisuudessa keskitytty erittäin yksinkertaiseen malliin (Kuvio 1). Royce ajatteli kevyemmän mallin olevan toimiva lähinnä pienemmille projekteille, tai jos laajemman mallin käytöstä ei projektille olisi nähtävissä hyötyjä. Roycen huomattavasti kattavampi, myös testauksen hyvin huomioon ottava malli (Kuvio 2) on lähes unohdettu. Lisäksi yksinkertaisempaakin mallia on tulkittu yleisesti siten, että prosessin vaiheet seuraavat toisiaan ilman kunnollista interaktiota, mikä ei ollut Roycen tarkoitus. (Royce 1970) Vesiputousmalliin kuuluu kiinteästi testaus, joka mallin yksinkertaisemmissa ja yleisimmin käytetyissä variaatioissa on ajateltu tuotantoprosessin viimeiseksi vaiheeksi ennen käyttöönottoa. Tämän tulkinnan mukaan toimittaessa ohjelmakoodia testataan kunnolla vasta, kun sen kuuluisi olla jo valmista.



Kuvio 1: Roycen kevyempi vesiputousmalli, kuten se kirjallisuudessa esitetään. Alkuperäiseen malliin kuuluu reitit testauksesta suunnitteluun ja siitä määrittelyyn (Royce 1970).



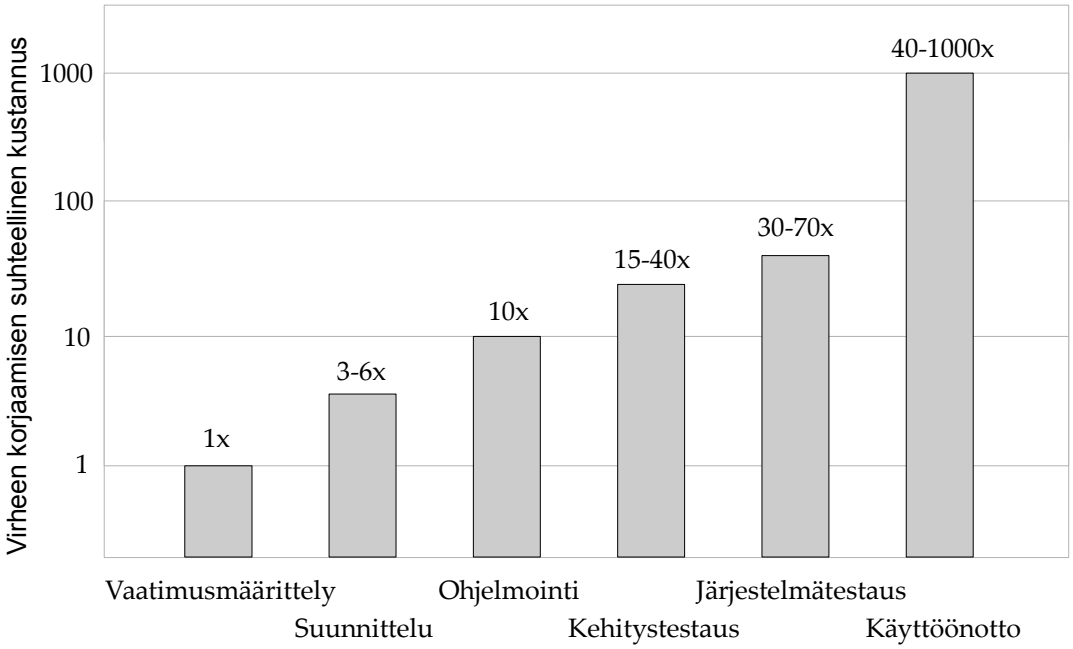
Kuvio 2: Roycen kattavamman vesiputousmallin askelet 3 ja 4 (Royce 1970).

1970-luvulla tapahtui edistystä myös metriikoissa, joiden avulla pystyttiin tunnistamaan virhealttiita moduuleita (Boehm 2006). Testaustoiminta kehittyi vuosikymmenen loppua kohti selkeästi, ja tavoitteeksi muodostui löytää virheitä ohjelman toteutuksesta (Gelperin, Hetzel 1988).

2.4 Testaus 1980-luvulla

1980-luvulla organisaatiot heräsivät huomaamaan, että jos esimerkiksi 60 % sovelluksen kehittämiseen käytetystä ajasta käytetään testaamiseen, kuluu tästä 70 % aiempien ohjelmistokehitysvaiheiden tuotosten korjaamiseen. Virheiden korjaamisen kustannuksista ohjelmistokehityksen eri vaiheissa tehtiin tutkimuksia jo 1970-luvulla. Tutkimuksista selvisi, että korjaamiseen kuluu vähemmän resursseja, jos virheet löydetään aiemmin, lähempänä sitä kehitysvaihetta, josta virhe on peräisin. (Boehm 2006) Boehmin vuonna 1976 kokoama kaavio ohjelmistovirheiden korjaamisen kustannuksista suhteessa kehitysvaiheeseen (Boehm 1976, 1228), on edelleen paikkansa pitävä hyvin pienin korjauksin (Kuvio 3).

Vuonna 1988 Boehm julkaisi ohjelmistotuotantoprosessin spiraalimallin (ks. Boehm 1988). Spiraalimalli poikkeaa huomattavasti Roycen vesiputousmallista, sillä se on vahvasti iteratiivinen ja prototyypittelyyn panostava. Spiraalimallin iteraatiot vastaavat pitkälti vesiputousmallin askeleita. Lisäksi jokaiseen iteraatioon kuuluu vähitellen kehittyvän prototyypin toteuttaminen ja arviointi sekä iteraation varsinaisen tuotoksen arviointi, tarkastaminen tai testaus. Myös spiraalimalli sijoittaa ohjelmiston testauksen vasta viimeisen iteraation lopulle. (Boehm 1988) Vaihetuotosten tarkastaminen, prototyyppien kehittäminen ja riskien hallinta luovat kuitenkin hyvän pohjan testauksen laajentamiselle koko ohjelmistotuotannon kattavaksi toiminnaksi.



Kuvio 3: Virheen korjaamisen suhteellinen kustannus (Boehm 1981).

Yhdysvaltojen puolustusministeriö huomasi tarvitsevansa keinon arvioida ohjelmistojensa tuottajien prosessien kypsyyttä ja tilasi tähän tarkoitukseen työkalun SEI:lta (Carnegie Mellon University, Software Engineering Institute). (Boehm 2006) Tulos oli SW-CMM (Capability Maturity Model for Software), tai yleisemmin vain CMM. Valitettavasti CMM (Chrissis ym. 1993) ei ota kantaa testaukseen. SW-CMM:sta myöhemmin kehittynyt CMMI-DEV (Capability Maturity Model Integration for Development) mainitsee testauksen joitain kertoja, antaen myös esimerkkejä yksikkö- sekä hyväksymistestauksen menetelmistä (CMMI 2006, 474, 497). Se ei kuitenkaan paneudu itse testausprosessiin. Ohjelmistotuotannon prosessien kypsyyden arviointi- ja kehitystyökaluiksi kyseiset mallit jättävät siis testauksen varsin vähälle huomiolle.

1980-luvulla testaaminen alkoi kuitenkin nousta selkeämmin esiin. Testaamiseen keskittyviä konferensseja alettiin järjestää, ja testaamiseen

keskittyviä kirjoja julkaistiin. Testaamisen asema näkyi myös siinä, että yritykset alkoivat hakea työntekijöitä nimenomaan testaajiksi. (Gelperin, Hetzel 1988) Ehkä selkein merkki testaamisen aseman kohoamisesta oli kuitenkin kolmen tärkeän IEEE-standardin kehittäminen. Ohjelmistojen testauksen dokumentaatiota varten kehitettiin IEEE 829-1983. Ohjelmistojen verifiointiin ja validointiin kehitettiin IEEE 1012-1986. Näitä kahta standardia on päivitetty ajanmukaisemmiksi vuonna 1998. (IEEE 829-1998 ja IEEE 1012-1998) Lisäksi kehitettiin yksikkötestauksen standardi ANSI/IEEE 1008-1987 (ANSI/IEEE 1008-1987). Testauksen tavoitteet etenivät ohjelmiston elinkaaressa toteutuksesta alkuun päin, ja pyrittiin löytämään toteutuksen lisäksi virheitä tuotantoprosessin aiempien vaiheiden tuotoksista. Vuosikymmenen lopulla tavoite kehittyi vielä siten, että pyrittiin virheiden löytämisen lisäksi aktiivisesti estämään niiden syntymistä. (Gelperin, Hetzel 1988)

2.5 Testaus 1990-luvulla

Eräs testauksen kannalta tärkeä malli on Beizerin vuonna 1990 julkaisemassa kirjassa (Beizer 1990) esitetty testaajan mielentilan malli (Tester's Mental Phases) (Taulukko 2). Malli kuvaa yksittäisen testaajan henkisen kehittymisen vaiheita, jotka vastaavat hyvin pitkälti Gelperinin ja Hetzelin testauksen evoluutiomallia (Taulukko 1). Mallin vaiheet 3 ja 4 poikkeavat hieman Gelperinin ja Hetzelin mallin tavoitteista. Vaiheessa 3 testaaja pyrkii aktiivisesti optimoimaan tuotteessa olevien virheiden määrää ja laatua siten, että tuote vastaa sille asetettuja vaatimuksia, mukaan lukien laatuvaatimuksia. Vaiheessa 4 tuntemus testaamisen mahdollisuuksista sekä ohjelmiston testattavuudesta johtaa ohjelmiston kehittämiseen tavalla, joka vähentää testaustyön tarvetta.

1990-luvulla alkoi ohjelmistojen testausalalla huomio kiinnittyä myös siihen, miten testausprosesseja kehitetään. Vuosikymmenen puolivälin jälkeen

julkaistiin useita hyviä kehitysmalleja. Tässä tutkimuksessa keskitytään näistä kolmeen, sekä yhteen 2000-luvulla julkaistuun malliin. Malleja kuvaillaan tarkemmin tutkimuksen luvussa 4.

Taulukko 2: Beizerin testaajan mielentilan malli (Tester's Mental Phases) (Beizer 1990).

Vaihe	**Tunnusmerkki**
Vaihe 0	Testaaminen = Virheestäminen
Vaihe 1	Testaaminen = Ohjelman toiminnan esittely
Vaihe 2	Testaaminen = Ohjelman toimimattomuuden esittely
Vaihe 3	Testaaminen = Toimimattomuuden havaitun riskin laskeminen hyväksyttävälle tasolle
Vaihe 4	Testaaminen = Henkinen itsekuri, joka johtaa vähäriskiseen ohjelmistoon ilman paljoa testaustyötä

Malleista ensimmäinen oli vuonna 1996 kehitetty TMM (Testing Maturity Model). Sen mallina on käytetty CMM:ia, jonka kumppaniksi se on tarkoitettu, ja taustalla on myös jonkin verran tutkimustuloksia. (Burnstein ym. 1996a, 1996b ja 1996c) TMM sai suunnitellusti laajennuksen kypsyystason arviointiin vuonna 1998 (Burnstein ym. 1998).

Vuonna 1997 julkaistu TIM (Test Improvement Model) on itsenäinen testausprosessin kehitysmalli (Ericson ym. 1997) ja toinen tutkimukseen valituista malleista. Se poikkeaa TMM:sta jonkin verran myös rakenteeltaan, joka on vähemmän jäykkä ja helpompi pienten kehitysaskeleiden kannalta. TIM sisältää myös työkalut testausprosessin kypsyystason arviointiin.

Kolmas käsiteltävä malli on vuonna 1997 kehitetty TPI-malli (Test Process Improvement). TPI poikkeaa TMM:sta ja TIM:sta selkeimmin kaupallisuudellaan. Siitä ei juurikaan ole saatavilla tarkkaa tietoa mallin kehittäneiden Koomenin ja Polin vuonna 1999 julkaiseman kirjan ”Test Process Improvement: A step-by-step guide to structured testing” (Koomen, Pol 1999) lisäksi. TPI on kuitenkin selkeästi kattavin 1990-luvun malleista, ja sisältää kehitysmallin sekä arvioinnin lisäksi myös käytännön vihjeitä mallin mukaiseen etenemiseen.

1980-luvun lopulle saakka testauksen tavoitteet kehittyivät ja kypsyivät. Testaus käsitteenä laajeni alun perin pelkästä virheestyksestä kattamaan huomattavan paljon laajemman joukon ohjattuja toimintoja. Testauksesta muodostui siis prosessi. 1990-luvulla tavoitteeksi testauksen suhteen muodostuikin organisaatioiden testausprosessien kehittäminen vastaamaan tuotteiden laatuvaatimuksia.

2.6 Testauksen tilanne 2000-luvulla

Vuonna 2008 TMMi Foundation julkaisi neljännen tähän tutkimukseen mukaan otetun testausprosessin kehitysmallin, TMMi:n (Test Maturity Model Integration). TMMi pohjautuu TMM:iin ja siinä on vahvoja kytköksiä CMMI:iin. (van Veenendaal ym. 2008) TMMi on varsin kattava malli, mutta myös melko raskas ja jäykkä verrattuna TIM:iin tai TPI:iin.

Testaajien kouluttamistarve on tiedostettu 2000-luvulla hyvin, ja yleisesti hyväksyttyjä opetusohjelmia ja -materiaaleja on aktiivisesti kehitetty sekä aloittelevien (ks. ISTQB 2007a) että edistyneempien (ks. ISTQB 2007b) testaajien kouluttamista varten. Kehitystä on hoitanut vuonna 2002 perustettu International Software Testing Qualifications Board (ISTQB), johon osallistuu

kirjoitushetkellä 38 eri kansallisuutta tai aluetta edustavaa ryhmää, Suomi mukaan luettuna. Testausalan termistölle on mainitun organisaation toimesta myös kehitetty standardi (ks. ISTQB 2007c), jota on jatkuvasti laajennettu ja joka on tuoreimmassa versiossaan jo varsin kattava. Opetusohjelmien pohjalta on kehitetty myös testaajien sertifiointiohjelma, johon ISTQB tarjoaa ohjeistuksen, hyväksytyt opettajat sekä opetusohjelman, ja jonka nimeä sertifiointi (ISTQB Certified Tester) kantaa.

Ohjelmistotuotteiden laatu on 2000-luvulla noussut aiempaa selkeämmin esille. Suuri osa järjestelmistä on Internetin välityksellä kytköksissä muihin järjestelmiin tai saman järjestelmän kauempana sijaitseviin osiin. Avoimuus julkiseen tietoverkkoon päin asettaa järjestelmille kovia tietoturvavaatimuksia. Nämä ovatkin olleet viime vuosina ja etenkin viime kuukausina erittäin paljon esillä. Onkin selvää, että vaikka testauksen teoria ja mallit ovat kehittyneet edellisten reilun neljän vuosikymmenen aikana todella paljon, on ajoittain laajojenkin järjestelmien kehityksessä jätetty testaus aivan liian vähälle huomiolle. Testausprosessin kehitysmallien laajamittaisen soveltamisen aloittaminen olisi erittäin hyvä teema vuosikymmenen lopulle.

3 TESTAUSPROSESSIN KEHITTÄMISESTÄ

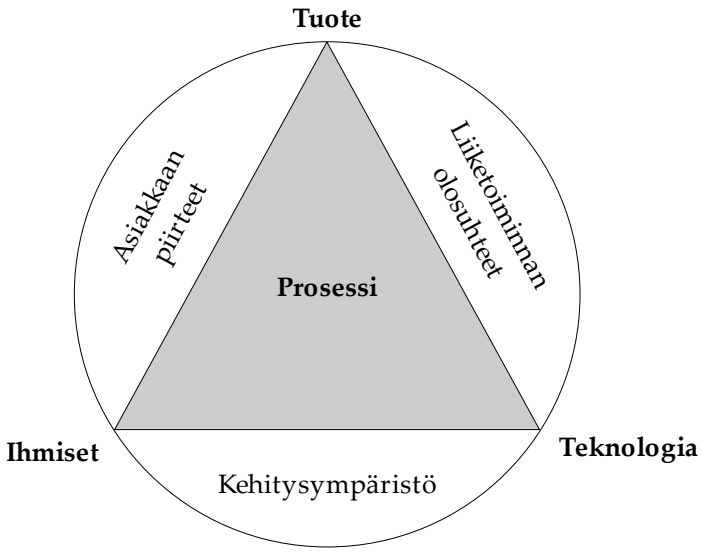
Prosessien kehittäminen on organisaatiolle työläs ja usein myös kallis toimenpide, eikä testausprosessin kehittäminen ole tästä poikkeus. Testausprosessin kehitykseen tulisi suhtautua, kuten organisaation muidenkin liiketoiminnan kannalta keskeisten prosessien kehittämiseen. Ei siis voida olettaa, että muutokset koskettaisivat vain testaajia sekä ohjelmoijia. Tulee tiedostaa, että kaikki tuotekehitykseen yhteydessä olevat henkilöt tulevat todennäköisesti kokemaan muutoksia omassa työssään. (ks. esim. Koomen, Pol 1999 ja van Veenendaal ym. 2008) Kyseisiin muutoksiin perehdytään myöhemmin tässä luvussa.

Muutokset prosessissa eivät välttämättä rajoitu työtehtävien muutoksiin. Prosessin ympäristöä kuvaa hyvin malli ohjelmistojen laadun ja organisaationaalisen tehokkuuden määräävistä tekijöistä (Kuvio 4). Sitä soveltamalla nähdään, että prosessin olennaisia osia ovat tuote, ihmiset ja teknologia. Prosessin, kuten testausprosessin, muutokset voivat siis vaikuttaa näihin kaikkiin. Testausprosessin muutosten vaikutukset saattavat ulottua myös prosessin ympäristöön, varsinkin kehitysympäristöön.

3.1 Motivaatio

Testausprosessin muutosten vaikutusten ollessa mahdollisesti hyvin kalliita ja aikaa vieviä, ei ole yllättävää, että kyseisiä muutoksia ei tehdä aivan kevein perustein. Muutoksiin voi kuitenkin olla sekä organisaation ulkoisia että sisäisiä syitä (Kuvio 4). Yleisiä syitä muutospaineille voivat olla esimerkiksi asiakkaan asettamat laatuvaatimukset, lakien asettamat vaatimukset, tarve säilyttää kilpailukyky tai yksinkertaisesti halu parantaa organisaation mainetta.

Olemassa olevassa prosessissa saatetaan myös havaita selkeitä puutteita alueilla, joita halutaan hallitusti kehittää vastaamaan organisaation tarpeita.



Kuvio 4: Ohjelmistojen laadun ja organisaationaalisen tehokkuuden määräävät tekijät (Pressman 2000, 80 ja Paulish, Carleton 1994, 51).

Valitettavan helposti käy kuitenkin niin, että kun motivaatio testausprosessin kehitykselle löytyy, ei organisaatiolla ole valmista mallia, jota kehityksessä voitaisiin noudattaa. Näin voi olla, vaikka yrityksellä olisi käytössä jokin tuotantoprosessin kehitysmalli, sillä nämä eivät ota riittävän tarkalla tasolla kantaa testausprosessiin (vrt. CMMI 2006). Jos organisaatiolla ei ole kehitysmallia, voi testausprosessin kehittämisestä aiheutua ylimääräisiä kustannuksia. Pahimmassa tapauksessa seurauksena voi olla toimimaton testausprosessi.

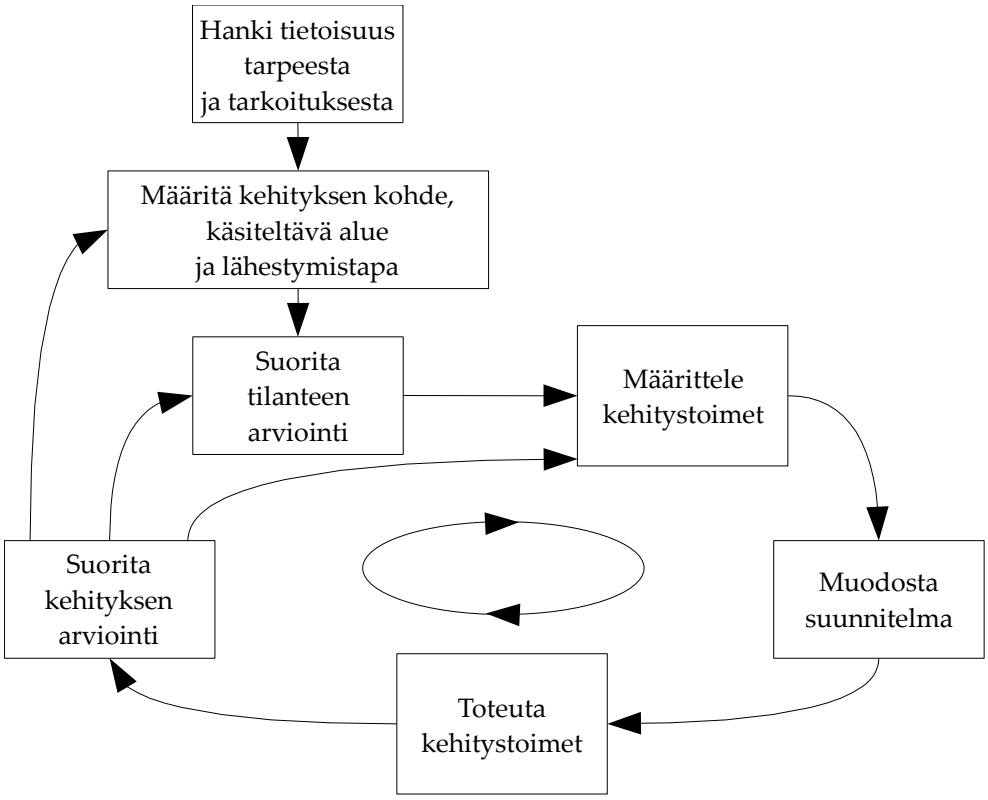
3.2 Kehittäminen

"Ohjelmistoprosessitkin ovat ohjelmistoja," sanoo Leon Osterweil lausahduksen mukaan nimetyssä artikkelissaan (Software Processes Are

Software Too, Osterweil 1987), joka käsittelee ohjelmistoprosessien kehittämistä ja sen samankaltaisuutta suhteessa ohjelmistojen kehittämiseen. Prosessien kehittämisellä todella on huomattavasti samoja piirteitä kuin ohjelmistojen kehittämisellä, ja tämä kannattaakin muistaa, kun testausprosessia kehitetään.

Käytetään sitten testausprosessin kehityksessä apuna jotain kehitysmallia tai ei, on kehitykselle syytä varata riittävästi aikaa ja huomiota. Kehityksessä voidaan noudattaa ohjelmistotuotannon prosessimalleja, kuten vesiputousmallia (ks. Royce 1970), spiraalimallia (ks. Boehm 1988) tai jotain monista kehittyneemmistä malleista. Pääasia kuitenkin on, että muutoksen tarpeet määritellään, muutos suunnitellaan, luodaan uusi prosessi ja testataan sitä pilottiprojektilla. Jos ilmenee määritysten toteutumisen suhteen puutteita, palataan prosessin suunnitteluun.

Jos testausprosessin nykytila ei ole tarkalleen selvillä, voi sen selvitykseen saada apua joltain kehitysmallilta. Nykytilanteen tunteminen on tarpeen, jotta kehitystä voidaan suunnitella realistisesti ja ottaa huomioon kaikki tavoitetilaan pääsyssä tarvittava (Koomen, Pol 1999, luku 6.1). Jos organisaatiolla on käytössä jokin testausprosessin kehitysmalli, sisältää se ohjeistusta muitakin prosessin kehitystoimia varten. Nyky- ja tavoitetilan määrittämisen ja tavoitteeseen pääsyn suunnittelun työkalut ovat testausprosessin kehitysmallien pääasiallinen anti. Prosessin kehittäminen kulkee yleisellä tasolla hyvin samankaltaisesti kehitysmallien käyttämisestä tai käyttämättä jättämisestä riippumatta (Kuvio 5). Yleisellä tasolla erona on lähinnä se, että kehitysmallista riippuen pystytään vaihtelevalla tarkkuudella vaikuttamaan siihen, mihin testausprosessin osa-alueisiin kehityksessä keskitytään.



Kuvio 5: Prosessin arviointi ja kehittäminen (Koomen, Pol 1998, 11).

4 TESTAUSPROSESSIN KEHITYSMALLIT

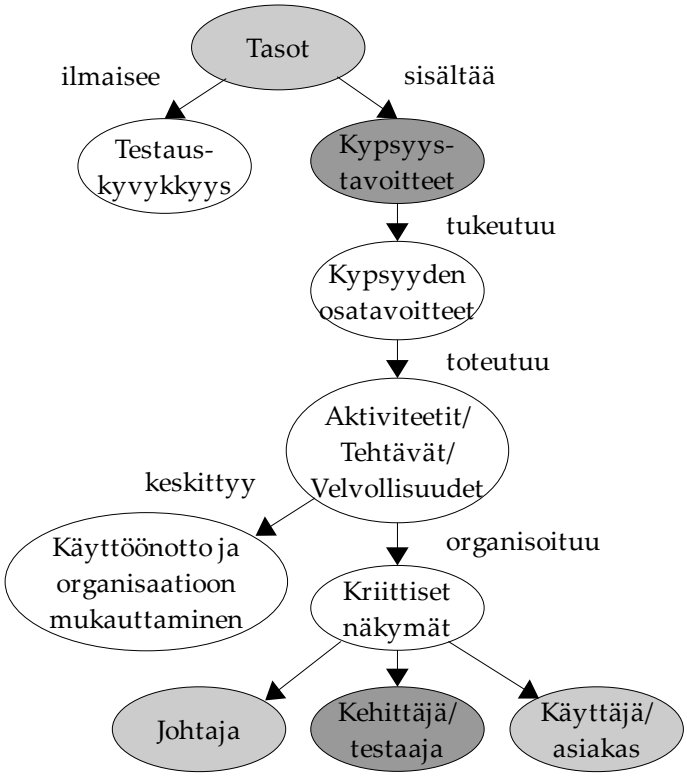
Tässä luvussa käsitellään neljää suosittua testausprosessin kehitysmallia niiden julkaisujärjestyksessä. Kehitysmalleja on olemassa huomattavasti suurempi määrä, mutta tutkimukseen on pyritty valitsemaan malleja, jotka ovat tunnettuja. Testausprosessin kehitysmallien käsittelyllä on kaksi tavoitetta. Ensimmäisenä tavoitteena on löytää kehitysmallien avulla tekijöitä, jotka testausprosessia kehitettäessä vaikuttavat organisaatioon, sen testausprosessia ympäröiviin prosesseihin, työkaluihin tai tuotteisiin. Tutkimuksessa pyritään painottamaan tekijöitä, joiden vaikutukset yltävät selkeästi henkilöstöön tai muihin resursseihin, sillä näitä organisaatiolla on käytettävissä rajallisesti. Toinen tavoite kehitysmallien käsittelyllä on löytää malleista mahdollisia ratkaisumalleja testausprosessin kehittämisen yleisiin kysymyksiin sekä muista malleista esiin nouseviin kysymyksiin. Havaittuja seikkoja kerätään yhteen ja niistä esitettyihin kysymyksiin vastataan ja esitetään ratkaisuja luvussa 5.

