Lappeenrannan teknillinen yliopisto

Teknistaloudellinen tiedekunta

Tietotekniikan osasto

Diplomityö

Lauri Kyttälä

KÄYTÄNNÖN OHJELMOINTI -KURSSIN KEHITTÄMINEN

Työn tarkastajat: Professori Kari Smolander

Tekniikan tohtori Uolevi Nikula

Työn ohjaaja: Tekniikan tohtori Uolevi Nikula

TIIVISTELMÄ

Lappeenrannan teknillinen yliopisto

Teknistaloudellinen tiedekunta

Tietotekniikan osasto

Lauri Kyttälä

Käytännön ohjelmointi -kurssin kehittäminen

Diplomityö

2011

107 sivua, 10 kuvaa, 2 taulukkoa, 2 liitettä

Tarkastajat: Kari Smolander

Uolevi Nikula

Hakusanat: ohjelmointi, ohjelmointikielet, luennot

Keywords: programming, programming language, lectures

Tämän tutkimuksen tavoitteena on selvittää opintojensa alussa olevien yliopisto-opiskelijoiden vaikeimpina pitämät käytännön ohjelmoinnin aihealueet sekä koostaa luentomoniste käytettäväksi seuraavalla alkavalla Käytännön ohjelmointi -kurssilla.

Tutkimusmetodina käytettiin konstruktiivista tutkimusmetodia, jossa tavoitteen spesifioinnin jälkeen implementoitiin luentomoniste koostamalla määriteltyjen aihekokonaisuuksien lähdemateriaalia yhtenäiseksi, luettavaksi kokonaisuudeksi.

Yliopistoissa ei yleisesti opeteta ohjelmistojen testausta ennen syventäviä ohjelmistotekniikan kursseja, mikä on kuitenkin puute työelämän kannalta. Tässä työssä esitetään perusteluja käytännönläheisten aihekokonaisuuksien painottamiselle ohjelmointikursseilla jo yliopisto-opintojen alkuvaiheessa.

Työssä käsitellään Käytännön ohjelmointi -kurssin kurssipalautetta, missä havaittiin opiskelijoiden pitävän kurssin hankalimpina aihealueina linkitettyä listaa, osoittimia, dynaamista muistinhallintaa, tietorakenteita ja versionhallintaa.

Työn avulla on pyritty kehittämään käytännön ohjelmoinnin yliopisto-opetusta Lappeenrannan teknillisessä yliopistossa luentomateriaalin avulla, jossa on muun muassa teoriaa, keskeisiä opiskelijoiden tarvitsemia komentoja, www-linkkejä sekä ohjelmoinnin tyyliopas.

ABSTRACT

Lappeenranta University of Technology

Faculty of Technology Management

Department of Information Technology

Lauri Kyttälä

The Development of Practical Programming Course

Master's thesis

2011

107 pages, 10 figures, 2 tables, 2 appendixes

Examiners: Kari Smolander

Uolevi Nikula

Keywords: programming, programming language, lectures

The aim of this Master's thesis is to investigate the early studies of university students in order to find the most difficult practical programming topics to learn, and to compile the study materials to be used in the next Practical Programming course.

The method used for this thesis was constructive research, where the objective was to implement lecture notes by combining the defined source material into a contiguous and readable form.

In universities it is not common to teach software testing before advanced software engineering courses, but this is a lack while having the working life in mind. This thesis provides a justification on emphasis of practical matters on programming courses in the early stages of university studies.

This thesis deals with the feedback given by the students who attended the course of Practical Programming. The topics which the students found the most demanding were linked list, pointers, dynamic memory management data structures and version control.

This thesis has aimed at developing the teaching of practical programming at Lappeenranta University of Technology with the help of lecture material, which covers among other things theory, some centric commands, www-links and a style guide for the C-language used in the course.

Alkusanat

Haluan kiittää Professori Kari Smolanderia tämän diplomityön aiheen tarjoamisesta minulle. Kiitän myös Tekniikan tohtori Uolevi Nikulaa asiantuntevasta työn ohjaamisesta ja neuvoista.

Kiitän kaikkia läheisiäni antamastanne tuesta ja kannustuksesta kesän aikana, tämän työn edetessä pikkuhiljaa kohti päätöstään.

Vantaalla 12.10.2011

Lauri Kyttälä

SISÄLLYSLUETTELO

LIITTEET

LYHENTEET

ACM	Association for Computing Machinery
ANSI-C	American National Standards Institute C-kielen standardi
CS1	Computer Science 1
gcc	GNU Compiler Collection -kääntäjä
GNU	Akronyymi sanoista GNU’s Not Unix
HOPS	Henkilökohtainen opintosuunnitelma
LTY	Lappeenrannan teknillinen yliopisto
op.	opintopiste
PK-sektori	Pieni ja keskisuuri sektori
SA	Structured Analyses
SD	Structured Development
SVN	Subversion, versionhallinta
TDD	Test Driven Development
TTY	Tampereen teknillinen yliopisto
UCA	Usability Capability Description
UML	Unified Modeling Language

1 Johdanto

1.1 Taustaa

Käytännön ohjelmointi -kurssi on Lappeenrannan teknillisen yliopiston ensimmäisen vuoden tietotekniikan opiskelijoille suunnattu laajuudeltaan viiden opintopisteen käytännönläheinen, tekemisen kautta opettava kurssi. Kurssin osallistujilta odotetaan Ohjelmoinnin perusteet-kurssin suoritusta, mutta se ei kuitenkaan ole ehdoton vaatimus. Opiskelijoilla oletetaan kuitenkin olevan ohjelmoinnin perustiedot, jotka he ovat hankkineet esimerkiksi Python-ohjelmointikielellä.

Tämän diplomityön tavoitteena oli selvittää kurssin suorituspisteiden vastaavuutta substanssiin nähden ja tehdä luentomoniste, konstruktio, Käytännön ohjelmointi -kurssille. Tälle konstruktiolle tehtiin empiirinen validointi, jonka suoritti ohjaaja toimiessaan kurssin luennoitsijana. Tutkimuksen tarkoituksena oli lisäksi kartoittaa käytännönläheisten ohjelmointi-kurssien sisällöllisiä ja metodisia eroja Suomen eri yliopistojen välillä, sekä perustella tehtyä konstruktiota havaituilla argumenteilla.

