

SAIMAAN AMMATTIKORKEAKOULU
Tekniikka Lappeenranta
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka

Laura Sainio

OHJELMISTOTESTAUKSEN MENETELMÄT JA TYÖVÄLINEET

Opinnäytetyö 2010

TIIVISTELMÄ

Laura Sainio

Ohjelmistotestauksen menetelmät ja työvälineet, 39 sivua, 2 liitettä

Saimaan ammattikorkeakoulu, Lappeenranta

Tekniikka, tietotekniikan koulutusohjelma

Ohjelmistotekniikka

Opinnäytetyö, 2010

Ohjaaja: lehtori Martti Ylä-Jussila

Tämän opinnäytetyön tavoitteena on ollut tutkia ohjelmistotestauksen merkitystä, sen työmenetelmiä ja nykyaikaisessa ohjelmistotestauksessa käytettäviä työvälineitä. Tutkimuksen perusteella on ollut tarkoitus tuottaa Saimaan ammattikorkeakoulun tietotekniikan koulutusohjelman käyttöön soveltuvaa opetusmateriaalia.

Ohjelmistotestauksen merkitys ohjelmistotuotannossa on korostunut, koska yhteiskunta on tullut entistä riippuvaisemmaksi tietojärjestelmistään, ohjelmistojen käytettävyys- ja luotettavuusvaatimukset ovat kasvaneet, ohjelmistorakenteet ovat monimutkaistuneet, ohjelmistojen koot ovat kasvaneet ja lisäksi tietojärjestelmät ovat integroituneet toisiinsa. Ohjelmistotuotanto on globalisoitunut kansainvälisten yritysintegraatioiden ja toteutustyön ulkoistamisen kautta. Runsaasti ihmistyötä vaativa ohjelmointityö tehdään siellä, missä sen teettäminen on edullisinta, ja kehittyneisiin maihin on jäämässä vain asiakasta lähellä oleva ohjelmistojen suunnittelu- ja määrittelytyö sekä projektien johtaminen. Ohjelmistoprojekteissa, jotka usein toteutetaan monikulttuuristen virtuaalitiimien voimin, nämä työtehtävät vaativat erityisosaamista, jota Suomessa ei ole riittävästi. Tästä syystä tietotekniikan opetuksen painopistettä on muutettu niin, että ohjelmistosuunnittelu ja laadunvalvonta korostuvat. Saimaan ammattikorkeakoulun tietotekniikan opetusohjelmaa on uudistettu ammattiaineiden osalta siten, että ohjelmistosuunnittelun sekä testauksen ja laadunvalvonnan osuutta on lisätty ja suuntautumisvaihtoehtoihin on lisätty erikoistumisvaihtoehdot.

Tehdyn tutkimuksen perusteella luotiin Saimaan ammattikorkeakoululle Ohjelmiston testauksen perusteet opintojaksolle opintomateriaali, joka sisältää lukumateriaalin (178 sivua) sekä diasarjat luentoja varten (256 sivua), jotka ovat opinnäytetyön liitteinä 1 ja 2.

Asiasanat: ohjelmistotestaus, ohjelmistotuotanto, ohjelmistotekniikka

ABSTRACT

Laura Sainio

Tools and techniques of software testing, 39 pages, 2 appendices

Saimaa University of Applied Sciences, Lappeenranta

Technology, Degree Programme in Information Technology

Software engineering

Bachelor thesis, 2010

Instructor: lecturer Martti Ylä-Jussila

The aim of this thesis was to study the importance of software testing and the working methods, techniques and tools used in modern software testing. The study was designed to produce teaching material for the Degree Programme in Information Technology at the Saimaa University of Applied Sciences.

The role of the software testing in the software engineering has become more important, because society has become increasingly dependent on information systems, software usability and reliability requirements have increased, the software structures are more complex, the software systems have grown and the information systems are integrated with each other. Software engineering has been globalized through international company integration and implementation outsourcing. Programming, which requires lots of manual labor, is made where it can be done most cheaply and the only work left in developed countries is software design and requirement and project management work, which is made close to the customer. In software projects, which are often done in multi-cultural virtual teams, this work requires special skills and there is not enough skills within this area in Finland. For this reason, the focus of IT studies has been changed to emphasize software design and quality control. The Information Technology curriculum at the Saimaa University of Applied Sciences has been changed so that there are more courses in software design, software testing and quality control. The Information Technology curriculum at the Saimaa University of Applied Sciences has been changed so that there are more courses in software design, software testing and quality control.

The teaching material for the course "Ohjelmiston testauksen perusteet" at Saimaa University of Applied Sciences was created based on this study. The material, which consists of reading materials and slides, can be found in Appendices 1 and 2.

Keywords: software testing, software engineering

SISÄLTÖ

TIIVISTELMÄ.....	2
ABSTRACT.....	3
1 JOHDANTO	7
1.1 Lähtökohdat.....	7
1.2 Opinnäytetyöraportin rakenne	9
2 ASIAKKAAN TOIMINNAN KUVAUS.....	9
2.1 Saimaan ammattikorkeakoulu	9
2.2 Tietotekniikan koulutusohjelma	11
3 OHJELMISTOTESTAUS	14
3.1 Historia	14
3.2 Näkökulmia testaukseen	15
3.3 Testaus osana perinteistä ohjelmistotuotantoa	17
3.4 Testauksen käytännöt ja tekniikat	20
3.5 Testauksen hallinta	21
3.6 Ketterät menetelmät ja testaus	23
3.7 Oliotestaus	24
3.8 Testauksen automatisointi.....	27
3.9 Testaustyökalut	29
4 TYÖN KULKU	32
5 LOPPUTULOKSET	33
6 POHDINTA	34
KUVAT	36
LÄHTEET	37

LIITTEET

Liite 1 Opintomateriaali

Liite 2 Diasarjat (vain cd:llä)

TERMIT JA LYHENTEET

GUI	Graphic User Interface. Graafinen käyttöliittymä.
IEEE	Institute of Electrical and Electronics Engineers. Maailman suurin tekniikan alan järjestö, jonka toimintaan kuuluu muun muassa tekniikan alan julkaisutoiminta, standardien luominen sekä konferenssi- ja koulutustoiminta.
IEEE 829	IEEE Standard for Software Test Documentation, versiot vuosilta 1983, 1998 ja 2008. Testausdokumentaation standardi.
ISTQB	International Software Testing Qualifications Board. Tutkintolautakunta, jonka tehtävänä on kehittää kansainvälinen testaajien sertifiointijärjestelmä.
Katselmointi	Ohjelmiston osien tai projektin tilan arviointi, jonka tarkoitus on tunnistaa tuotosten eroavuudet suunnitelmiin nähden sekä tuottaa kehitysehdotuksia.
Ketterä kehitys	Agile development. Nopeita ja muutoksiin nopeasti vastaavia ohjelmistokehitysmenetelmiä tarkoittava termi.
Mock-objekti	Korvikeolio, jota voidaan käyttää olioiden rajapintoja testatessa oikean olion sijasta.
TDD	Test Driven Development t. Test Driven Design. Yksikkötestauksen menetelmä, jossa testit kirjoitetaan ennen ohjelmakoodia.

Tutkiva testaus

Exploratory testing. Kokemusperäinen testaustekniikka, jossa käytetään ihmisen tietämystä ja kokemusta testitapausten määrittämiseksi.

Ketterissä menetelmissä tutkivalla testauksella kutsutaan kokonaista testausprosessia, jossa prosessin rakenne perustuu tarpeeseen suunnata testausta sen mukaan, mitä testausprosessissa opitaan testattavasta kohteesta.

V&V Verifiointi ja validointi, ohjelmistotuotannon työvaihe, jossa varmistetaan, että ohjelmisto täyttää sille asetetut vaatimukset ja tarpeet.

V-malli Ohjelmistoprosessin elinkaarimalli, jossa testaus on integroitu kaikkiin prosessin osavaiheisiin.

Vesiputousmalli

Perinteinen ohjelmistoprosessin elinkaarimalli.

XP Extreme Programming. Eräs ketterän ohjelmistokehityksen menetelmistä.

Ääritestaus

XP-perustainen yksikkötestausmenetelmä, jossa testit laaditaan TDD:n avulla.

1 JOHDANTO

Opinnäytetyön tarkoituksena on tutkia ohjelmistotestauksen nykytilaa ja sen menetelmiä ja työkaluja. Työ on tehty tutkimalla alan kirjallisuutta ja www-julkaisuja sekä kokeilemalla työmenetelmiä ja apuvälineitä oppilaitoksella ja käytännössä ohjelmistoalan yrityksessä.

1.1 Lähtökohdat

Nyky-yhteiskunnassa tietojärjestelmiä on kaikkialla ja niistä ollaan entistä riippuvaisempia. On selvää, että esimerkiksi julkishallinnon ja terveydenhuollon tietojärjestelmien on oltava varmatoimisia, luotettavia ja tietoturvallisia, mutta myös kaupallisilta ohjelmistotuotteilta vaaditaan samaa. Laatu tuo kilpailuetua: huonosti toimivat ja epävakaat ohjelmistot eivät mene kaupaksi, ja laatuongelmat karkottavat asiakkaat kilpailevan tuotteen pariin. Tästä syystä ohjelmistojen testaamiseen ja laatuksymyksiin on alettu kiinnittämään huomiota entistä enemmän.

Laadukkaan ohjelmiston tuottaminen ei ole helppoa. Syyt tähän löytyvät osittain jo ohjelmistotuotannon luonteesta. Tekniikan alana ohjelmistotuotanto on nuori ja kehityksessään keskeneräinen. Lisäksi se poikkeaa muista tekniikan aloista erityispiirteillään, kuten äärimmäisellä monimutkaisuudella: ohjelmistot ovat usein laajoja ja modulaarisia, koostuen mahdollisesti useista eri aikakausina ja eri ohjelmointikielillä toteutetuista osista, joiden fyysiset toiminta-alustat poikkeavat toisistaan. Varsinkin järjestelmäintegraatiot tuottavat monimutkaisia systeemejä. Myös ohjelmistotyön näkymättömyys, ohjelmistojen näennäisen helppo muunneltavuus, ohjelmistoprojektien ainutkertaisuus ja se, että työvälineet ja –menetelmät muuttuvat jatkuvasti, aiheuttavat omat hankaluutensa (Haikala, Märijärvi. 2000, s.15-17). Ohjelmistoprojektia aloitettaessa voidaan olla tilanteessa, jossa ratkaistava ongelma-alue tai välineet ja menetelmät, joita on tarkoitus ratkaisussa käyttää, ovat projektiryhmälle entuudestaan täysin tuntemattomia. Kun tähän lisätään vielä se, että henkilökohtaiset erot ihmisten tuottavuudessa ovat suuria, on työmäärien ja projektin valmiusasteen määrittelemi-

nen haasteellista. Tämä johtaa siihen, että luotettavia aikatauluja on vaikea tehdä.

Nykypäivän ohjelmistoprojektit ovat usein laajoja ja monikansallisia. Yritysin-tegratioiden kautta muodostuneiden, monella mantereella toimivien yritysten lisäksi ohjelmistotuotannon kalleus on aiheuttanut yksinkertaisen toteutustyön ulkoistamista halvan työvoiman maihin, kuten Intiaan. Kun ohjelmiston suunnittelijan ja toteuttajan fyysinen ja kulttuurinen välimatka ovat pitkiä, vaatii se projektinhallinnasta, suunnittelusta ja laadunvarmistuksesta vastaavilta henkilöiltä erityistaitoja. Kun monimutkaista teknistä ongelmaa ratkotaan monikulttuurisen virtuaaliitiimin voimin, on selvää, että toimintaprosessien kypsyys on elinehto toiminnan onnistumiselle.