Kehitysmalleja käsitellään esittelemällä malli lyhyesti ja tuomalla sitten esiin mallista tai mallin avulla havaittuja seikkoja ja niistä esiin nousevia kysymyksiä. Luvun lopussa yhteenvedossa käydään läpi valittujen testausprosessin kehitysmallien selkeimmät keskinäiset erot ja samankaltaisuudet. Mallien kuvauksissa käytetään usein lähes suoria käännöksiä alkuperäisistä lähteistä.

4.1 TMM

TMM (Testing Maturity Model) on Illinois Institute of Technology:ssa kehitetty malli. Kehitystyössä on otettu vaikutteita CMM:sta, Gelperinin ja Hetzelin evoluutiomallista (Taulukko 1, sivu 13), alan testauskäytännöistä sekä Beizerin testaajan mielentilan mallista (Taulukko 2, sivu 20). TMM on kehitetty CMM:n

kumppaniksi ja sitä täydentäväksi malliksi, joten se lainaa CMM:sta myös ajatuksen kehitysprosessin jakamisesta helpommin hallittaviin osiin (Kuvio 6). (Burnstein ym. 1999)

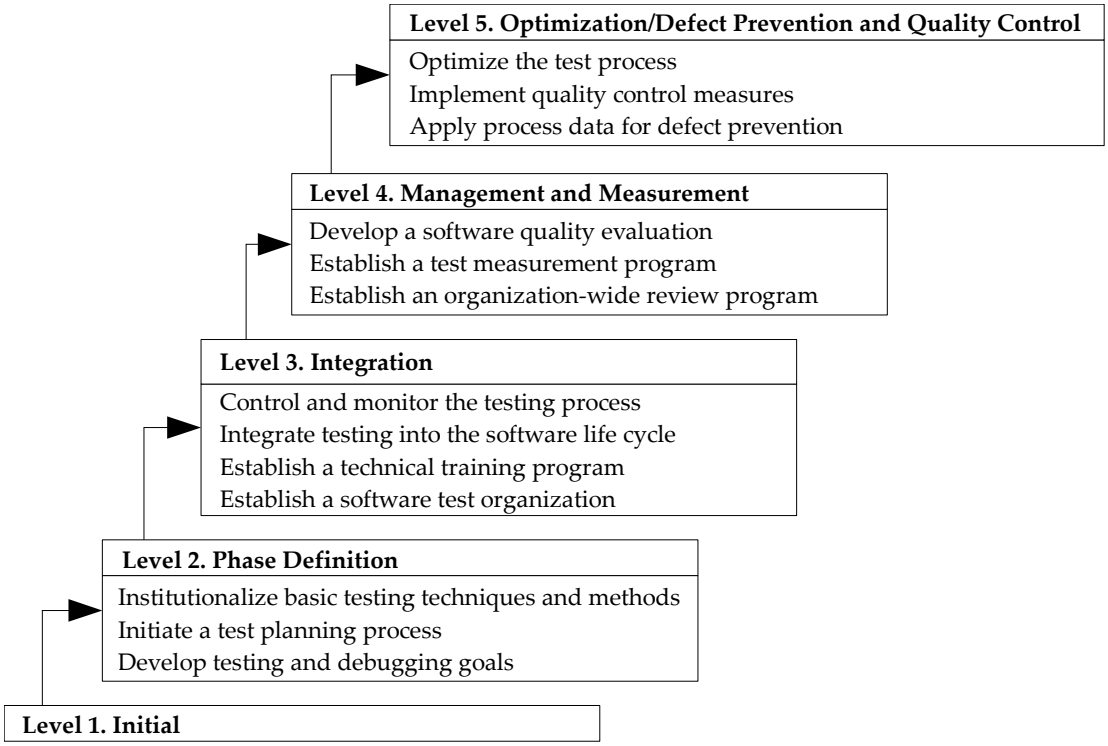


Kuvio 6: TMM-viitekehys (Burnstein ym. 1996a).

TMM:n rakenteen osia ovat kypsyystasot, jotka jakautuvat kypsyystavoitteisiin (MG, maturity goal) ja näiden osatavoitteisiin (MSG, maturity subgoal). Kypsyystasot ilmaisevat organisaation testausprosessin kehittyneisyyttä. Kypsyystasolle päästäkseen organisaation on täytettävä kaikki kyseisen tason ja sitä alempien tasojen kypsyystavoitteet. Kypsyystavoitteille on määritetty useita osatavoitteita, joiden täyttämistä vaaditaan ennen kuin kypsyystavoite voidaan katsoa täytetyksi. Aktiviteetit, tehtävät ja velvollisuudet (ATR, activies and tasks with responsibilities), ohjaavat saavuttamaan kypsyyden osatavoitteet. (Burnstein ym. 1996a) ATR:t on valitettavasti kuvattu ainoastaan

Teratip Suwannasartin julkaisemattomassa väitöskirjassa (Suwannasart 1996) vuodelta 1996 (Swinkels 2000). Tämä on TMM:n käyttöönoton kannalta erittäin ongelmallista.

TMM seuraa CMM:ia viidellä kypsyystasollaan (Kuvio 7), jotka on nimetty niiden edustamien testausprosessin kehityksen teemojen mukaan. TMM:n kypsyystasoa vastaa aina CMM:n kypsyystaso (van Veenendaal 2006). Käytännössä tämä tarkoittaa, että organisaatiolla tulee olla myös CMM käytössä, tai se joutuu tekemään ylimääräistä työtä ja muuntamaan mahdollisen toisen prosessikehitysmallin mukaisen tilanteen CMM:n mukaiseksi suunniteltaessa TMM:n mukaista kehitystä. Toinen mahdollinen ongelma TMM:n käytössä on, että se on melko karkealla tasolla toimiva kehitysmalli. Mallin mukaan ei siis välttämättä saa kovin tarkkaa kuvaa organisaation nykytilasta.



Kuvio 7: TMM:n kypsyystavoitteet tasoittain (Burnstein ym. 1999).

Seuraavaksi käydään lyhyesti läpi TMM:n kypsyystasot sekä organisaation kypsyystason arviointi. Tasoille määriteltyihin kypsyystavoitteisiin sisältyy myös osatavoitteita. Näihin keskitytään tarpeen mukaan myöhemmin, siltä osin kuin se koetaan tarpeelliseksi kysymysten taustojen selvittämisessä.

4.1.1 Taso 1

Tasoista ensimmäinen on lähtötaso (Initial), jolla organisaation katsotaan olevan, jos se ei täytä toisen tason kypsyystavoitteita. Tällä tasolla testausprosessia joko ei ole tai sitä ei noudateta. Testien kehittäminen on yleensä lähinnä ad hoc -tyyppistä ja testaus itsessään ajatellaan osaksi virheestystä. Testauksen tavoite on osoittaa, että ohjelmistossa ei ole pahoja vikoja. Testaukseen ei ole riittävästi resursseja, työkaluja eikä koulutettua henkilöstöä. (van Veenendaal 2006)

4.1.2 Taso 2

Toisella tasolla (Phase Definition) organisaatio alkaa ottaa huomioon testauksen teknisiä ja hallinnollisia piirteitä. Testausprosessi on määritetty ja selkeästi erotettu virheestyksestä. (Swinkels 2000) Kypsyystavoitteita tasolle 2 ovat:

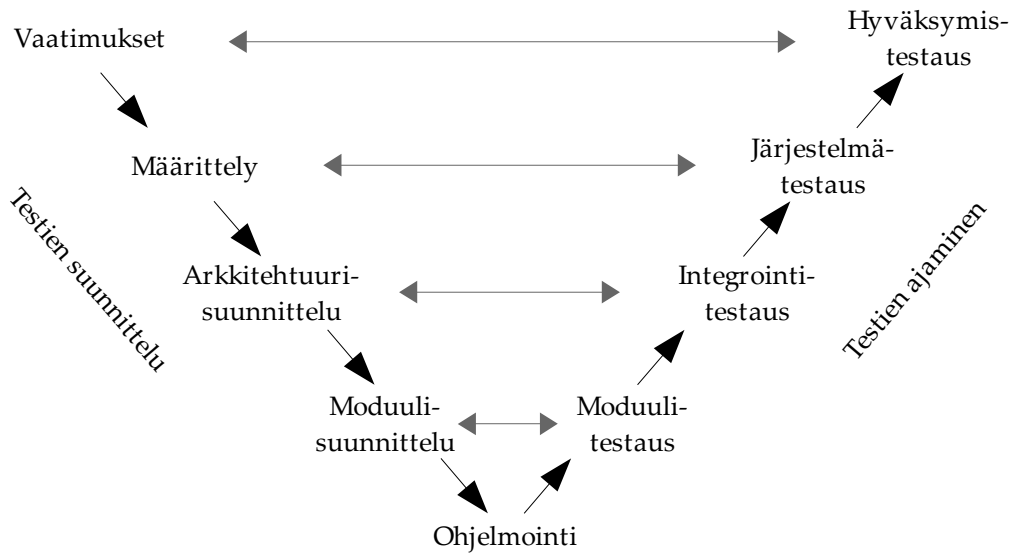
- Testaus- ja virheestystavoitteiden kehittäminen.
- Testien suunnitteluprosessin käynnistäminen.
- Perustason testaustekniikoiden ja -metodien vakiinnuttaminen.

4.1.3 Taso 3

Kolmas taso (Integration) pyrkii rakentamaan tuotteisiin laatua jo tuotekehitysprosessin aikaisissa vaiheissa. Testaus on integroitu tuotteen koko

elinkaareen joukkona hyvin määriteltyjä toimia, esimerkiksi V-mallin (Kuvio 8) mukaisesti. Organisaation johto tukee testausryhmän muodostamista ja kouluttamista. (Swinkels 2000) Kypsyystavoitteita tasolle 3 ovat:

- Testausryhmän luominen.
- Teknisen koulutusohjelman laatiminen.
- Testauksen integroiminen osaksi tuotteen elinkaarta.
- Testausprosessin valvonta ja hallitseminen.



Kuvio 8: V-malli.

4.1.4 Taso 4

Neljännellä tasolla (Management and Measurement) testaus on kauttaaltaan määritelty, tukevalle pohjalle rakennettu ja mitattavissa oleva prosessi. Katselmoinnit, tarkastukset ja läpikäynnit on liitetty tuotteen elinkaaren kaikkiin vaiheisiin ja katsotaan osaksi testausprosessia. Testaus ymmärretään

arviointina, johon sisältyy kaikki ohjelmistoon ja sen tuotantoon liittyviä tuotoksia tarkastavat toimet. (van Veenendaal 2006) Testauksen laajentuneeseen määritelmään sisältyy osa toiminnoista, joiden katsotaan yleensä kuuluvan verifioinnin ja validoinnin alle. Kypsyystavoitteita tasolle 4 ovat:

- Koko organisaation kattavan katselmointiohjelman perustaminen.
- Testauksen arviointiohjelman perustaminen.
- Ohjelmiston laadun arviointi.

4.1.5 Taso 5

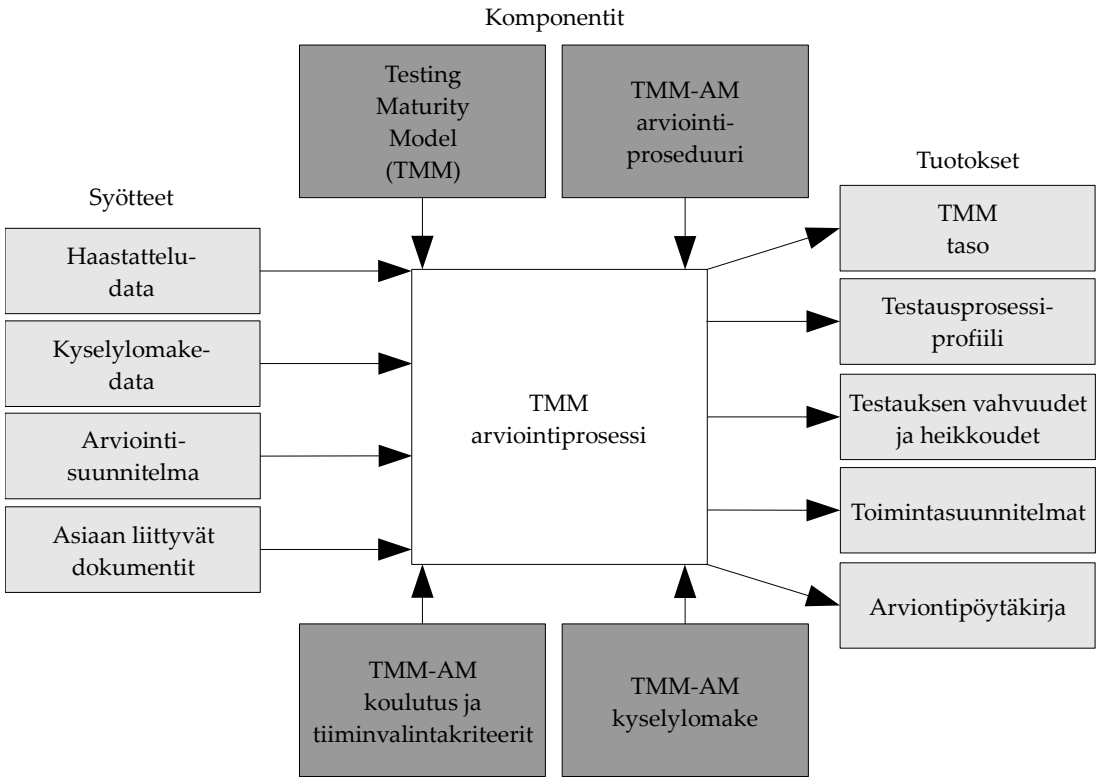
Viidennellä ja korkeimmalla tasolla (Optimization, Defect Prevention and Quality Control) testausprosessi pyrkii virheiden ennaltaehkäisyyn. Organisaatio tukee testausprosessin ja tuotteiden laadun jatkuvaa arviointia ja kehittämistä. (Swinkels 2000) Kypsyystavoitteita tasolle 5 ovat:

- Virheiden ennaltaehkäisy.
- Laadunhallinta.
- Testausprosessin optimointi.

4.1.6 Kypsyystason arviointi

TMM:n mukaista organisaation testausprosessin kypsyystason arviointia varten on TMM:lle tehty laajennos, TMM-AM (TMM Assessment Model, Homyen 1998). Arviointimalli perustuu arviointitiimin kouluttamiseen SPICE-suositusten (ks. ISO/IEC 15504-3:2004) mukaisesti, jäljitettävyysmatriisiin, kyselylomakkeeseen sekä tarkkoihin ohjeisiin arviointitoiminnasta.

Organisaation kypsyyttä arvioidaan arviointitiimin toimesta kyselylomakkeella ja muilla tavoin kerätyn informaation pohjalta. Jäljitettävyysmatriisia käytetään kerättyjen tietojen tallentamiseen, tarkastamiseen sekä tiedoissa havaittujen ristiriitojen ratkaisemiseen. (Burnstein ym. 1998) Arviointiprosessi ei selvitä pelkästään organisaation TMM:n mukaista nykytasoa, vaan lisäksi muun muassa sen testausprosessin heikkoudet ja vahvuudet (Kuvio 9), joiden perusteella organisaation on helpompi suunnitella prosessin kehittämistä (Homyen 1998).



Kuvio 9: TMM-AM-arviointiprosessin komponentit, syötteet ja tuotokset (Burnstein ym. 1998).

Arviointitiimi ja sen toiminta kuvataan TMM-AM:ssa varsin tarkasti. Mallin mukaisen tiimin tulisi olla neljästä kahdeksaan henkeä kattava. Myös sekä tiimin yhteiselle että jäsenkohtaiselle kokemukselle esitetään suosituksia.

Tiimin koulutus käydään läpi ja sille esitetään jopa aikatauluesimerkki. Tiimin tulisi valita neljästä viiteen organisaation toimintaa edustavaa vähintään kuusi kuukautta kestävää projektia seurattavaksi arviointia varten. Lisäksi muutama projekti tulisi valita varalle ja näitä käytetään jos jokin arviointiin valituista projekteista ei syystä tai toisesta pysty arviointiin osallistumaan. (Homyen 1998)

TMM-AM kyselylomake (Homyen 1998, 182) koostuu kahdeksasta osiosta, joista viisi keskittyy organisaatioon sekä vastaajaan, ja loput itse kyselyyn. Kaksi osiota viidestä edellä mainitusta keskittyvät vastaajan sekä organisaation taustoihin. Jäljelle jäävät kolme osaa liittyvät kypsyystavoitteisiin ja osatavoitteisiin, testaustyökaluihin sekä testauksen suuntauksiin. Kyselylomaketta ei kuitenkaan tule pitää TMM-AM:n ainoana tietolähteenä. Kyselylomakkeen lisänä tulisi käyttää haastatteluja ja oleellisten dokumenttien läpikäyntiä. (Homyen 1998 ja Swinkels 2000)

4.1.7 Kysymyksiä TMM:sta

Kun TMM:ia aletaan soveltamaan, ensimmäisenä tulee vastaan kysymys dokumentaatiosta. Näyttää siltä, että valtaosa tarkemmasta dokumentaatiosta on olemassa ainoastaan julkaisemattomassa väitöskirjassa (Suwannasart 1996). Jos kyseisen väitöskirjan hankkiminen ei onnistu, tai tarvittavaa tietoa ei löydy muuta kautta, voi kehitysmallin käyttäminen kaatua heti alkuunsa informaation puutteeseen. Tästä voidaan johtaa ensimmäinen testausprosessin kehittämiseen liittyvä kysymys:

- Onko kehittämistä ajatellen saatavilla riittävästi informaatiota?

Seuraava kysymys koskee kehitysmallin rakeisuutta. TMM on viisitasoinen malli, joista neljä tasoa koskee varsinaista kehitystä. On todettu, että

organisaatiolla voi kestää useita vuosia päästä CMM:n toiselle tasolle, jonka jälkeen etenemiseen kuluu yleensä aikaa noin kaksi vuotta tasoa kohti (Chrissis ym. 1993). TMM:lla on tiiviitä liitoksia CMM:iin, joten TMM:n seuraaminen vaatii usein tueksi CMM:n mukaisia toimia. Liitosten vuoksi TMM:n tasoilla etenemiseen voidaan olettaa kuluvan aikaa yhtä paljon kuin CMM:n mukaan etenemiseen. Vaikka testausprosessi TMM:n mukaan edetessä tietysti kehittyy vähitellen kun organisaatio pyrkii pääsemään seuraavalle tasolle, ei TMM:ssa ole mahdollisuutta ilmoittaa kypsyyttä kuin täysinä tasoina. Voidaan siis katsoa että testausprosessin kehittäminen TMM:n mukaan tarjoaa investoinnille organisaation ulkopuolelle näkyvää vastetta vasta usean vuoden prosessikehityksen jälkeen. Usean vuoden kehitystyö yhden tason edistymistä varten ei kuitenkaan todennäköisesti innosta varsinkaan nuorempia yrityksiä, tai yrityksiä, jotka pyrkivät kehittämään prosessejaan esimerkiksi yhteistyökumppaneiden tai asiakkaiden johdosta. Toisaalta on myös mahdollista, että organisaatio hakee pelkästään kehitystä, eikä sille ole käytännön merkitystä sillä, kuinka nopeasti kehitysmallin täysillä tasoilla edetään. Toinen kysymys on:

- Tarjoaako käyttöön valittu kehitysmalli haettuja etuja riittävän nopeasti?
- Pystyykö organisaatio suunnittelemaan prosessin kehitystä seuraavalle tasolle asti?
- Pystyykö organisaatio varaamaan riittävästi resursseja testausprosessin kypsyyden seuraavalle tasolle nostamista varten?

TMM:n karkeasta rakenteesta johtuen sitä noudattavalla organisaatiolla ei ole mahdollista valita testausprosessista organisaatiolle olennaisia osa-alueita kehitystä varten. Tämä ei ole ongelma, jos organisaatio hakee kokonaisvaltaista

kehitystä testausprosessin kaikilla osa-alueilla. On kuitenkin esitettävä kysymys:

- Kuinka hyvin valittu kehitysmalli pystyy mukautumaan organisaation kannalta olennaisiin kehitystarpeisiin?

TMM-AM vaatii koulutetun arviointitiimin, mikä voi varsinkin pienemmälle organisaatiolle olla turhan kova vaatimus. Arviointi järjestetään kyselylomakkeilla, haastatteluilla sekä olemassa olevan dokumentaation läpikäynnillä, joten arviointiprosessi saatetaan kokea organisaatiossa toimintaa häiritseväksi. Tarkka arviointiprosessi, joka käyttää hyväkseen jäljitettävyysmatriisia ja seuraa organisaation toimintaa vähintään puoli vuotta vaikuttaa varsin luotettavalta. Organisaation testausprosessin kypsyyden nostaminen keinotekoisesti arviointiprosessia varten ei ole helppoa. Tämä kannattaa huomioida, sillä organisaatiolle voi olla esimerkiksi taloudellista hyötyä saada arvioinnin mukaan mahdollisimman korkea kypsyystaso. Pitkällä tähtäimellä arviointiprosessin ”huijaaminen” kuitenkin lähes varmasti aiheuttaa enemmän haittoja kuin hyötyjä. Puoli vuotta kypsyystason arviointiin on joka tapauksessa varsin pitkä aika. Arviointimallista seuraa neljä kysymystä:

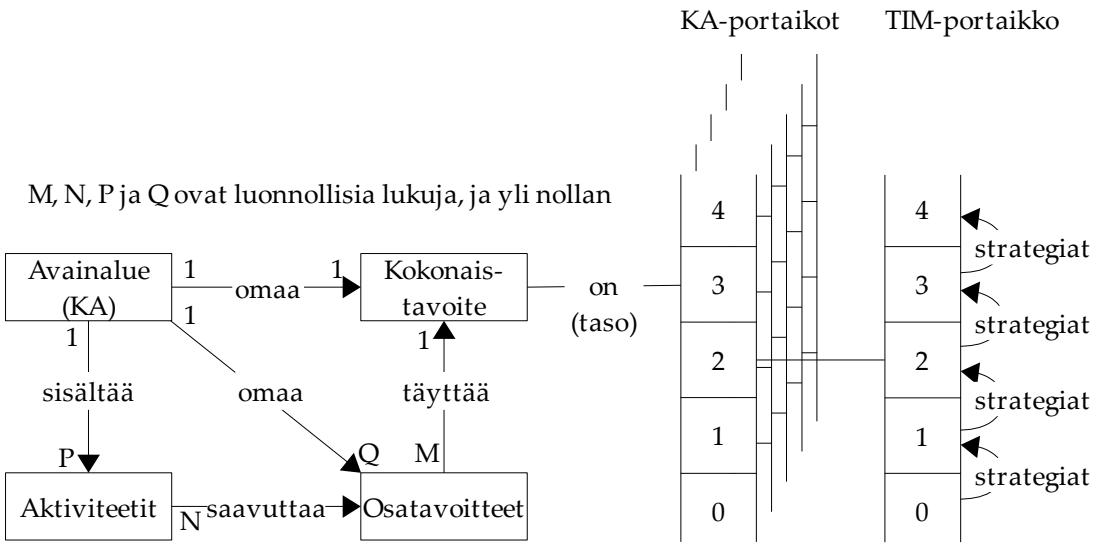
- Pystytäänkö kehitysmallin mukainen testausprosessin nykytilan arviointiprosessi viemään luotettavasti läpi organisaation resursseilla?
- Koetaanko arviointiprosessi organisaation toimintaa häiritseväksi?
- Toimiiko arviointiprosessi riittävän nopeasti?
- Voidaanko arviointiprosessin tuloksiin varmasti luottaa?

4.2 TIM

TIM on Ruotsissa Linkopingin yliopistolla ja Saab Combitechillä kehitetty testausprosessin kehitysmalli. TIM:n kehitykseen on otettu vaikutteita CMM:sta sekä Gelperinin TSM:sta (ks. Gelperin 1996). TIM ei kuitenkaan ole riippuvainen mistään tietystä tuotantoprosessin kehitysmallista. (Ericson ym. 1997)

TIM koostuu neljä kypsyystasoa sisältävästä kehitysportaikosta, johon lähtötasoa ei lasketa mukaan. Tasojen lisäksi TIM:lla on toinen ulottuvuus: avainalueet (KA, key area), joita on viisi. Jokainen avainalue on erillinen kehitysportaikko, joka keskittyy tiettyyn testauksen kannalta tärkeään alueeseen. Mallissa kypsyystasoilla on kuitenkin yhtenäinen teema avainalueesta riippumatta, ja tasojen saavuttamiseksi kuvataan tietyt avainalueriippumattomat strategiat. TIM:lla kypsyystaso on itsessään tavoite, jolla on osatavoitteita. Kypsyystaso voidaan katsoa saavutetuksi, kun sen osatavoitteet on täytetty. Osatavoitteiden täyttämiseen malli tarjoaa apua aktiviteettien muodossa. (Ericson ym. 1997) TIM:n korkean tason rakennetta kuvaa kuvio 10.