Ohjelmistoteollisuuden viimeaikainen nopea kehittyminen Suomessa ja sen myötä ohjelmistokokojen jatkuva kasvaminen ovat luoneet tarvetta testaamisen ja analysoinnin tarkempaan huomioimiseen jo koulutusvaiheessa. Kehittämällä ohjelmistotekniikan kursseja työelämälähtöisesti on saatava hyöty tulevaisuudessa mitattavissa ennen kaikkea PK-sektorin ohjelmistoyrityksissä.

Lähtökohtana tehdylle konstruktiolle oli tarve lisätä käytännön tekemistä kurssin sisältöön. Tätä tavoitetta lähestyttiin kahdella konkreettisella tavalla, lisäämällä raportoitava moduulitestaus integrointitestausvaiheen yhteyteen ja ottamalla käyttöön ohjelmiston versionhallinta. Moduulitestauksen ja versionhallinnan käyttäminen kurssilla antaa opiskelijoille konkreettisen käsityksen ohjelmistoprojektin kehitysvaiheesta jo opintojen

alkuvaiheessa, ennen syventäviä ohjelmistotekniikan opintoja. Lisäksi konstruktion avulla pyritään parantamaan ohjelmointikurssin rakennetta (Nikula et al. 2009) ja tarjoamaan opiskelijoille yhtenäinen oppimateriaali käytettäväksi yhdessä oppaan, C-kieli ja käytännön ohjelmointi osa 1, kanssa (Kasurinen 2011).

1.2 Työn rakenne

Luku 2 koostuu kirjallisuuskatsauksesta, minkä lisäksi siinä käsitellään käytännön ohjelmistotekniikan opetusta vertaillen Suomen yliopistoja, niissä käytettäviä ohjelmointikieliä sekä järjestelyjä yleisesti. Luku 3 koostuu tutkimusmetodista sekä hypoteesista ja luku 4 Lappeenrannan teknillisen yliopiston Käytännön ohjelmointi -kurssin rakenteesta. Luvussa 5 käsitellään konstruktion rakennetta sekä sisältöä, siitä saatua palautetta ja kuinka se otettiin huomioon konstruktiota edelleen muokattaessa paremmin opetuskäyttöön sopivaksi. Työn lähtökohtana olleen olettamuksen testaus perustuu vuoden 2011 kurssista saatuun opiskelijapalautteeseen, jota käsitellään kymmenen kysymyksen avulla.

2 Kirjallisuuskartoitus

Tämän diplomityö rakentuu keskeisesti koostetun oppimateriaalin, konstruktion, ympärille. Koska C-kielestä ei ollut saatavilla hyvää suomenkielistä oppikirjaa, eikä Käytännön ohjelmointi -kurssin toisen periodin oppisisällöstä ollut yhtenäistä materiaalia, oli selkeä tarve tehdä oma rajattuun kokonaisuuteen keskittyvä oppimateriaali jälkimmäiselle periodille.

2.1 Lähdekirjallisuus

Konstruktion implementointiprosessi edellytti kirjallisuuskatsausta ja lähdemateriaalien koostamista. Koska konstruktion aihealue oli melko laaja, asetti se puolestaan lähtökohdat materiaalien keräämiselle, toisaalta ajankäyttö ja resurssit rajoittivat konstruktiossa käytettävien teosten määrää.

Kurssimateriaalin sisältö pyrittiin rakentamaan mahdollisimman hyvin tukemaan opiskelijoiden kehittyviä tietotekniikan taitoja noudattaen muun muassa ACM:n (Association for Computing Machinery) julkaisujen Computing Curriculum 2001 (The Joint Task Force on Computing Curricula 2001) ja Computer Science – Computer programming (Koffman 1984, Koffman 1985) runkoja.

Lähdemateriaali valittiin Computing Curriculum'n (2001) CS1 ja CS2 (Computer Science) kurssiaiheisiin perustuen ja käytetyt hakusanat olivat: debuggaus, dokumentointi, driver (ajuri), katselmointi, moduulien käyttö, ohjelmointityyli, stub (tynkä), suunnittelu, tarkastus, testaus ja testidata. Materiaali, jota kertyi melko runsaasti, oli pääsääntöisesti kirjoina mutta lisäksi kurssi- ja tutkimusmateriaaleina. Runsasta lähdemateriaalia piti karsia jo alkuvaiheessa lähinnä materiaalien ymmärrettävyyteen ja havaittuihin viittausmääriin perustuen. Aihekokonaisuuksista pyrittiin käyttämään mahdollisuuksien mukaan suomenkielistä materiaalia sen helpomman referoitavuuden tähden, mutta tämä onnistui hyvin harvoin, olihan lähdemateriaali lähes kokonaan englanninkielistä. Poikkeuksena

mainittakoon teokset Ohjelmistotuotanto (Haikala 2004) ja Ohjelmistoarkkitehtuurit (Koskimies 2005). Suomenkielistä materiaalia löytyi myös muutamasta diplomityöstä ja tohtorinväitöksestä. Tämä lähdekirjallisuuden kokoaminen ja läpikäyminen oli kiistatta yksi tämän diplomityön haastavimmista ja työllistävimmistä vaiheista.

2.2 Käytännön ohjelmointi -kurssit ja ohjelmistotestaus eri yliopistoissa

Alla oleva selvitys on tehty Suomen teknisen alan yliopistojen opetussuunnitelmien 2010 - 2011 pohjalta. Käytännön ohjelmointi -kurssien ja CS2 ohjelmistotestaus-kurssien sisältöjä sekä opetusmetodeja on vertailtu keskenään siltä osin kun tarkempaa tietoa oli saatavilla. Kerätyn materiaalin pohjalta muotoutui käsitys, että vain kahdessa yliopistossa kolmesta-toista, Lappeenrannan ja Tampereen teknillisissä yliopistoissa, opetetaan kandidaatin tutkintoon kuuluvilla kursseilla ohjelmien testausta. Ohjelmointitekniikan diplomi-insinööriopinnoissa, erillisiä ohjelmiston testauskursseja puolestaan on pääsääntöisesti kaikissa yliopistoissa, paitsi Lappeenrannan teknillisessä yliopistossa ja Lapin yliopistossa. Suomen eri yliopistojen ohjelmointikurssit, joissa opetettiin ohjelmatestausta lukuvuotena 2010 – 2011, esitetään seuraavana taulukkona (ks. taulukko 1):

Taulukko 1. Käytännön ohjelmointi- ja ohjelmistotestaus-kurssit eri yliopistoissa.