Alan erityispiirteiden ongelmallisuuden, järjestelmien monimutkaisuuden ja globalisaation ongelmien aiheuttamia riskejä ohjelmistotuotannossa pyritään hallitsemaan kiinnittämällä erityishuomiota ohjelmistojen suunnitteluun, laadunvarmistukseen ja ohjelmistotestaukseen. On ensiarvoisen tärkeää saada ohjelmistotuotannon prosessit selkeiksi ja ennustettaviksi, kun ala itsessään on sen luonteista, että yllättävät ongelmat ovat enemmän sääntö kuin poikkeus.

Työn lähtökohtana on Saimaan ammattikorkeakoulun pyrkimys vastata työelämän tarpeisiin ohjelmistotekniikan alalla. Toimivien prosessien ja laadukkaiden ohjelmistotuotteiden aikaansaamista voidaan parantaa opettamalla tulevia ohjelmistoalan ammattilaisia kiinnittämään erityishuomiota kyseisiin asioihin. Saimaan ammattikorkeakoulun tietotekniikan opetusohjelmaa on uudistettu ammat-tiaineiden osalta siten, että ohjelmistosuunnittelun sekä testauksen ja laadunvalvonnan osuutta on lisätty ja suuntautumisvaihtoehtoihin on lisätty erikoistumisvaihtoehdot. Ohjelmistotestauksesta ei kuitenkaan ole vielä tarjolla ammatti-korkeakouluun soveltuvaa suomenkielistä oppikirjaa, joten opinnäytetyön tavoitteena on tehdä tutkimus ohjelmistotestauksen menetelmistä ja työvälineistä sekä tuottaa materiaalia, jota voidaan käyttää opetusmateriaalina testauskurssilla.

1.2 Opinnäytetyöraportin rakenne

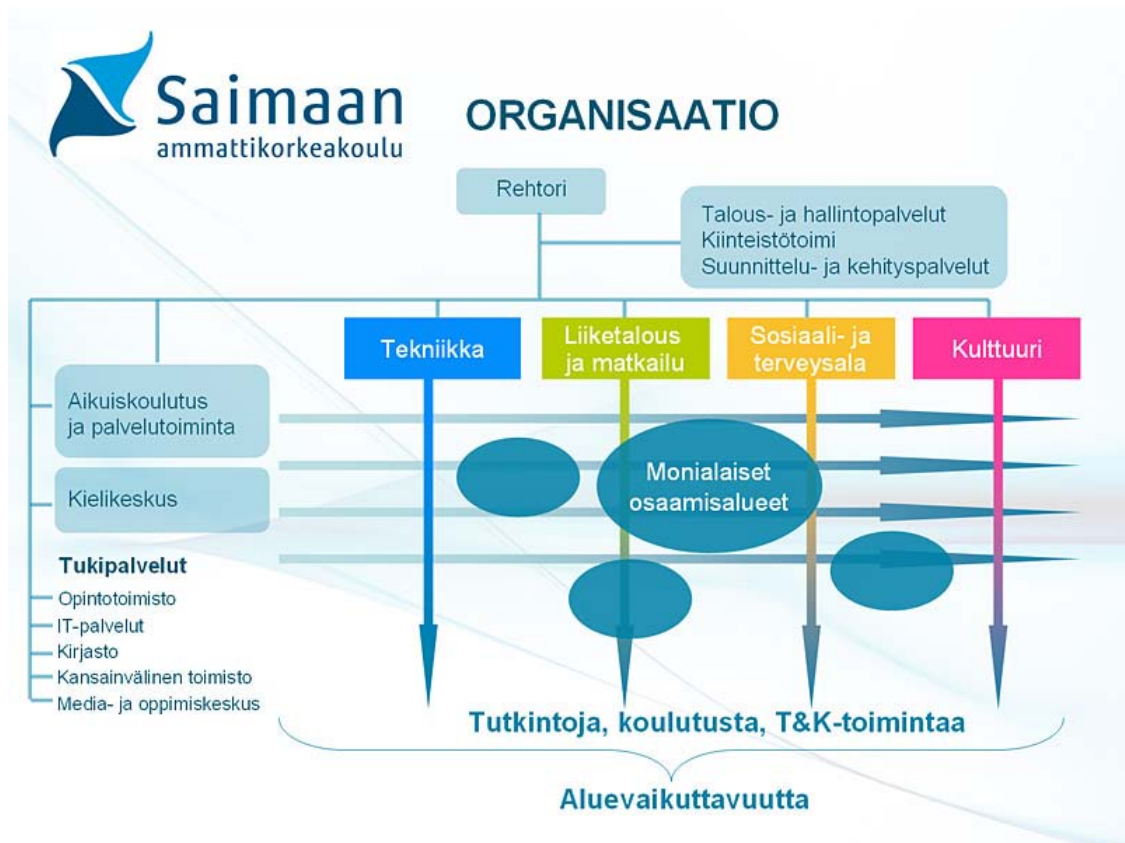
Opinnäytetyöraportin luvussa 1 käsitellään työn lähtökohdat ja tavoitteet. Luku 2 kuvaa Saimaan ammattikorkeakoulun toimintaa ja tietotekniikan koulutusohjelman opetussuunnitelman muutoksia viime vuosina. Luvussa 3 käydään tutkimustulokset läpi vain pääpiirteittäin, koska tutkimustulokset on esitetty seikka-peräisesti liitteenä 1 olevassa ohjelmistotestauksen opintomateriaalissa. Opinnäytetyöraportin luvussa 4 on kuvattu työn kulku ja luvussa 5 työn tulokset. Luku 6 sisältää opinnäytetyön loppupäätelmät.

2 ASIAKKAAN TOIMINNAN KUVAUS

Opinnäytetyön asiakkaana on Saimaan ammattikorkeakoulun tietotekniikan koulutusohjelma. Asiakkaan edustajana lehtori Martti Ylä-Jussila, joka toimii Saimaan ammattikorkeakoulun tietotekniikan osaston opettajana.

2.1 Saimaan ammattikorkeakoulu

Saimaan ammattikorkeakoulu Oy on Etelä-Karjalan koulutuskuntayhtymän omistama oppilaitos, jossa on viisi koulutusala (kuva 2.1): tekniikka, sosiaali- ja terveysala, liiketalous, matkailu- ja ravitsemispalvelut ja kulttuuri. Ammattikorkeakoulussa on 18 ammattikorkeakoulututkintoon johtavaa koulutusohjelmaa, joista neljä on kokonaan englanninkielisiä, ja neljä ylempään ammattikorkeakoulututkintoon johtavaa koulutusohjelmaa joista yksi kokonaan englanninkielinen. Ennen vuotta 2009 oppilaitos toimi nimellä Etelä-Karjalan ammattikorkeakoulu.



Kuva 2.1 Saimaan ammattikorkeakoulun koulutusalat (Saimaan AMK)

Ammattikorkeakoulun opetus jakautuu neljälle koulutuskampukselle, joista kaksi sijaitsee Imatralla ja kaksi Lappeenrannassa. Oppilaitoksen kokonaisopiskelijamäärä vuonna 2009 on noin 3000 opiskelijaa, joista tekniikkaa opiskelee 39 %, sosiaali- ja terveysalaa 26 %, liiketaloutta 21 %, matkailu- ja ravitsemuspalveluita 9 % ja kulttuuria 5 %.(Saimaan AMK).

2.2 Tietotekniikan koulutusohjelma

Tietotekniikan koulutusohjelman opetussuunnitelma on muuttunut radikaalisti viime vuosina. Ohjelmistotekniikan ja viestinnän suuntautumisvaihtoehdot ovat vaihtuneet tietojärjestelmien kehityksen ja organisaation IT-palveluiden suuntautumisvaihtoehdoiksi. Opintoja toteutetaan entistä enemmän käytännön projekteissa sekä tietojärjestelmäklinikalla, jossa opetellaan erityyppisten tietojärjestelmien suunnittelua ja hallintaa. Suuntautumisvaihtoehtoihin on lisätty erikoistumisopinnot, jotka sijoittuvat neljännelle opiskeluvuodelle. Laajat, kymmenen opintopisteen projektityöt tehdään erikoistumisalaan liittyen. (Saimaan AMK).

Tietotekniikan koulutusohjelman suuntautumisvaihtoehtojen erikoistumisalat (Saimaan AMK):

Perusopinnot	58 op
Ammattiopinnot	122 op
- yhteiset ammattiopinnot	78 op
- suuntautumisvaihtoehdon opinnot	44 op
Tietojärjestelmien kehityksen sv	
- sisältötuotantojärjestelmät	
- palvelinjärjestelmät	
Organisaation IT-palveluiden sv	
- terveydenhuollon tietojärjestelmät	
- toiminnanohjausjärjestelmät	
Vapaasti valittavat opinnot	15 op
Harjoittelu	30 op
Opinnäytetyö	15 op
Yhteensä	240 op

Kun vertaillaan vuoden 2005 (kuva 2.2) tietotekniikan opetussuunnitelmaa vuoden 2009 (kuva 2.3) vastaavaan ammattiaineiden osalta, nähdään, että ohjelmointitekniikan opetusta on vähennetty kun taas tietojärjestelmien suunnittelua opetetaan entistä enemmän. Laadunhallinta ja testaus ovat tulleet mukaan uusina aineina opetussuunnitelmaan.

AMMATTIOPINNOT	TR0032	0.0	31	26	55	38
TIETOTEKNIikka	000014	21.0				
Tietotekniikan perusteet	KTI0064	4.0	4			
Laitetekniikka	KTE0681	3.0	3			
Digitaalinen media	KTE0063	3.0	3			
Linux	KTE1070	3.0	3			
Käyttöjärjestelmät T	KTI0066	4.0			4	
Tietotekniikan työharjoitus	KTI0065	3.0	3			
<u>Tietokoneet ja kääntäjät</u>	KTE0570	3.0				3
OHJELMISTOTEKNIKAN SUUNTAUTUMISVAIHTOEHTO	217113	0.0				
Systeemis suunnittelu II	KTE1320	5.0			5	
Olio-ohjelmointi II	KTE1326	4.0			4	
<u>Olio-ohjelmoinnin suunnittelumallit</u>	KTE1152	3.0				3
<u>Hajautetut sovellukset</u>	KTE1327	4.0				4
Ohjelmistotekniikan erikoiskurssi	KTE1328	3.0				3
<u>J2ME-ohjelmointi</u>	KTE1410	3.0				3
Tietorakenteet ja algoritmit II	KTE1330	3.0			3	
<u>Projektityö</u>	KTE1332	10.0				10
VIESTINTÄTEKNIKAN SUUNTAUTUMISVAIHTOEHTO	217112	0.0				
Graafisen suunnittelun perusteet	KTE0732	3.0			3	
Videon käsittely	KTE1158	3.0			3	
Digitaalinen valokuvaus	KTE1155	3.0			3	
Sisältötuotanto	KTE1333	4.0			4	
<u>Multimediaajäriestelmät</u>	KTE1156	3.0				3
<u>Multimedian erikoiskurssi</u>	KTE1334	3.0				3
<u>Animaatiot</u>	KTE1154	3.0				3
OHJELMOINTITEKNIikka	217101	18.0				
Ohjelmoinnin perusteet I	KTE0731	3.0	3			
Ohjelmoinnin perusteet II	KTE0733	3.0	3			
Ohjelmoinnin jatko	KTE1386	4.0		4		
Olio-ohjelmointi I	KTE1316	5.0		5		
Tietorakenteet ja algoritmit I	KTE1317	3.0			3	
TIEDONHALLINTATEKNIikka	217102	13.5				
Relaatiotietokannat I	KTE0061	3.0	3			
Relaatiotietokannat II	KTE0231	3.0		3		
Tietokantaohjelmointi	KTE1318	5.0			5	
Tietokannan suunnittelu	KTE0591	3.0			3	
SYSTEMITYÖ	217103	7.5				
Systeemis suunnittelu I	KTE1319	5.0		5		
Projektityöskentely	KTE0250	3.0			3	
TIEDONSIIRTOTEKNIikka	217104	13.5				
Tiedonsiirron perusteet	KTE0234	3.0	3			
Lähiverkot	KTE0235	3.0		3		
Tiedonsiirtojärjestelmät	KTE1321	4.0			4	
<u>Tietoturvaluissuus</u>	KTE0523	3.0				3