TIM suosittelee tasapainoista kehitystä kaikki avainalueet huomioiden siten, että väli alimmalla ja korkeimmalla tasolla olevin avainalueiden välillä ei ole liian leveä. Mallissa alemmat kypsyystasot tukevat ylempiä, joten kokonaisten tasojen yli hyppääminen on vaikeata, vaikka tätä ei suoranaisesti kielletäkään. Organisaatiot voivat kuitenkin hyötyä ottamalla prosessiinsa mukaan ominaisuuksia mallin korkeammilta tasoilta. (Ericson ym. 1997) Seuraavaksi käydään läpi TIM:n kypsyystasojen yleiset tavoitteet, sen avainalueet ja niiden kypsyystasojen tarkemmat tavoitteet lyhyesti, minkä jälkeen kuvataan TIM:n kypsyystason arviointi.



Kuvio 10: TIM:n rakenne (Ericson ym. 1997, 3).

4.2.1 Kypsyystasot

TIM määrittelee kypsyystasojen yleiset tavoitteet avainalueista riippumattomina. Näin avainalueiden tarkempia tavoitteita sisältävillä kypsyystasoilla on kuitenkin yhtenäinen kantava ajatus. TIM:n rakenne on siis varsin lähellä TMM:ia, mutta siinä TMM:n yksi yhtenäinen kypsyystasoportaikko on jaettu selkeästi viiteen osaan, tuoden etenemiseen joustavuutta. Tässä kuvataan TIM:n kypsyystasojen tavoitteet ja strategiat niiden saavuttamiseksi. Kypsyystasojen kuvaukset perustuvat lähteeseen Ericson ym. 1997.

4.2.1.1 Taso 0

Organisaatio, jolla testausprosessin jokin avainalue ei täytä TIM:n ensimmäisen tason vaatimuksia, on kyseisen avainalueen osalta tasolla 0. Tätä tasoa ei kuvata, sillä organisaation testausprosessi ei vielä tässä vaiheessa noudata

TIM:ia. Taso vastaa TMM:n tasoa 1, jolla testausprosessin katsotaan olevan vielä täysin epäkypsä.

4.2.1.2 Taso 1

Ensimmäinen taso (Baselining) keskittyy perustason testausprosessin luomiseen. Organisaatiolla täytyy olla ryhmä, joka omistautuu testaukselle. Näille henkilöille on määritetty roolit sekä vastuualueet ja heidän työtapansa on dokumentoitu selkeästi ja standardien mukaan. Perustasolle kuuluu myös vaatimusten katselmoinnin sisällyttäminen testaukseen. Strategiat tason 1 tavoitteiden saavuttamiseksi ovat:

- Standardien, menetelmien ja toimintatapojen dokumentointi.
- Tuotteiden ongelmien analysointi ja luokittelu.

4.2.1.3 Taso 2

Toinen taso (Cost-effectiveness) keskittyy testausprosessin kustannustehokkuuden nostamiseen virheiden tehokkaampaa havaitsemista edistämällä. Tässä virheiden aikainen tunnistaminen on erittäin tärkeää, sillä valtaosa virheistä syntyy kehitysprosessin aikaisissa vaiheissa. Näiden virheiden korjauskustannukset kohoavat huimasti (Kuvio 3, sivu 18), mitä pidemmälle ne kehityksessä pääsevät ennen kuin ne havaitaan. Strategiat tason 2 tavoitteiden saavuttamiseksi :

- Aikainen virheiden tunnistaminen.
- Testaustyön tietokoneistettu avustaminen.
- Koulutus.

- Uudelleenkäyttö.

4.2.1.4 Taso 3

Kolmas taso (Risk-lowering) keskittyy riskien hallintaan. Tällä tasolla testausprosessin tulisi olla mukana tuotteen kehityksessä heti alusta alkaen, jotta se saa tuotteesta riittävästi tietoa pystyäkseen tunnistamaan mahdollisia ongelma-alueita hyvissä ajoin. Testausprosessi pystyy jossain määrin varmistamaan tuotteen laatua. Tällä tasolla metriikat otetaan käyttöön testausprosessin yhteydessä. Niillä pystytään seuraamaan testaustavoitteiden täyttymistä ja testaustoimien hyötysuhdetta sekä tunnistamaan virhealttiita testikohteita. Strategiat tason 3 tavoitteiden saavuttamiseksi ovat:

- Aikainen osallistuminen.
- Kustannusten perustelu.
- Tuotteen ja prosessin ongelmien analysointi.
- Tuotteen, prosessin ja resurssien mittaaminen.
- Riskien analysointi ja hallinta.
- Kaikkien projektiin osallisena olevien tahojen kanssa kommunikointi.

4.2.1.5 Taso 4

Neljäs taso (Optimising) tavoittelee testausprosessin jatkuvaa optimointia ja kehittämistä. Testausprosessi pystyy varmistamaan tuotteelle siltä vaaditun laadun. Testausta tehdään jokaisessa kehitysprosessin siirtymävaiheessa yhteistyössä vaihetuotokset tuottaneiden tahojen sekä ne vastaanottavien tahojen kanssa. Strategiat tason 4 tavoitteiden saavuttamiseksi ovat:

- Tietämystä ja ymmärrystä kokeiden ja mallinnuksen kautta.
- Jatkuva kehittyminen.
- Perimmäisten syiden analysointi.
- Testauksen tasapainottaminen tuotteen tarpeiden ja organisaation laatupolitiikan mukaan.
- Projektin kaikkien tahojen yhteistyö kehityksen kaikissa vaiheissa.

4.2.2 Avainalueet

Mallin mukaan edettäessä testausprosessin kehitys voi kulkea eri avainalueilla eri kehitystasoilla, mikä tekee TIM:sta jonkin verran TMM:ia joustavamman mallin. Se muun muassa helpottaa organisaatiota keskittymään sille olennaisten asioiden kehittämiseen. Lisäksi hienojakoisempi malli tarkoittaa sitä, että organisaatio näkee mallin mukaan edetessään testausprosessin tilanteen tarkemmin saavutettujen tasojen kautta. Myös organisaation sidosryhmät saavat näin tarkempaa tietoa testausprosessin kypsyydestä. Tässä kuvataan TIM:n avainalueet ja niiden saavuttamiseksi vaaditut aktiviteetit lyhyesti. Avainalueiden kuvaukset perustuvat lähteeseen Ericson ym. 1997.

4.2.2.1 Organisointi

Tämä avainalue (Organisation), keskittyy testausprosessin organisointiin suhteessa sitä ympäröiviin prosesseihin. Se pyrkii irrottamaan testauksen tuotekehityksestä, mutta siten, että testaus ja tuotekehitys ovat kuitenkin tiiviissä yhteistyössä. Avainalueen eri tasojen saavuttamiseksi tarvittavat aktiviteetit on kuvattu taulukossa 3.

Taulukko 3: Organisointi-avainalueen aktiviteetit (Ericson ym. 1997).

Taso	Aktiviteetit
Taso 4	Testaajat monialaisten tiimien osa. Jatkuva organisationaalinen kehitys. Prosessien kehittämistä varten on olemassa ryhmiä.
Taso 3	Resurssien varaaminen ja tehtävien suunnittelu suoritetaan kaikilla tasoilla testit huomioon ottaen. Testaustoiminnalle pyritään saamaan hallinnon kaikkien tasojen tuki. Testaajat osallistuvat kehityksen kaikkiin vaiheisiin ja kehittäjät osallistuvat testaukseen. Henkilöstö kiertää testauksen ja kehitystyön välillä.
Taso 2	Testaustoiminta on sijoitettu fyysisesti yhteen paikkaan. Avainhenkilöt työskentelevät täysipäiväisesti yhden projektin parissa ja resursseja ei jaeta. Testaustoiminta on itsenäistetty. Tiimin rakenne perustuu resurssien inventointiin ja projektin tarpeisiin. Vastuut on selkeästi määritelty, dokumentoitu ja ymmärretty. Testausryhmälle on omistettu työalue, asianmukaiset työkalut ja tarvikkeet. Koulutusta organisaation tarpeiden mukaan.
Taso 1	Testaustoiminta on organisoitu dokumentoidun standardin mukaisesti. Testipäällikkö ja ydintestitiimi ovat olemassa.

4.2.2.2 Suunnittelu ja seuranta

Suunnittelu on tärkeä osa testausta. Ilman hyvin määriteltyjä tavoitteita ja strategioita näiden saavuttamiseksi, saatetaan resursseja heittää hukkaan. Seurantaa tarvitaan valvomaan toiminnan etenemistä kohti asetettuja tavoitteita. Suunnittelussa suositellaan käytettävän jotain dokumentointistandardia, kuten IEEE 829 (ks. IEEE 829-1998). Avainalueen eri tasojen saavuttamiseksi tarvittavat aktiviteetit on kuvattu taulukossa 4.

Taulukko 4: Suunnittelu ja seuranta -avainalueen aktiviteetit (Ericson ym. 1997).

Taso	Aktiviteetit
Taso 4	Kustannuksista, resursseista, riskeistä ym. luodaan malleja. Suunnittelu- ja seurantatoimia kehitetään jatkuvasti metriikoiden tulosten ja kokemusten perusteella. Jälkiselvittelyt toteutetaan ja niiden tulokset jaetaan asiaankuuluvasti.
Taso 3	Riskianalyysi suoritetaan ja se vaikuttaa suunnitteluun sekä resurssien järjestelyyn. Suunnitelmat, mukaan lukien testauksen tavoitteet, hyväksytetään tahoilla, joihin ne vaikuttavat. Testauksen kokonaistavoitteiden toteutumista seurataan. Suunnittelun tekee kokenut testaaja tai suunnittelija. Suunnitelmat muuttuvat todellisten olosuhteiden johdosta.
Taso 2	Suunnittelu ja seuranta on tietokoneavusteista. Käytetään geneerisiä suunnitelmia. Testausmenetelmät ja testauksen tasot valitaan testattavien kohteiden ja testauksen tavoitteiden pohjalta. Etenemistä mitataan ja verrataan suunnitelmiin. Suunnittelun tekee koulutettu testaaja tai suunnittelija. Suunnittelu tehdään evolutionäärisellä tavalla. Resurssien ennustusta tehdään dokumentoituja käytäntöjä noudattaen.
Taso 1	Yksinkertaista suunnittelua tapahtuu. Aloitus- ja lopetuskriteerit määritellään kaikille testauksen tasoille. Testien tulokset dokumentoidaan sekä käsitellään ja jaellaan asianmukaisesti. Suunnitelmat luodaan standardeja noudattaen, poikkeamat kirjataan ja niille löytyy syyt. Suunnittelu tehdään dokumentoitujen käytäntöjen mukaisesti. Suunnitelmat muuttuvat vaatimusten mukana. Testaussuunnitelman muutokset tehdään dokumentoidun käytännön mukaisesti.

4.2.2.3 Testitapaukset

Dynaaminen testaus tulisi aina suorittaa etukäteen määriteltyjä testitapauksia hyödyntäen. Testitapausten suunnittelu ja valinta on erittäin tärkeää, sillä testitapausten laatu määrää suoraan testauksen laadun ja vaikuttaa sitä kautta tuotteen laatuun. Tämä avainalue keskittyy testitapausten suunnitteluun ja dokumentointiin. Avainalueen eri tasojen saavuttamiseksi tarvittavat aktiviteetit on kuvattu taulukossa 5.

Taulukko 5: Testitapaukset -avainalueen aktiviteetit (Ericson ym. 1997).

Tasot	Aktiviteetit
Taso 4	Testitapausten suunnittelutekniikoita mitataan, arvioidaan ja kehitetään.
Taso 3	Testitapaukset rankataan ja valitaan kriittisyysasteen mukaan. Testitapaukset suunnitellaan vastauksena riskialueille.
Taso 2	Testitapaukset suunnitellaan käyttäen dokumentoituja tekniikoita. Testattavuusnäkökannat vaikuttavat vaatimuksiin ja suunnitteluun. Testitapauksia kirjataan ja uudelleenkäytetään. Testitapaukset organisoidaan suhteessa testaustasoihin, vaatimuksiin, testikohteisiin, tavoitteisiin, jne.
Taso 1	Testitapauksia tarkistetaan kun vaatimukset muuttuvat. Testitapaukset pohjautuvat vaatimuksiin. Testitapaukset suunnitellaan ja dokumentoidaan ennen suoritusta dokumentoidun käytännön mukaisesti. Testitapaukset suoritetaan dokumentoitujen ohjeiden mukaisesti.

4.2.2.4 Testausohjelmisto

Testausohjelmistoon kuuluu testausmenetelmät, tukiohjelmisto, testien ajoissa käytettävät tietojoukot sekä testauksen tukena toimiva dokumentaatio.

Avainalue sisältää testausohjelmiston kokoonpanonhallinnan sekä testausohjelmiston ja testauksen työkalujen käytön. Työkalut auttavat toistuvien tehtävien suorituksessa, kuten samojen testitapausten useaan kertaan suorittamisessa. Työkalut voivat siis tehostaa testausprosessia ja tehdä siitä mielenkiintoisempaa lisäämällä luovuutta vaativan työn osuutta suhteessa ei-luovaan työhön.

Konfiguraation hallinta (CM, configuration management) on tärkeä osa ohjelmistotuotanto-organisaatiota. Konfiguraation hallinta varmistaa, että ohjelmistoon tehtäville muutoksille on aina olemassa tunnettu syy, ja että muutoksesta seuraavat muihin ohjelmiston osiin tarvittavat muutokset tulevat myös toteutetuiksi. Konfiguraation hallinta varmistaa myös, että ohjelmistosta saadaan tarvittaessa rakennettua mikä tahansa tarvittava versio. Avainalueen eri tasojen saavuttamiseksi tarvittavat aktiviteetit on kuvattu taulukossa 6.

Taulukko 6: Testiohjelmistot -avainalueen aktiviteetit (Ericson ym. 1997).

Tasot	**Aktiviteetit**
Taso 4	Tietokoneavusteinen tulosten tarkistus ja arviointi. Tietokoneavusteinen testausohjelmiston suunnittelu. Testausympäristö on integroitu. Kokeiluja työkaluilla.
Taso 3	Tietokoneavusteinen riskien analysointi. Testausohjelmisto ja ohjelmisto on tehokkaan CM:n hallinnassa. Tietokoneavusteinen regressiotestaus. Käytetään vain kypsää teknologiaa.

Tasot	Aktiviteetit
Taso 2	Tietokoneavusteinen ongelmien seuraaminen, koodin jäljitys ja testien suoritus. Tietokoneavusteinen testitapausten tulosten kerääminen, kirjaaminen, käsittely ja raportointi. Käytössä on kattavuusanalyysityökaluja. Laitteisto- ja ohjelmistojärjestelmiä simuloidaan. (Nykyisin esimerkiksi emulaattorit ja virtuaalikoneet sopivat tähän erinomaisesti.) Tietokantoja käytetään testausohjelmiston konfiguraation hallintaan. Testityökaluja arvioidaan ennen osto- ja käyttöönottopäätöstä. Staattisen analyysin työkaluja käytetään esimerkiksi koodin katselmointien valmistelussa.
Taso 1	Ongelmien raportointi on tietokoneistettu. Tiedostojärjestelmiä käytetään "konfiguraation hallintaan".

4.2.2.5 Katselmoinnit

Avainalueena katselmoinnit kattavat kaikki tekniikat, jotka liittyvät ohjelmiston dokumenttien lukemiseen tai visuaaliseen tarkastelemiseen. Joitain esimerkkejä katselmointitekniikoista ovat tarkastukset, tekniset katselmoinnit, läpikäynnit, vertaiskatselmoinnit ja ohjelmakoodin lukeminen. Katselmoinnit ovat ajan kuluessa osoittautuneet tehokkaiksi, mutta niiden taloudelliset näkökohdat ovat vielä epävarmoja. Katselmoinnit ovat kuitenkin tärkeitä, sillä niillä voidaan testata vaihetuotoksia, joita ei voida suorittaa. Avainalueen eri tasojen saavuttamiseksi tarvittavat aktiviteetit on kuvattu taulukossa 7.

Taulukko 7: Katselmoinnit-avainalueen aktiviteetit (Ericson ym. 1997).

Tasot	Aktiviteetit
Taso 4	Katselmointitekniikoita kehitetään jatkuvasti. Tiimin ja tekniikan valinta perustuu faktoihin.

Tasot	Aktiviteetit
Taso 3	Testiohjelmisto ja testitapaukset tarkastetaan. Katselmointitekniikat arvioidaan. Katselmointiprosesseja, tuotoksia ja resursseja mitataan.
Taso 2	Kaikki katselmointihenkilöstö koulutetaan katselmointitekniikoissa. Tarkistuslistoja, skenaarioita, jne. räätälöidään ja käytetään. Suunnittelu- ja koodidokumentit tarkastetaan. Organisaatiolla on joitakin katselmointitekniikoita, joista voidaan valita.
Taso 1	Vaatimukset katselmoidaan. Katselmointijohtajat ja johtoporras koulutetaan katselmointitekniikoissa. Ainakin ensimmäistä projektia varten ostetaan koulutusta ja tukea. Käytetään standardoituja ja dokumentoituja katselmointitekniikoita.

4.2.3 Kypsyystason arviointi

TIM:iin sisältyy testausprosessin kypsyystason arviointimenettely, joka on mahdollista toteuttaa kahdella tavalla. Laajemmalla tavalla toteutettuna menettelyn mukaan tehdään ensin kattava arviointi, jonka jälkeen luodaan suunnitelma testausprosessin kehittämiseksi. Kevyempi tapa koostuu pelkästään lyhyestä, päivän kestävästä kehitysmallin ja kyselylomakkeen läpikäynnistä ilman kehityssuunnitelmaa. Kevyempi arviointimenettely sopii käytettäväksi organisaatiossa, joka ei halua käydä läpi formaalia arviointia tai esimerkiksi pelkää arviointimenettelyn vievän liikaa resursseja tai häiritsevän toimintaa. Haittapuoli kevyessä menettelyssä on, että sitä noudatettaessa ei voida luoda kunnollista kehityssuunnitelmaa. (Ericson ym. 1997)

Normaali arviointimenettely on kattava, mutta ei lähesty TMM-AM -arvioinnin laajuutta. Se ei toisaalta ole lähellekään yhtä raskas kuin TMM-AM. Menettelyn mukaan organisaatiosta valitaan koko organisaatiota kattavasti edustava

joukko, jonka jälkeen näille edustajille selvitetään testausprosessin kehitysmalli sekä arviointimalli. TIM:n kyselylomaketta käytetään pohjana haastatteluissa, joissa kyselylomakkeen avustuksella päätetään mihin suuntaan keskustelua ohjataan. Kyse on todellakin keskustelusta, sillä yksinkertaisia kyllä/ei vastauksia ei mallin mukaan hyväksytä. Keskusteluilla pyritään auttamaan paitsi haastattelijaa, myös haastateltavaa ymmärtämään organisaation testausprosessin mahdolliset ongelma-alueet. Tällä tavoin haastateltavat saadaan myös sitoutettua testausprosessin kehityssuunnitelman luomiseen. Haastattelujen tulokset arvioidaan ja esitetään eräänlaisena kypsyysprofiilina, jossa organisaation testausprosessi pisteytetään avainaluekohtaisesti. Kypsyysprofiilia myös verrataan organisaation kokemaan kypsyyden tasoon. (Ericson ym. 1997)

Kypsyystason selvittämisen jälkeen TIM:n menettelyssä luodaan testausprosessille kehityssuunnitelma, jonka pohjana käytetään organisaatiolle luotua kypsyysprofiilia. Kypsyysprofiilin perusteella TIM:sta voidaan saada ehdotuksia strategioista, joilla prosessia voidaan kehittää. Nämä strategiat arvioidaan ja järjestetään sen mukaan, mitä organisaatio kokee tärkeäksi ja mihin muun muassa resurssien ja kykyjen koetaan riittävän. (Ericson ym. 1997) Kypsyystason arviointi TIM:n menetelmän mukaisesti on varsin joustava ja henkilöstöä hyvin mukaan innostava toimenpide. Arvioinnin toteuttamisen valitettavasti tekee varsin vaikeaksi se, että sen pohjana käytettäväksi tarkoitettua kyselylomaketta ei ole julkisesti saatavilla (ks. myös Swinkels 2000). Lomakkeen tosin kerrotaan rakentuvan avainalueiden aktiviteettien ympärille (Ericson ym. 1997). Tämän tiedon pohjalta on varmasti mahdollista soveltaa arviointimenetelmää.

4.2.4 Kysymyksiä TIM:sta

TIM:n kuvauksesta käy ilmi, että se kattaa hiukan laajemman alueen testaukseen liittyvää toimintaa ja toiminnan ympäristöä kuin TMM. Esimerkiksi testaustiimin toimintaympäristöön, sijoitukseen ja palkkausmalliin otetaan kantaa. Myös testauksen välineet ovat paremmin esillä TIM:ssa kuin TMM:ssa. Organisaatioiden tarpeet voivat kuitenkin poiketa toisistaan varsin radikaalisti muun muassa erilaisten palvelumallien ja teknologioiden käytön vuoksi. Tämän takia, ja koska mallien huomioimat kehityskohteet poikkeavat jonkin verran, voidaan esittää kaksi varsin samankaltaista kysymystä, joista kuitenkin kumpikin on tärkeä:

- Ottaako kehitysmalli huomioon organisaation kannalta oleelliset kehityskohteet?
- Ovatko kehitysmallin huomioon ottamat kehityskohteet organisaation kannalta oleellisia?

Kehitysmalli pyrkii ottamaan kantaa myös testattavamman ohjelmakoodin tuottamiseen. Tähän pyritään kierrättämällä kehitys- ja testaustoiminnan parissa työskenteleviä henkilöitä toistensa töissä ja näin tutustuttamaan myös ohjelmiston kehittäjät testauksen tarpeisiin. Tämä voi olla sekä hyvä että huono puoli. Asetetuin väliajoin vaihtuvat tehtävät voivat ylläpitää mielenkiintoa sekä lisätä dokumentaation ja koodin laatua, sillä kehittäjän vaihtuessa on hänen päästävä työhön hyvin kiinni. Toisaalta on mahdollista, että jos kaikki ovat osallistuneet sekä kehitystyöhön että testaukseen, saattaa johdon mielestä olla hyväksyttävissä vaihtaa työtehtäviä erittäin lyhyellä varoitusajalla tarpeiden mukaan. Tätä malleissa pyritään estämään erottamalla testaustoiminta ohjelmistokehityksestä, mutta houkutus voi olla kiiretilanteessa suuri, jos

tiedetään, että kaikki voivat toimia kehittäjinä. Testaus saattaa siis valitettavasti myös kärsiä työtehtävien kiertämisestä. Seuraa kysymyksiä:

- Kuinka pystytään takaamaan riittävä välimatka testaus- ja kehitystoimen välillä, jotta kehityksen paineet eivät pääse vaikuttamaan testaukseen?
- Mikä on sopiva aikaväli tehtävien vaihtumiseen töitä kierrätettäessä?
- Kuinka saadaan motivoitua kehittäjät ja testaajat toimimaan toistensa rooleissa?
- Mitä täytyy tehtävien kierrättämisessä ottaa huomioon, jotta se varmasti onnistuisi?

Yllä mainittu tavoite testattavammasta ohjelmakoodista on erittäin hyvä. Tämä voi kuitenkin aiheuttaa jonkin verran mietittävää organisaatiossa, joka kehittää esimerkiksi SaaS-mallin (Software as a Service) mukaan lisensoitavaa palvelua Tällaisessa palvelussa ohjelmisto on jatkuvan kehityksen alla ja asiakkaat käyttävät ohjelmistoa tuottajaorganisaation palvelimilta Internetin yli. Kyseisenlainen palvelumalli aiheuttaa ongelmia varsinkin silloin, jos on toivottavaa muuttaa kehitettävän ohjelmiston arkkitehtuuria testattavammaksi. Ohjelmiston kehitykselle on todennäköisesti tällaisessa tilanteessa paineita tuottaa jatkuvasti uutta, eikä resursseja välttämättä ole kertaluontoista muutosta varten. Arkkitehtuuria muutettaessa on myös pystyttävä varmistamaan, että muutetussa tuotteessa on samat toiminnot kuin aiemmassa ja että nämä toimivat samalla tavoin. Tähän liittyen esitetään seuraavat kysymykset:

- Milloin ja miksi arkkitehtuurin muuttamista helpommin testattavaksi kannattaa harkita?

- Mitä arkkitehtuurimuutoksessa tulee ottaa huomioon, jotta muutetun tuotteen toiminnallisuuden voidaan varmistaa vastaavan tilannetta ennen muutoksia?

Malleilla on taipumus korostaa testauksen apuna käytettävien työkalujen osuutta varsin nopeasti kypsyystasoja noustaessa. Tämä on ymmärrettävää, sillä sopivien työkalujen käytöllä voidaan saavuttaa hyötyjä sekä kustannusten, että testauksen laadun osalta. Työkalujen hankintaan ja valintaan TIM ottaa kevyesti kantaa. Työkaluihin liittyen on kuitenkin esitettävä kysymyksiä:

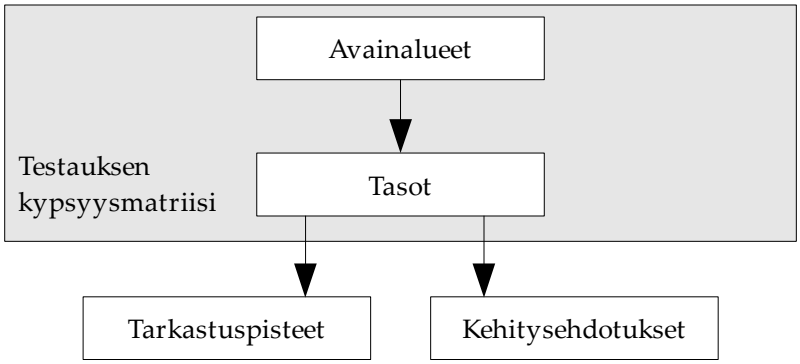
- Millaisia työkaluja organisaatiolla tulisi testaustoiminnan tueksi olla käytettävissä?
- Missä tapauksissa organisaation on hyödyllisempää kehittää itse työkalut testausta tukemaan?
- Mitä tulee ottaa huomioon testauksen työkaluja hankittaessa tai kehitettäessä?
- Mitä tulee ottaa huomioon testauksen työkalujen käyttöönotossa?