	Kand.	**DI**	**ohj.kieli**	**op.**	**Kurssin nimi**
Aalto-yliopisto	-				-
		X	Java, C++	5	Ohjelmien testaus ja laadunvarmistus
Helsingin yliopisto	-				-
		X		4	Ohjelmistoprosessit ja ohjelmien laatu
Itä-Suomen yliopisto	-				-
		X		4	Johdatus testaukseen
		X	Java, C#	3	Ohjelmistokehitysvälineet
Jyväskylän yliopisto	-				-
		X	Java	5	Ohjelmistotestaus
		X	Java	5	Ohjelmistojen test. ja laadunvarmistus
Lapin yliopisto	-				-
		-			-
LTY	X		C	5	Käytännön ohjelmointi
		X		7	Software Quality Processes and Organizations
Oulun yliopisto	-				-
		X		3	Ohjelmistojen testaus
		X		3	Ohjelmistojen laatu ja laatutekniikat
		X		5	Ohjelmistoprosessin parantaminen
TTY	X		C++	4	Ohjelmointi 1
		X	Java	5	Ohjelmien testaus
Tampereen yliopisto	-				-
		X		5 / 6	Testing Security and Trust
		X		5	Software Tools and Evaluation
Turun yliopisto	-				-
		X		5	Ohjelmistotestaus
Vaasan yliopisto	-				-
		X	Java	3 / 5	Ohjelmistotestaus
Åbo Akademi	-				-
		X		5	Software Quality
		X		5	Software Testing

Aalto-yliopistossa Tietotekniikan osastolla oli Ohjelmistotekniikan syventävässä moduulissa kurssi, Ohjelmistojen testaus ja laadunvarmistus (5 opintopistettä) (Aalto-yliopisto 2010, 37). Aalto-yliopistossa Ohjelmoinnin perusteet opetettiin Javalla ja jatkokurssi joko Javalla tai C++:lla, kurssien tarkemmasta rakenteesta ei ollut mainintaa opinto-oppaassa. (Aalto-yliopisto 2010, 83)

Helsingin yliopistossa Tietojenkäsittelytieteen laitoksella oli keväällä 2011 kurssi Ohjelmistoprosessit ja ohjelmistojen laatu (4 op.), joka on tietojenkäsittelytieteen maisteriopintojen valinnainen ohjelmistojärjestelmien erikoistumislinjan kurssi (Helsingin Yliopisto 2010, 44).

Itä-Suomen yliopistossa oli kurssi nimeltään Johdatus testaukseen 4 op. (Introduction to Software Testing), sekä Ohjelmistokehitysvälineet (Software Developmnet Tools) (Itä-Suomen yliopisto 2010, 175).

Jyväskylän yliopistossa oli kurssi Ohjelmistotestaus 5 op. (Jyväskylän yliopisto 2010). Jyväskylän yliopistossa Ohjelmointi I kurssi käytti vaihtoehtoisesti joko Java- tai C#-ohjelmointikieltä (Jyväskylän yliopisto 2010, 130-132).

Lapin yliopistossa Informaatioteknologiaa voi opiskella sivuaineena perusopintojen (25 op.) verran, kurssit eivät sisältäneet ohjelmistotekniikkaa eivätkä ohjelmistotestausta (Lapin yliopisto 2010, 31).

Lappeenrannan teknillisessä yliopistossa Ohjelmoinnin perusteet-kurssin opetuskielenä on ollut Python. Käytännön ohjelmointi -kurssi, joka on suunniteltu jatkokurssiksi edelliselle, opetettiin C-kielellä, kurssin sisältöön kuului ohjelmien testaus. Maisteriopinnoissa oli kurssi, Software Quality Processes and Organizations, jossa ryhmätyönä toteutetaan ja dokumentoidaan ohjelmistoprojekti (7 op.) (Lappeenrannan teknillinen yliopisto 2010, 418-423).

Oulun yliopistossa oli ohjelmistotekniikan kursseja runsaasti tarjolla. Yksityiskohtaiset kurssikuvaukset löytyivät vain WebOodista joten käytössä olleet tiedot olivat siltä osin puutteelliset. Ohjelmistotuotannon suuntautumisopinnoissa oli tarjolla muun muassa seuraavat kurssit; C-ohjelmointi (4 op.), C++-kielen perusteet (6 op.), Ohjelmistojen testaus (3 op.), Ohjelmiston laatu ja laatutekniikat (3 op.) sekä Ohjelmistoprosessin parantaminen (5 op.). (Oulun yliopisto 2010, 232-233)

Tampereen teknillisessä yliopistossa (TTY) kurssilla Ohjelmointi I (4 op.) luentoaiheena oli pienten ohjelmien suunnittelu ja testaus, Ohjelmointi II-kurssi (5 op.) puolestaan käsitti muun muassa aiheet modulaarinen suunnittelu, dynaamiset tietorakenteet ja työkaluohjelmat. Opetuskielenä molemmilla kursseilla oli C++. TTY:ssä oli lisäksi testauksesta oma kurssi, Ohjelmistojen testaus (5 op.), jolla käsiteltiin muun muassa dynaamisen testauksen tekniikat, yksikkötestauksen ja järjestelmätestauksen tärkeimmät työkalut sekä koodikattavuus ja sen mittaaminen. (Tampereen teknillinen yliopisto 2010)

Tampereen yliopistossa Tietojenkäsittelyopin perusopinnoissa Lausekielinen ohjelmointikurssilla (9 op.) ja pakollisiin opintoihin kuuluvalla Olio-ohjelmoinnin-perusteet kurssilla (6 op.) opetuskielenä oli Java, Ohjelmoinnin tekniikka kurssilla (3-6 op.) opetuskielet olivat C ja C++. Kursseilla ei käsitelty ohjelmistojen testausta (Tampereen yliopisto 2010). Ohjelmistotekniikan syventävät opintojaksot Testing, Security and Trust (5–6 op.) ja Software Tools and Evaluation (5 op.) kurssit koostuivat luennoista ja seminaareista, käsitellen ohjelmistojen testaukseen liittyviä teemoja.