Kuva 2.2 Tietotekniikan koulutusohjelman opetussuunnitelma 2005 (Saimaan AMK, SoleOPS)

	AMMATTIOPINNOT	TR0032	0.0	31	30	14	3
🔍	TIETOTEKNIikka	000014	21.0				
🔍	<u>Tietotekniikan perusteet</u>	KTI0064	4.0	4			
🔍	Digitaalitekniikka T	KTE1315	3.0		3		
🔍	<u>Digitaalinen media</u>	KTE0063	3.0	3			
🔍	<u>Linux</u>	KTE1070	3.0	3			
🔍	Käyttöjärjestelmät T	KTI0066	4.0		4		
🔍	OHJELMOINTITEKNIikka	217101	18.0				
🔍	<u>Ohjelmoinnin perusteet</u>	KTI0063	4.0	4			
🔍	<u>Ohjelmoinnin jatko</u>	KTE1386	4.0	4			
🔍	Olio-ohjelmoinnin perusteet	KTI0071	4.0		4		
🔍	Olio-ohjelmoinnin jatko	KTI0072	4.0		4		
🔍	TIETORAKENTEET JA TIETOHALINTA	217118	0.0				
🔍	Tietorakenteet ja niiden käyttö	KTI0074	3.0		3		
🔍	Algoritmien käyttö	KTI0073	3.0		3		
🔍	<u>Relaatiotietokannat</u>	KTI0075	4.0	4			
🔍	Tietokantaohjelmointi	KTE1446	4.0			4	
🔍	Tietokannan suunnittelu	KTE0591	3.0		3		
🔍	OHJELMISTOTEKNIikka	217119	0.0				
🔍	<u>Ohjelmistotuotannon perusteet</u>	KTI0076	3.0	3			
🔍	Ohjelmiston määrittely ja suunnittelu	KTI0077	3.0		3		
🔍	<u>Projektityöskentely</u>	KTE0250	3.0	3			
🔍	Olio-pohjaisen ohjelmistosuunnittelun perusteet	KTI0078	3.0			3	
🔍	Testaus ja laadunvalvonta	KTI0079	3.0			3	
🔍	TIEDONSIIRTOTEKNIikka	217104	13.5				
🔍	<u>Tiedonsiirron perusteet</u>	KTE0234	3.0	3			
🔍	Lähiverkot	KTE0235	3.0		3		
🔍	Tiedonsiirtojärjestelmät	KTE1321	4.0			4	
🔍	Tietoturvallisuus	KTE0523	3.0				3
	SUUNTAUTUMISVAIHTOEHDOT	SV	44.0			18	26
	<u>Tietojärjestelmien kehityksen sv</u>	TIKE	44.0				
🔍	TIETOJÄRJESTELMIEN KEHITYS	217120					
🔍	Tiedonhallinnan jatko	KTI0080	3.0			3	
🔍	Ohjelmistoarkkitehtuurit ja suunnittelumallit	KTI0082	4.0			4	
🔍	Ihminen vuorovaikutteisessa teknologiassa	KTI0084	3.0			3	
🔍	Olio-pohjaisen ohjelmistosuunnittelun jatko	KTI0081	4.0			4	
🔍	Käytettävyys ja käyttöliittymäsuunnittelu	KTI0083	4.0			4	
🔍	Järjestelmäprojekti	KTI0092	10.0				10
🔍	Tietojärjestelmien kehityksen erikoistumisopinnot	KTI0095	16.0				16
	Organisaation IT-palveluiden sv	ORIT	44.0				
🔍	ORGANISAATION IT-PALVELUT	217121					
🔍	Organisaation oppiminen	KTI0085	3.0			3	
🔍	Organisaation tietojärjestelmät ja tietohallinto	KTI0086	4.0			4	
🔍	Tietojärjestelmien käyttöönotto	KTI0087	4.0			4	
🔍	Järjestelmäintegraatio	KTI0088	4.0			4	
🔍	IT-Palveluprosessit	KTI0089	3.0			3	
🔍	IT-palveluprojekti	KTI0093	10.0				10
🔍	IT-palveluiden erikoistumisopinnot	KTI0094	16.0				16

Kuva 2.3 Tietotekniikan koulutusohjelman opetussuunnitelma 2009 (Saimaan AMK, SoleOPS)

3 OHJELMISTOTESTAUS

Tässä luvussa käydään läpi ohjelmistotestauksen teoria mukaillen liitteen 1 rakennetta.

3.1 Historia

Ohjelmistotestaus, kuten automaattinen tietojenkäsittelykin, on alana suhteellisen nuori ja alkuaikoina testaus keskittyi enemmän laitteistoihin kuin ohjelmiin. Testauksen kehityksen on katsottu alkaneen 1950-luvun alussa, ja sen kehityskulku on esitettävissä aikakausina, joita ovat virheenjäljityksen, havainnollistamisen, hajottamisen, arvioinnin ja ennaltaehkäisyn aikakaudet (kuva 3.1).

Vuosi	Aikakausi
– 1956	The debugging-oriented period Virheenjäljityksen aikakausi
1957–1978	The demonstration-oriented period Havainnollistamisen aikakausi
1979–1982	The destruction-oriented period Hajottamisen aikakausi
1983–1987	The evaluation-oriented period Arvioinnin aikakausi
1988–	The prevention-oriented period, Ennaltaehkäisyn aikakausi

Kuva 3.1 Ohjelmistotestauksen kehitysvaiheet. (Gelperin, Hezel 1988)

Ohjelmistotestaus on ammattimaistunut 1970 –luvulta lähtien. Vuonna 1972 järjestettiin Pohjois-Carolinan yliopistossa Chapel Hillsissä ensimmäinen aiheetta käsittelevä konferenssi, jonka sisältöön perustuva kirja, G.J. Myersin the Art of Software Testing, julkaistiin vuonna 1979. Kirja on edelleen alansa klassikkoteos, ja siitä on tehty uudistettu versio vuonna 2004. (Myers, Sandler, Badgett 2004).

Testauksen tunnustaminen omana erikoisosaamisen alanaan on näkynyt myös sen standardointina: vuonna 1983 julkaistiin ensimmäinen IEEE-standardi, joka käsitteli ohjelmistotestausta, ”829-1983 IEEE Standard For Software Test Documentation”. Standardia on päivitetty vuosina 1998 ja 2008 (IEEE 1998, IEEE 2008). Myöhemmin myös yksikkötestaukselle ja katselmoinneille on julkaistu omat IEEE-standardinsa. Standardien lisäksi ohjelmistotestausta on sertifioitu, jotta testaajien ammattitaito pystyttäisiin parantamaan ja mittaamaan kansainvälisesti. Tätä varten perustettiin vuonna 2002 Edinburghissa International Software Testing Qualifications Board, ISTQB (ISEB 2009).

3.2 Näkökulmia testaukseen

Ohjelmistotestauksen lähtökohta ja määritelmät ovat muuttuneet historian saatossa: kun alkuaikoina testauksella varmistettiin ohjelmiston toimivuus, niin nykyaikaisen testauksen lähtökohta on paljastaa ohjelmiston toimimattomuus ja löytää mahdollisimman paljon virheitä. Lisäksi testauksella varmistetaan, että ohjelmisto on määrittelyään vastaava ja toimii riittävän hyvin täyttäen sille asetetut laatuvaatimukset.

Ohjelmistotestaajia on jaettu lähtökohtiensa ja näkökulmiensa mukaan eri koulukuntiin, jotta olisi helpompi ymmärtää, miksi eri testaajilla voi olla niin erilaiset käsitykset siitä, mitä ohjelmistotestaus perimmiltään on. Pettichordin (2007) mukaan koulukunnat ovat analyttinen, standardilähtöinen, laatulähtöinen, kontekstiohjattu ja ketterän testauksen koulukunta (kuva 3.2).

Koulukunta	Periaate
Analytic school	Testaus on ohjelmistotiedettä
Standard school	Testaus on hallittu prosessi
Quality school	Testaus on laadun varmistamista
Context-driven school	Testaus on ohjelmistokehityksen yksi haara
Agile school	Testaus todistaa, että tuote on valmis

Kuva 3.2 Testauksen koulukunnat (Pettichord 2007)

Analyttisen koulukunnan ydinajatus on, että ohjelmisto on looginen kokonaisuus ja testaus matematiikan sivuhaara, ja analytikkojen mielestä ohjelmiston pitää olla erittäin tarkasti määritelty, että sitä voidaan testata.

Standardilähtöisen, toiselta nimeltään tehdaskoulukunnan, mielestä testaus taas on sarja toistettavia toimenpiteitä, ja testauksen oltava helposti hallittavaa.

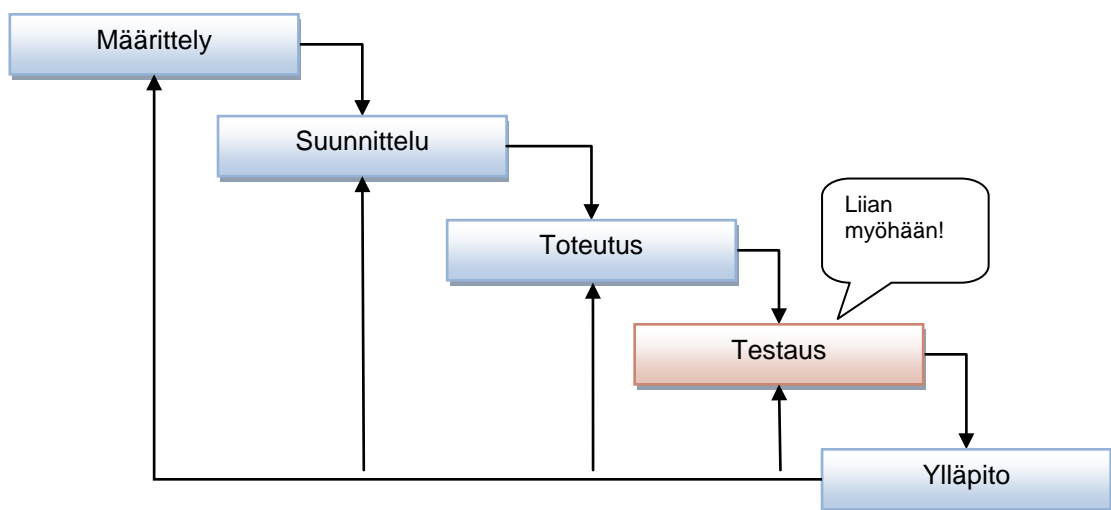
Laatulähtöinen koulukunta ajattelee, että testausyksikkö toimii ohjelmistoprojektin portinvartijana ja että ohjelmisto on valmis sitten, kun laadunvarmistus näin sanoo.

Kontekstiohjatun eli tilannelähtöisen koulukunnan ajatus on, että testaamalla pyritään oppimaan uusia asioita testattavasta aineistosta ja testausta pidetään vaativana, älyllisenä toimintona.