4.3 TPI

TPI (Test Process Improvement) on Alankomaista peräisin olevan Sogeti-yrityksen luoma testausprosessin kehitysmalli, joka tukeutuu saman yrityksen TMap-malliin. TMap (Test Management approach) on organisaation tarpeisiin räätälöitävä testauksen prosessimalli. TPI:lla ei ole kytköksiä tuotantoprosessin kehitysmalleihin, joten sitä voi tarpeen mukaan käyttää esimerkiksi CMMI:n kanssa.

Mallin rakenne (Kuvio 11) koostuu 20 avainalueesta, joilla jokaisella on yhdestä neljään kypsyystasoa (A – D). Avainalueet voidaan esimerkiksi raportointia varten ryhmitellä viiteen ryhmään TMap:n mukaisesti. Kypsyystasot koostuvat vaatimuksista, joita TPI:ssa kutsutaan tarkastuspisteiksi (Checkpoint). Tarkastuspisteet ovat ikään kuin kysymyksiä, joihin on pystyttävä vastaamaan myöntävästi, jotta taso voidaan katsoa saavutetuksi. Tarkastuspisteiden läpäisemisen avuksi TPI tarjoaa varsin kattavasti kehitysehdotuksia ja esimerkkejä. Jos organisaatio ei läpäise tason A tarkastuspisteitä, sen katsotaan olevan kyseisen avainalueen osalta alimmalla, määrittelemättömällä tasolla. Avainalueiden kypsyystasoilla on usein pieni määrä hyvin perusteltuja riippuvuuksia muiden avainalueiden tasoihin. Avainalueen sisällä kypsyystasot tukeutuvat alempiin tasoihin, joten esimerkiksi tason B saavuttaakseen, organisaation täytyy täyttää myös tason A vaatimukset. (Koomen, Pol 1999)

Eri avainalueiden väleillä kypsyystasojen tasapainoisen etenemisen ei katsota tapahtuvan samassa tahdissa. Kypsyystasot toisin sanoen kulkevat limittäin, mistä seuraa 13 kypsyysluokkaa. Nämä ottavat huomioon kaikkien avainalueiden kypsyystasot. Kypsyysluokat voidaan jakaa kolmeen koko prosessin kattavaan tasoon, joista kerrotaan kypsyystason arvioinnin yhteydessä luvussa 4.3.2. TPI tukee valikoivaa testausprosessin kehittämistä hyvin ja rohkaiseekin organisaatiota selvittämään sille tärkeät kehityskohteet. (Koomen, Pol 1998)



Kuvio 11: TPI:n rakenne (Koomen, Pol 1998, 3).

TPI on TMM:ia ja TIM:ia huomattavasti hienojakoisempi malli, mikä helpottaa prosessin kehittämistä. Se saattaa myös vaikeuttaa testausprosessin kypsyyden ilmaisemista ulkoisille sidosryhmille. Kahtakymmentä avainaluetta ja limittäin kulkevia kypsyystasoja voi olla asiaan perehtymättömän vaikea käsitellä. Kypsyyden ilmaisemista helpottaa avainalueiden jako TMap:n mukaisesti sekä kypsyysluokkien jakaminen kolmeen koko prosessia kuvaavaan tasoon (ks. luku 4.3.2).

Tässä kohdassa kuvataan ensin TPI:n avainalueet sekä niiden kypsyystasot. Kypsyystasojen tarkastuspisteisiin perehdytään vain niiltä osin, kuin tälle jonkin tarkastuspisteen osalta erikseen ilmenee tarvetta. Tässä tapauksessa asia otetaan esille luvussa 4.3.3, kun käsitellään TPI-mallista esille nousevia kysymyksiä. Kypsyystasojen läpikäynnin päätteeksi esitetään testauksen kypsyysmatriisi, joka kuvaa TPI:n kypsyysluokat, sekä esitetään kuinka matriisia käytetään testausprosessin suunnittelun apuvälineenä.

4.3.1 Avainalueet

TPI:n avainalueet ovat lähes täysin itsenäisiä ja niissä onkin vain satunnaisesti liitoksia toisten avainalueiden kanssa. Liitokset avainalueiden välillä koskevat

kypsyystasojen tarkastuspisteitä, joiden toteutuminen ajoittain vaatii toiselta avainalueelta vähintään jotain tiettyä kypsyystasoa. Avainalueiden itsenäisyys tekee mallista testausprosessin kehittämisen kannalta erittäin joustavan. Organisaation tarpeet on mallissa helppo ottaa huomioon ja malli rohkaiseekin tähän. Mallin mukaan toteutettava testausprosessin kehitystyö voi edetä nopeammin organisaation tärkeiksi katsomilla avainalueilla. Tässä kohdassa kuvataan TPI:n avainalueet ja niiden kypsyystasot lyhyesti. Kuvaukset perustuvat lähteeseen Koomen, Pol 1999.

4.3.1.1 Testausstrategia

Testausstrategia-avainalue keskittyy tärkeimpien virheiden mahdollisimman aikaiseen havaitsemiseen mahdollisimman edullisesti. Testausstrategia määrittelee, mitkä vaatimukset ja riskit katetaan milläkin testeillä. Mitä paremmin testitasot määrittelevät omat strategiansa ja mitä paremmin erilaiset testausstrategiat sovitetaan yhteen, sitä korkeampilaatuinen kokonaistestausstrategia on. Testausstrategian tasot ovat:

A. Strategiat yksittäisille korkean tason testeille.

B. Yhdistetty strategia korkean tason testeille.

C. Yhdistetty strategia korkean tason testeille sekä matalan tason testeille tai arvioinnille.

D. Yhdistetty strategia kaikille testien ja arvioinnin tasoille.

4.3.1.2 Elinkaarimalli

Elinkaarimalli koostuu testausprosessille sisäisesti määritettävistä vaiheista, kuten suunnittelusta, valmistelusta, tarkasta määrittelystä, suorituksesta ja

valmistumisesta. Jokaisessa vaiheessa suoritetaan useita toimintoja. Jokaista toimintoa kohden tulisi olla määritettynä muun muassa tarkoitus, syöte, prosessi, tuloste, riippuvuudet, soveltuvat tekniikat ja työkalut, vaadittavat välineet sekä dokumentaatio. Elinkaarimallin käyttäminen on tärkeää sillä se parantaa testausprosessin ennustettavuutta ja hallittavuutta. Ennustettavuus ja hallittavuus kasvavat, sillä erilaisia toimintoja voidaan suunnitella ja tarkkailla yhtenevällä tavalla. Elinkaarimallin tasot ovat:

A. Suunnittelu, määrittely, toteutus.

B. Suunnittelu, valmistelu, määrittely, toteutus, valmistuminen.

4.3.1.3 Osallistumisajankohta

Vaikka testien varsinainen suorittaminen alkaa normaalisti vasta tuotteen toteutuksen jälkeen, testausprosessin tulisi käynnistyä jo huomattavasti aiemmin. Testauksen aikaisempi osallistuminen ohjelmiston kehityksessä auttaa löytämään virheitä mahdollisimman aikaisin ja helposti. Se saattaa jopa auttaa estämään virheitä. Tämän seurauksena voidaan suorittaa tehokkaammin testien keskinäistä säätöä ja pystytään pitämään testauksen aika ohjelmiston kehityksen kriittisellä polulla mahdollisimman lyhyenä. Osallistumisajankohdan tasot ovat:

A. Testien perustan valmistuminen.

B. Testien perustan aloittaminen.

C. Vaatimusmäärittely aloittaminen.

D. Projektin alullepano.

4.3.1.4 Arviointi ja suunnittelu

Testien suunnittelu ja arviointi osoittavat mitä toimintoja tulee suorittaa milloinkin sekä mitä resursseja toimintojen suorituksessa tarvitaan. Hyvä arviointi ja suunnittelu on erittäin tärkeää, sillä ne ovat pohja esimerkiksi resurssien varaamiselle tietyllä aikavälillä. Arvioinnin ja suunnittelun tasot ovat:

A. Arviointi ja suunnittelu perustellusti.

B. Arviointi ja suunnittelu statistiikalla perustellusti.

4.3.1.5 Testien määrittelytekniikat

Testien määrittelytekniikka on standardoitu tapa johtaa testitapaukset lähdeinformaatiosta. Näiden tekniikoiden käyttäminen antaa näkemystä testien laatuun ja kattavuuteen sekä nostaa testien käytettävyyttä. Testien määrittelytekniikoiden tasot ovat:

A. Epäformaalit tekniikat.

B. Formaalit tekniikat.

4.3.1.6 Staattisen testauksen tekniikat

Staattisen testauksen tekniikat keskittyvät tuotosten arviointiin ilman ohjelmakoodin suorittamista. Nämä tekniikat soveltuvat lähinnä testaamaan kehitysprosessin erilaisia vaihetuotoksia sekä laatuun vaikuttavia tekijöitä, joita ei dynaamisesti edes voida testata. Staattisessa testauksessa tarkistuslistat ovat erittäin hyödyllisiä. Staattisen testauksen tekniikoiden tasot ovat:

A. Testien perustan katselmointi.

B. Tarkistuslistat.

4.3.1.7 Metriikat

Metriikat ovat tuotteen tai prosessin piirteiden tilan havainnointia ja mittaamista. Testausprosessin osalta erittäin tärkeitä ovat prosessin edistymisen ja testattavan ohjelmiston laadun metriikat. Metriikoita käytetään hyväksi testausprosessin hallinnassa ja ohjaamisessa sekä ohjelmistojen ja prosessien vertailussa. Lisäksi testausprosessia kehitettäessä metriikat ovat tärkeitä, kun arvioidaan kehitysaskelien vaikutuksia. Metriikoiden tasot ovat:

A. Projektimetriikat (tuote).

B. Projektimetriikat (prosessi).

C. Järjestelmämetriikat.

D. Organisaatiometriikat (enemmän kuin yksi järjestelmä).

4.3.1.8 Testaustyökalut

Testaustyökalut ovat testausprosessia automatisoivia apuvälineitä. Automaatiolla voi olla tavoitteena esimerkiksi kustannus- ja resurssisäästöt, parempi testausprosessin seurattavuus tai joustavuus, kattavampi testaus sekä testaajien motivaation parantaminen. Testauksen automatisoinnin tasot ovat:

A. Suunnittelun ja hallinnan työkalut

B. Suorituksen ja analysoinnin työkalut.

C. Testausprosessin laajamittainen automatisointi.

4.3.1.9 Testausympäristö

Testien suorittaminen tapahtuu testausympäristössä. Testausympäristö koostuu lähinnä laitteistosta, ohjelmistosta, kommunikaatiotavoista, menettelytavoista sekä tietokantojen ja tiedostojen hallinnan työkaluista. Ympäristö tulisi rakentaa sellaiseksi, että testien tuloksista voidaan helposti päätellä kuinka hyvin testien kohteet täyttävät niille asetetut vaatimukset. Ympäristö vaikuttaa paljon testausprosessin laatuun, läpimenoaikaan ja kustannuksiin. Tärkeitä ympäristöön liittyviä asioita ovat vastuut, hallinta, edustavuus, joustavuus sekä sopiva saatavuus. Testausympäristön tasot ovat:

A. Hallittu ja valvottu ympäristö.

B. Testaus sopivimmassa ympäristössä.

C. Ympäristö päivystää.

4.3.1.10 Toimistoympäristö

Testaushenkilöstö tarvitsee muun muassa tilat, pöydät, tuolit sekä laitteiston. Hyvin organisoidulla toimistoympäristöllä on positiivinen vaikutus testaushenkilöstön motivaatioon, tehokkuuteen ja kommunikointiin. Toimistoympäristöllä on vain yksi kypsyystaso, mutta tärkeä sellainen:

A. Asianmukainen toimistoympäristö.

4.3.1.11 Sitoutuminen ja motivaatio

Kaiken testaukseen osallistuvan henkilöstön sitoutuminen ja motivaatio ovat tärkeitä vaatimuksia hyvin toimivalle testausprosessille. Tämä ei tarkoita vain itse testaajia, vaan myös esimerkiksi ohjelmistoprojektin johtoa ja muuta

hallintohenkilöstöä. Hallintohenkilöstön sitoutuminen auttaa takaamaan testaukselle riittävän hyvät olosuhteet, jolloin testausprosessille järjestetään riittävästi aikaa, rahaa ja resursseja testaustoiminnan suorittamiseksi hyvin. Sitoutumisen ja motivaation tasot ovat:

A. Budjetin ja ajan varaaminen.

B. Testauksen integroituminen projektiorganisaatioon.

C. Testien suunnittelutoiminta.

4.3.1.12 Testaustoiminnot ja koulutus

Testaustoiminnot ja koulutus -avainalueen tavoitteena on testaustiimien sopiva rakenne ja kouluttaminen. Tiimiin vaaditaan sekoitus eri aloja, toimintoja, tietämystä ja kykyjä. Testauskokemuksen lisäksi tarvitaan testattavan ohjelmiston aihepiirin tuntemusta, organisaation tuntemusta, yleistä IT-alan tietämystä sekä sosiaalisia taitoja. Sopivan kykyrakenteen saavuttamiseksi tarvitaan muun muassa koulutusta. Testaustoimintojen ja koulutuksen tasot ovat:

A. Testauksen johtaja ja testaajat.

B. Systemaattinen, tekninen ja toiminnallinen tuki sekä hallinta.

C. Formaali sisäinen laadunvarmistus.

4.3.1.13 Työskentelytapojen laajuus

Organisaatiossa käytetään jokaista testausprosessin läpivientikertaa kohti jotain tiettyä työskentelytapaa. Työskentelytapa koostuu muun muassa käytetyistä toiminnoista, käytännöistä, säännöksistä sekä tekniikoista. Työskentelytavat

voivat heikentää testausprosessin tehokkuutta, jos ne ovat jokaisella läpivientikerralla erilaiset tai jos ne ovat niin yleisluontoisia että suuri osa niistä tarvitsee jokaisella läpivientikerralla soveltaa uudestaan vastaamaan tilannetta. Avainalueen tavoite on, että organisaatio käyttäisi työskentelytapaa, joka on riittävän yleisluontoinen, että se sopii kaikkiin tilanteisiin, mutta joka on silti riittävän yksityiskohtainen, että samojen asioiden soveltamista ei tarvitse miettiä uudestaan jokaisella läpiviennillä. Työskentelytapojen laajuuden tasot ovat:

A. Projektikohtainen.

B. Organisaatiolle yleisluonteinen.

C. Organisaatiolle optimoiva.

4.3.1.14 Viestintä

Testausprosessissa täytyy kaikkien prosessin liittyvien henkilöiden ja ryhmien keskinäisen viestinnän tapahtua monella tavalla. Erilaiset viestintätavat ovat tärkeitä toimivan testausprosessin kannalta. Näiden avulla saadaan luotua hyvät olosuhteet, optimoitua testausstrategiaa sekä viestittyä etenemisestä ja laadusta. Viestinnän tasot ovat:

A. Sisäinen viestintä.

B. Projektiviestintä (puutteet, muutosten hallinta).

C. Organisaation sisäinen testausprosessin laatuun liittyvä viestintä.

4.3.1.15 Raportointi

Testauksen ei tulisi olla niinkään puutteiden havaitsemista kuin näkemyksen hakemista tuotteen laadusta. Raportoinnin tulisi keskittyä antamaan asiakkaille perusteltuja neuvoja tuotetta ja jopa tuotteen kehitysprosessia koskien. Raportoinnin tasot ovat:

A. Puutteet.

B. Eteneminen (testauksen ja tuotteiden tilat), toimet (kustannukset ja aika, välietapit) ja priorisoidut puutteet.

C. Metriikoilla tuetut riskit sekä suositukset.

D. Suositukset ovat prosessin kehittämiseen tähtääviä.

4.3.1.16 Puutteiden hallinta

Puutteiden hallinta liittyy tiiviisti sekä ohjelmiston kehitykseen että testaukseen. Se on osa projektia, mutta testaajat ovat sen pääasiallinen käyttäjäkunta. Hyvän hallinnan pitäisi pystyä seuraamaan puutteen elinkaarta sekä tukemaan havaittujen puutteiden laatutrendien analysointia. Analysoinnin pohjalta voidaan esimerkiksi antaa perusteltuja neuvoja laatuun liittyen. Puutteiden hallinnan tasot ovat:

A. Sisäinen puutteiden hallinta.

B. Laajennettu puutteiden hallinta, joka sisältää joustavat raportointitoiminnot.

C. Projektitason puutteiden hallinta.

4.3.1.17 Testausohjelman hallinta

Testauksen tuotosten tulisi olla ylläpidettävissä ja uudelleenkäytettävissä, joten niille täytyy olla olemassa hallinta. Testauksen tuotoksiksi katsotaan muun muassa testaussuunnitelmat ja määritelmät sekä testauksessa tarvittavat tietokannat ja tiedostot, joista yhdessä syntyy testausohjelma. Testauksen omien tuotosten lisäksi on tärkeää, että myös testattavan tuotteen vaihetuotokset hallitaan hyvin. Hallinta on tärkeää, sillä testausprosessi voi häiriintyä, jos sille tuodaan vääriä versioita esimerkiksi ohjelmakoodista. Hyvä versionhallinta eri vaihetuotoksille parantaa tuotteen testattavuutta. Testausohjelman hallinnan tasot ovat:

A. Sisäinen testausohjelman hallinta.

B. Testauksen perustan ja kohteen ulkoinen hallinta.

C. Uudelleenkäytettävä testausohjelma.

D. Ohjelmiston vaatimusten jäljitettävyys testitapauksiin.

4.3.1.18 Testausprosessin hallinta

Prosessien ja toimintojen hallinnan oleellinen osa on PDCA-malli ja sen neljä toistuvaa askelta: suunnittele, tee, tarkista, toimi (Plan, Do, Check, Act). Prosessin hallinta on optimaalisen testauksen toteutumiselle elintärkeää usein sekasortoisen testausprosessin keskellä. Testausprosessin hallinnan tasot ovat:

A. Suunnittelu ja suoritus.

B. Suunnittelu, suoritus, seuranta ja mukauttaminen.

C. Seuranta ja mukauttaminen organisaation laajuisesti.

4.3.1.19 Arviointi

Arviointi tarkoittaa vaihetuotosten, kuten vaatimusten ja toiminnallisen suunnittelun katselmointia. Arviointi on tärkeää, jotta virheitä löydettäisi huomattavasti varsinaista testausta aikaisemmassa kehitysprosessin vaiheessa. Jos virheet löydetään aiemmassa vaiheessa, ovat niiden korjaamisen kustannukset huomattavasti alhaisemmat (Kuvio 3). Arviointi on testausta helpompi toteuttaa, sillä esimerkiksi ohjelmakoodia ei ole tarpeen suorittaa, eikä testausympäristöä tarvitse pystyttää. Arvioinnin tasot ovat:

A. Arviointitekniikat.

B. Arviointistrategia.

4.3.1.20 Alhaisen tason testaus

Alhaisen tason testejä suorittavat lähes poikkeuksetta kehittäjät. Tunnettuja alhaisen tason testejä ovat yksikkötesti ja integraatiotesti. Arvioinnin tapaan alhaisen tason testit löytävät virheitä aiemmassa vaiheessa kehitystä kuin korkean tason testit. Alhaisen tason testaus on tehokasta, sillä se ei tarvitse juurikaan viestintää, ja koska virheiden löytäjä on usein sama henkilö, joka virheen on tuottanut ja joka sen myös korjaa. Alhaisen tason testauksen tasot ovat:

A. Alhaisen tason testauksen elinkaari (suunnittelu, määrittely ja suorittaminen).

B. Sisäistestaustekniikat.

C. Alhaisen tason testausstrategia.

4.3.2 Kypsyystason arviointi

TPI:n kypsyystason arviointi käyttää hyväkseen testauksen kypsyysmatriisia (Taulukko 8). Arviointi pyrkii selvittämään organisaation testausprosessin vahvuudet ja heikkoudet. TPI kuvaa kaksi arviointimenettelyä, joista organisaatio voi valita omaan tilanteeseen ja tarpeisiin sopivamman.

Normaali arviointimenettely koostuu neljästä vaiheesta: valmistelusta, informaation keräämisestä, analysoinnista sekä raportoinnista. TPI ohjeistaa jokaisen vaiheen toimet tarkasti. Arvioinnin tuloksena saadaan raportti organisaation testausprosessin vahvuuksista ja heikkouksista. Lisäksi arviointi tuottaa testausprosessin kypsyystason kuvauksen kypsyysmatriisille. (Koomen, Pol 1999)

Nopeassa arviointimenettelyssä käytetään hyväksi TPI-asiantuntijaa, joka suorittaa kevyen arvioinnin. Tässä suoritetaan varsinaista arviointia edeltävät toimet mahdollisimman kevyesti. Nopea arviointi toimii hyvin pikaisena testausprosessin laadun arviointivälineenä, jos normaalin arviointimenettelyn käytölle ei ole tarvetta tai halua. (Koomen, Pol 1999)

Kypsyysmatriisissa (Taulukko 8) TPI:n avainalueet voidaan jakaa TMap:n neljän kulmakiven mukaisesti ryhmiin, joiden kehityksen suositellaan tapahtuvan tasapainoisesti. Ryhmiä ovat: elinkaari (E), organisaatio (O), infrastruktuuri ja työkalut (I) sekä tekniikat (T). Lisäksi pienen osan avainalueista katsotaan vaikuttavan kaikkiin kulmakiviin (K). Kypsyysmatriisin kypsyysluokat voidaan jakaa kolmeen ryhmään, jotka kuvaavat testausprosessin kypsyyttä karkeammalla tasolla. (Koomen, Pol 1999) Yhdessä avainalueiden ryhmäjaon kanssa, kypsyysluokkien ryhmittely tarjoaa esimerkiksi sidosryhmäviestintään sopivan karkean tason tietoa testausprosessin kypsyydestä.

Taulukko 8: Testauksen kypsyysmatriisi (Koomen, Pol 1999). Avainalueet voidaan jakaa ryhmiin sen mukaan, miten ne vaikuttavat TMap:n kulmakiviin: elinkaari (E), organisaatio (O), infrastruktuuri ja työkalut (I), tekniikat (T) sekä kaikki (K). Tyhjillä soluilla ei ole käytännön merkitystä.

	Kypsyysluokka	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
	Avainalue		Hallittu					Tehokas					Optimoiva		
E	Testausstrategia		A					B				C		D	
E	Elinkaarimalli		A			B									
E	Osallistumisajankohta			A				B				C		D	
T	Arviointi ja suunnittelu				A							B			
T	Testien määrittelytekniikat		A		B										
T	Staattisen testauksen tekniikat					A		B							
T	Metriikat						A			B			C		D
I	Testaustyökalut					A			B			C			
I	Testausympäristö				A				B						C
I	Toimistoympäristö				A										
O	Sitoutuminen ja motivaatio		A				B						C		
O	Testaustoiminnot ja koulutus				A			B				C			
O	Työskentelytapojen laajuus					A						B			C
O	Viestintä			A		B							C		
O	Raportointi		A			B		C					D		
O	Puutteiden hallinta		A				B		C						
O	Testausohjelman hallinta			A			B				C				D
O	Testausprosessin hallinta		A		B								C		
K	Arviointi							A			B				
K	Alhaisen tason testaus					A		B		C					

Kypsyysmatriisi (Taulukko 8) on TPI:n keskeinen osa. Sitä käytetään testausprosessin kypsyyden arvioinnissa, sen kehityksen suunnittelussa sekä tilanteen viestinnässä. TPI:n tyhjillä soluilla ei ole käytännössä merkitystä. Avainalueen kypsyystasojen väliin jäävistä tyhjistä soluista ei siis voida suoraan päätellä kyseisten tasojen vaatiman kehityksen raskautta. Matriisin käyttäminen testausprosessin kehityksen seurannassa tarjoaa havainnollisen visuaalisen esityksen organisaation testausprosessin kypsyydestä. Havainnollisuuden ja ymmärrettävyyden takia TPI suositteleekin, että kypsyysmatriisi olisi se asia, jonka ympärille testausprosessin kypsyyden ja kehittämissuunnitelmien esittely johdolle painottuisi. (Koomen, Pol 1999)

4.3.3 Kysymyksiä TPI:sta

TIM:n yhteydessä esitettiin kysymys siitä, ovatko kehitysmallin huomioimat kehityskohteet organisaation kannalta oleellisia. TPI:n kanssa tämä kysymys korostuu, sillä se on erittäin laaja testausprosessin kehitysmalli. TPI on myös varsin joustava kehitysmalli, mikä johtuu sen hienojakoisuudesta. Joustavuus on usein toivottavaa, mutta esitetään silti kysymys:

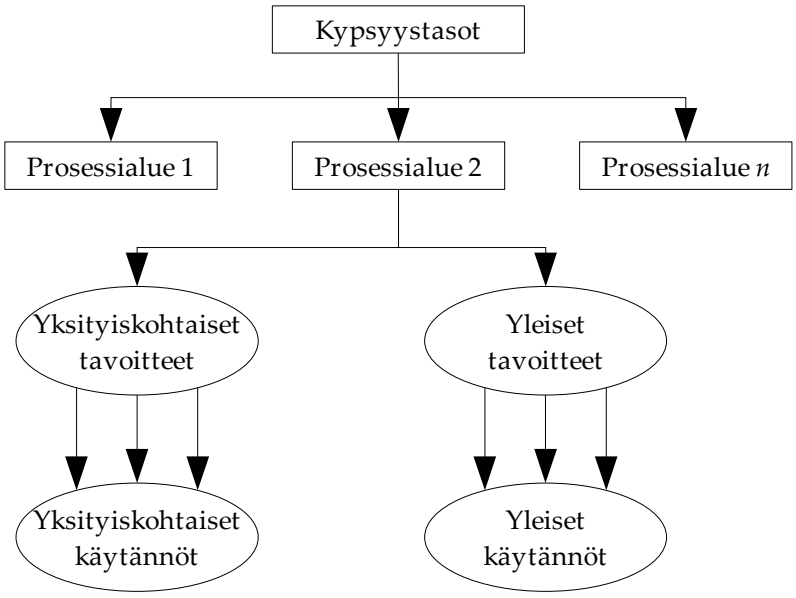
- Paljonko joustavuutta organisaatio tarvitsee testausprosessin kehitysmallilta?