Turun yliopistossa oli Tietojenkäsittelytieteiden koulutusohjelmassa Algoritmien ja ohjelmoinnin peruskurssi (5 op.) sekä Olio-ohjelmoinnin perusteet kurssi (5 op.) joiden opetuskielenä oli Java. Tietotekniikan, elektroniikan ja tietoliikennetekniikan koulutusohjelmasta löytyi kurssi Ohjelmistotestaus (5 op.). (Turun yliopisto 2010)

Vaasan yliopistossa opetussuunnitelmaan kuului seuraavat ohjelmointikurssit; Ohjelmointi (5 op.), The Basics of C-programming, Olio-ohjelmointi (5 op.), sekä Ohjelmistotestaus

(3 / 5 op.). Kurssien tarkemmasta rakenteesta ei ollut mainintaa opinto-oppaassa. (Vaasan yliopisto 2010)

Åbo Akademissa järjestettiin muun muassa kurssit; Programmering i C/C++ (5 op.), Programmering grundkurs (5 op.) opetuskielenä Gamma, Software Quality (5 op.) ja Software testing (5 op.). Huomioitavaa oli että Datorteknik-kurssilla (5 op.) opetuskielenä oli Cobol ja Systemdesign grundkurs-kurssilla (5 op.) Fortran. (Åbo Akademi 2010)

2.3 Päätelmä yliopistojen kurssitarjonnasta

Verrattaessa ohjelmistotekniikan kurssitarjontaa on havaittavissa, että yliopistoilla on pääsääntöisesti erilliset kurssit ohjelmistotestauksesta. Tämä menettely on sikäli perusteltu että ohjelmien testaus yhtenä aihekokonaisuutena on niin laaja.

Perusteita ohjelmistotestauksen laajemmalle opettamiselle on olemassa Suomen ohjelmistoteollisuuden jatkuvan kehittymisen ja kasvun tähden. Tuotannon laajentumisen myötä myös itse ohjelmistojen koot ja arkkitehtuuri kasvavat niin suuriksi, että niiden hallinta ja testaus tulee viemään yhä enemmän resursseja. Samoin ohjelmistojen kattava testaus tulee näin välttämättömäksi ja korostetusti esiin, jotta laatu saadaan pysymään korkealla tasolla.

3 Konstruktiivinen tutkimusote

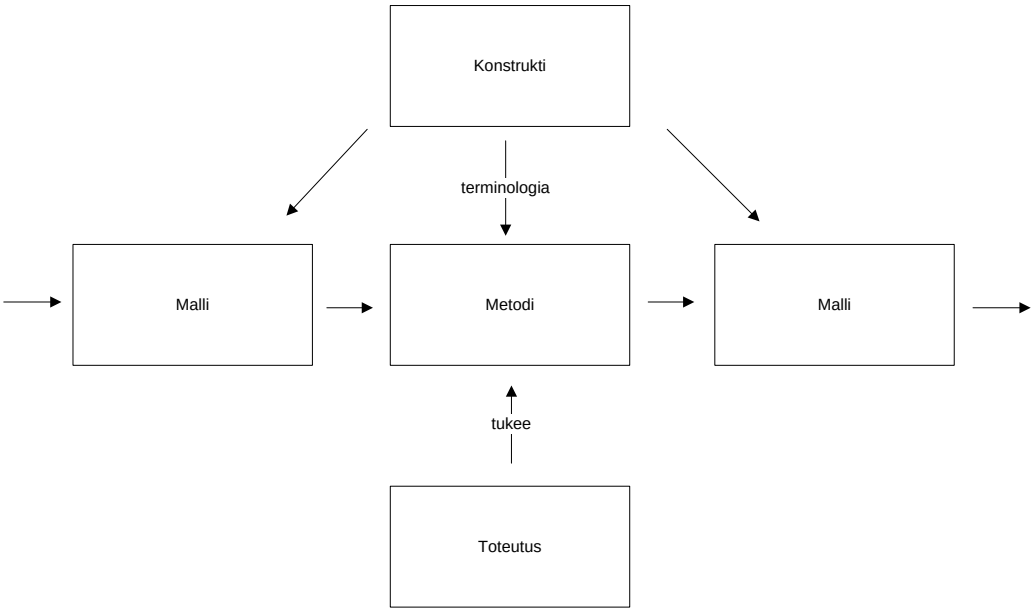
March ja Smith (1995) kuvaavat artikkelissaan informaatiotekniikan kahta tutkimusmetodia seuraavasti: luonnontieteet (natural science) pyrkivät kuvaamaan laajempia kokonaisuuksia tai ilmiöitä ja suunnittelutieteissä (design science) luodaan toteutuksia sekä metodeja. Suunnittelutieteen tuotteet March ja Smith jakavat neljään luokkaan: konstruktit, mallit, metodit ja toteutukset. Tutkimustoiminnan he jakavat seuraavasti: rakentaminen, arviointi, teorian luominen ja teorian testaus (March 1995, 253). He tunnistavat siis neljä kuvausta tutkimuksen tuloksellisuudesta sekä tutkimustoiminnasta ja esittävät tutkimuksen luokittelun seuraavana taulukkona (ks. taulukko 2).

Taulukko 2. March & Smith'n tutkimuskehys (March 1995, 255).

		Tutkimustoiminnot			
		Rakentaminen	Arviointi	Luoda teoriaa	Testata teoriaa
Tutkimus-suoritteet	Konstruktit				
	Mallit				
	Metodit				
	Toteutukset				

Konstruktit (käsitteistöt), mallit, metodit ja toteutukset ovat kukin tutkimussuoritteita jotka osoittavat jonkin tutkimustoiminnan. Rakentaminen ja arviointi ovat suunnittelutieteiden tutkimustoimintaa tarkoituksena parantaa suorituskykyä. Teorian luominen ja testaus ovat luonnontieteiden tutkimustoimintaa tarkoituksena erottaa yleistietoa ehdottamalla ja testaamalla teorioita. (March 1995, 260)