Ketterä koulukunta nojautuu nimensä mukaisesti ketteriin menetelmiin, joissa toteutus jaetaan lyhyisiin iteraatioihin, jotka kehitetään nopeassa määrittele-suunnittele-toteuta-testaa-julkaise-syklissä. Ketterässä kehityksessä testauksen tarkoitus on osoittaa, että iteraatio on valmis julkaistavaksi.

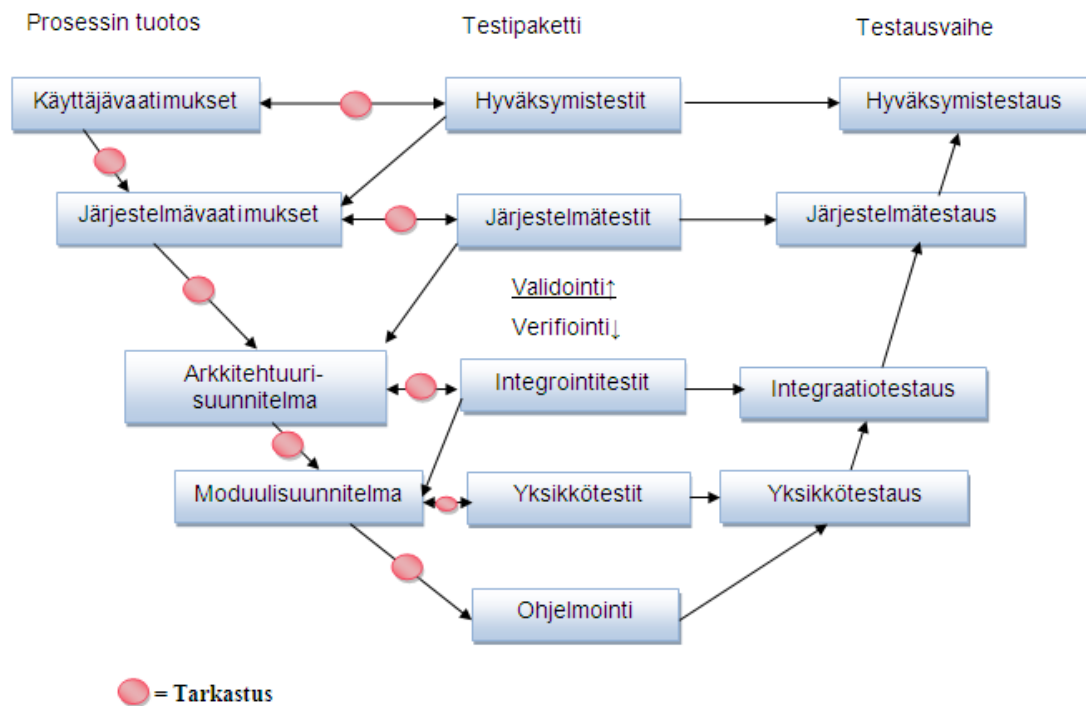
3.3 Testaus osana perinteistä ohjelmistotuotantoa

Perinteisesti testaus on nähty ohjelmistotuotannossa yhtenä vesiputousmallin vaiheena. Vesiputousmallilla tarkoitetaan ohjelmistotuotannon prosessimallia, jossa vaiheet kulkevat vesiputouksen lailla tasolta toiselle. Malli on peräisin vuodelta 1970, jolloin sen esitteli Winston W. Royce. Vesiputousmallin (kuva 3.3), vaiheet ovat määrittely, suunnittelu, toteutus, jolla on tarkoitettu varsinaisen ohjelmakoodin kirjoittamisen ohella yksikkötestausta ja toteutuksen verifiointia sekä viimeisenä testaus, joka on järjestelmätestausta sekä toteutuksen validointia. Kun testaus on prosessin viimeinen vaihe, karsitaan siitä määrällisesti eniten testaukseen käytettävästä ajasta. Tämä vaikuttaa huonontavasti ohjelmiston laatuun, ja myöhäisessä vaiheessa löydetyt viat tulevat kalliiksi korjata.



Kuva 3.3 Ohjelmistotuotannon vesiputousmalli (Royce)

Koska myöhäisessä vaiheessa aloitettu testaus on riski ohjelmistotuotantoprosessissa, on vesiputousmallia paranneltu luomalla testauksen V-malli (kuva 3.4), jossa testaus on integroitu prosessin kaikkiin vaiheisiin. V-mallissa jokaiselle prosessin tuotokselle on vastinparina testausvaihe. Prosessin tuotoksia, eli vaatimusmäärittelyjä ja suunnitelmia, tarkastetaan ja tarkennetaan prosessin edetessä vaiheesta toiseen. Näin viat on mahdollista löytää aikaisemmassa vaiheessa kuin perinteistä vesiputousmallia noudattaen olisi. (Taina, 2004).



Kuva 3.4 Testauksen V-malli (Taina, 2004)

V-mallin mukaan testaus jaetaan yksikkötestaukseen, integrintitestaukseen, järjestelmättestaukseen sekä hyväksymistestaukseen. Yksikkötestaus testaa alimman tason moduuleja ja integrintitestaus yhteen liitettyjä moduuleja. Järjestelmättestauksessa testataan koko järjestelmän toimivuutta ja hyväksymistestauksessa käydään läpi samoja asioita kuin järjestelmättestauksessa, mutta asiakkaan toimesta.

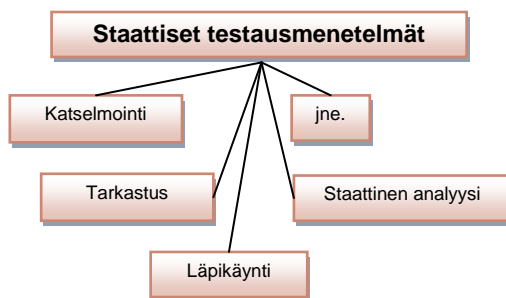
V-malli päättyy siihen, että ohjelmisto on valmis ja hyväksytetty. Testaustoiminta ei sen sijaan päätty, sillä on odotettavissa, että ennen pitkää ohjelmistoon on tehtävä korjaus- ja ylläpitotoimenpiteitä, jotka vaativat lisätestausta. Regres-

siotestauksella ja ylläpitotestauksella tarkoitetaan jo aiemmin testatun ohjelman uudelleentestaamista siihen tehtyjen muutosten jälkeen. Näin pyritään selvittämään, ovatko muutokset aiheuttaneet uusia vikoja sellaisissa ohjelmiston osissa, joihin muutostöissä ei ole puututtu (Holopainen, 2005). Regressiotestit ovat yleisimmin yksikkö- ja integrointitestejä. Ylläpitotestauksesta puhutaan, kun muutoksia tehdään järjestelmätasolla, esimerkiksi käyttöympäristön muuttumisen aiheuttamien mukautustöiden jälkeen.

Edellä selostetun vaihejaon lisäksi testaustoiminta voidaan jaotella vielä verifiointiin ja validointiin. Verifiointilla varmistetaan, että ohjelmisto on määrittelynsä mukainen, ja validoinnilla varmistetaan, että ohjelmisto täyttää asiakkaan asettamat odotukset. Termi V&V, verification and validation, yhdistää nämä osat alueet. V&V:n tavoitteena on, että tarkastamalla ja testaamalla ohjelmisto koko prosessin elinkaaren ajan säännöllisesti varmistutaan siitä, että ohjelmisto täyttää sille asetetut tarpeet, ja että se on riittävän virheetön omassa kontekstissaan. (Watkins, 2001).

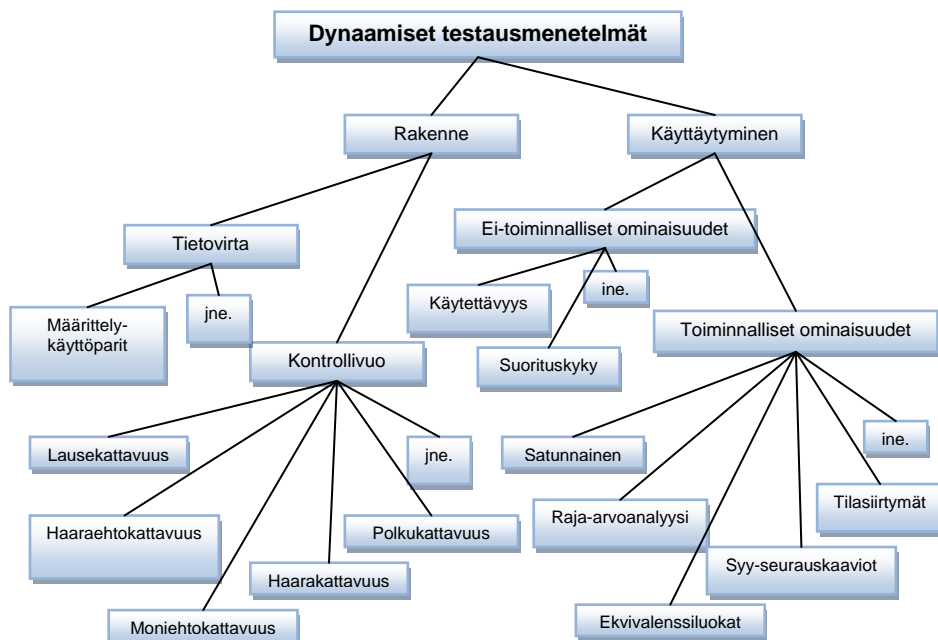
3.4 Testauksen käytännöt ja tekniikat

Testauksen käytännöt on jaettavissa staattisiin ja dynaamisiin testausmenetelmiin. Staattinen testaus (kuva 3.5) tarkoittaa komponentin tai järjestelmän testausta määrittely- tai toteutustasolla suorittamatta ohjelmistoa. (ISTQB, 2007).



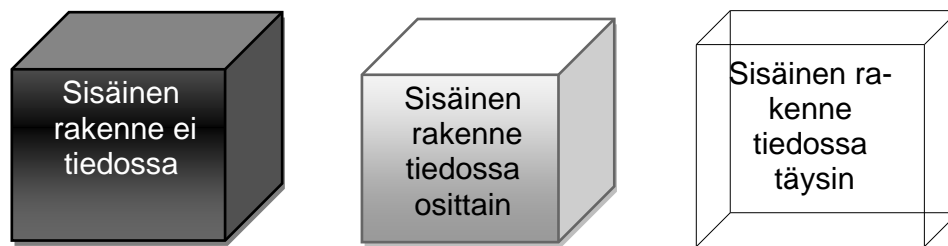
Kuva 3.5 Staattiset testausmenetelmät (liite 1, s.57)

Dynaaminen testaus on testausta, joka tehdään suorittamalla testattavaa ohjelmaa selvittääkseen ohjelman ja ohjelmakoodin suorituksen aikaista käyttäytymistä (ISTQB, 2007). Kuvassa 3.6 on esitetty dynaaminen testaus jaoteltuna rakenteen ja käyttäytymisen testaukseen.



Kuva 3.6 Dynaamiset testausmenetelmät (liite 1, s.63)

Testauksen tekniikat voidaan jakaa mustalaatikko- harmaalaatikko- ja lasilaatikkotestaukseen. Mustalaatikkotestauksessa ohjelman sisäisestä rakenteesta ei tiedetä mitään, ja testit laaditaan esimerkiksi käyttötapauskuvausten perusteella. Harmaalaatikkotestaus on käsitteellisesti mustan ja lasilaatikon välimaastossa, eli ohjelman sisäinen rakenne on osittain tiedossa. Lasilaatikkotestauksessa ohjelman sisäinen rakenne tiedetään täysin, ja lähdekoodit ovat testauksen käytävissä. Kuva 3.7 havainnollistaa käsitteiden eroavaisuuksia.