4.4 TMMi

TMMi (Testing Maturity Model Integration) on TMMi-säätiön (TMMi Foundation) kehittämä malli, joka tunnettiin aiemmin kehityksen aikana myös nimellä V2M2 (Verification and Validation Maturity Model). Malli on vielä kehityksen alla. TMMi:lla on vahvoja vaikutteita TMM:sta ja sillä on tiiviitä liitoksia CMMI-malliin, jonka se käytännössä tarvitsee tuekseen. TMMi

perustuu kypsyystasoihin, kuten TMM sekä CMMI:n asteittainen malli. CMMI:n jatkuvaa mallia vastaavaa TMMi:ssa ei vielä ole. Mallin kehityksen tukena on käytetty EU-rahoitteisen MB-TMM-projektin tuotoksia (ks. Swinkels 2000) sekä IEEE 829 -standardia (ks. IEEE 829-1998), ISTQB:n ohjelmistotestauksen standardisanastoa (ks. ISTQB 2007c) ja käytännön kokemuksia TMM:n käytöstä. (van Veenendaal ym. 2008)

TMMi:n rakenne (Kuvio 12) koostuu viidestä kypsyystasosta, joista alin on aloitustaso. Kypsyystasoille on määritelty vaihtelevasti prosessialueita, jotka vastaavat ajatukseltaan TPI:n ja TIM:n avainalueita. Ne ovat siis testausprosessin osa-alueita, joihin keskitytään. Toisin kuin kahdella edellä mainitulla mallilla, TMMi:lla prosessialueet otetaan kehitykseen mukaan vähitellen, ja niiden kehittämiseen keskitytään pääasiassa vain kypsyystasolla, jolla ne tulevat vastaan (Kuvio 13). (van Veenendaal ym. 2008) Jokaisen prosessialueen muodostaa joukko yksityiskohtaisia tavoitteita, jotka koostuvat vastaavasti yksityiskohtaisista käytännöistä.



Kuvio 12: TMMi:n rakenne (van Veenendaal ym. 2008, 12).

Kuvio 13: TMMi:n prosessialueet kypsyystasoittain (van Veenendaal ym. 2008, 7).

Yllä kuviossa 12 esitetty rakenne, jossa prosessialue pitää sisällään yleiset tavoitteet, on hiukan hämäävä. Todellisuudessa mallissa kuvatut yleiset tavoitteet ovat nimittäin yhteisiä kypsyystason sekä sitä alempien kypsyystasojen kaikille prosessialueille. Yleiset käytännöt koskevat samaan tapaan kaikkia kypsyystason ja sitä alempien tasojen prosessialueita, mutta ne realisoituvat eri tavoin prosessialueesta riippuen. (van Veenendaal ym. 2008) Mallin rakennetta ja sisältöä tarkasteltaessa on havaittavissa, että TMMi on TMM:ia selvästi laajempi malli. Laajuuden hallintaa ei ole kuitenkaan juuri kehitetty, joten TMMi on TMM:iakin huomattavasti raskaampi ja jäykempi malli käyttää. Kovin syvällisen vertailun tekeminen tässä on valitettavasti mahdotonta jo aiemmin mainitun Suwannasartin väitöskirjan kroonisen

saatavuusongelman vuoksi sekä TMMi:n vielä hiukan keskeneräisen luonteen takia.

Seuraavaksi käydään lyhyesti läpi TMMi:n kypsyystasot sekä organisaation kypsyystason arviointimahdollisuudet. Prosessialueiden sisältöön otetaan kantaa vain siltä osin, kuin se on tarpeen mallista esille nousevien kysymysten esittämisen yhteydessä. Tasojen kuvaukset perustuvat lähteeseen van Veenendaal ym. 2008.

4.4.1 Taso 1

Taso 1 vastaa täysin TMM:n tasoa 1. Testausprosessi ei täytä TMMi:n tason 2 vaatimuksia, joten se katsotaan kaoottiseksi toiminnaksi. Testausprosessia ei ole kunnolla määritelty ja testauksen laatu sekä tulokset riippuvat täysin yksittäisistä henkilöistä. Testejä tehdään yleensä ennakkoon suunnittelematta ja ohjelmiston laadusta sekä riskeistä ei ole kunnollista tietoa. Testaukseen ei ole käytettävissä riittävästi resursseja ja julkaistujen ohjelmistojen laatu ei usein täytä sille asetettuja odotuksia ja tavoitteita.

4.4.2 Taso 2

Tasolla 2 (Managed) testauksesta tulee hallittu prosessi, joka on selvästi erotettu virheestyksestä. Testausprosessista on saatava riittävän kypsä, jotta käytäntöjä noudatetaan myös paineen alla. Koko yrityksen kattava testausstrategia luodaan ja testaussuunnitelmia kehitetään. Testaussuunnitelmissa määritetään testaukselle lähestymistapa, joka perustuu arvioon testattavan ohjelmiston riskeistä. Riskienhallintaa käytetään tunnistamaan riskejä ohjelmiston dokumentoitujen vaatimusten perusteella.

Testaussuunnitelmalla määritellään kuka testaa, mitä ja miten testataan sekä milloin testaus suoritetaan. Testaussuunnitelman toteutusta valvotaan ja hallitaan, jotta voidaan varmistaa testauksen oikea suunta sekä pysyä ajan tasalla mahdollisista suunnitelmasta poikkeamisista. Testauksen suunnittelutekniikoilla tuotetaan ja valitaan tilanteeseen sopivat testitapaukset. Testausta harjoitetaan usealla tasolla. Suoritetaan yksikkö-, integraatio-, järjestelmä- sekä hyväksymistestaukset. Kaikille näille testauksen tasolle on asetettu selkeät tavoitteet testausstrategialla.

Tälle kypsyystasolle asetetuista laajoista vaatimuksista huolimatta testaus aloitetaan usein varsin myöhäisessä vaiheessa kehitysprosessia, ja sen tavoitteena on vain todistaa, että tuote täyttää sille asetetut vaatimukset. Laatuongelmia aiheutuu tällä tasolla myöhäisestä testauksen aloitusvaiheesta, minkä ansiosta virheet pääsevät määrityksistä ja suunnitelmista ohjelmakoodiin. Tason 2 prosessialueet ovat:

- Testausstrategia ja menettelytavat.
- Testauksen suunnittelu.
- Testauksen valvonta ja hallinta.
- Testien suunnittelu ja suoritus.
- Testausympäristö.

4.4.3 Taso 3

Tasolla 3 (Defined), organisaation testausprosessin toimintaa standardoidaan organisaation sisäisesti. Edellisellä tasolla standardien kattavuus, prosessien kuvaukset sekä menettelytavat saattoivat vaihdella prosessin instanssien, kuten

projektien välillä. Nyt yritykselle luodaan näistä standardit, joita räätälöidään projektien tai yksiköiden tarpeiden mukaan tiettyjen räätälöintiohjeiden rajoissa.

Testaus ei tällä tasolla ole enää vaihe, joka seuraa toteutusta, vaan se on integroitu organisaation koko kehitysprosessiin ja sen välietappeihin. Testaus suunnitellaan hyvin aikaisessa vaiheessa projektia, esimerkiksi vaatimusten määrittelyn yhteydessä. Testaajien koulutuksesta huolehditaan ja testaus mielletään todelliseksi ammatiksi.

Organisaatio alkaa tällä tasolla ymmärtää katselmointien tärkeyden laadun varmistuksessa. Katselmointiohjelma käynnistetään, ja katselmointiin otetaan kehitysprosessin kaikkien vaiheiden tuotokset. TMMi:n kolmannella tasolla organisaation testausprosessi on yleensä määritelty huomattavasti tarkemmin kuin tasolla 2. Tarkemman määrittelyn takia TMMi:n yleiset käytännöt koskevat aina myös alempia tasoja. Näin ollen alempien tasojen prosessialueita joudutaan myös kehittämään. Tason 3 prosessialueet ovat:

- Testausorganisaatio.
- Testauksen opetusohjelma.
- Testauksen elinkaari ja integraatio.
- Ei-toiminnallinen testaus.
- Vertaiskatselmukset.

4.4.4 Taso 4

Taso 4 (Management and Measurement) keskittyy testausprosessin hallinnan ja mittaroinnin kehittämiseen. Testausprosessi toimii hyvällä pohjalla, täysin

määritellysti ja mittaroitavasti. Tuotteiden laadulle ja prosessien suorituskyvylle asetetaan mitattavia tavoitteita, joita käytetään hyväksi näiden hallinnassa koko elinkaaren läpi. Katselmoinnit ja tarkistukset luetaan osaksi testausta ja niitä käytetään mittaamaan dokumenttien laatua. Tällä tasolla tuotteista mittaan laadullisia ominaisuuksia, kuten luotettavuutta, käytettävyyttä ja ylläpidettävyyttä. Testausprosessin toimintaa mitataan organisaation laajuisesti, millä saadaan tuotua tietoa ja näkyvyyttä prosessin toiminnasta. Testaus koetaan arvioinniksi, joka koostuu kaikesta kehityksen tuotosten ja vaihetuotosten arviointiin liittyvästä. Tason 4 prosessialueet ovat:

- Testauksen mittaaminen.
- Ohjelmiston laadun arviointi.
- Edistyneet vertaiskatselmukset.

4.4.5 Taso 5

Tasolla 5 (Optimization) testausprosessin menetelmiä ja tekniikoita optimoidaan. Prosessin tavoite on virheiden ennaltaehkäisy. Testaus on prosessina täysin määritelty ja sen kustannuksia sekä tehokkuutta voidaan kontrolloida. Organisaatio kehittää testausprosessia jatkuvasti eri prosessien toiminnan vaihteluiden yleisten syiden pohjalta. Testausprosessia tarkastellaan jatkuvasti hienosäädön sekä kehittämisen kannalta. Prosessille on tunnusomaista otantaan perustuva laadun mittaaminen.

Tällä tasolla testausprosessia tuetaan työkaluilla mahdollisimman paljon läpi koko prosessin. Testaustyökalujen valintaan ja arviointiin on olemassa tarkka menettelytapa. Tason 5 prosessialueet ovat:

- Virheiden ennaltaehkäisy.

- Testausprosessin optimointi.
- Laadun hallinta.

4.4.6 Kypsyystason arviointi

TMMi:iin ei nykyisellään kuulu kypsyystason arviointia. TMMi Foundation on julkaissut arviointimalleille asetettavat vaatimukset TAMAR-kehikkona (TMMi Assessment Method Application Requirements, Wells ym. 2008), joissa muun muassa otetaan kantaa muodollisiin sekä epämuodollisiin arviointeihin ja mihin näiden tuloksia voidaan käyttää. TAMAR-kehikkoa noudattavat arviointimenetelmät noudattavat myös ISO 15504-2 -standardia (ks. ISO/IEC 15504-2:2003) siten, kuin sitä on kehikossa tulkittu. (Wells ym. 2008)

Kehikkoa noudattava muodollinen arviointi tuottaa tarkistettavissa olevan kypsyysarvion organisaation testausprosessista. Kypsyysarvio sisältää tarkan arvion organisaation testausprosessin vahvuuksista ja heikkouksista sekä sen kypsyystasosta. Muodollinen arviointi vaatii henkilöstön laaja-alaisten haastattelujen sekä oleellisten dokumenttien käyttämistä arvioinnissa. Muutakin materiaalia voidaan käyttää edellä mainittujen lisäksi. Arviointi suoritetaan tiimillä, johon kuuluu vähintään kaksi henkilöä. Tiimissä täytyy olla yksi TMMi Foundationin valtuuttama arvioinninjohtaja, minkä lisäksi tarvitaan vähintään yksi TMMi Foundationin valtuuttama arvioija. (Wells ym. 2008)

Kehikkoa noudattava epämuodollinen arviointi tuottaa nopeasti karkean arvion organisaation testausprosessin tilasta. Arvioinnista on hyötyä lähinnä sisäisessä kypsyyden arvioinnissa. Epämuodollista arviointia varten tarvitaan esimerkiksi henkilöstön haastatteluja, oleellisia dokumentteja tai kyselyitä. Arviointitiimiin ei tarvitse kuulua kuin yksi henkilö, jolta vaaditaan kokemusta

arviointien suorittamisesta. Arvioijien ei kuitenkaan tarvitse olla TMMi Foundationin valtuuttamia. (Wells ym. 2008)

4.4.7 Kysymyksiä TMMi:sta

TMMi on luotu tukemaan CMMI:ia, ja käytännössä vaatii, että organisaatiolla on CMMI myös käytössä. TMMi myös rajoittaa CMMI:n käyttöä sen asteittaiseen malliin, sillä TMMi ei sisällä CMMI:n jatkuvan mallin vastinetta. Jos organisaatio joutuu ottamaan CMMI:n käyttöön TMMi:n käyttöönoton takia, voi tästä aiheutua ylimääräistä rasitetta. Toisaalta jos organisaatio jo käyttää CMMI:ia ja nimenomaan sen asteittaista mallia, on TMMi:lla jonkinlainen etulyöntiasema muihin testausprosessin kehitysmalleihin nähden. Tästä seuraa kysymyksiä:

- Vaatiiko käyttöön valittu testausprosessin kehitysmalli rinnalleen jonkin tietyn prosessinkehitysmallin?
- Rajoittaako käyttöön valittu testausprosessin kehitysmalli käytössä olevan prosessinkehitysmallin soveltamista?
- Tukeeko käytössä oleva prosessinkehitysmalli jonkin tietyn testausprosessin kehitysmallin käyttämistä ja miten tämä ilmenee?

TMMi on dokumentaation mukaan versiossa 1.0, mutta syystä tai toisesta on tarkasti dokumentoitu vasta TMMi:n taso 2 ja siihen liittyvät prosessialueet sekä vaatimukset (van Veenendaal ym. 2008). Arviointimenetelmää mallin mukaisen kypsyystason selvittämiselle ei ole olemassa. Keskeneräisyys voi yllättää, sillä versionumeron perusteella voisi ajatella mallin olevan kypsä käyttöä varten. Näin ei kuitenkaan ole, ja TMMi:iin on suunnitteilla vielä

muutoksiakin, mukaan lukien jatkuva kehitysmalli CMMI:n tapaan (TMMi Foundation 2008). Näistä huomioista seuraa kysymyksiä:

- Onko kehitysmalli riittävän kypsä, että se pystytään ottamaan käyttöön?
- Jos kehitysmalli on edelleen kehityksessä, onko odotettavissa muutoksia, jotka vaikuttavat organisaation saavuttamaan luokitukseen suuntaan tai toiseen?
- Jos kehitysmalli on edelleen kehityksessä, onko kehityksen suunta sellainen, jonka organisaatio kokee hyväksyttäväksi tai hyödylliseksi?
- Jos kehitysmalli otetaan käyttöön keskeneräisenä, onko takeita siitä, että malli myös valmistuu hyväksyttävällä aikataululla?

TMMi:n vaatimukset toiselle kypsyystasolle pääsemiseksi ovat selkeästi TMM:iakin raskaammat. Esimerkki, joka kuvaa hyvin etenemisen raskautta TMMi:lla suhteessa TPI:iin, esitetään TMMi:n kuvauksessa (ks. van Veenendaal ym. 2008, 74-78). Kyseisessä esimerkissä tosin on selkeitä puutteita, sillä siinä katsotaan TPI:n taso saavutetuksi, jos se on osittain katettu TMMi:ssa. Esitetään kysymykset:

- Kuinka organisaation kannattaa valita itselleen sopiva testausprosessin kehitysmalli?
- Onko kehitysmallin käyttämisestä yleensä nähtävillä riittävästi hyötyjä sen vaatimiin resursseihin nähden, jotta sellainen kannattaa ottaa käyttöön?

4.5 Yhteenveto

Luvussa esiteltiin neljä testausprosessin kehitysmallia. Malleilla on selkeitä eroja, mutta myös paljon samankaltaisia piirteitä. Malleilla on kaikilla sama tavoite, optimoiva testausprosessi, mutta mallien kattavuus ja etenemistahti vaihtelevat huomattavasti. Myös kehitysmallien rakeisuudessa on tuntuvaa vaihtelua, varsinkin karkeimman TMM:n ja hienojakoisimman TPI:n välillä. Karkeusasteen vaihtelusta johtuen myös mallien havaittava raskaus sekä jäykkyys vaihtelevat huomattavasti. Eroista riippumatta kaikilla malleilla on kuitenkin sama perustavaa laatua oleva tavoite: optimoituva testausprosessi. Mallien eroista johtuen, niiden soveltuvuus eri tilanteissa oleville organisaatioille vaihtelee.

4.5.1 Jatkuva vs. asteittainen malli

Jatkuvat kehitysmallit, kuten TIM ja TPI soveltuvat paremmin organisaatioille, joille suurten askelten ottaminen prosessin kehityksessä tuottaisi ongelmia. Ne soveltuvat organisaatiolle paremmin myös silloin, jos halutaan kehittää organisaation kannalta oleellisia testauksen osa-alueita. TIM ja TPI ovat jatkuvina kehitysmalleina myös huomattavasti TMM:ia ja TMMi:ia kevyempiä ottaa käyttöön. Lisäksi ne ovat riippumattomia muista prosessinkehitysmalleista, joten niiden kanssa on hyvin mahdollista käyttää esimerkiksi CMMI:ia.

Mukaan otetuista malleista loput kaksi, TMM ja TMMi, eivät ole jatkuvia, joten prosessin kehittäminen niitä seuraten on raskaampaa ja jäykempää. Ne tarvitsevat kumppaneikseen CMM:n tai CMMI:n, joiden tueksi ne on kehitetty. CMM tai CMMI ovat kumpikin raskaita vaatimuksia, mutta yhdessä testausprosessin kehitysmallien kanssa ne myös tarjoavat erinomaisen kattavan

ja yhtenäisen keinon kehittää organisaation koko ohjelmistotuotantotoimintaa. Koska TMM ja TMMi on luotu sisällöiltään ja rakenteiltaan CMM:ia sekä CMMI:n asteittaista mallia täydentäviksi malleiksi, on niiden valinta kyseisiä prosessinkehitysmalleja seuraavissa organisaatioissa hyvin todennäköistä.

4.5.2 Havaintoja testausprosessin kehitysmalleista

CMM:iin ja CMMI:iin tutustuminen syvällisemmin ei kuulu tämän tutkimuksen aihepiiriin, joten on vaikea sanoa, toimiiko TMM tai TMMi näiden kanssa paremmin kuin TIM tai TPI. On kuitenkin selvää, että organisaatiolle on muun muassa kehityksen seurannan ja sidosryhmäviestinnän kannalta yksinkertaisempaa käyttää toisiaan tiiviisti tukevia kehitysmalleja. Valitettava puute TMMi:n osalta on kuitenkin jatkuvan kehitysmallin puuttuminen. Jos organisaatio seuraa CMMI:n jatkuvaa mallia, soveltuu sille testausprosessin kehitysmallin rakenteen osalta tällä hetkellä käyttöön paremmin TPI kuin TMMi.

TMMi:n ja TMM:n vertailu tarkalla tasolla antaisi objektiivisen näkemyksen sille, mitä hyötyjä tai haittoja TMMi:n käytöstä TMM:n sijaan todellisuudessa on. Eritoten mallien toisen tason vaatimusten ja niihin liittyvien resurssitarpeiden vertailu voisi tuoda hyvää tietoa mallien käyttöönoton suhteellisesta raskaudesta. Täysin objektiivisia vertailuita tutkimukseen valituista kehitysmalleista ei ole saatavilla. Swinkels (Swinkels 2000) teki vertailua tavoitteena kehittää TMM:ia. Luonnollisesti vertailu keskittyi lähinnä siihen, mitä ajatuksia muista malleista voisi TMM:n kehittämiseen saada. Se ei vertailevana tutkimuksena näin ollen ole riittävän objektiivinen. Testausprosessin kehitysmalleista on tehty muitakin vertailevia tutkimuksia, joissa tähän tutkimukseen valittuja malleja on ollut mukana. Valitettavasti kyseisten tutkimusten tuloksia ei ole saatavilla.

Tutkituilla malleilla on erilainen tapa merkitä aloitustaso, vaikka kyseisestä tasosta kaikilla onkin sama ajatus. Aloitustaso mielletään aina tasoksi, jolla toimitaan hallitsemattomasti. TMM ja TMMi merkitsevät aloitustason tasoksi 1, kun TIM ja TPI merkitsevät sen tasoksi 0. Näistä jälkimmäinen tapa on ehkä selkeämpi, sillä kyseinen taso on kaikissa malleissa ajateltu tasoksi, joka ei malleja vielä noudata. TMM:n ja TMMi:n merkintätapa voi ehkä antaa kuvan, että ollaan jo ensimmäisellä todellisella kypsyystasolla, kun ollaan aloitustasolla.

TMMi:lla ei ole määritettyä arviointimenetelmää. Muodollisen arvioinnin TAMAR-kehikkoa noudattava menetelmä voi kuitenkin olla kevyempi kuin TMM-AM (vrt. Wells ym. 2008, luku 2.2 ja Burnstein ym. 1998). Lisäksi epämuodollisen arvioinnin kehikkoa noudattava menetelmä on huomattavan helppo käydä läpi. Arviointimenetelmien mahdollinen keventyminen tekee TMMi:sta siltäkin osin TMM:ia houkuttelevamman mallin. Se, että TMMi:lle löytyy kaikesta huolimatta enemmän dokumentaatiota kuin TMM:lle, tekee TMMi:n näistä kahdesta suositeltavammaksi malliksi. Samoin on laita TMMi:n ja TIM:n välillä. TIM saattaa tosin keveytensä ansiosta olla TMMi:ia miellyttävämpi kehitysmalli joissain tilanteissa. TMMi ja TPI ovat rakenteeltaan hyvin erilaisia. Sisällön kannalta kyseiset mallit ovat laajoja, kumpikin omalla tavallaan. Siksi TMMi:n ja TPI:n välillä ei ole täysin selkeää paremmuussuhdetta. TMMi:n keskeneräisyyden vuoksi TPI on tarkastelluista malleista tällä hetkellä helpoin suositella käytettäväksi. TPI:n joustava ja kattavasti dokumentoitu malli kehitysehdotuksineen sopii hyvin myös CMMI:ia noudattavan organisaation käyttöön.

TMMi:n kehitys on vielä sen verran kesken, että mallin käyttöönottoa ei voi vielä täysin varauksetta suositella. Valmiiden ja julkisesti saatavilla olevien

kypsyystason arviointimenetelmien puuttuminen on myös vakava puute. TMMi:sta on kuitenkin selkeästi tulossa varsin kattava malli, ja CMMI:tä vastaavan jatkuvan kehitysmallin myötä siitä todennäköisesti tulee myös varsin joustava. Tällä hetkellä TPI on tarkastelluista testausprosessin kehitysmalleista selkeästi monipuolisin ja käytettävin, mutta TMMi:sta tulee sille varmasti hyvä kilpailija. Ainoa näköpiirissä oleva selkeä ongelma TMMi:n käytössä on, että se vaatii lähes ehdottomasti käyttöön myös CMMI:n, mikä voi tarkoittaa organisaatiolle ylimääräisiä kustannuksia.

5 TESTAUSPROSESSIN KEHITYKSEN KYSYMYKSIÄ

Edellisessä luvussa esiteltiin ja lyhyesti käsiteltiin neljää testausprosessin kehitysmallia. Mallien yhteydessä nostettiin esille joitain niihin liittyviä seikkoja ja kysymyksiä, joihin testausprosessia kehitettäessä tai mallia valittaessa olisi hyödyllistä saada vastauksia. Esitettyihin kysymyksiin annetaan mahdollisuuksien mukaan tässä luvussa vastauksia. Tämän lisäksi on otettu esille joitain testausprosessin kehitykseen yleisesti liittyviä huomioitavia aiheita. Aiheita pyritään käsittelemään suhteellisen loogisessa järjestyksessä. Luvun lopuksi esitetään perusteita testausprosessin kehitysmallin käyttämiselle ja annetaan neuvoja organisaatiolle sopivan kehitysmallin valintaan.

5.1 Vaikutukset henkilöstöön

Testausprosessin kehitys, kuten mikä tahansa organisaation prosessin kehitys, voi vaikuttaa laajasti organisaation toimintaan. Kun organisaation toimintaan tulee muutoksia, koskettavat vaikutukset myös organisaation henkilöstöä. Tässä selvitetään lyhyesti ja vapaamuotoisesti, millaisia vaikutuksia testausprosessin kehittämisellä näyttäisi henkilöstöön olevan. Pohjana käytetään testausprosessin kehitysmallien kuvauksia ja niissä mainittuja, selkeästi henkilöstöön tai työtehtäviin vaikuttavia seikkoja. Jos ei toisin mainita, esitetyt asiat pohjautuvat kehitysmallien kuvauksiin lähteissä Burnstein ym. 1996a sekä 1996b ja 1996c, Ericson ym. 1997, Koomen, Pol 1999 ja van Veenendaal ym. 2008.

5.1.1 Uusia työtehtäviä

Testausprosessin mallien mukainen kehittäminen tuo poikkeuksetta katselmoinnit mukaan organisaation ohjelmistokehitysprosessiin.

Katselmoinnit (ks. luku 5.4) vaikuttavat laajasti koko ohjelmiston kehitykseen osallistuvaan henkilöstöön. Selkeimmin vaikutuksen kokevat henkilöt, jotka luovat vaatimusmäärittelyjä, suunnitelmia, käyttötapauksia, testitapauksia tai mitä tahansa muuta kirjallista tai graafista ohjelmistokehityksessä tai testauksessa käytettävää materiaalia. Katselmointien tavoitteena on kaiken tämän materiaalin tarkistaminen organisaation muun henkilöstön toimesta. Yleistä testausprosessin kehitysmalleissa on, että katselmointiin pyritään saamaan mukaan myös testauksen edustajia. Katselmointi vaatii, että katselmoitava materiaali on riittävän laadukasta. Lisäksi katselmoinnin seurauksena kyseiseen materiaaliin lähes poikkeuksetta joudutaan tekemään korjauksia (Gilb 2000). Katselmoinnit aiheuttavat siis mahdollisesti lisää työtä pohjamateriaalin tuottajille. Työtä ei kuitenkaan tule heille lisää paljon, jos tuotettu materiaali on jo valmiiksi laadukasta. Testaajille ja mahdollisesti kehittäjille katselmoinnit aiheuttavat lisää työtä katselmointeihin osallistumisen muodossa, mutta myös huomattavasti vähentävät työmäärää pohjamateriaalin virheiden vähentymisen myötä. Pohjamateriaalin tuottajille aiheutuva työmäärän kasvu on huomattavasti pienempi kuin kyseistä materiaalia hyödyntävän henkilöstön saavuttama työmäärän väheneminen.