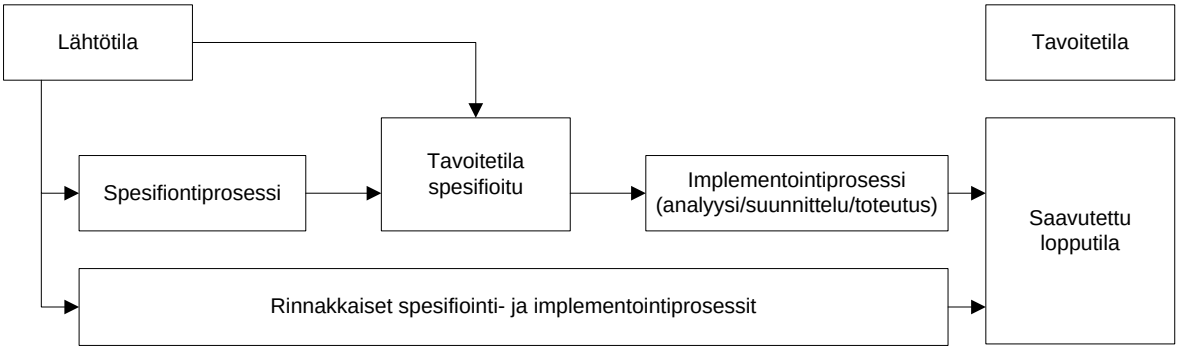
Suunnittelutiede käsittää yritykset luoda malleja, jotka palvelevat ihmisten tarpeita, joten mallit arvioidaan niiden hyödyntämisen kannalta. Tätä edellä mainittua kutsutaan myös *konstruktiiviseksi tutkimukseksi*. March & Smith'n (1995) mukaan edistystä on saavutettu suunnittelutieteen avulla, kun nykyinen teknologia korvataan tehokkaammilla. Sen sijaan että suunnittelutieteen tutkijat tukisivat teorioita, he pyrkivät luomaan artefakteja (innovaatioita) joiden toteutukset ovat uusia ja arvokkaita. Jokela (2001, 23) on esittänyt March ja Smith'n taulukon hieman yksinkertaistetummassa muodossa (ks. kuva 1).



Kuva 1. Riippuvuudet eri toteutusten välillä (Jokela 2001, 24).

Tehtäessä käytettävyyden ja ominaisuuksien arviointi UCA-tutkimuskehyksen (Usability Capability Description) mukaisesti tulisi konstruktion olla täydellinen, yksinkertainen ja helposti ymmärrettävä. Mallien tulisi olla täydellisiä, kestäviä ja johdonmukaisia sekä metodien toimivia ja tehokaita. (Jokela 2001, 30)

Konstruktiiviselle tutkimukselle on siis luonteenomaista uuden rakentaminen olemassa olevan tiedon pohjalta, samalla päättäen mitä uutta halutaan luoda. Tutkijat ja päätöksentekijät määrittelevät tavoitetilan mihin pyritään. Konstruktiota arvioidaan sen käyttäjäyhteisölle tuottaman arvon ja hyödyn perusteella (Järvinen 2004, 105). Konstruktiivinen tutkimusmetodi oli siten tässä diplomityössä perusteltu ja luonnollinen valinta kun tavoitteena oli koostaa lähdemateriaalista konstruktio, luentomoniste, käytettäväksi Käytännön ohjelmointi -kurssilla. Konstruktion innovaatio eteni käytännössä kuvan 3 mukaisesti. Aluksi rakennettiin aiherunko seitsemälle luentokerralle ja hyväksytettiin se ohjaajalla, joka toimi kurssin luennoitsijana keväällä 2011. Prosessimalli erottaa spesifionnin implementoinnista millä korostetaan tavoitetilaa (Järvinen 2004, 108). Voidaan siis todeta, että konstruktio on innovaation realisointi joka on jaettu osaprosesseihin esimerkiksi kuvan 2 mukaisesti.



Kuva 2. Innovaatioprosessi (Järvinen 2004, 108).

3.1 Hypoteesi

Hypoteesi, joka muodostettiin helmikuun alussa 2011, oli olettamus johon perustuen muodostettiin Käytännön ohjelmointi -kurssin toisen periodin luentokokonaisuus.

> ***Hypoteesi***: luentokokonaisuuden tulee sisältää keskeiset käytännönläheisen ohjelmoinnin osa-alueet kuten arkkitehtuuri, suunnittelu, hyvä ohjelmointityyli, kooditarkastus ja testaus.

Hypoteesi muodostettiin tukemaan kurssin kehitystä, jotta opiskelijoilla olisi kurssin suoritettuaan paremmat valmiudet käytännön työelämään. Hypoteesin seurauksena luentokerrat muotoutuivat seuraavanlaisiksi:

- Luento 8: Arkkitehtuuri
- Luento 9: Suunnittelu
- Luento 10: Hyvä ohjelmointityyli
- Luento 11: Kooditarkastus
- Luento 12: Testaus1
- Luento 13: Testaus2
- Luento 14: Yhteenveto, kertaus, lopetus.

3.2 Hypoteesin testaus

Asetettu hypoteesi validoitiin opiskelijapalautteen avulla. Hypoteesin pohjalta etsittiin evidenssiä opiskelijapalautteesta (ks. luku 5.5), oliko keväällä 2011 käytössä oppimateriaalia riittävästi ja oikeista aihekokonaisuuksista. Tämän pohjalta luentokokonaisuus ja tehtävä konstruktio tarkentuivat asiasisällöltään kevään aikana. Opiskelijapalaute ja lopullinen hypoteesin testaus konstruktiosta toteutuu seuraavan kurssin yhteydessä keväällä 2012 järjestettävässä opiskelijakyselyssä. Tämä konstruktion lopullinen testaus ei ole osana tätä työtä, mutta käytännön kannalta se on tärkeä.

4 LTY:n opetusympäristö ja kurssin rakenne

Tämä diplomityö on tietotekniikan diplomityö eikä käsittele ohjelmointikurssin pedagogiikkaa lähemmin. Mainittakoon tässä yhteydessä kuitenkin Kirsi Ala-Mutkan tutkimusartikkeli A Study of the Difficulties of Novice Programmer (Ala-Mutka 2005) ja Milnen tutkimus Difficulties in Learning and Teaching Programming (Milne 2002), joita olen käyttänyt referenssiaineistoina tarkastellessani kevään 2011 kurssipalautetta sekä eri ohjelmointikielten käyttöä.