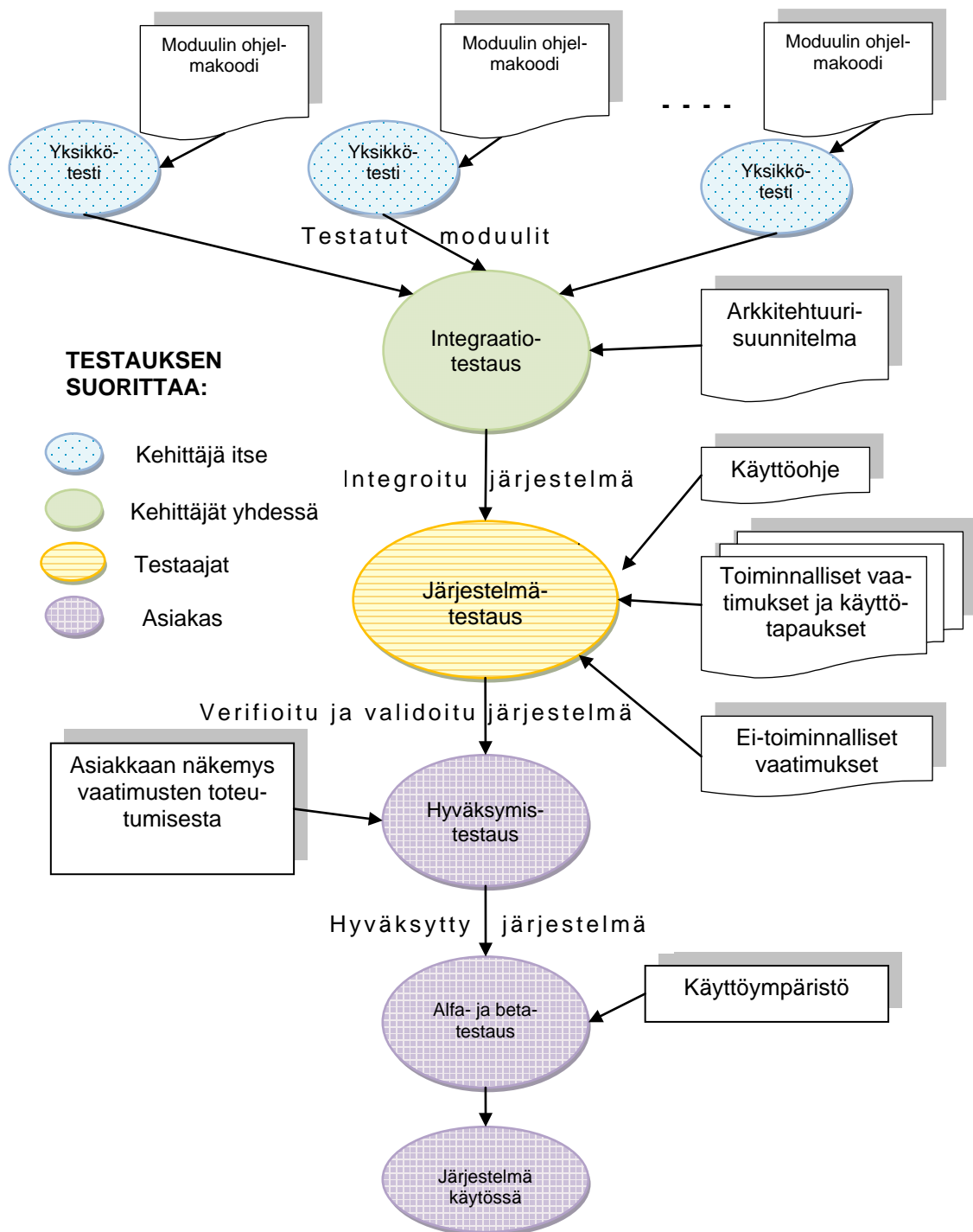


Kuva 3.7 Eri testaustekniikoiden käsitteelliset eroavaisuudet (liite 1, s.64)

Mustalaatikkotestaus on vaatimuksiin perustuvaa ja testitapaukset suoritetaan ohjelmaa käyttämällä. Harmaalaatikkotestauksen testitapauksilla saadaan sitä enemmän ulottuvuutta testaukseen, mitä enemmän ohjelman rakenteesta tiedetään. Lasilaatikkotestaus vaatii mahdollisuuden suorittaa koodia syötteillä, joita se vastaanottaa käytännössä ja siinä ohjelman sisäinen rakenne on testaajan tiedossa.

3.5 Testauksen hallinta

Testauksen hallinnalla tarkoitetaan toimintoja, joiden tarkoitus on pitää kustannukset ja ajankäyttö budjetin rajoissa sekä mitata testauksen edistymistä ja sen tuloksia. Testauksen hallintaan kuuluu testimateriaalin hallinta, testauksen resurssien hallinta, testivaatimusten hallinta, testauksen suunnittelu, testitapaus-ten suoritus ja suoritettujen testien seuranta sekä testauksen tuloksien raportointi ja seuranta. Hallintaan liittyy myös testauksen organisointi sen osalta, kenen vastuulla testaus on missäkin prosessin vaiheessa. Kuvassa 3.8 on kuvattu tyypillisen testausprojektin aktiviteetit ja niiden suorittajat.



Kuva 3.8 Testauksen vaiheet ja suorittajat (liite 1, s. 87)

Testauksen suunnittelu

Jotta testauksen hallinta onnistuisi, on ennen testauksen aloittamista laadittava testaussuunnitelma, jossa määritellään testauksen kohteet, testausstrategia, testien läpäisy- ja epäonnistumiskriteerit, laadittavat dokumentit, testaustehtävät, testausympäristö, vastuut, henkilöstön koulutustarpeet, aikataulut, riskit ja testauksen päätöskriteerit. Suositukset testauksen dokumenttien sisällöstä löytyvät IEEE:n standardista 829 (IEEE 1998).

Testausstrategialla tarkoitetaan yleistä lähestymistapaa testaukseen. Se määrittää, millä tavoin huolehditaan siitä, että kriittisimmät ominaisuudet tulee testattua, sekä sen, mitä tekniikoita ja työkaluja käytetään testauksessa. Lisäksi strategiassa kuvataan testauksen kattavuuden vähimmäisvaatimukset sekä testauksen lopetusehdot. Testausstrategiaan kirjataan myös takarajat testauksen valmistumiselle.

Testauksen aikataulua suunnitellessa käytetään apuna ohjelmistoprojektin varsinaista aikataulua, ja siihen sisällytetään kaikki ohjelmistoprojektin aikataulun virstanpylväät kuten osaluovutusten ajankohdat. Jos testaus tarvitsee omia virstanpylväitä, lisätään ne aikatauluihin.

3.6 Ketterät menetelmät ja testaus

Ketterän testauksen määritelmä on, että se on mitä tahansa testausta, joka on sidottu ketteriin arvoihin. Ketterät arvot on koottu ketteryyden julistukseen (Manifesto for Agile Software Development), joka sisältää ketterän kehityksen arvot sekä periaatteet, joita ketterät menetelmät noudattavat. Periaatteiden julistus on seuraavanlainen :

"Me etsimme parempia keinoja ohjelmistojen kehittämiseen tekemällä sitä itse ja auttamalla siinä muita. Tässä työssämme olemme päätyneet arvostamaan

***Yksilöitä ja vuorovaikutusta enemmän kuin prosesseja ja työkaluja
Toimivaa sovellusta enemmän kuin kokonaisvaltaista dokumentaatiota
Asiakasyhteistyötä enemmän kuin sopimusneuvotteluita
Muutokseen reagoimista enemmän kuin suunnitelman noudattamista.***

Vaikka oikeallakin puolella on arvoa, me arvostamme vasemmalla olevia asioita enemmän.”

Varsinainen ketterä kehitys tarkoittaa nopeaa, itseohjautuvaan tiimityöhön perustuvaa ohjelmistokehitystä, jossa dokumentaatio pyritään pitämään kevyenä ja kehittämään ohjelmistoa nopeissa sykleissä niin, että joka iteraatio toistaa määrittele-suunnittele-toteuta-testaa-sykliä. Menetelmät pyrkivät joustavaan, muutoksiin nopeasti vastaavaan ohjelmistokehitykseen. Ketterä testaus ei kuitenkaan tarvitse kehiksekseen ketterää kehitystä, vaan menetelmiä voidaan käyttää testauksessa myös projekteissa, joissa ohjelmistokehitys seuraa perinteistä mallia. Ketterän testauksen pääsuuntaukset ovat XP-menetelmään (eXtreme Programming) perustuva ääritestaus ja tutkiva testaus (Pyhäjärvi, Pöyhönen, 2004).

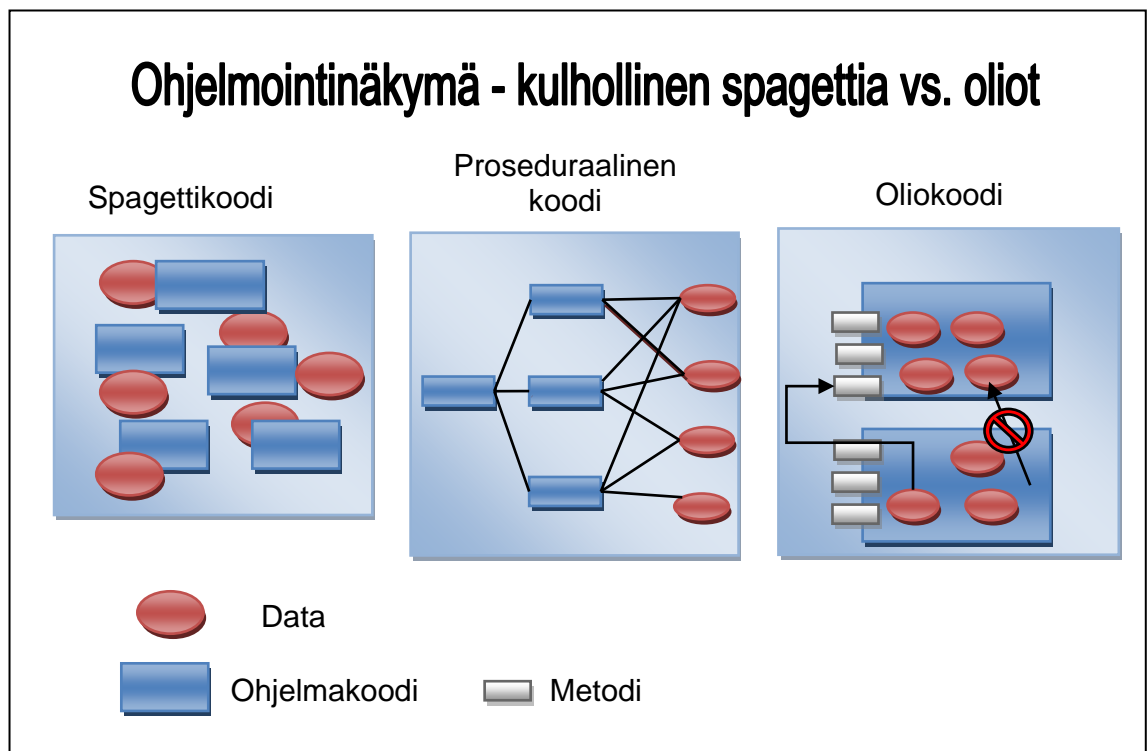
Ääritestaus ottaa kantaa yksikkötestaukseen. Se on menetelmä, jossa yksikkötestaus suoritetaan TDD eli Test Driven Development – menetelmällä. TDD on itse asiassa ohjelmointi- eikä testausmenetelmä, ja siinä testitapauksen koodi kirjoitetaan ennen varsinaista ohjelmakoodia. Ohjelmakoodia muokataan niin pitkään, että se läpäisee uuden testin, ja sen jälkeen kirjoitetaan uusi testitapaus. Ääritestauksessa testattava ohjelmisto koostetaan päivittäin, ja koostamisen yhteydessä ajetaan automatisoidut regressiotestit, jolloin iteraatio on julkaistavissa milloin tahansa.