Malleja, eritoten TIM:ia (Ericson ym. 1997), tarkasteltaessa tulee esille ajatus testaus- ja kehityshenkilöstön työtehtävien kierrättämisestä. Tämä on perusteltu hyvin sillä, että kehittäjien tutustuminen testaukseen lisää kehittäjien ymmärrystä testauksen tarpeista ja näin ollen parantaa tuotettavan ohjelmakoodin testattavuutta. Testaajien tutustuminen kehitystyöhön puolestaan kehittää testaajien ymmärrystä testattavasta ohjelmakoodista ja sen käyttötavoista. Työtehtävien kierrättäminen ei suoranaisesti lisää kehittäjien tai testaajien työmäärää, mutta se voi alkuun vaatia hiukan tutustumista ja koulutusta. Työtehtävien kierrättäminen saattaa kuitenkin jopa parantaa

työmoraalia kasvattamalla töiden mielenkiintoisuutta. Testaajien ja kehittäjien tehtävien kierrättämiselle on olemassa myös perusteltua vastustusta (ks. luku 5.2). Tehtävät ovat luonteeltaan varsin erilaisia ja saattavat vaatia työtä tekeviltä henkilöiltä jonkin verran erilaisia luonteenpiirteitä (Cohen ym. 2004).

Jos esimerkiksi yksikkötestaus otetaan ensimmäistä kertaa käyttöön, aiheutuu tästä lisää työtä henkilöstölle, joka on vastuussa yksikkötestien luomisesta. Yleensä vastuullinen taho on joko kehittäjät tai testaajat. Yksikkötestit voivat kuitenkin helpottaa ohjelmakoodin luomista sekä virheiden aiempaa kiinnisaamista, jos testit tehdään ennen ohjelmakoodin kirjoittamista. Suunnitellut toiminnot täysin kattavien automaattisten testitapausten luomista ennen ohjelmakoodin kirjoittamista kutsutaan TDD:ksi (Test Driven Development). TDD helpottaa virheiden aiempaa kiinnisaamista, sillä testien kirjoittaminen voi paljastaa virheitä suunnitteludokumenteissa ennen kuin kyseiset virheet pääsevät ohjelmakoodiin. Jos ohjelmakoodiin ei pääse virhettä, sitä ei sieltä tarvitse myöskään korjata, ja näin säästyy kehittäjän aikaa. Lisäksi TDD:ssa ohjelmakoodiin syntyvät virheet havaitaan usein nopeasti.

5.1.2 Uusia työkaluja

Testauksen työkalut ovat esillä kaikissa testausprosessin kehitysmalleissa jollain tasolla. Uusia työkaluja hankittaessa on tavoitteena tavallisesti joko töiden helpottaminen, laadun parantaminen tai kustannussäästöt. Työkalujen käytön kouluttamiseen kuluu jonkin verran aikaa, mutta huolella valitut työkalut usein helpottavat testaajien ja muun prosessiin osallistuvan henkilöstön töitä. Työkalut voivat myös lisätä testaushenkilöstön mielenkiintoa työtänsä kohtaan ja parantaa näin työmoraalia.

5.1.3 Lisää informaatiota

Testausprosessin kehitysmalleissa on kaikissa jollain tavalla mukana kaksi informaation keräämiseen ja levitykseen vaikuttavaa osa-aluetta: metriikat ja raportointi. Kehittyvillä metriikoilla pystytään keräämään tarkkaa tietoa lähes mistä tahansa testausprosessin osa-alueesta. Raportoinnin kehittyessä taas kyetään tätä tietoa hyödyntämään paremmin. Informaatiota pystytään myös tehokkaammin levittämään sitä hyödyntäville tahoille.

5.2 Testaus vs. kehitys

TIM (Ericson ym. 1997) ehdottaa testaus- ja kehitystoimien suhteen sekä työtehtävien kierrättämistä, että riittävän etäisyyden säilyttämistä. Organisaatiosta riippuen tämä voi olla toimiva ratkaisu, mutta siihen liittyy kaksi suunnittelua ja käytännön kokeilua vaativaa kysymystä. Ehdotetuille testaus- ja kehitystoimien järjestelyille on myös varsin perusteltua vastustusta (esim. Cohen ym. 2004). Työtehtävien kierrättäminen ei ehkä onnistu aivan helposti, minkä lisäksi etäisyys testauksen ja kehityksen välillä saattaa kasvattaa konfliktin riskiä. Tässä kohdassa käsitellään ensin TIM:n ratkaisumalli ja kuinka sen voidaan ajatella toimivan, jonka jälkeen asiaa hieman eri näkökulmasta tarkastellen perustellaan toisenlainen ratkaisu.

5.2.1 Työtehtävien kierrättäminen ja eristäminen

TIM:stä (Ericson ym. 1997) esiin nousseet kysymykset testaajien ja kehittäjien tehtävien kierrättämisen nopeudesta ja siinä huomioitavista asioista osoittautuivat liian vaikeiksi vastattaviksi. Kierrättämisestä haetaan etuja tutustuttamalla testaus- ja kehityshenkilöstöä toistensa töihin. Tästä ajatellaan seuraavan tehokkaampaa testausta sekä testattavampaa ohjelmakoodia.

Työtehtävien kierrättämistahdin voisi ajatella olevan jossain määrin organisaatio- ja projektikohtainen muuttuja. Liian nopealla kierrolla ei henkilöstö saa kunnolla kiinni uusista työtehtävistään. Järkevänä lyhimpänä kiertoaikana toimisi luultavasti kahden viikon väli. Tämä mahdollistaa töihin kiinnittymisen ja riittävän vauhdin saavuttamisen. Lisäksi kahdessa viikossa pitäisi olla mahdollista viedä ainakin yksi kehitettävä tai testattava kohde alusta loppuun, modulaarisuudesta riippuen. Kierron ei kannata olla liian pitkä, sillä silloin voi alkaa esiintyä haluttomuutta irrottautua tutuista ja turvallisiksi koetuista tehtävistä. Pisintä toimivaa kiertoväliä on vaikea ehdottaa, mutta sen ei liene järkevää olla paljon yli kuukauden, jos kierrosta halutaan saavuttaa TIM:ssa esitettyjä hyötyjä.

TIM esittää myös ajatuksen testaustoimen sijoittamisesta fyysisesti samaan ympäristöön ja sen toimintojen riittävää eristämistä kehitystyöstä (Ericson ym. 1997). Ajatuksena on, että jos testaustoimi pystytään sopivasti eristämään, voidaan taata testauksen toimiminen, vaikka kehitykseen kohdistuisi esimerkiksi aikataulupaineita. Testauksen eristäminen ja tehtävien kierrättäminen tuntuisivat jossain määrin sotivan toistensa kanssa. Organisaatiolle ja projektille sopivan pitkällä kiertoajalla voidaan kuitenkin todennäköisesti ylläpitää sopivaa eristyksen tasoa, vaikka tehtäviä kierrätettäisikin.

5.2.2 Työtehtävien eriyttäminen ja yhteistyön tiivistäminen

Työtehtävien kierrättäminen ei ehkä kuitenkaan ole toimivin ratkaisu kaikkiin tilanteisiin. Kuten Cohen kumppaneineen toteaa, testaus- ja kehitystehtävien vaatimat luonteenpiirteet poikkeavat selkeästi. Tämän ansiosta on luonnollista, että testaajilla ja kehittäjillä on omia tehtäviään vastaavat luonteet. Tehtävien kierrättäminen ei siis ehkä onnistu henkilöiden luonteiden ja tehtävien

vaatimusten aiheuttaman ristiriidan takia. (Cohen ym. 2004) Epäilemättä on kuitenkin organisaatiosta ja etenkin henkilöstöstä kiinni, kuinka hyvin tehtävien kierrättäminen voi toimia.

Tehtävien eristämisen sijaan voi olla hyödyllisempää pyrkiä ylläpitämään tiiviitä suhteita testauksen ja kehityksen välillä. Tämä sotii selkeästi TIM:n esittämää irrallaan pitämistä vastaan. Yhteistyö kuitenkin vähentää testauksen ja kehityksen välistä konfliktin riskiä. Cohenin ja kumppaneiden mukaan eräs tehokas tapa pitää testaajat ja kehittäjät tiiviissä, hyvin motivoidussa yhteistyössä on luoda tiimejä, jotka koostuvat sekä testaajista että kehittäjistä. Integroidut tiimit toimivat hyvin varsinkin, jos tiimi sidotaan pysyvästi tiettyyn sovellukseen tai sovelluksiin. Tällä pyritään siihen, että sama tiimi vastaa sovelluksesta sen elinkaaren alusta loppuun saakka, ylläpito mukaanlukien. Pysyvä koko tiimin sitoutuminen luo yhteenkuuluvuutta ja yhteistä onnistumisen tunnetta. Jos testaus- ja kehitystiimit ovat erilliset, saattaa tiimien välille muodostua kitkaa, minkä lisäksi kehitys alkaa helposti syödä testaukselle varattua aikaa. (Cohen ym. 2004)

5.3 Mittarointi ja metriikat

Organisaation ja sen prosessien toiminnassa, samoin kuin tuotettavan ohjelmiston toiminnassa on lähes rajaton määrä asioita, joita voidaan mitata. Mittarointi tulee pitää riittävän geneerisenä, jotta sillä ei voida mitata henkilöstön suorituskykyä yksilöivästi. Metriikoiden virheellisen tulkinnan riski on liian suuri, minkä lisäksi yksilöivät metriikat voivat aiheuttaa kiusauksen muokata mittaustuloksia. Mittarointi onkin syytä aloittaa kevyesti ja mitattavat asiat sekä niitä mittaavat metriikat valita harkiten. (Koomen, Pol 1999) Joitain hyvin yksinkertaisia kontrollimetriikoita ovat muun muassa:

- Resurssien käyttö esimerkiksi projektia tai koodiyksikköä kohden: tunnit, kustannukset, yms.
- Ohjelmiston elinkaaren vaiheiden läpimenoajat.
- Elinkaaren eri vaiheessa löydettyjen virheiden määrä. Virheitä on hyvä myös ryhmitellä kriittisyysasteen ja niiden lähteen mukaan.
- Testitapauksella löydettyjen virheiden määrä ja laatu.

Mittaroinnin toteuttaminen suoritetaan usein omana projektinaan, sillä se vaikuttaa laajasti organisaation toimintaan. Osittain tämä johtuu myös siitä, että mittaroinnin käyttöön ottamisen mahdollisia ongelmia ei tule aliarvioida, ja niitä on helpompi hallita projektina. (Koomen, Pol 1999) Mittarointi ja metriikat ovat kuitenkin tieteen alue, johon organisaation on syytä tutustua tämän tutkimuksen käsittelyä kattavammin niitä otettaessa käyttöön.

5.4 Katselmoinnit

Jos testausprosessia kehitetään jonkin testausprosessin kehitysmallin mukaisesti, joutuu organisaatio jossain vaiheessa miettimään katselmointeja. Katselmoinneilla pystytään löytämään virheitä ohjelmiston elinkaaren aikaisempien vaiheiden tuotoksista. Näitä ovat käytännössä vaatimus- ja suunnitteludokumentit.

Kehitysmallit korostavat katselmointien tärkeyttä (ks. esim. van Veenendaal ym. 2008 ja Koomen, Pol 1999). Tärkeyden syy on se, että jos virheistä pystytään löytämään edes osa aikaisessa vaiheessa, pystytään pudottamaan tällä tavoin löydettyjen virheiden korjauskustannuksia mahdollisesti murto-osaan (Kuvio 3, sivu 18). Organisaation todellisista dokumentointikäytännöistä riippuen katselmointien käyttöönotto voi aiheuttaa suurtakin vastustusta. Katselmoinnit

saattavat aiempiin käytäntöihin verrattuna vaatia suuren määrän työtä vaatimusten määrittelydokumenttien, suunnitteludokumenttien, käyttötapausten ja testitapausten luojilta. Katselmoinneilla saavutettavat edut ovat kuitenkin niin suuret pitkällä tähtäimellä, jopa 13:1 kustannussäästöt kehityksessä (Gilb 2000), että lisääntyvä työn määrä elinkaaren aikaisessa vaiheessa on helposti perusteltavissa. Katselmoinneilla voidaan saada jo vuoden käytön jälkeen kiinni 50 % aikaisen vaiheen virheistä. Jatkuvalla käytöllä voidaan kiinnijäävien virheiden osuus saada nostettua 90 % yli. (Gilb 2000) Eräs varsin toimivalta vaikuttava menetelmä katselmointien kehittämistä varten on esitelty Gilbin artikkelissa (ks. Gilb 2000).

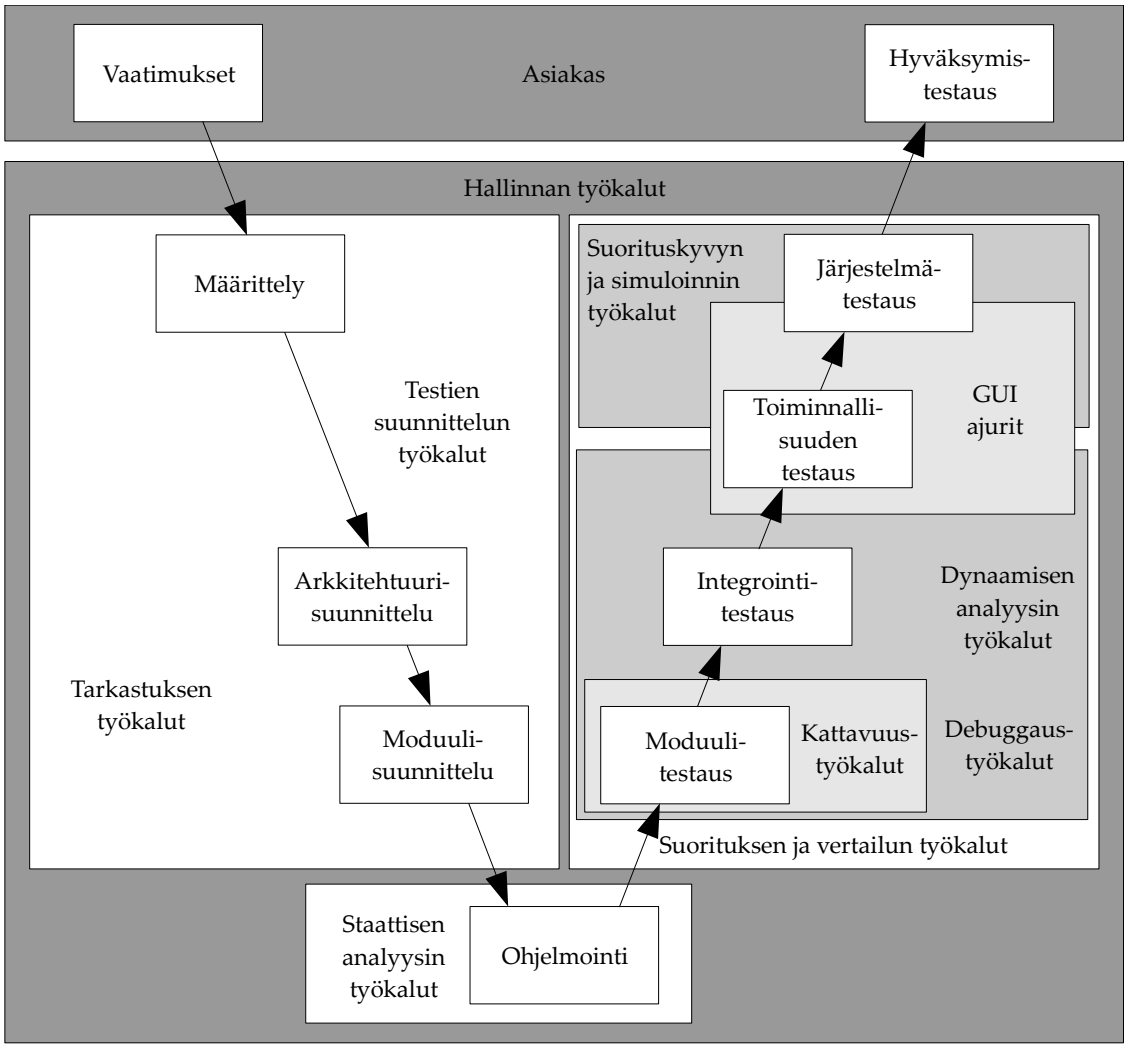
5.5 Testauksen työkalut

Testauksen tueksi on tarjolla suuri määrä erilaisia työkaluja eri tarkoituksiin. Yksi työkalu voi myös vastata toiminnoiltaan useita muita työkaluja, tai olla integroitavissa muihin työkaluihin. Selkeän ja organisaation tarpeita vastaavan työkalukokonaisuuden osien kehittäminen tai valitseminen ja hankkiminen sekä työkalujen hallinta ovat tämän kohdan aiheita.

5.5.1 Testauksen työkaluja

Testauksen työkaluja voidaan ryhmittää usealla tavalla (ks. esim. ISTQB 2007b; Pohjolainen 2003 ja Pol ym. 2002). Tässä käytetään yhdistettynä ja huomattavasti karsittuna ISTQB:n ja TMap:n varsin erilaisia ryhmittelyjä (ISTQB 2007b ja Pol ym. 2002). Testauksen työkalujen kuvauksessa keskitytään kuvaamaan työkaluja, joita yleisimmin käytetään tai jotka ovat testaustoiminnan kannalta olennaisimpia. Esitetyt työkalut eivät siten kata työkalutarjontaa täydellisesti. Eräänlainen V-malliin sovitettu kuvaus

testauksen työkaluista esitetään kuviossa 14 (mukaillen Fewster, Graham 1999 ja Pohjolainen 2003).



Kuvio 14: Testauksen työkalut V-mallissa (mukaillen Fewster, Graham 1999 ja Pohjolainen 2003).

5.5.1.1 Puutteiden hallinta

Puutteiden hallintatyökalut ovat olennainen osa ohjelmiston kehitystä ja testausta. Nämä työkalut hallinnoivat ohjelmiston koko elinkaaren aikana havaittuja virheitä ja puutteita sekä niiden vaatimia toimia. Työkalut kattavat

puutteen koko elinkaaren havainnon kirjaamisesta priorisoinnin ja tehtyjen päätösten kautta mahdollisiin korjaus- tai muutostoimiin. (Pol ym. 2002) Puutteiden hallinnan työkalujen avulla pystytään ohjelmiston puutteista luomaan raportteja esimerkiksi puutteiden kriittisyyden ja perimmäisten syiden perusteella. Raporttien pohjalta voidaan kehitystyötä sekä testausta ohjata keskittämään voimavaroja puutteiden perimmäisten syiden korjaamiseen tai runsaasti kriittisiä virheitä sisältäneiden alueiden testaamiseen.

5.5.1.2 Konfiguraation hallinta

Konfiguraation hallinta on työkalu, joka voi sisältää muun muassa dokumentaation, ohjelmakoodin, testien sekä ympäristön versionhallinnan. Konfiguraation hallinta on testauksen kannalta olennainen työkalu sillä sen kautta voidaan hallita sekä testattava kohde, että sitä testaavat testit ja niiden eri versiot, muutokset ja jopa mahdolliset riippuvuussuhteet (Pol ym. 2002).

5.5.1.3 Testauksen hallinta

Testauksen hallintatyökalu kokoaa usein yhteen työkaluun toimintoja, jotka liittyvät testauksen suunnitteluun, etenemisen hallintaan, puutteiden hallintaan sekä konfiguraation hallintaan. Tällaisen työkalun osa-alueet eivät usein ole yhtä vahvoja, kuin erilliset työkalut. Toimintojen integrointi kuitenkin tuottaa usein säästöjä sekä hankintakuluissa että resurssien käytössä. (Pol ym. 2002)

5.5.1.4 Dynaamisen analyysin työkalut

Dynaamisen analyysin työkalut toimivat tutkimalla ohjelmakoodin suoritusta. Suorituksesta tarkkaillaan usein ensi sijassa ohjelman tai sen osan tuloksia tunnetulla testidatalla. Näitä tuloksia verrataan odotettuihin tuloksiin tai

vaihteluväliin. Dynaamisen analyysin työkaluilla voidaan myös seurata esimerkiksi muistivuotoja. (ISTQB 2007b) Dynaamisen analyysin työkaluihin kuuluvat esimerkiksi yksikkötestauksen, integraatiotestauksen sekä järjestelmätestauksen työkalut ja virheestystyökalut.

5.5.1.5 Staattisen analyysin työkalut

Staattisen analyysin työkalut toimivat analysoimalla ohjelmakoodia suoraan. Ohjelmakoodista voidaan näin havaita riskialttiita ohjelmointitapoja ja koodialueita. Staattisen analyysin työkalut keskittyvät erilaisiin alueisiin ohjelmakoodin analysoinnissa. Ne voivat arvioida esimerkiksi ohjelmakoodin rakennetta, tyyliä tai kokoa. (Pol ym. 2002)

5.5.1.6 Kuormitus- ja rasitustestauksen työkalut

Kuormitus- ja rasitustestauksen työkaluja käytetään simuloimaan suurta käyttäjämäärää, jotta nähtäisiin, miten järjestelmä käyttäytyy rasituksen alla. (Pol ym. 2002) Kuormitus- ja rasitustestaus testaa koko järjestelmän toimintaa, mukaan lukien ohjelmisto, laitteisto ja verkkokomponentit. Kuormitustestaustauksen tavoitteena on selvittää, kuinka korkealla normaalia käyttöä vastaavalla kuormituksella järjestelmä kykenee toimimaan. Testattaessa tarkkaillaan esimerkiksi vasteaikojen pysymistä hyväksyttävissä rajoissa. Rasitustestauksessa pyritään häiritsemään järjestelmän toimintaa ja aiheuttamaan virhetilanteita nostamalla kuormitusta yli odotettujen rajojen. Lisäksi rasitustestauksessa järjestelmää kuormitetaan käyttämällä sitä tavoilla, jotka eivät välttämättä vastaa normaalia käyttöä. (Pressman 2000) Rasitustestauksessa tarkkaillaan muun muassa rajoja, jotka ylitettäessä järjestelmä alkaa toimia virheellisesti sekä virheiden vaikutusta järjestelmän toimintaan. Testattavan järjestelmän tulisi selviytyä virhetilanteista ilman datan

häviämistä tai korruptoitumista. Rasitustestauksella voidaan myös löytää virheitä, jotka eivät normaalikäytössä tai -testauksessa helposti ilmenisi. (Sommerville 1995)

5.5.1.7 Nauhoita ja toista -työkalut

Nauhoita ja toista -työkalut toimivat nauhoittamalla käyttäjän toimia järjestelmässä. Näitä nauhoituksia voidaan myöhemmin käyttää testattaessa järjestelmän toimintaa, esimerkiksi muutoksen jälkeen. Näin ollen nauhoita ja toista -työkalut ovat erinomaisia varsinkin regressiotestausta ajatellen. (Koomen, Pol 1999) Niiden käytössä ja käyttöä suunniteltaessa tulee kuitenkin ottaa huomioon paljon asioita testattavasta ohjelmistosta (ks. Pol ym. 2002).

5.5.2 Työkalujen hankkimisen riskit

Testauksen työkalut voivat tuoda testausprosessille ja koko organisaatiolle selkeitä hyötyjä. Jos työkalujen valintaa ja hankintaa ei toteuteta huolella, voi lopputulos kuitenkin olla jotain muuta kuin odotettiin. Työkalua hankittaessa, saatetaan tehdä esimerkiksi seuraavanlaisia virheitä (ISTQB 2007b):

- Epärealistiset odotukset tai uskomukset työkalulle. Näitä voivat olla esimerkiksi odotukset työkalun sisältämästä toiminnallisuudesta tai luottaminen työkalun helppokäyttöisyyteen ja käytettävyyteen.

- Työkalun käyttöönoton vaatimien resurssien aliarviointi. Käyttöönotto vaatii henkilöstön kouluttamista sekä työkaluun tutustumista.

- Työkalusta saatavien hyötyjen vaatiman ajan ja vaivan aliarviointi. Työkalu saattaa vaatia muutoksia testausprosessiin sekä jatkuvaa työkalun käytön kehittämistä. Lisäksi testauksen automaation

tapauksessa testiskriptien luominen vaatii resursseja niin paljon, että varsinaista hyötyä työkalusta saadaan vasta useiden testiajojen jälkeen.

- Työkalun tuottaman materiaalin hallitsemiseen vaadittujen resurssien aliarviointi.
- Liiallinen usko siihen, että työkalulla voidaan esimerkiksi täysin korvata osia manuaalisesta testauksesta tai testauksen suunnittelusta.