4.1 Tekninen opetusympäristö

Käytännön ohjelmointikurssilla on perinteisesti käytetty joko Windows-ympäristössä avoimen lähdekoodin Codeblocks-kehitysympäristöä (CodeBlocks) joka sisältää kääntäjän ja debuggerin tai Ubuntu Linux-ympäristöä, jonka käyttöliittymä on muokattavissa lisäsovelluksilla kuten esimerkiksi Anjuta.

Windows- tai Linux-ohjelmointiympäristöillä on omat kannattajansa, kyse on lähinnä opetuksen tavoitteista, kumpi ympäristö valitaan. Codeblocks 10.05 kehitysympäristöön on integroitu periaatteessa kaikki tarpeellisimmat toiminnot, eikä käyttäjän tarvitse modifioida sitä, kun taas Linuxin kehitysympäristö vaatii joitain asennuksia ja sallii variaatioita valintojen suhteen.

Valittaessa Linux kehitysympäristöksi voidaan käyttöjärjestelmän asennus tehdä kahdella luotettavaksi todetulla tavalla: asentamalla Windowsiin virtuaalilaatikko, jossa Linuxia ajetaan, tai asentamalla se niin kutsuttuna dual-boottina Windowsin rinnalle.

4.2 Ohjelmointikurssin substanssi

Opintojen laajuus määritellään opintopisteinä. Opintojaksot pisteytetään niiden edellyttämän työmäärän mukaan siten, että yhden lukuvuoden opintojen suorittaminen edellyttää keskimäärin 1600 tunnin työpanosta, mikä vastaa 60 opintopistettä. Yksi opintopiste vastaa keskimäärin opiskelijan 26 työtuntia. (Lappeenrannan teknillinen yliopisto 2010, 19)

Opinto-oppaan mukaan ehdottomia vaatimuksia kurssiin osallistuville ei ollut, suositeltiin kuitenkin Ohjelmoinnin perusteet-kurssia. Opiskelijoiden lähtötaso saattoi siis vaihdella kurssilla suhteellisen paljon, koska opiskelija voi olla ensimmäisen vuosikurssin tietotekniikan opiskelija, sivuaineopiskelija, jatko-opiskelija tai avoimen yliopiston opiskelija. Opiskelijoiden lähtötaso, tarkoittaen ohjelmoinnin osaamistasoa kurssin alkaessa, vaikuttaa määräävästi heidän käsitykseensä kurssin vaativuudesta. Esimerkkinä voidaan mainita 123 opiskelijan ilmoittautuminen Lappeenrannan teknillisen yliopiston Käytännön ohjelmointi -kurssille keväällä 2011, kuitenkin vain 45 opiskelijaa teki kaikki pakolliset harjoitustehtävät (ks. luku 5.5).

Käytännön ohjelmointi -kurssin luennoitsijan vaihduttua lukuvuosien (2008-2010) jälkeen oli kurssin sisällön tarkistaminen luonnollista ja sille nähtiin myös tarvetta. Ohjelmoinnin perusteet-kurssin kehittämisestä on useampia Lappeenrannan teknillisessä yliopistossa tehtyjä tutkimuksia ja tieteellisiä artikkeleita, (Nikula 2011, Nikula et al. 2009, Kasurinen 2008). Aiemmin tehtyjen tutkimusten havaintoja ja tuloksia on siis tarkoituksellista hyödyntää soveltuvin osin muokattaessa Käytännön ohjelmointi -kurssia. Ajateltaessa ohjelmoinnin peruskurssien valikoimaa Lappeenrannan teknillisessä yliopistossa, on kurssien sisällöissä ja nimeämisissä tapahtunut vuosien myötä muutoksia. Käytännön ohjelmointi -kurssia vastasi vuoden 2006 - 2007 opintosuunnitelmassa Tietorakenteet ja C-kieli, jolloin jatkokurssina oli Algoritmien suunnittelu (nykyään Tietorakenteet ja algoritmit). (Lappeenrannan teknillinen yliopisto 2006, 284)

Kuinka pitkälle opiskelijoiden osaamistason vaihtelu sitten tulisi ottaa huomioon kurssin rakenteessa? Oppimistavoite opinto-oppaan mukaan oli se, että kurssin suoritettuaan opiskelijalla on tietyt perusvalmiudet ohjelmoinnissa ja perusosaaminen ainakin seuraavissa aihealueissa: osoittimet, rekursio, dynaaminen muistinhallinta, virheenjäljittimet ja versionhallinta. Sisällöksi opinto-oppaaseen oli kirjattu C-ohjelmointikielen rakenteet ja kielioppi, ohjelmoinnin työkalut, modulaarisuus, kytkentä ja koheesio, suunnittelu, toteutus sekä testaus (Lappeenrannan teknillinen yliopisto 2010, 418). Opiskelijoiden osaamistason vaihtelu ei ollut osana testattavaa hypoteesia, vaan opiskelijoiden oletettiin olevan ensimmäisen vuosikurssin tietotekniikan opiskelijoita.