Tutkiva testaus (exploratory testing) ottaa kantaa järjestelmätestaukseen. Se on yksinkertaistettuna testien suunnittelemista ja suorittamista samalla kertaa, joka on vastakohtaista perinteiseen suunnitelmalliseen testaukseen (scripted testing) verrattuna, jossa testausproseduurit ja testitapaukset määritellään etukäteen, ennen testauksen suorittamista (Bach, 2003).

3.7 Oliotestaus

Ohjelmistojen rakenne on läpikäynyt kuvan 3.9 mukaisen kehityskaaren. Ohjelmistokehityksen alkuaikoina ohjelmakoodi oli rakenteeltaan epämääräistä, vaikeasti luettavaa ja ylläpidettävää. Kun ohjelmistojen koko kasvoi, kehitettiin koodin selkeyttämiseksi rakenteinen, eli proseduraalinen koodi, jossa pääohjelma ajaa aliohjelmia ja tietorakenteet ovat erillään ohjelmakoodista. Ohjelmis-

tojen kompleksisuus ja koko ovat kuitenkin jatkaneet kasvuaan, ja tarve koodin uudelleenkäytettävyydelle ja koodin paremmalle ylläpidettävyydelle on kasvanut entisestään. Tämän mahdollistamiseksi on kehitetty olio-ohjelmointi, jossa jokainen ohjelman moduuli huolehtii omista tietojäsenistään, ja moduulit viestivät keskenään metodiensa avulla.



Kuva 3.9 Ohjelmistojen rakenteen kehittyminen (Pöyhönen ym. 2002)

Oliokielisen ohjelman testaaminen poikkeaa lausekielisen ohjelman testaamisesta, koska oliokielet ovat ilmaisuvoimaisempia kuin lausekieliset.

Olioihin liittyviä erityistilanteita ovat:

- perintä
- polymorfismi ja dynaaminen sidonta
- tilariippuvuus
- tiedon kapselointi
- abstraktit luokat
- poikkeuskäsittelyn vaikutus testaukseen
- rinnakkaisuuden vaikutus testaukseen.

Perintä, polymorfismi ja dynaaminen sidonta mahdollistavat monimutkaisten rakenteiden laatimisen helposti, mutta virheiden tekeminen helpottuu samalla. Perinnässä koko metodin toiminta voi muuttua ei-toivotuksi perintahierarkiaan johtuen ja dynaaminen sidonta aiheuttaa sen, että metodin kutsu voi vaihtua suorituskertojen välillä.

Oliot ovat tilariippuvia, ja niiden tila täytyy huomioida testauksessa. Ongelmia aiheuttaa se, että kapselointi piilottaa olion tilatiedot, ja niihin, kuten muihinkin olion yksityisiin tietoihin, on hankalaa päästä käsiksi.

Myös se, että luokka on suhteellisen pieni kokonaisuus, aiheuttaa ongelmia: koska rajapintoja on huomattavasti enemmän kuin lausekielisessä ohjelmoinnissa, tulee kutsurajapinnan virheitäkin enemmän. Se, että olio-ohjelmoinnissa on käytettävissä myös abstrakteja luokkia ja rajapintoja, hankaloittaa testaajan työtä entisestään: suora testaus ei onnistu, vaikka luokat ja rajapinnat sisältävät ohjelman suorituksen kannalta oleellista tietoa. Testauksen toteuttamiseksi voidaan silloin käyttää korvikeolioita eli Mock-objekteja, joilla rajapintoja voidaan simuloida.

Poikkeuskäsittelyssäkin on omat haasteensa: metodin suoritus voi keskeytyä tai siirtyä uuteen paikkaan ilman, että siirtymää voi nähdä suoraan koodissa. Rinnakkaisuus on myös yksi olio-ohjelmistojen testauksen haaste: useita toisistaan riippuvia metodeita suoritetaan samaan aikaan. Näin voi tapahtua proseduraalissakin ohjelmoinnissa, mutta oliomaailmassa se on yleistä. (Pöyhönen ym, 2002).

Yksittäisen olion elinkaari testataan seuraavalla tavalla:

- Luodaan olio
- Kirjoitetaan vähintään yksi testitapaus jokaiselle muodostimelle
- Alustetaan olio perustilaan, tai johonkin muuhun sopivaan tilaan
- Käydään läpi kaikki tilasiirtymät
- Edellä mainittu toistetaan jokaiselle olion attribuutille

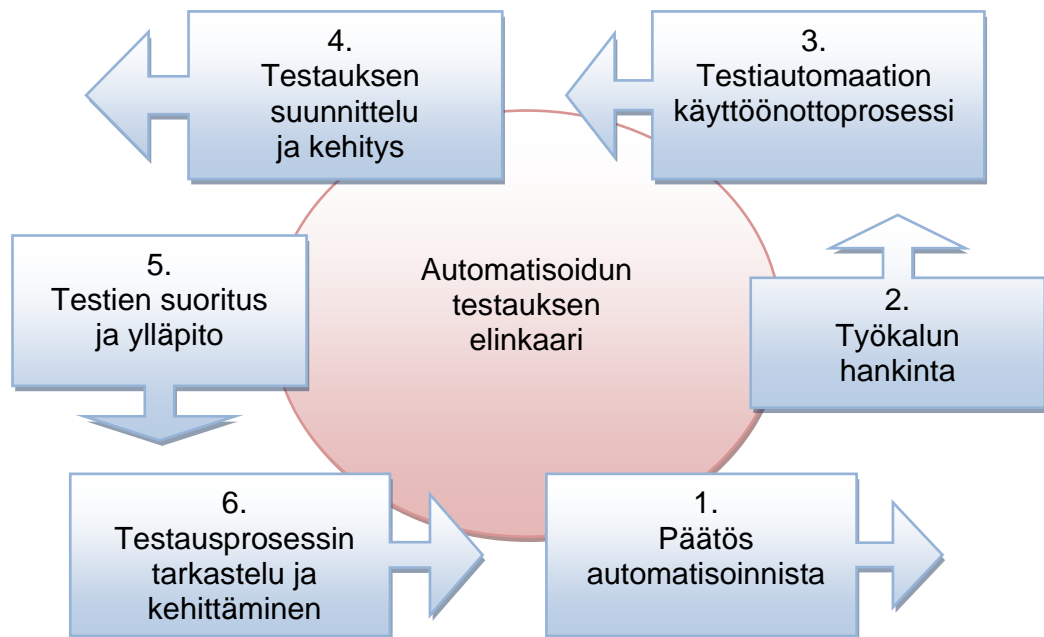
- Kutsutaan testattavaa metodia
- Testijoukkoa karsitaan eri tekniikoin, kuten ekvivalenssiluokkiin jakamisella ja raja-arvoanalyysillä.
- Tarkistetaan kutsutun olion attribuuttien tila ja sivuvaikutukset jokaisesta kutsusta
- Tehdään sama olion kaikille parametreille
- Tehdään sama jokaiselle testattavalle metodille
- Tuhotaan olio
- Testataan hajottaja ja tarkistetaan, että kaikki olion varaamat resurssit on todella vapautettu
- Tarkistetaan kattavuusluvut ja tarvittaessa lisätään testitapauksia.

Olio-ohjelman testaamisessa tärkeää on, että ohjelma on kuvattu riittävän tarkasti erilaisin mallinnusvälinein: sekvenssikaaviot ja UML-mallit ovat tarpeen testitapausten luomiseksi käyttötapauskuvausten ohella. Mallit on syytä myös katselmoida kunnolla niiden oikeellisuuden varmistamiseksi. Luokkatasolla testaus kuuluu luokan suunnittelijalle, ja luokkien yksikkötestit on syytä automatisoida yksikkötestaustyökalulla, jolloin luokan toimivuus muutosten jälkeen on helppo testata. Rajapintojen toiminnalliset testit ovat myös tärkeitä, sillä olio-ohjelmoinnissa rajapintoja on paljon. (Pöyhönen ym, 2002).

3.8 Testauksen automatisointi

Kun suunnitellaan testiautomaation käyttöönottoa, on mietittävä tarkasti mitä kannattaa automatisoida. Muuten on vaarana, että automatisoidaan testejä, joita on helppo automatisoida, mutta joilla ei löydä vikoja. Paras hyöty automaatiosta saadaan, kun laitetaan tietokone tekemään asioita, joihin ihminen ei kykene, kuten esimerkiksi kuormitustestejä. Ilman hyvää testauksen suunnittelua automaation tuloksena voi olla paljon testaustoimintaa, jonka tulokset ovat mitättömät. Testauksen suunnitteleminen ymmärtämättä automaation tarjoamia erityismahdollisuuksia voi aiheuttaa sen, että ne mahdollisuudet jäävät käyttämättä. Onnistuneeseen testiautomaatioon vaaditaan sekä hyviä testauksen suunnittelijoita että hyviä automatisoijia. Tarvitaan erikoisosaamista, jotta kyetään valitsemaan testeistä ne, jotka kannattaa automatisoida ja että automati-

soidut testit saadaan suunniteltua ja toteutettua tarkoituksenmukaisiksi. (Kaner, ym. 2002, 93-94). Kuvassa 3.10 selvitetään testaustyökalujen valintaa ja automatisoidun testauksen elinkaarta. Vaihejako perustuu teokseen “Automated Software Testing: Introduction, Management and Performance”. (Dustin, Rashka, Paul, 2003).



Kuva 3.10 Automatisoidun testauksen elinkaari (Dustin ym. 2003)

Ensimmäisessä vaiheessa tehdään päätös testiautomaation käyttöönotosta testausprojektissa.

Toisessa vaiheessa pohditaan, mikä työkalu hankitaan. Tässä vaiheessa testataan eri vaihtoehtoja ja mietitään, onko työkalun rakentaminen itse varteenotettava vaihtoehto. Kun vaihtoehtoiset työkalut on tutkittu ja pisteytetty sen mukaan, miten ne soveltuva kyseiseen projektiin, valitaan niistä parhaiten soveltuva ja käynnistetään pilottiprojekti.

Kolmannessa vaiheessa analysoidaan nykyinen testauskäytäntö ja etsitään sen kehityskohteita. Valittuun työkaluun tutustutaan lähtemällä liikkeelle testattavan sovelluksen vaatimuksista, tutkimalla sen kelpoisuus testattavan sovelluksen suhteen ja kokeilemalla työkalua käytännössä.