5.5.3 Perusteet työkalujen käytölle

Testausprosessia voidaan tukea sopivilla työkaluilla. Työkaluja käyttämällä voidaan helpottaa ja tehostaa testaustoimintaa sekä nostaa lopputuloksen laatua. Työkaluilla voidaan myös parantaa testausprosessin hallittavuutta, seurantaa ja suunnittelua. (ISTQB 2007a) On kuitenkin tärkeää, että työkaluja ei oteta käyttöön työkalujen itsensä takia. Käyttöön ottamisen täytyy olla perusteltavissa tarpeella sekä odotetuilla hyödyillä. (Koomen, Pol 1999)

Työkalun käyttöä tiettyyn tarpeeseen ja tarkoitukseen voidaan pyrkiä perustelemaan suorittamalla kustannus-hyötyanalyysi harkitulle työkalulle tai työkaluille. Kustannus-hyötyanalyysi tuottaa tulokseksi arvion työkalun käytöstä odotettavista säästöistä tai lisäkustannuksista. Analyysia on mahdollista käyttää apuna myös työkaluja vertailtaessa. Kustannus-hyötyanalyysi suoritetaan ottaen huomioon käytettyjen työtuntien kustannukset, suorat kustannukset ja toistuvat sekä kertaluontoiset kustannukset. Työkalun käyttöönoton ja käytön arvioituja kustannuksia tulee verrata kustannuksiin, jotka syntyvät testauksesta ilman työkalua. (ks. ISTQB 2007b)

5.5.4 Työkalujen valitseminen

Työkaluja valittaessa kannattaa pyrkiä yhtenäiseen ja riittävän kattavaan työkalukokonaisuuteen, jonka osat tukevat toisiaan. Mitä paremmin työkalut integroituvat keskenään, sitä vaivattomampaa niiden käyttäminen yhdessä on. Työkalujen integroituminen myös vähentää virheiden tapahtumista testausprosessissa kun tietoa siirretään työkalujen välillä. (ISTQB 2007b) Integroitumisen tavoittelu rajaa työkaluja niiden valmistajan osalta helposti käytössä olevien työkalujen valmistajiin. Usea työkalu kuitenkin pystyy integroitumaan myös muiden valmistajien tuotteiden kanssa.

Jos on kerralla tarpeen hankkia useampi työkalu, kannattaa ottaa huomioon, kuinka työkalut toimivat yhdessä. Lisäksi on hyvä selvittää, onko tarjolla työkaluja tai työkalupaketteja, jotka sisältävät kaikki tarvittavat toiminnot tai useampia niistä. Työkalujen keskinäinen integrointi ei kuitenkaan saa kriteerinä olla tärkeämpi kuin työkalujen riittävä laatu. Työkaluja hankittaessa on usein pakko tasapainotella ja tehdä kompromisseja muun muassa integroitumisen, laadun ja kustannusten välillä.

Myös ohjelmistokehityksessä käytettävä ohjelmointikieli tai kielet voivat vaikuttaa työkalujen valitsemiseen. Osittain kielen vaikutus johtuu kielen suosiman kehitysympäristön integrointimahdollisuuksista, osittain kielestä itsestään. Esimerkiksi yksikkötestauksen työkaluista osa on vahvasti kielikohtaisia. Alusta, jolle ohjelmistoa kehitetään, vaikuttaa myös testauksen työkalujen valintaan. Perinteisen työpöytäsovelluksen testaustyökaluille asettamat vaatimukset poikkeavat paljon esimerkiksi pda-, puhelin- tai www-pohjaisen sovelluksen vaatimuksista.

Työkalun käyttökohteesta riippumatta voidaan testausta tukevilla työkaluilla ajatella olevan tietyt perustavaa laatua olevat piirteet. Näiden piirteiden avulla

on mahdollista suorittaa työkalujen välistä arviointia. Vertaamalla kahden tai useamman työkalun samaa piirrettä, voidaan selvittää niiden keskinäinen paremmuusjärjestys kunkin piirteen suhteen. Työkalujen piirteet ja niistä vertailtavat asiat ovat (ISTQB 2007b):

- Analysointi: Kuinka hyvin työkalu tukee analysointia, mitä se pystyy analysoimaan ja kuinka se ottaa vastaan tarvittavan informaation.
- Suunnittelu: Onko suunnittelu täysin manuaalista vai tuetaanko sitä automaatiolla.
- Valinta: Valitaanko käsiteltävät kohteet manuaalisesti vai automaattisesti perustuen joihinkin kriteereihin, kuten kattavuuteen.
- Suoritus: Onko työkalun toiminto suoritettava manuaalisesti vai onko se jollain tapaa automatisoitu.
- Arviointi ja esitys, jotka ymmärretään usein suorituksen tietojen kirjaamiseksi ja raportoinniksi. Ovatko nämä toiminnot manuaalisia vai automaattisia, kuten esimerkiksi vertailevia tai tiettyjen kriteereiden mukaan luotuja.

Ehdolla olevien työkalujen valitseminen on siis mahdollista tehdä hyvinkin perusteellisesti. Organisaatiolla ei välttämättä ole mahdollisuutta käyttää työkalujen valitsemiseen resursseja niin paljon, kuin esimerkiksi kattavan kustannus-hyötyanalyysin (ks. luku 5.5.3) tekeminen jokaiselle ehdolla olevalle työkalulle niitä vaatisi. Soveltamalla on kuitenkin mahdollista saada aikaan valintaprosessi, jota käyttämällä saadaan vähitellen rajattua vaihtoehtoja:

1. Varmista, että työkalut vastaavat organisaation tarpeita nyt ja lähitulevaisuudessa.

2. Varmista, että työkalujen käyttöön on saatavilla koulutusta ja tukea.

3. Varmista, että työkalut toimivat olemassa olevien sekä muiden hankittavien työkalujen kanssa hyvin yhteen. Mieti myös tulevaisuutta.

4. Analysoi työkalujen piirteiden vahvuudet ja heikkoudet, kuten yllä on ohjeistettu. Rajaa pois työkalut, jotka vaikuttavat näiden perusteella organisaatiolle sopimattomilta.

5. Vertaa työkalujen kustannus-hyötyanalyysejä (ks. luku 5.5.3).

6. Kokeile työkalua tai työkaluja pilottiprojekteissa.

5.5.5 Työkalujen hankkiminen vs. kehittäminen

Organisaation on hankkimisen sijasta mahdollista myös itse kehittää työkaluja testauksen tueksi. Riippuu täysin organisaatiosta ja tilanteesta, kannattaako näin tehdä. Voidaan sanoa, että jos vain on valmiina saatavilla organisaation testaustarpeita vastaava työkalu, kannattaa ainakin harkita tarkasti, onko organisaatiolle hyötyä kehittää itse vastaavaa työkalua. Tässäkin voi olla apua kustannus-hyötyanalyysistä (ks. luku 5.5.3). Omia työkaluja kehitettäessä on tärkeää ottaa huomioon riittävä ja ymmärrettävä dokumentaatio sekä työkalun toimintojen pitäminen selkeänä, yhtenäisenä kokonaisuutena (ISTQB 2007b).

5.5.6 Huomioitavaa työkaluista

Työkalujen käyttö voi huomattavasti helpottaa testaajien taakkaa ja tehostaa testausprosessia. Tässä listataan muutama testauksen työkalujen käyttöön liittyvä neuvo (ISTQB 2007b):

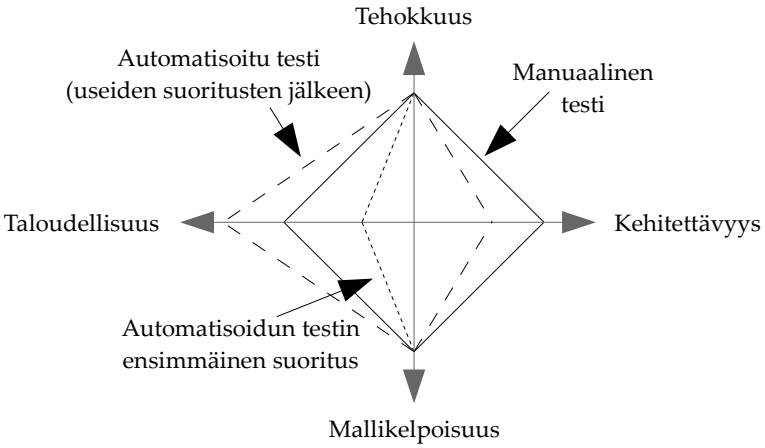
- Ota työkalu käyttöön hallitusti ja vähitellen.

- Päätä ja määrittele työkalun käytölle standardit tavat, esimerkiksi nimeämiskäytännöt.

- Seuraa työkalujen käyttöä ja käytöstä saavutettuja hyötyjä.

5.6 Automatisointi

Usein testausprosessia kehitettäessä tulee vastaan kysymys testauksen automatisoinnista. Testaustoimien automatisoinnilla pyritään lähes poikkeuksetta saavuttamaan joko kustannussäästöjä tai ohjelmiston laadun kasvamista. Usein ei osata ottaa huomioon testauksen automatisoinnin vaatimuksia varsinkaan resurssien kannalta. Automatisointi vaatii alkuun huomattavan paljon aikaa ja vaivaa. Kustannussäästöjä ei tulisi odottaa heti, vaan vasta useiden automatisoitujen testiajojen seurauksena (Kuvio 15). Kustannussäästöjä testauksen automatisoinnista syntyy usein, jos automatisoitavat testitapaukset valitaan huolella. Joitain testityyppejä ei käytännössä ole mahdollista suorittaa ilman automaatiota. Tällainen testityyppi on varsinkin suorituskyky- ja kuormitustestaus. (Fewster, Graham 1999)



Kuvio 15: Automatisoinnin vaikutus testitapauksen mitattaviin atribuutteihin. Vaikutukset rajoittuvat taloudellisuuteen ja kehitettävyyteen. (Fewster, Graham 1999)

Automatisoinnin korkeat aloituskustannukset huomioon ottaen kannattaa testauksen automatisointi aloittaa varovaisesti. Eräs käytännönläheinen tapa ottaa testauksen automatisointi käyttöön organisaatiossa on aloittaa yksikkötestauksen vähittäisellä automatisoinnilla. Kun luodaan uusia yksiköitä tai tehdään muutoksia olemassaoleviin, luodaan samalla yksiköitä vastaavat yksikkötestit. Yksikkötestejä voidaan suorittaa joko aina muutosten jälkeen tai tietyllä aikataululla. Kun yksikkötestejä saadaan luotua ja suoritettua riittävän kattavasti, voidaan niistä kokemusten perusteella valita ja täydentää testitapauksia, joita tullaan käyttämään regressiotestauksessa.

Automatisoitavat testitapaukset tulisi valita huolellisesti. Valinnassa kannattaa keskittyä testitapauksiin, joihin ei ole odotettavissa suuria muutoksia. Automatisoitavia testitapauksia on mahdollista valita usein perustein. Voidaan esimerkiksi pyrkiä valitsemaan testitapauksia, joiden suoritus manuaalisesti on hidasta tai vaikeata sekä testitapauksia, joiden automatisoinnista voidaan saada suurimmat kustannushyödyt.

Testauksen automatisoinnissa on tärkeää huomioida myös testien suorituskyvyn mittarointi. Mittaroinnin perusteella voidaan ryhmitellä ja priorisoida testitapauksia ja niiden suorittamista. Jos tietyllä testillä saadaan usein kiinni virheitä, on sen prioriteetin hyvä olla korkea, jotta sitä ajettaisiin usein. Jos testitapauksia priorisoidaan näin, voidaan pitkäkestoisistakin automatisoiduista testeistä ajaa lyhyemmässä ajassa olennaisimmat ja ne, joilla todennäköisimmin saadaan virheitä kiinni. Muitakin tapoja automaattisesti suoritettavien testitapausten priorisointiin on. Pääasia on, että jos testausta automatisoidaan, on testitapaukset syytä priorisoida jollain hyvin perustellulla menetelmällä, sillä priorisoinnilla voidaan vaikuttaa testauksen tehokkuuteen. (Rothermel, Elbaum 2003)

5.7 Testattavuus ja refaktorointi

Testausta kehitettäessä saattaa ilmetä, että ohjelmisto ei ole yksikkö- tai moduulitestauksen tarpeet huomioiden riittävän modulaarinen, tai että moduulien rajapinnat eivät ole riittävän selvärajaiset. Tällöin ohjelmakoodi ei ole riittävän testattavaa ja saattaa tarvita refaktorointia. Modulaarisuuden puutteisiin voi olla useita syitä, joihin ei tässä perehdytä.

Tässä keskitytään refaktoroinnin kautta toteutettaviin ohjelmiston arkkitehtuuriin tehtäviin muutoksiin. Arkkitehtuurin hienosäätö ohjelmiston elinkaaren alussa on jopa suotavaa (Pressman 2000). Ongelmia arkkitehtuurin muutostarpeista syntyy silloin, kun ohjelmiston kehitys on jo pitkällä tai ohjelma on kenties jo käytössä. Ohjelmakoodin laajamittainen refaktorointi vaatii huomattavan paljon resursseja. Resurssitarvetta tosin voidaan jossain määrin jakaa pidemmälle aikavälille jos refaktorointi voidaan suorittaa vähitellen.

Esimerkiksi www-pohjaisen jatkuvasti kehitettävän ohjelmiston tapauksessa mainitunlainen tarve saattaa testausta kehitettäessä syntyä. Jos ohjelmiston arkkitehtuuriin joudutaan tekemään muutoksia ohjelmiston elinkaaren loppupään vaiheilla, on muutosten tekemiseen käytännössä kaksi vaihtoehtoa. Muutokset voidaan toteuttaa vähitellen refaktoroimalla olemassaolevan sovelluksen koodia modulaarisemmaksi tai ne voidaan toteuttaa kerralla kehittämällä uusi sovellus.

Jos kehitetään uusi sovellus, voidaan olemassaolevaa sovellusta käyttää hyväksi sen vaatimusten määrittelyssä. Lisäksi olemassaolevaa sovellusta voidaan käyttää hyväksi testauksessa. Testausta voidaan suorittaa vertailevana, niin sanottuna back-to-back-testauksena, jossa samat testitapaukset suoritetaan

kummallekin järjestelmälle, ja testien tuloksia verrataan keskenään. Toinen vaihtoehto olemassaolevan järjestelmän käyttämiseen testauksessa on käyttää nauhoita ja toista -työkaluja nauhoittamaan testitapaukset ja odotetut tulokset olemassaolevalla järjestelmällä (Meszaros 2003). Nauhoitetut testitapaukset voidaan sitten suorittaa regressiotestinomaisesti uudella järjestelmällä tarkistaen, että tulokset vastaavat odotuksia.

Jos kehitetään olemassaolevaa sovellusta refaktoroimalla sen rakennetta vähitellen modulaarisemmaksi, voidaan tässäkin käyttää hyväksi nauhoita ja toista -työkaluja yllä kuvatulla tavalla. Testitapaukset nauhoitetaan sovelluksen versiolla, johon ei modularisointia ole tehty, ja niiden tuloksia verrataan modularisoidulla sovelluksella suoritettujen testien tuloksiin. Jos sovellusta modularisoidaan vähitellen, saattaa tästä seurata hieman ylimääräistä työtä, sovelluksen alkuperäisestä arkkitehtuurista riippuen. Jos modularisoitua ohjelmakoodia joudutaan näin käyttämään alkuperäisen ohjelmakoodin kanssa, saatetaan joutua kehittämään väliaikaisia rajapintoja tai kääreitä (*engl. wrapper*), jotta erilaiset arkkitehtuurit pystyvät toimimaan rinnakkain.

Nauhoita ja toista -työkalujen käyttäminen ohjelmiston refaktoroinnin yhteydessä saattaa tapauksesta riippuen olla erittäin käytännöllinen tapa testata sovelluksen versioiden toiminnan yhdenmukaisuutta. Usein kyseiset työkalut kuitenkin vaativat sovelluksen eri versioilta saman tai erittäin samankaltaisen käyttöliittymän toimiakseen luotettavaksi. Jos näitä työkaluja siis käytetään, on syytä huomioida niiden vaatimukset sovelluksen käyttöliittymän suhteen. Se on mahdollisesti pystyttävä pitämään yhdenmukaisena arkkitehtuurin muutoksista huolimatta. (Meszaros 2003)

5.8 Sitouttaminen

Sitouttamisella voidaan saada testausprosessin kehittäminen etenemään helpommin ja pienemmällä vastarinnalla. Jossain määrin sitouttaminen on myös prosessin kehittämisen ehtona. Ilman johdon tukea on prosessin kehittämistä mahdotonta tai erittäin vaikeaa viedä onnistuneesti läpi. Sitoutuminen kehitykseen vähentää tehokkaasti muutosvastarinnan ja kehityksen epäonnistumisen riskiä. (Koomen, Pol 1999) Tässä kuvataan syitä ja tapoja sitouttaa erilaisia ohjelmiston tuotantoon yhteydessä olevia ryhmiä.

5.8.1 Henkilöstön sitouttaminen

Testausta kehitettäessä muun henkilöstön voi olla vaikea ymmärtää, miksi heidän toimintatapojaan ollaan muuttamassa tai miksi heidän pitäisi kouluttautua. Prosessia kehitettäessä voi siis esiintyä muutosvastarintaa, kuten mitä tahansa toimintatapoihin vaikuttavaa muutosta toteutettaessa. Johdon perusteltu tuki voi auttaa sitouttamaan muuta henkilöstöä. Henkilöstölle kannattaa tehdä selväksi testauksen kehittämisestä odotettavat edut, kuten laadukkaammat vaatimukset ja suunnitelmat, kiireessä tehtävien ohjelmiston korjausten väheneminen sekä lopputuotteen laadun koheneminen.

Vaatimusten ja suunnitelmien laadun kasvaminen helpottaa huomattavasti ohjelmiston kehittäjien työtä. Se vähentää kehityksen aiemmista vaiheista johtuvia virheitä ohjelmakoodissa. Lisäksi se vähentää ylimääräisen selvittelyn tarvetta puutteellisten tai selvästi virheellisten vaatimusten ja suunnitelmien tapauksessa. Ohjelmakoodin tuottaminen sujuu helpommin, jos sen pohjana toimiva dokumentaatio on kunnossa.

Kiireisten korjausten väheneminen vaikuttaa positiivisesti kaikkeen tuotekehitykseen yhteydessä olevaan henkilöstöön. Kiirekorjaukset aiheuttavat stressiä, minkä lisäksi ne voivat vaikuttaa häiritsevästi muun tuotekehityksen aikatauluihin ja resursseihin.

Lopputuotteen laadun koheneminen parantaa asiakkaiden kuvaa organisaatiosta ja sen tuotteista. Henkilöstön kannalta tärkeämpiä vaikutuksia ovat kuitenkin todennäköisesti esimerkiksi asiakkaiden vähentyvät negatiiviset yhteydenotot sekä myyntityön ja tuotteiden esittelyn helpottuminen.

5.8.2 Johdon sitouttaminen

Johdon täytyy ymmärtää testauksen tärkeys, oli se sitten koulutuksen tai testausprosessin suunnitteluun ja kehitykseen osallistumisen kautta. Testauksen leikkaaminen aikataulu- tai resurssipaineiden takia kostautuu helposti, ja tämä on tarpeen myös ymmärtää. Jos johto ymmärtää testausta, voi johto myös tehdä järkeviä päätöksiä. Järkeviin päätöksiin voi kuulua myös päätös testauksen leikkaamisesta, mutta tärkeää on, että päätös tehdään perustellusti, ymmärtäen sen seuraamukset. Johdon on myös oman ymmärtämisensä kautta mahdollista perustellusti vaatia muun henkilöstön huomioivan testauksen tarpeet.

5.8.3 Asiakkaan sitouttaminen ja sen riskit

Prosessin kehittämisen aiheuttamien muutosten perustelu voi olla tarpeen myös asiakkaille. Asiakkaille saattaa olla tarpeen perustella, miksi tuotteen kehitys hidastuu väliaikaisesti. Ajoissa ja asiakkaan huomioiden tapahtuvalla viestinnällä voidaan välttää tilanteita, joissa asiakas tuntee tulleensa huijatuksi ja maksavansa tyhjästä.

On mahdollista, että asiakkaan kuva organisaatiosta ja sen toiminnasta jopa kohenee, jos hänelle perustellaan hyvin testausprosessin kehittäminen ja sen aiheuttamat muutokset. Vaarana testausprosessin kehityksen viestimisessä asiakkaalle kuitenkin on, että asiakas odottaa tuotteen laadun kehittyvän heti. On siis hyvä punnita mahdollisia hyötyjä ja haittoja tarkkaan, ennen kuin testausprosessin kehittymisestä viestitään asiakkaalle.

5.9 Koulutus

Testaajien asianmukainen koulutus testausprosessia kehitettäessä on tärkeää. Testausprosessiin on kuitenkin osallisena paljon muutakin henkilöstöä, joiden riittävästä koulutuksesta huolehtiminen on myös syytä muistaa. Tässä käydään läpi testausprosessia kehitettäessä tarvittava erilainen koulutus, ja henkilöt, joiden tulisi koulutukseen osallistua.

5.9.1 Testauskoulutus

Testaajia varten on kehitetty ISTQB:n toimesta kattava opinto-ohjelma, jolla on kaksi tasoa. Tasoista suppeampi, ISTQB Certified Tester Foundation Level Syllabus, kattaa testausalan jo erittäin hyvin. Ohjelman mukaisesti testaajan koulutuksessa huomioidaan muun muassa testien suunnittelu, testausmenetelmät eri vaiheissa ohjelmiston elinkaarta, testauksen työkalut sekä testauksen hallinta ja johtaminen. Jos halutaan varmistaa organisaation testaajien tarvittavien tietojen ja taitojen hallitseminen, on ISTQB:n koulutus ja sertifiointi siihen erittäin toimiva keino.

ISTQB Certified Tester Advanced Level Syllabus on ISTQB:n opinto-ohjelman tasoista kattavampi. Tällä tasolla ohjelma pureutuu testaukseen todella tarkasti. Se syventyy isoon osaan alemman tason aiheista tarkemmin, minkä lisäksi se

huomioi esimerkiksi testauksen automaatiota, testaustiimin kokoamista sekä testausprosessin kehittämistä. Varsinkin testausprosessia kehitettäessä organisaatiolle on kattavamman tason aiheiden käsittelystä hyötyä. Myös testauksen johdolle on nähtävissä hyötyä tarkemman tason opinnoista ja sertifioinnista.

Henkilöstön osaamisen sertifioimisesta on organisaation usein mahdollista hyötyä muutenkin kuin henkilöstön kehittyneen tietotaidon kautta. Organisaation mahdolliset asiakkaat saattavat toivoa tai jopa vaatia tiettyjen sertifikaattien olemassaoloa. Vaikka asiakkaat eivät tällaisia vaatimuksia esittäisikään, on asiakkaiden luottamusta organisaation testaustoimintaan ja tuotteiden laatuun mahdollista kasvattaa viestimällä ISTQB-sertifikaattien olemassaolosta.

5.9.2 Katselmointikoulutus

Katselmointikoulutuksen tavoitteena on laadukkaampi dokumentaatio ja ohjelmakoodi. Kun henkilöstöä koulutetaan katselmointitekniikoissa, saadaan siitä kahdenlaista hyötyä. Henkilöstö oppii katselmoimaan muiden tuotoksia järjestelmällisesti, millä saadaan nostettua muiden tuottaman dokumentaation tai ohjelmakoodin laatua. Lisäksi henkilöstö oppii katselmointitekniikoiden kautta tutkimaan kriittisesti omia tuotoksiaan ja kehittämään niiden laatua oma-aloitteisesti.

5.9.3 Työkalujen koulutus

Testausprosessia kehitettäessä saatetaan valita ja ottaa käyttöön uusia työkaluja testausprosessin tueksi. Testaushenkilöstöä on tarpeen kouluttaa uusien työkalujen käyttöön ja selvittää niiden käyttämisen merkitys. Työkalujen

ottaminen käyttöön ilman koulutusta lähes takaa sen, että kyseisistä työkaluista ei tule organisaatiolle olemaan juurikaan hyötyä. Usein työkalujen kehittäjältä on mahdollista hankkia itse työkalun lisäksi myös työkalun käytön koulutusta sekä tukea. Joidenkin työkalujen käyttämiselle saattaa olla tarjolla myös sertifiointeja.

5.9.4 Testauksen pohjamateriaalin tuottajien koulutus

Testaushenkilöstö ei ole ainoa osa organisaation henkilöstöä, jolle testausta tukevien työkalujen ja testaustapojen tunteminen on hyödyksi. Kaiken henkilöstön, jonka tuotoksia testataan tai testauksessa hyödynnetään, olisi hyvä ymmärtää testauksen tarpeet ja käyttötavat kyseisten tuotosten osalta. Usein tämä tarkoittaa ohjelmakoodin, suunnitelmien ja määritysten tuottajia. Ymmärryksen kautta voi näiden tuotosten tuottaja kehittää ulosantiaan siten, että sen käyttö testauksessa tehostuu.

5.10 Testausprosessin kehittäminen ilman kehitysmallia

Voi olla, että organisaatio ei syystä tai toisesta halua ottaa käyttöön mitään testausprosessin kehitysmallia. Testausprosessin kehittäminen voi silti olla tarpeen. Tällaista tilannetta varten voidaan testausprosessin kehitysmalleista johtaa joukko neuvoja. Neuvojen tavoitteena on selkiyttää testausprosessin kehittämisen yleisiä tavoitteita sekä helpottaa kehitystyötä ja prosessin hallintaa.

- Kehitä prosessia maltillisesti. Liian ison palan haukkaamisesta on vain haittaa prosessin kehittämiselle. Turhan kunnianhimoiset tavoitteet johtavat helposti epäonnistumiseen. Prosessin kehityksen epäonnistuminen voi saada johdon epäilemään myös tulevien

kehityssuunnitelmien onnistumista, jolloin niitä varten voi olla vaikea saada riittävästi resursseja.

- Älä pyri aina uusimpien ja hienoimpien testausmenetelmien ja työkalujen käyttöön. Olemassa olevat työkalut eivät välttämättä tue uusia menetelmiä kunnolla. Uudet työkalut taas saattavat olla vielä liian epäkypsiä laajaa käyttöä ajatellen.

- Älä yritä automatisoida kaikkea. Älä varsinkaan yritä automatisoida kaikkea kerralla. Automatisointi testausprosessin yhteydessä voi pitkällä tähtäimellä säästää testauksen resursseja. Lyhyellä tähtäimellä se kuitenkin huomattavasti lisää resurssien tarvetta. Automatisointi on hyvä aloittaa yksikkötestauksesta, sillä niin voidaan lisätä virheiden kiinnijäämisen todennäköisyyttä heti ohjelmakoodin tuottamisen jälkeen. (ks. luku 5.6)

- Muista pilotoida uusi testausprosessi. Testausprosessin muutosten toimivuuden kokeileminen on erittäin tärkeä vaihe ennen testausprosessin laajamittaista käyttöönottoa. Varsinkin testausprosessia kehitettäessä olisi todella paha virhe olla testaamatta uutta prosessia pilottiprojektilla. Pilotointi tulisi suorittaa valitsemalla yksi tai muutama yrityksen normaalia projektikantaa hyvin edustava, suhteellisen lyhyt ja ei kriittinen projekti. Valituille projekteille otetaan käyttöön uusi testausprosessi ja projektin läpiviennin jälkeen kerätään kokemukset. Testausprosessiin tehdään muutoksia, jos se on tarpeen kerättyjen kokemusten perusteella tai pilottiprojektien aikana.