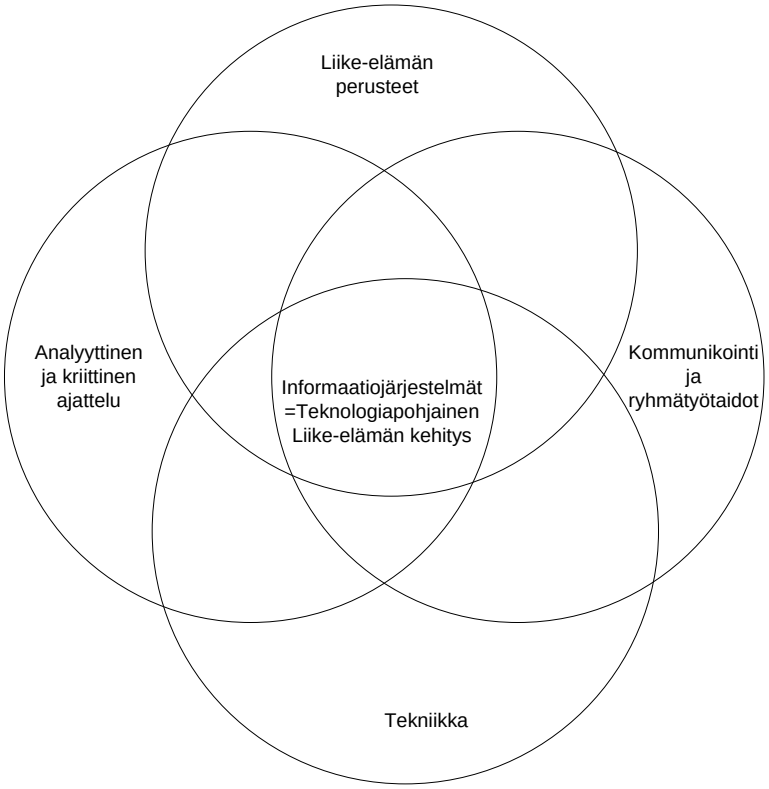
4.3 Yliopistot vs teollisuus

Ohjelmistoteollisuus on viime vuosina kehittynyt ja kansainvälistynyt Suomessa nopeasti. Ohjelmoinnin alihankinta ja kilpailutus on isommilla ohjelmistopalveluja käyttävillä yrityksillä ollut käytäntönä jo useiden vuosien ajan, mistä aiheesta on kirjoitettu myös alan lehdissä melko runsaasti. Millä tekijöillä suomalainen ohjelmistoteollisuus tulee sitten menestymään yhä kansainvälistyvillä markkinoilla ja miten kilpailukykyä voitaisiin vahvistaa?

Tekesin tutkimusraportissa Lester ja Sotarauta (2007) kuvaavat yliopistojen valmiutta vastata koulutustarpeisiin yritysten sekä muiden organisaatioiden kyvyllä sopeutua jatkuvasti muuttuvaan toimintaympäristöön ja pysymiseen mukana teknologisessa kehityksessä. Samaa tahtia innovaatiotoiminnan tiivistymisen kanssa myös yliopistoihin kohdistuvat odotukset ovat kasvaneet. (Lester 2007, 32) Keskeisinä tekijöinä Lester ja Sotarauta pitävät:

- *koulutusta*
- *uuden tiedon tuottamista*
- *paikallisen tieteellisen teknologisen ongelmaratkaisukapasiteetin vahvistamista*
- *tulkitsevien tilojen tarjoamista tulevaisuuksien etsintään (esimerkiksi työtilat, konferenssit).*

Gorgone et al. (2002, 13) taas kuvaavat informaatiojärjestelmien alalta valmistuvien opiskelijoiden osaamiskenttää, tärkeimpiä osaamisalueita, kuvan 3 mukaisesti (ks. kuva 3):



Kuva 3. Valmistuvien opiskelijoiden osaamiskenttä (Gorgone et al. 2002, 13).

Kirjallisuuskatsauksessa Anderson (2001, 239) korostaa, että monet teollisuus-yliopisto yhteistyöalat toimivat markkinavetoisilla sektoreilla, viitaten kirjaan Academic capitalism (Slaughter 1997), jonka mukaan seuraavat tekijät ovat kriittisiä globaalissa kaupallisessa kilpailussa: tietokoneet, telekommunikaatio, elektroniikka, kehittyneet materiaalit, tekoäly ja bioteknologia.

Aiemmin tyypillinen urapolku tietotekniikan ammattilaisella on ollut lineaarinen alkaen ohjelmoijasta jatkuen järjestelmäanalyytikoksi ja projektipäälliköksi. Tietotekniikan ammattilaiset etenivät siis vaiheittaisesti kohti samaa päämäärää. Monet muutokset konsernien tietohallinnoissa kuten lisääntynyt telekommunikaation käyttö, järjestelmien integrointi ja ulkoistukset ovat kuitenkin luoneet monia uusia urapolkuja (Trauth 1993). Tietotekniikan alan työmarkkinoiden muutosten tähden tulisi taata se, että valmistuvilla opiskelijoille olisi riittävät osaamisvalmiudet myös käytännön ohjelmistotöihin heidän siirtyessään työelämään.

4.4 Ohjelmointikieli

Lappeenrannan teknillisessä yliopistossa on menestyksellisesti käytetty jo useamman vuoden ajan Python-ohjelmointikieltä ohjelmoinnin perusteiden opettamiseen. Olio-ohjelmointikielen käyttö on viime vuosina yleistynyt yliopistoissa ohjelmoinnin peruskurs-seilla, suosituimpina ohjelmointikielinä C++ ja Java (Ala-Mutka 2005, 15). Oliopohjaisiin kieliin siirtyminen on sikälikin perusteltua, että ohjelmistojen kokojen jatkuvasti kasvaessa ja rakenteiden monimutkaistuessa on niiden mallintaminen ja hallinta helpompaa kun sekä suunnittelu että toteutus tehdään käyttäen UML-työkaluja ja metodeja (Unified Modeling Language).

Käytännön ohjelmointi -kurssin ohjelmointikielen valintavaihtoehtoja on kolme vertailtaessa Suomen yliopistoja; C, C++ ja Java (ks. luku 2.3). Pythonia ei löydettyjen tietojen mukaan käytetty ohjelmoinnin jatkokursseilla. Ala-Mutkan (2005) tutkimuksen mukaan, kattaa kuusi yliopistoa ja 559 opiskelijaa, käytetyimmät ohjelmointikielet 1-2 vuosikurssin opiskelijoiden keskuudessa olivat; C++ 73,4 %, Java 17,3 %, Pascal 2,9 % ja muut 4,8 %:a. Mitkä ovat sitten perusteet käyttää C-ohjelmointikieltä?

Ohjelmointikielenä perinteinen C on alkujaan suunniteltu proseduraaliseen järjestelmäoh-jelmointiin, kirjoitettiinhan Unix aikoinaan C-kielellä. Tuskinpa kukaan ohjelmistotekniikan alalla työskentelevä voisi väittää, ettei C-kielen perusosaaminen olisi tärkeä osa tietotekniikan ammattilaisen työrutiineja. C on ohjelmointikielenä laiteläheisempi kuin monet olio-ohjelmointikielet ja esimerkiksi laiteajurit on useasti kirjoitettu C-kielellä. Tietotekniikan opiskelijan opintosuunnitelmaan, HOPS:iin (henkilökohtainen opintosuun-nitelma) sisältyy pakostakin Python, C++- tai Java- sekä muita olio-ohjelmointikielten kursseja. Tällöin on perusteltua myös tulevien osastokohtaisten opiskeluvalintojen tähden, esimerkiksi konenäkö ja hahmontunnistus, hallita hyvin laiteläheinen ohjelmointi.