Neljännessä vaiheessa tehdään testaussuunnitelma, jonka sisältönä on muun muassa:

- roolit ja vastuut
- testauksen aikataulutus
- testauksen suunnittelun ja testien suunnittelun toiminnot
- testauksen riskien kartoitus
- testauksen hyväksymiskriteerit

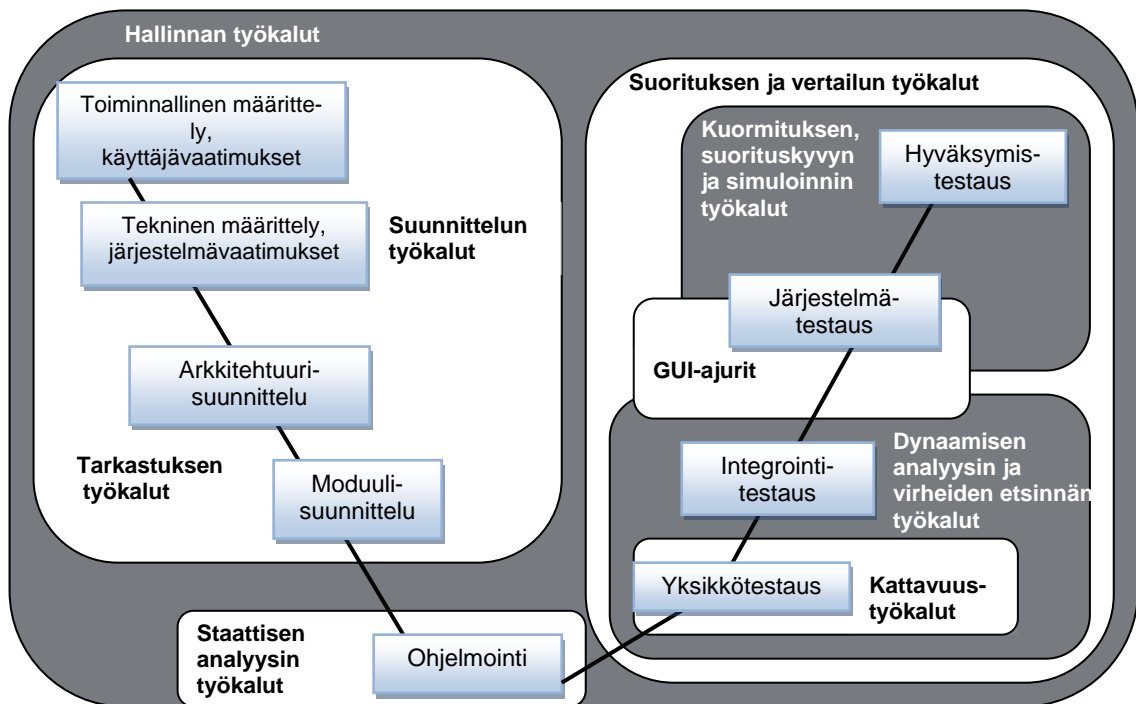
Testaussuunnitelmaan voidaan liittää myös testausprosessit, nimeämiskäytännöt, testauksessa noudatettavat standardit ja testauksen jäljitysmatriisit.

Viidennessä vaiheessa testaustiimi suorittaa suunnitellut yksikkö-, integrointi-, järjestelmä- ja hyväksymistestit pyrkien pysymään suunnitellussa aikataulussa. Kun nämä vaiheet on käyty läpi, ja testaussuunnitelmassa määritellyt hyväksymiskriteerit on saavutettu, on koko järjestelmän testaus suoritettu.

Kuudes ja viimeinen vaihe on testausprosessin arviointi ja kehittäminen. Testaustiimin keräämät tiedot testauksen edistymisestä kerätään yhteen ja katselmoidaan sen tutkimiseksi, kuinka hyvin testaus lopulta vastasi suunnitelmia. On myös syytä kirjata ylös onnistumiset ja toimiviksi todetut käytännöt, jotta niitä voitaisiin hyödyntää tulevilla testausprojekteilla.

3.9 Testaustyökalut

Testaustyökalulla tarkoitetaan sovellusta, joka tukee yhtä tai useampaa testauksen toimintaa, kuten suunnittelua, valvontaa, määrittelyä, testaustietokantojen luomista, testien suorittamista tai niiden analyysia. Automaation perimmäinen tarkoitus on parantaa testausprosessia esimerkiksi ajansäästönä tai mahdollisuutena testata tavoilla, joihin manuaalinen testaus ei kykene. Testaustyökalujen kirjo on laaja. Niiden tyypit ja niiden sijoittuminen testauksen V-mallin tasolle selvitetään kuvassa 3.11.



Kuva 3.11 Testaustyökalujen tyypit (mukaillen Pohjolainen, 2003)

Hallinnan työkaluilla tarkoitetaan kaikkia niitä työkaluja, jotka liittyvät testausprosessin hallintaan. Hallintatyökalut, jotka kattaisivat kaikki testauksen osa-alueet, ovat vasta kehitteillä, ja toisaalta testauksen hallinta voi käyttää samoja projektinhallintatyökaluja kuin muukin ohjelmistoprojekti. Näitä työkaluja ovat erilaiset versionhallintajärjestelmät ja ajanhallintajärjestelmät. WWW-pohjaisten hallintajärjestelmien käyttö mahdollistaa sen, että kaikki testausta käsittelevä tieto on ajasta ja paikasta riippumatta kaikkien projektiin kuuluvien henkilöiden käytettävissä. Testauksen hallinnan työkalut automatisoivat testauksen suunnittelua, testitulosten analysointia, testauksen dokumentointia sekä raportointia.

Suunnittelun työkaluja ovat työkalut, joilla voidaan mallintaa ohjelmistoja. Esimerkiksi UML-mallin perusteella voidaan generoida koodia, ja mallista voidaan johtaa testitapauksia.

Tarkastuksen työkalut sisältävät erilaisia tarkastuskäytäntöjä helpottavia, usein WWW-pohjaisia sovelluksia, jotka mahdollistavat esimerkiksi fyysisesti eri paikoissa olevien henkilöiden osallistumisen katselmointiin.

Staattisen analyysin työkalut tutkivat ohjelmakoodin eri ominaisuuksia suorittamalla itse ohjelmaa. Analyysin kohteena voi olla esimerkiksi standardinmukaisuus, koodin kompleksisuus tai tietoturvaheikkoudet. Staattisen analyysin työkalut ovat usein integroitavissa ohjelmointiympäristöihin.

Suorituksen ja vertailun työkalut ovat testien suorittamisen automaation työvälineitä. Toisinaan on vaikea vetää rajaa, mitkä työkalut tähän kategoriaan kuuluvat, sillä alue on hyvin laaja, kattaen koko testauksen V-mallin oikean haaran kokonaan: esimerkiksi GUI-ajurit kuuluvat osaltaan myös tähän ryhmään. Muita tähän kategoriaan kuuluvia työkaluja ovat esimerkiksi yksikkötestaustyökalut, näytön kaappaus- ja nauhoitussovellukset, SOA-testerit ja vertailuvälineet, tietoturvatestaustyökalut ja kuormitustyökalut.

GUI-ajurit ovat työkaluja, joilla voidaan automatisoida graafisen käyttöliittymän testausta. Niiden toiminta perustuu GUI-karttojen nauhoittamiseen. GUI-kartalla tarkoitetaan tiedostoa, joka kuvaa graafisen käyttöliittymän komponenttien sijainteja sovellusikkunassa.

Dynaamisen analyysin työkalut tarjoavat ajonaikaista tietoa testattavasta ohjelmistosta ja sen resurssien käytöstä. Dynaaminen analyysi on laajimmin automatisoitu testauksen osa-alue, ja työkaluja on paljon. Työkaluilla analysoidaan esimerkiksi muistin käyttöä tai ohjelmiston suorituskyyä.

Kattavuustyökalut ovat apuvälineitä, jotka mittaavat erilaisia koodin kattavuuslukuja, esimerkiksi sitä, mikä osa koodista on katettu testitapauksilla.

Erillisten työkalujen lisäksi testaustyökaluja integroidaan ohjelmointiympäristöihin eli IDE:ihin. Esimerkiksi Microsoft Visual Studio 2010 Ultimate –version ominaisuuksista löytyy muun muassa muutoksenhallinnan, testidatan generoinnin, yksikkötestauksen, kuormitustestauksen, staattisen analyysin ja koodin kattavuuslukujen analysointityökalut sekä projektin versionhallinnan ja testiympäristön luomisen työkaluja (Microsoft, 2009).

4 TYÖN KULKU

Opinnäytetyöni kulku seurasi Saimaan ammattikorkeakoulun opinnäytetyöprosessia. Prosessi koostuu aiheen valinnasta, suunnittelusta, toteutuksesta, raportoinnista ja työn viimeistelystä. Aiheen työhöni sain Saimaan ammattikorkeakoululta, joka opetussuunnitelman muutosten takia tarvitsi materiaalia ohjelmistojen testausta käsittelevälle opintojaksolle. Kun aihe oli valittu, pidettiin aloituspalaveri ja laadittiin projektisuunnitelma ja – aikataulu mukaan lukien opinnäytetyön ohjauspalaverit.

Työ aloitettiin kirjallisuustutkimuksena etsimällä ja lukemalla kirjallisuutta ja www-sisältöä, joka koskee ohjelmistotestausta. Koska aihe oli minulle ennestään tuntematon, vei lukeminen paljon aikaa, sillä halusin tutustua mahdollisimman kattavasti sekä alan kirjallisuuteen että muiden suomalaisten ammattikorkeakoulujen ja korkeakoulujen ohjelmistotestausta käsitteleviin opetusmateriaaleihin. Näin pyrin saamaan selkeän kokonaiskuvan siitä, mitkä asiat olisivat oleellisia mainita opetusmateriaalissa. Lähdekirjoista keskeisimpiä olivat perinteisen ohjelmistotestauksen klassikko ”Art of Software Testing” sekä kontekstiohjatun koulukunnan edustajien kokoama, vahvasti käytännönläheinen teos ”Lessons learned in software testing: a context-driven approach”.

Teoriaosuuden lisäksi tutkin testausohjelmia ja automatisointityökaluja, jotka sopisivat opetustarkoitukseen. Koekäyttämällä ohjelmia tutkittiin, voisiko niitä hyödyntää testauskurssilla harjoitusten läpiviennissä.

Opintomateriaalin laatimisen jälkeen kirjoitin opinnäytetyöraportin ja viimeistelin luentomateriaalin julkaisukelpoiseksi, jotta sen voi liittää opinnäytetyöraporttiin. Raporttia laadin työharjoittelun ohella, sillä hakeuduin opinnäytetyöprosessin loppuvaiheella testaajaharjoittelijaksi ohjelmistoalan yritykseen. Syy tähän oli se, että halusin käytännön kokemusta ohjelmistotestauksesta. Samalla sain mahdollisuuden tutkia, miten oppimateriaaliin keräämäni teoria pätee käytännön testaustyössä.

5 LOPPUTULOKSET

Opintomateriaalin (liite 1) kolme ensimmäistä lukua johdattelevat lukijan ohjelmistotestaukseen ja määrittelevät sen keskeiset termit. Ensimmäinen luku on johdanto, jossa kerrotaan ohjelmistotuotannon erityispiirteistä ja laatunäkökulman tärkeydestä ohjelmistotuotannossa. Toinen luku kertoo testauksen historiasta, ja kolmas luku sisältää testauksen määritelmien ja koulukuntaesittelyiden lisäksi varoittavia esimerkkejä huonosti testattujen ohjelmistojen aiheuttamista vahingoista.

Opintomateriaalin kolme seuraavaa lukua käsittelevät käytännön testausta: neljännessä luvussa selvitetään testauksen sijoittumista ohjelmistoprojektissa, määritellään virheet ja niiden vakavuusluokittelu sekä selitetään testauksen eri vaiheet samassa luvussa esiteltävän ohjelmistotestauksen V-mallin mukaisesti. Viides luku esittelee testauksen käytäntöjä ja tekniikoita ja kuudes luku testausprosessin organisointia ja hallintaa.

Opintomateriaalin kolme viimeistä lukua esittelevät muutamia testauksen erityisilanteita: luku seitsemän esittelee ketterien menetelmien perusperiaatteet ja sen, kuinka ketteriä periaatteita toteutetaan ohjelmistotestauksessa. Kahdeksannessa luvussa selitetään olio-ohjelmointia ja esitellään oliotestauksen erityispiirteitä. Yhdeksäs luku käsittelee testauksen automatisointia, testaustyökaluja ja niiden luokittelua.