- Dokumentoi testausprosessi riittävällä tarkkuudella. Testausprosessi on tarpeen dokumentoida, jotta sen läpivientiä voidaan seurata ja poikkeamiin puuttua. Dokumentaation tarkkuuteen on kuitenkin syytä

kiinnittää huomiota, sillä liian tarkka dokumentaatio tekee prosessista jäykän. Liian tarkasta dokumentaatiosta oleellisen asian löytäminen voi myös olla hankalaa. Liian epätarkka dokumentaatio taas pakottaa soveltamaan testausprosessia jatkuvasti, kun toimintamallit eivät ole riittävän hyvin selvillä. Pohjana on hyvä käyttää IEEE:n dokumentointistandardia (ks. IEEE 829-1998), jota kannattaa muokata organisaation ja prosessin tarpeisiin sopivaksi. Hyvin dokumentoitua prosessia on helpompi seurata ja ylläpitää.

- Testausprosessin on tuotettava ja saatava dokumentaatiota riittävällä tarkkuudella. Kaikki oleellinen tulisi olla jollain tasolla dokumentoituna, testaussuunnitelmista testauksen tuloksiin. Testauksen tuloksista voidaan dokumentaatiota tuottaa myös automaattisesti. Pääasia on, että dokumentaatiosta löytyy kaikki testausprosessin seurannan kannalta tärkeä tieto. Testausprosessi myös tarvitsee dokumentaatiota sopivalla tarkkuudella. Testitapauksia luodaan vaatimusten ja suunnitelmien pohjalta, joten näiden on syytä olla dokumentoituna vaaditulla tavalla. Myös testausprosessin tuottamassa dokumentaatiossa on hyvä käyttää organisaation tarpeisiin muokattuna IEEE:n standardia (ks. IEEE 829-1998).

- Seuraa testausprosessin toimintaa ja säädä sitä tarpeen mukaan. Testausprosessin tuottaman dokumentaation ja esimerkiksi testaushenkilöstön palautteen perusteella voi olla tarpeen säätää tai kehittää testausprosessia edelleen. Muutosten tulisi olla perusteltuja ja tähdätä prosessin toiminnan tehostamiseen tai laadun parantamiseen.

- Huolehdi testaajien ja muun testauksen kanssa tekemisissä olevan henkilökunnan riittävästä koulutuksesta (ks. luku 5.9).

- Pyri vakiinnuttamaan testausprosessin läpivienti menetelmäksi, jota noudatetaan myös paineen alla. Ohjelmistotuotannossa tulee aina tilanteita, jolloin on kiire. Kiire ei kuitenkaan saisi hallitsemattomasti heikentää testauksen laatua. (Swinkels 2000) Testausta kannattaa siksi pyrkiä priorisoimaan ja järjestämään siten, että huomioidaan testattavien kohteiden virhealttius, tärkeys ja testaamiseen kuluvat resurssit. Näin voidaan pitää huoli siitä, että jos testaus joudutaan keskeyttämään, on silti testattu ohjelmisto käytettyyn aikaan nähden mahdollisimman hyvin. Testausta ei kuitenkaan tulisi keskeyttää ilman hyvin informoitua päätöstä asiasta, sillä testauksen rajoittaminen ei säästä kustannuksia.

- Suunnittele testausprosessista riittävän joustava, jotta sillä pystytään tehokkaasti testaamaan sekä suuria että pieniä muutoksia. Jos testausprosessi suunnitellaan vain uuden tuotteen testausta silmällä pitäen, se voi olla liian raskas kyseisen tuotteen seuraavan päivityksen tehokkaaseen testaamiseen. (Koomen, Pol 1999)

- Kehitä vaatimusten ja suunnitelmien laatua tarpeen mukaan ja katselmoi ne. Yllä on mainittu, että testausprosessi tarvitsee riittävän tarkat vaatimukset ja suunnitelmat testauksen suunnittelua varten. Lisäksi laadukkaat vaatimukset ja suunnitelmat vähentävät niiden pohjalta myöhemmin kehitysprosessissa syntyviä virheitä. Katselmoinnilla saadaan kiinni virheitä vaatimuksissa ja suunnitelmissa, mikä säästää kustannuksia huomattavasti myöhemmässä vaiheessa ohjelmiston kehittämistä.

- Tärkeitä testauksen kohteita ovat myös ei-toiminnalliset vaatimukset. Tietoturva, tehokkuus, käytettävyys ja ylläpidettävyys vaikuttavat tuotteen laatuun. Lisäksi niihin saattaa kohdistua odotuksia ja

vaatimuksia asiakkaalta tai jopa säädöksiä viranomaistahoilta. Puutteet tai virheet ei-toiminnallisissa vaatimuksissa voivat tulla organisaatiolle erittäin kalliiksi. Ei-toiminnallisia vaatimuksia onkin hyvin perusteltua testata huolella.

- Pyri sisällyttämään katselmoinnit osaksi testausprosessia. Katselmoinneilla voidaan saada kiinni jopa yli 90 % ohjelmiston lähdedokumentaation virheistä hyvin aikaisessa vaiheessa. Virheiden kiinnisaaminen aikaisessa vaiheessa voi johtaa huomattaviin säästöihin virheiden korjauskustannuksissa. (ks. luku 5.4)

- Sitouta testaukseen liittyvä henkilöstö mukaan testausprosessin kehitykseen. Tähän sisältyy testaajien lisäksi johto ja kehittäjät. Sitoutuminen kehitykseen vähentää tehokkaasti muutosvastarinnan ja kehityksen epäonnistumisen riskiä. (ks. luku 5.8)

5.11 Perusteet testausprosessin kehitysmallin käytölle

Ohjelmistotuotanto-organisaatioissa pyritään jatkuvasti kehittämään prosesseja tehokkaammiksi ja laadukkaammiksi. Testaus on prosessina tärkeä, mutta usein aliarvostettu osa ohjelmistotuotantoyrityksen toimintaa. Testausta kehittämällä voidaan kuitenkin saada nostettua koko tuotantoprosessin tehokkuutta ja lopputulosten laatua selkeästi.

Testausprosessin kehittäminen on kuitenkin haastava toimenpide. Kehitystä suunniteltaessa on syytä huomioida paljon prosessiin ja sen toimintaympäristöön liittyviä seikkoja (ks. luku 5.10). Testausprosessin kehitystä voidaan kuitenkin tukea ja ohjata erilaisilla kehitysmalleilla (ks. luku 4). Kehitysmallit auttavat organisaatiota selvittämään testausprosessin kehitystarpeita ja organisoimaan prosessin kehitystä. Lisäksi mallit tarjoavat

välineet prosessin kehityksen seurantaan sekä kypsyyden kommunikointiin ulkoisille sidosryhmille. Organisaation tarpeisiin sopivaa kehitysmallia käyttämällä voidaan myös säästää testausprosessin kehityksen suunnittelussa tarvittavia resursseja.

5.12 Testausprosessin kehitysmallin valitseminen

Tutkielmaan mukaan valituissa testausprosessin kehitysmalleissa on suuria eroja. Esimerkiksi TPI:ssa on 20 avainaluetta ja niillä vaihtelevasti yhdestä neljään kypsyystasoa (Koomen, Pol 1999). TMM:ssa puolestaan on yksinkertaisesti neljä kypsyystasoa (Burnstein ym. 1996a). Rakenteellisesti ja sisällöllisesti ero TMM:n ja TPI:n välillä on valtava.

Mallien järjestäminen yleisesti hyväksyttävällä tavalla ei ole mahdollista, vaan mallien keskinäinen paremmuusjärjestys on lähes täysin organisaatiokohtaista. Testausprosessin kehitysmalleja on myös olemassa huomattavasti tutkimuksessa esiteltyä neljää mallia enemmän. Seuraavaksi annetaan joitain yleispäteviä ohjeita mallin valitsemiseen. Ohjeisiin on liitetty esimerkkejä tutkimuksessa mukana olleista malleista.

- Mieti, onko organisaatiolla tarvetta CMMI:n mukaiseen kypsyystasoluokitukseen testausprosessin osalta. Jos on, valitse kehitysmalli, jossa tämä on mahdollista. CMMI:n tueksi on kehitetty esimerkiksi TMMi-malli (van Veenendaal ym. 2008).

- Jos organisaatio haluaa kehittää testausprosessia mahdollisimman hyvin omien prioriteettien mukaan, valitse jatkuva kehitysmalli. Jatkuvia kehitysmalleja on esimerkiksi TPI ja TIM. Lisäksi suunnitteilla on TMMi:iin jatkuva kehitysmalli (TMMi Foundation 2008).

- Jos organisaatio tarvitsee kehitysmallin, jossa on mahdollisimman vähän työtä kypsyystasoa kohti, kannattaa valita mahdollisimman hienojakoinen jatkuva kehitysmalli. Esimerkiksi TPI on hienojakoisempi malli kuin TIM.

- Jos organisaation tarvitsee viestiä testausprosessin kypsyystasosta ulkoisille sidosryhmille, on hyvä valita malli, joka tarjoaa tähän yksinkertaisen ratkaisun. Tällaisia malleja ovat yleisesti asteittaiset kehitysmallit. Asteittaisia kehitysmalleja ovat esimerkiksi TMMi ja TMM. Lisäksi jotkin jatkuvat mallit, kuten TPI, voivat tarjota mahdollisuuden ryhmitellä kypsyystasoja ja avainalueita ymmärrettävämmin (Koomen, Pol 1999).

- Selvitä kattaako harkittu kehitysmalli organisaation tarpeelliseksi katsomat osat organisaation toiminnasta. Selvitä myös, kattaako malli liian suuren osan organisaation toiminnasta. Mitä tarkemmin malli istuu organisaation kehitystarpeisiin, sitä parempi. Tutkimuksessa esitellyt neljä mallia kattavat jokainen organisaation toimintaa hieman eri tavalla. Näistä TPI ja TMMi ovat huomattavasti kattavampia kuin TIM ja TMM. TIM on puolestaan TMM:ää kattavampi.

- Jos harkittu malli on edelleen kehityksessä, varmista, että mallin valmistumisesta on takeita. On myös syytä ottaa selvää mallin kehityksen suunnasta, sillä se voi olla ainoa tapa varmistaa kehityksessä olevan mallin soveltuminen organisaation tarpeisiin myös tulevaisuudessa.

- Selvitä muut organisaation testausprosessissa käytetyt mallit ja ohjeistot. Testausprosessin kehittämismallin valitseminen siten, että mallit tukevat ja eivät rajoita toistensa käyttöä voi helpottaa prosessin kehittämistä.

Esimerkiksi TPI tukee rakenteisen testausprosessin TMap-mallia (Koomen, Pol 1999) ja TMMi tukee CMMI:ia, joka on prosessinkehitysmalli (van Veenendaal ym. 2008).

- Tarkista mahdollisimman aikaisessa vaiheessa, onko harkinnassa olevalle mallille saatavissa riittävä dokumentaatio. Jos malliin perustuvia kirjoja on myynnissä, tätä voidaan pitää hyvänä merkkinä. Esimerkiksi TIM:lle ei ole helposti saatavilla arviointimenetelmän dokumentaatiota ja TMM:lle on vaikea saada mallin täydellistä kuvausta. Myös TMMi on toistaiseksi keskeneräinen, mutta kehityksen alla. TMMi:n taso 2 on kuitenkin jo täysin kuvattu. Esitellyistä malleista kirjoja on saatavilla vain TPI:stä.

- Mahdollisimman aikaisessa vaiheessa on myös syytä selvittää muiden organisaatioiden kokemuksia harkinnassa olevasta mallista.

- Ennen kaikkea on hyvä tutustua useaan lupaavalta vaikuttavaan malliin.

6 YHTEENVETO

Tutkielmassa esiteltiin testausprosessin kehitystä sekä yleisesti, että testausprosessin kehitysmalleja apuna käyttäen. Testausprosessin kehitysmallien käytölle nähdään useita hyviä perusteita. Tärkeimpiä perusteita ovat:

- Valmis kartta, jonka mukaan testausprosessin kehitys voi suunnistaa.
- Kustannussäästöt testausprosessin kehityksen suunnittelussa.
- Mahdollisuus formaaliin testausprosessin kypsyystason arviointiin.

Yllä mainitut perusteet eivät kenties vaikuta kovin ihmeellisiltä. Kuitenkin, kun otetaan huomioon se kehityskohteiden määrä, joka testausprosessia onnistuneesti kehitettäessä on huomioitava, saadaan huomattavaa etua kehitysmallien tarjoamasta opastuksesta.

Tarkastellut testausprosessin kehitysmallit erosivat toisistaan rakenteeltaan ja jossain määrin myös sisällöltään. Malleissa otetaan kuitenkin huomioon valtaosin samoja asioita, vaikkakin joissain tapauksissa muista poikkeavasta näkökulmasta. Kehitysmallit tuntuvat korostavan tiettyjä kehityskohteita. Näitä ovat katselmoinnit ja testauksen integrointi koko ohjelmistotuotantoprosessiin, testaustoiminnan optimointi, mittarointi ja metriikat sekä testauksen automatisointi. Jos ajatellaan ohjelmistotuotantoprosessin yhtä iteraatiota, huomataan, että kehitysmallit pyrkivät tuomaan testausta sen sisällä jatkuvasti aiempaan vaiheeseen katselmointien muodossa.

6.1 Rajoitteita ja kehitettävää

Tutkielma toi esiin useita testausprosessin kehityksessä ja sen suunnittelussa huomioon otettavia asioita. Käsitellyt asiat ovat kuitenkin vain osa siitä, mitä tutkielmassa olisi voitu käsitellä. Valitettavasti rajoitukset käytettävissä olevien resurssien suhteen pakottivat jonkin verran rajaamaan käsittelyä. Eräs alue, jonka puuttuminen luvusta 5 aiheuttanee ihmetystä, on testausmenetelmät. Niiden pois rajaamiseen oli kuitenkin syynsä. Testausmenetelmiä on paljon, minkä lisäksi testaukseen perehtynyt henkilö normaalisti tuntee useita testausmenetelmiä. Menetelmien kuvaaminen ja käytön selvittäminen ei siis olisi ollut tarkoituksenmukaista. Testausmenetelmistä olisi kuitenkin ollut paljon sanottavaa. Olisikin toivottavaa, että testausprosessin kehittämisestä myös testausmenetelmien näkökulmasta tehtäisi tutkimusta. Myös tätä tutkimusta järjestelmällisempi aihealueen tarkastelu voisi olla paikallaan. Materiaalia riittäisi helposti kirjaksi asti, sillä alue on laaja.

Tutkielmassa pyrittiin keskittymään testausprosessin kehittämiseen tarkastelemalla vaikutuksia organisaatioon ja sen henkilöstöön. Tässä onnistuttiin osittain. Organisaation ominaisuuksiin ei kuitenkaan voitu ottaa kantaa, sillä kyseessä ei ollut case-tutkimus. Tästä johtuen organisaatioon itseensä liittyvät aiheet on kuvattu hyvin yleisellä tasolla, eikä kaikkiin luvussa 4 esitettyihin kysymyksiin ole saatu vastauksia.

6.2 Kysymyksistä ja vastauksista

Valitettavasti kaikkiin luvussa 4 esitettyihin kysymyksiin ei pystytty löytämään objektiivisia, perusteltuja vastauksia. Näitä ovat kysymykset, jotka liittyvät organisaation kykyyn tai haluun ottaa käyttöön jokin tietty testausprosessin

kehitysmalli. Seuraavassa esitetään kommentoitu lista kysymyksistä, joihin ei luvussa 5 esitetty vastausta:

- *Tarjoaako käyttöön valittu kehitysmalli haettuja etuja riittävän nopeasti?*
  Kysymykseen vastaaminen vaatii organisaatiolta riittävää tutustumista kehitysmalliin, jotta sen etenemistä voidaan arvioida organisaation omien kykyjen ja tarpeiden pohjalta.

- *Pystyykö organisaatio suunnittelemaan prosessin kehitystä seuraavalle tasolle asti?*
  Jotta kysymykseen pystytään vastaamaan, täytyy olla riittävä tuntemus organisaation resursseista sekä resurssien käytöstä. Jos organisaation resurssien käyttö on riittävän tasaista, ja toiminta niin suunnitelmallista, että organisaatio pystyy tekemään luotettavia usean vuoden kattavia suunnitelmia, vastaus on: ”kyllä”.

- *Pystyykö organisaatio varaamaan riittävästi resursseja testausprosessin kypsyyden seuraavalle tasolle nostamista varten?*
  Katso edellisen kysymyksen vastaus. Suunnittelu ja resurssien varaaminen ovat normaalissa tilanteessa selkeästi kytköksissä.

- *Pystytäänkö kehitysmallin mukainen testausprosessin nykytilan arviointiprosessi viemään luotettavasti läpi organisaation resursseilla?*
  Vastaus tähän kysymykseen riippuu jossain määrin organisaation resursseista ja resurssien käytöstä. Paljon riippuu myös käytetystä testausprosessin kehitysmallista ja sen arviointimenetelmistä. Jos tarjolla on useampi arviointimenetelmä, on vastaus todennäköisesti kiinni organisaation arvioinnin laadun tarpeesta.

- *Koetaanko arviointiprosessi organisaation toimintaa häiritseväksi?*

Testausprosessin kehitysmallien arviointimenetelmät ovat erilaisia. Luotettavia tuloksia antavat menetelmät tarkkailevat organisaation toimintaa varsin läheltä ja joissain tapaukissa myös pitkän aikaa. Luotettavat arviointimenetelmät ovat siis luonnostaan jossain määrin organisaation toimintaa häiritseviä. On kuitenkin useita organisaatioon liittyviä tekijöitä, jotka voivat vaikuttaa siihen, miten häiritseväksi arviointiprosessia koetaan. Nopeat ja kevyet arviointiprosessit eivät todennäköisesti häiritse organisaation toimintaa, mutta niiden tulokset eivät ole kovin luotettavia.

- *Toimiiko arviointiprosessi riittävän nopeasti?*

  Tähän kysymykseen tietää vastauksen vain organisaatio, joka on testausprosessin kehitysmallia ottamassa käyttöön. On kuitenkin varmasti totta, että puoli vuotta kestävä arviointiprosessi TMM-AM:n tapaan on usean organisaation mielestä liian hidas.

- *Voidaanko arviointiprosessin tuloksiin varmasti luottaa?*

  Lyhyt vastaus kysymykseen on: ”ei”. Arviointiprosessin tulokset riippuvat täysin organisaatiosta. Jos organisaatiossa on riittävästi pyrkimystä ”huijata” arviointiprosessia, se varmasti myös onnistuu. Hyötyä tällaisesta toiminnasta ei pitkällä tähtäimellä todennäköisesti ole. Mitä kattavampi arviointiprosessi, sitä luotettavammat sen tulokset yleensä ovat.

LÄHDELUETTELO

ANSI/IEEE 1008-1987, 1987. *IEEE Standard for Software Unit Testing*. IEEE.

Beizer B., 1990. *Software Testing Techniques*. New York, NY, USA: Van Nostrand Reinhold Company.

Boehm B., 1976. *Software engineering*. IEEE Transactions on Computers, Volume C-25, Issue 12. Los Alamitos, CA, USA: IEEE Computer Society Press, 1226-1241.

Boehm B., 1981. *Software Engineering Economics*. Upper Saddle River, NJ, USA: Prentice Hall PTR.

Boehm B., 1988. *A Spiral Model of Software Development and Enhancement*. Computer, Volume 21, Issue 5, 61-72.

Boehm B., 2006. *A View of 20th and 21st Century Software Engineering*. International Conference on Software Engineering. New York, NY, USA: ACM Press, 12-29.

Burnstein I., Homyen A., Grom R., Carlson C., 1998. *A Model to Assess Testing Process Maturity*. Crosstalk - The Journal of Defense Software Engineering, November 1998.

Burnstein I., Homyen A., Suwannasart T., Sazena G., Grom R., 1999. *A Testing Maturity Model for Software Test Process Assessment and Improvement*. Software Quality Professional, Volume 1, Number 4, 8-21.

Burnstein I., Suwannasart T., Carlson R., 1996a. *Developing A Testing Maturity Model For Software Test Process Evaluation And Improvement*. Proceedings of the

IEEE International Test Conference on Test and Design Validity. Washington, DC, USA: IEEE Computer Society Press, 581-589.

Burnstein I., Suwannasart T., Carlson R., 1996b. *Developing a Testing Maturity Model: Part I*. Crosstalk - The Journal of Defense Software Engineering, August 1996.

Burnstein I., Suwannasart T., Carlson R., 1996c. *Developing a Testing Maturity Model: Part II*. Crosstalk - The Journal of Defense Software Engineering, September 1996.

Chrissis M., Curtis B., Paulk M., Weber C., 1993. *Capability Maturity Model for Software, Version 1.1*. Pittsburgh, PA, USA: Carnegie Mellon University, Software Engineering Institute (CMU/SEI).

CMMI Product Team, 2006. *CMMI® for Development, Version 1.2*. Pittsburgh, PA, USA: Carnegie Mellon University, Software Engineering Institute (CMU/SEI).

Cohen C., Birkin S., Garfield M., Webb H., 2004. *Managing conflict in software testing*. Communications of the ACM, Volume 47, Issue 1, 76-81.

Ericson T., Subotic A., Ursig S., 1997. *TIM - A Test Improvement Model*. Software Testing, Verification and Reliability, Volume 7, Number 4, 229-246.

Fewster M., Graham D., 1999. *Software Test Automation: Effective use of test execution tools*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co.

Gelperin D., 1996. *A Testability Maturity Model*. 5th International Conference on Software Testing Analysis & Review, Orlando, Florida, May 13-17, 1996. USA: Software Quality Engineering.

Gelperin D., Hetzel B., 1988. *The Growth of Software Testing*. Communications of the ACM, Volume 31, Number 6, 687-695.

Gilb T., 2000. *Planning to Get the Most Out of Inspection*. Software Quality Professional, Volume 2, Number 2, 7-19.

Homyen A., 1998. *An Assessment Model to Determine Test Process Maturity*. Chicago, IL, USA: Illinois Institute of Technology, Väitöskirja.

IEEE 1012-1998, 1998. *IEEE Standard for Software Verification and Validation*. IEEE.

IEEE 829-1998, 1998. *IEEE Standard for Software Test Documentation*. IEEE.

ISO/IEC 15504-2:2003, 2003. *Information technology -- Process assessment -- Part 2: Performing an assessment*. ISO/IEC.

ISO/IEC 15504-3:2004, 2004. *Information technology -- Process assessment -- Part 3: Guidance on performing an assessment*. ISO/IEC.

ISTQB, 2007. *Certified Tester Foundation Level Syllabus, Version 2007*. International Software Testing Qualifications Board.

ISTQB Advanced Level Working Party, 2007. *Certified Tester Advanced Level Syllabus, Version 2007*. International Software Testing Qualifications Board.

ISTQB Glossary Working Party, 2007. *Standard glossary of terms used in Software Testing, Version 2.0*. International Software Testing Qualifications Board.

Koomen T., Pol M., 1998. *Improvement of the test process using TPI*. Sogeti Nederland B.V.

Koomen T., Pol M., 1999. *Test Process Improvement: A practical step-by-step guide to structured testing*. Boston, MA, USA: ACM Press/Addison-Wesley Longman Publishing Co., Inc.

Meszaros G., 2003. *Agile regression testing using record & playback*. Companion of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications. New York, NY, USA: ACM Press, 353-360.

Osterweil L., 1987. *Software processes are software too*. Proceedings of the 9th international conference on Software Engineering. Los Alamitos, CA, USA: IEEE Computer Society Press, 2-13.

Paulish D., Carleton A., 1994. *Case studies of software-process-improvement measurement*. Computer, Volume 27, Issue 9, 50-57.

Pohjolainen P., 2003. *Testauksen automatisointi ja sen työkalut*. Testauspäivä 2003 seminaarimateriaali. Kuopio, Finland: Kuopion yliopisto, Tietojenkäsittelytieteen laitos, PlugIT-Teho.

Pol M., Teunissen R., van Veenendaal E., 2002. *Software Testing: A Guide to the TMap Approach*. Boston, MA, USA: Addison-Wesley Professional.

Pressman R., 2000. *Software Engineering: A Practitioner's Approach 5th Edition, European Adaptation*. London, England: McGraw-Hill.

Rothermel G., Elbaum S., 2003. *Putting Your Best Tests Forward*. IEEE Software, Volume 20, Issue 5, 74-77.

Royce W., 1970. *Managing the development of large software systems: concepts and techniques*. Proceedings of IEEE WESCON. USA: TRW Inc., 1-9.

Sommerville I., 1995. *Software Engineering 5th edition*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc.

Suwannasart T., 1996. *Towards the Development of a Testing Maturity Model*. Chicago, IL, USA: Illinois Institute of Technology, Julkaisematon väitöskirja.

Swinkels R., 2000. *A comparison of TMM and other Test Process Improvement Models*. MB-TMM Project Report. Frits Philips Institute.

TMMi Foundation Home [online]. TMMi Foundation [viitattu 26.5.2008]. Saatavilla www-muodossa <http://www.tmmifoundation.org/>.

van Veenendaal E., 2006. *Guidelines for Testing Maturity*. STEN Journal, Volume IV, 2006.

van Veenendaal E., Goslin A., Olsen K., O'Hara F., Miller M., Thompson G., Wells B., 2008. *Test Maturity Model Integration (TMMi) version 1.0*. TMMi Foundation.

Wells B., Weller N., Goslin A., Olsen K., O'Hara F., Miller M., Thompson G., van Veenendaal E., 2008. *TMMi (Version 1.0) Assessment Method Application Requirements (TAMAR)*. TMMi Foundation.