Liite 2 koostuu diaesityksistä, jotka seuraavat samaa jaottelua kuin opintomateriaalin liite 1.

6 POHDINTA

Laatimaani opintomateriaalia käytettiin ensimmäisen kerran opetuksessa syyslukukauden 2009 Ohjelmiston testaus –opintojaksolla. Kurssin osallistujilta pyydettiin välikokeiden yhteydessä palautetta materiaalista, jotta mahdolliset virheet tai puutteet olisivat korjattavissa ennen opintojakson seuraavaa toteutusta. Kehitysehdotuksina palautteenantajat kaipasivat opintomateriaalin tueksi valmiita testaukseen liittyviä dokumentteja, kuten testaus- ja testisuunnitelmia sekä testausraportteja.

Ensimmäisessä toteutuksessa yhtenä opintojakson harjoituksena opiskelijat testasivat ja esittelivät valitsemiaan testausohjelmia. Näitä esittelyjä voidaan käyttää hyödyksi opintojakson jatkokehityksessä, sillä kokemuksen perusteella testiympäristöt ja keskeisimmät työvälineet kannattaisi asentaa valmiiksi, jotta aikaa säästyisi muuhun toimintaan. Myös testattavat sovellusohjelmistot kannattaisi asentaa valmiiksi ja suunnitella valmiita testaustehtäviä opiskelijoille. Opintojaksolle tarvittaisiin siis lisämateriaalia ja harjoituksia opetuksen tueksi. Itselläni ei aikaa riittänyt enää niiden laatimiseen teoriaosuuden kirjoittamisen ohella.

Materiaalia kootessani ilmeni, että testaus ja laadunhallinta ovat niin laaja ja oleellinen osa ohjelmistotuotantoa, että 3 opintopisteen opintojakso riittää ainoastaan ohjelmistotestauksen peruskurssiksi. Ohjelmistotestaus sisältää paljon osa-alueita, joista moni olisi oman opintojaksonsa arvoinen, ja tulevaisuudessa on syytä pohtia, mihin suuntaan testauksen ja laadunhallinnan opetusta Saimaan ammattikorkeakoulussa kannattaa kehittää, ja kenties laajentaa. Pohdinnan tukena voisi käyttää kaupallisten testauskurssien sisältöä. Kursseja järjestävät muun muassa FC Sovelto Oy ja Tieturi Oy, ja tällä hetkellä tarjolla on muun muassa testaussertifikaattivalmennuksia sekä testausprosessin kehittämistä, ketterää testausta, tietoturvatestausta, web-sovellusten testausta, suorituskykytestausta ja käytettävyydestestausta koskevia kursseja. Ohjelmistotestaus ja laadunhallinta ovat niin tärkeä osa nykyaikaista ohjelmistotuotantoa, että siitä olisi mahdollisuus luoda oma erikoistumisalansa tietotekniikan koulutusohjelman tietojärjestelmien kehityksen suuntautumisvaihtoehtoon. Testaus ja laa-

dunhallinta olisi syytä soveltuvin osin ottaa huomioon myös muilla opintojaksoilla: yksikkötestausta, sen työkaluja ja TDD:tä voisi opettaa olio-ohjelmointikursseilla, ja koulun Visual Studio 2008 Professional-ohjelmointiympäristöt päivittää testaustyökaluja sisältäviksi Visual Studio 2010 Ultimate-versioiksi. Testausprosessi tulisi myös käydä läpi Järjestelmäprojekti-opintojakson (10 op) toteutuksessa, niin että se tulisi tutuksi laajassa käytännön toteutuksessa.

Testaajaharjoittelussani opinnäytetyöprosessin loppuvaiheessa sain myös vahvistusta ajatukselleni, että Tietojärjestelmien kehitys suuntautumisvaihtoehdon opiskelijoille on ehdottomasti opetettava testaus- ja versionhallintatyökalujen käyttöä Järjestelmäprojekti-opintojaksolla. Työskennellessäni ohjelmistoalan yrityksessä totesin, että toimiva versionhallinta helpottaa projektinhallinnan lisäksi testauksen hallintaa ja parantaa mahdollisuuksia valmistaa hyvälaatuinen ohjelmistotuote.

Testaajaharjoittelussa vakuutuin, että opintomateriaaliin keräämäni asiat ovat hyödyllisiä ohjelmistotestauksen opiskelussa. Perehdytys työpaikalla oli nopeaa, kun peruskäsitteet olivat selvillä ja työtehtävien sijoittuminen kokonaisuuteen oli helppo ymmärtää. Opettavaista oli myös nähdä käytännössä globaalisti hajautettujen ohjelmistonkehitystiimien toimintaa käytännössä. Tiimeissä suurin osa ohjelmointityöstä tehtiin Intiassa, ja suunnittelu- kehitys- ja testaustyö pääosin Suomessa. Harjoittelujakso osoitti käytännössä, että hyvät sosiaaliset taidot, kielitaito ja eri kulttuurien ja tapojen ymmärtäminen ovat teknisen osaamisen ohella tärkeitä taitoja tämän päivän ohjelmistotyöläiselle.

KUVAT

Kuva 2.1 Saimaan ammattikorkeakoulun koulutusalat (Saimaan AMK), s.10

Kuva 2.2 Tietotekniikan koulutusohjelman opetussuunnitelma 2005 (Saimaan AMK, SoleOPS) s.12

Kuva 2.3 Tietotekniikan koulutusohjelman opetussuunnitelma 2009 (Saimaan AMK, SoleOPS) s.13

Kuva 3.1 Ohjelmistotestauksen kehitysvaiheet. (Gelperin, Hezel 1988) s.14

Kuva 3.2 Testauksen koulukunnat (Pettichord 2007) s.16

Kuva 3.3 Ohjelmistotuotannon vesiputousmalli (Royce) s.17

Kuva 3.4 Testauksen V-malli (Taina, 2004) s.18

Kuva 3.5 Staattiset testausmenetelmät (liite 1, s.57) s.20

Kuva 3.6 Dynaamiset testausmenetelmät (liite 1, s.63) s.20

Kuva 3.7 Eri testaustekniikoiden käsitteelliset eroavaisuudet (liite 1, s. 64) s.21

Kuva 3.8 Testauksen vaiheet ja suorittajat (liite 1, s. 87) s.22

Kuva 3.9 Ohjelmistojen rakenteen kehittyminen (Pöyhönen ym. 2002) s.25

Kuva 3.10 Automatisoidun testauksen elinkaari (Dustin ym. 2003) s.28

Kuva 3.11 Testaustyökalujen tyypit (mukaillen Pohjolainen, 2003) s.30

LÄHTEET

Bach, James. 2003. Exploratory Testing Explained.
<http://www.satisfice.com/articles/et-article.pdf>
(Luettu 4.9.2009)

Dustin, Rashka, Paul. 2003.
Automated Software Testing. Introduction, Management and Performance.
Addison-Wesley.

Gelperin, Hezel. 1988. The growth of software testing.
Communications of ACM, vol. 31, No. 6. s.687 – 695
<http://portal.acm.org/citation.cfm?doid=62959.62965>
(Luettu 4.8.2009)

Haikala, Märijärvi. 2000. Ohjelmistotuotanto.
Talentum Media Oy.

Holopainen, Juha. 2005. Regressiotestaus ja testien valintatekniikat.
Pro Gradu-tutkielma, Kuopion yliopisto.
http://www.uku.fi/tike/his/avointa/julkaisut/Regressiotestaus_ja_testien_valintatekniikat.pdf
(Luettu 27.8.2009)

IEEE 1998. 829-1998 Standard for Software Test Documentation.
http://wilma.vub.ac.be/~se1_0607/svn/bin/cgi/viewvc.cgi/documents/standards/IEEE/IEEE-STD-829-1998.pdf?revision=45 (Luettu 4.8.2009)

IEEE 2008. 829-2008 Standard for Software and System Test Documentation.
<http://ieeexplore.ieee.org/servlet/opac?punumber=4578271>
(Luettu 4.8.2009)

ISEB 2009. ISEB Foundation Certificate in Software Testing, overview
<http://www.bcs.org/server.php?show=conWebDoc.2299>
(Luettu 4.8.2009)

ISTQB 2007. Suomennos ISTQB:n testaussanastosta "Standard glossary of terms used in Software Testing Version 2.0"
http://ttlry-fi-bin.directo.fi/@Bin/bf55cf1cfaae232ff73073310d4cc02e/1250671363/application/pdf/14155799/istqb_sanasto.pdf
(Luettu 19.8.2009)

Kaner, Bach, Pettichord. 2002. Lessons learned in software testing: a context-driven approach. Wiley Computer Publishing

Manifesto for Agile Software Development

<http://agilemanifesto.org/>

(Luettu 4.3.2010)

Pettichord, Brett. 2007. Schools of software testing

http://www.io.com/~wazmo/papers/four_schools.pdf

(Luettu 11.8.2009)

Microsoft corporation. 2009. Microsoft Visual Studio 2010 First look.

<http://www.microsoft.com/visualstudio/en-us/products/2010/default.mspx>

(Luettu 12.3.2009)

Pohjolainen, Pentti. 2003. Ohjelmiston testauksen automatisointi.

Pro Gradu-tutkielma, Kuopion yliopisto.

http://www.cs.uku.fi/tutkimus/Teho/PenttiPohjolainen_Gradu.pdf

(Luettu 5.11.2009)

Pöyhönen, Stenberg. 2002. Laajojen oliojärjestelmien testaus-diasarja.

Nokia Research Center, SW Technology Laboratory

<http://www.cs.tut.fi/tapahtumat/olio2002/poyhonen.pdf>

(Luettu 15.9.2009)

Pyhäjärvi, Pöyhönen. 2004. Strategioista suunnitelmiin - selkeyttä käsitteiden sekamelskaan.

testausosy.ttlry.fi/webfm_send/101

(Luettu 4.3.2010)

Royce, Winston W. Managing the development of large software systems

<http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf>

(Luettu 2.3.2010)

Saimaan AMK, Tietotekniikan koulutusohjelman kokous.

\\Saimaa\Projects\Tekniikka\Tietotekniikka\Koulutusohjelman

kokoukset\Suuntautumisvaihtoehdot 9.2.2010.pp

(Luettu 8.3.2010)

Saimaan AMK, Saimaan amk lyhyesti.

<http://www.saimia.fi/fi-FI/tietoja-saimaan-amk/saimaan-amk-lyhyesti>

(Luettu 10.12.2010)

Saimaan AMK, Organisaatiokaavio,

http://www.saimia.fi/fi-FI/images/saimaan_amk_organisaatio.jpg

(luettu 02.03.2010)

Saimaan AMK, SoleOPS, Tietotekniikan koulutusohjelma, ryhmä D171S05.

https://www.saimia.fi/opsnet/disp/fi/ops_KoulOhjOps/tab/tab/sea?ryhma_id=16078027&koulohj_id=16076937&stack=push

(Luettu 2.3.2010)

Saimaan AMK, SoleOPS, Tietotekniikan koulutusohjelma, ryhmä D171S09.

https://www.saimia.fi/opsnet/disp/fi/ops_KoulOhjOps/tab/tab/sea?ryhma_id=16455356&koulohj_id=16076937&stack=push
(Luettu 2.3.2010)

Taina, Juha. 2004. Ohjelmistojen testaus-luentokalvot.
<http://www.cs.helsinki.fi/u/taina/ohte/s-2004/luennot/Luku02.pdf>
(Luettu 3.3.2010)

Watkins, John. 2001. Testing IT: An Off-the-Shelf Software Testing Handbook.
Cambridge University Press