Jos opiskelija ei ole tietotekniikan opiskelija, saattaa C ohjelmointikielenä osoittautua haastavaksi, mutta tässä vaikuttavat pitkälti opiskelijan osaamistaso ennen kurssin aloi-tusta. Lappeenrannan teknillisen yliopiston Olio-ohjelmointi-kurssin esitietovaatimuksina

(Lappeenrannan teknillinen yliopisto 2010) oli Käytännön ohjelmointi -kurssi ja vastaavasti Object-Oriented Programming Techniques -kurssilla, jolla opetuskielenä oli Java, esitietietovaatimuksina oli Olio-ohjelmointi-kurssi tai vastaavat tiedot. Tällä tavoin ohjelmointikielten opiskelu ja hallinta alkaa Python-kielestä Ohjelmoinnin-perusteet-kurssilla, päätyen C-kielen kautta olio-ohjelmointikielien hallintaan.

Cheng (2009) painottaa lehtiartikkelissaan C-ohjelmointikielen vankan osaamisen merkitystä mekaanisen suunnittelun perustietoihin kuuluvana osana, onhan laajalti käytetty Matlab-ohjelma toteutettu C-kielellä. Opiskelijoiden hallitessa C-ohjelmointikielen on heidän paljon helpompi oppia käyttämään Matlab-ympäristöä. Cheng (2011) listaa lisäksi useampia syitä miksi tulisi opettaa ja opiskella C-ohjelmointikieltä:

- *C on yleisimmin käytetty ohjelmointikieli teollisuudessa*
- *C on paras valinta kun ohjelmoidaan sulautettujen ja mekatronisten järjestelmien laitteistorajapintoja*
- *C on yksi yleisimmin käytetyistä ohjelmointikielistä korkeakouluissa ja yliopistoissa.*
- *Kun opiskelijat ovat oppineet C-ohjelmointikielen, he voivat tarvittaessa opiskella muita kieliä omatoimisesti.*

4.5 Windows vs Unix

Käytännön ohjelmointi -kurssilla opetusympäristön valinnalla pyritään ohjaamaan opiskelijoita käyttämään ja oppimaan Unix-pohjaista käyttöjärjestelmää, Ubuntu Linuxia, jotta mahdollinen kynnys jatko-opinnoissa sekä työelämässä Unixin käyttöön madaltuisi. Ubuntun virtuaaliasennus Windowsin rinnalle on opetuskäytössä perusteltu valinta, sillä silloin opetustilanteessa luennoitsijan tekemät nauhoitukset voidaan tallentaa normaaliin tapaan ja muut mahdollisesti tarvittavat Windows-ympäristön sovellukset ovat käytettävissä.

Ubuntua käyttäessään opiskelijat tutustuvat ja saavat käsityksen monista Unixin perustoiminnoista kuten esimerkiksi erilaisesta tiedostojärjestelmästä ja editoreista. Lisäksi opiskelijat saattavat opetella käyttämään komentorivejä (skriptejä) helpottamaan ohjelmistoprojektin suoritusta.

4.6 Opetusmateriaalit

Opetusmateriaaleihin sisältyivät kurssin luentomateriaali eli luentokalvot, C-kieli ja käytännön ohjelmointi osa 1 luentomoniste (Kasurinen 2011), nauhoitetut luennot sekä Viope-opetusympäristö. Viope-opetusympäristö sisälsi harjoitustehtäviä, teoriaa sekä opettajan kokeita. Lisäksi kurssiin kuului laajempi kolmen hengen ryhmissä tehtävä harjoitustyö. Kurssimateriaaliin katsotaan myös kuuluvaksi luennoitsijan ilmoittavat kurssikirjat, The C Programming Language (Kernighan 1988) ja The Practice of Programming (Kernighan 1999.). Tämän diplomityön osana toteutettu konstruktio, C-kieli ja käytännön ohjelmointi osa 2 luentomoniste (ks. liite 1), tulee olemaan lisänä seuraavalla alkavalla kurssilla.

Luentojen nauhoittaminen ja videotiedostojen tallentaminen Noppa-portaaliin tukee oppimista, onhan tällöin opiskelijoilla jotka eivät osallistuneet luennolle, mahdollisuus

saada käytännössä sama informaatio kuin jos he olisivat olleet luennolla. Nauhoitettu luentomateriaali tukee oppimista myös siten, että opiskelija voi halutessaan käyttää videotiedostoja kertausmateriaalina esimerkiksi valmistautuessaan tenttiin tai tarkentaessaan, että on ymmärtänyt tietyn asiakokonaisuuden oikein. Kun luentomateriaali on valmiina ja luentosalin tekniikka toimii, on luennoitsijalla mahdollisuudet pedagogisten taitojensa avulla motivoida opiskelijat seuraamaan opetusta ja saada heidät omaksumaan uusia asioita.

4.7 Palautettujen ohjelmien testaaminen ja virheiden etsintä

Ohjelmistoprojektia lukuun ottamatta palautettujen ohjelmien testaaminen Käytännön ohjelmointi -kurssilla perustuu Viope-oppimisympäristön automaattiseen koodin tarkastukseen, joka pohjautuu merkkijonosyötteen ja vasteen vertailuun. Viope, joka toimii integroituna selaintoimintoon, on siinä mielessä varsin hyvä ja toimiva oppimisympäristö ohjelmointiin, että opiskelija voi tehdä harjoitustehtäviä riippumatta ajasta ja paikasta. Riittää kun opiskelijalla on tietokone ja Internet-yhteys käytössään.

Tarkoituksenmukaista on myös opettaa opiskelijoille virheiden etsintää eri perustyökalujen avulla, kuten esimerkiksi assert, Valgrind ja eri debuggerit. Lisäksi erillisten testaussovellusten käyttöön tutustuminen olisi tärkeää, jotta opiskelijoilla olisi käsitys ohjelmistotestaustyökaluista ennen työelämää.

4.8 Kurssin rakenne

Käytännön ohjelmointikurssi vastaa viiden opistopisteen suoritusta jaksottuen tietotekniikan opinnoissa ensimmäiselle vuodelle, periodeille kolme ja neljä. Rakenne muodostuu kahden oppitunnin viikkoluennoista, jotka sisältävät sekä teoriaa että soveltavia harjoitusesimerkkejä, ja viikkoharjoituksista jotka ohjaa assistentti. Viikkoharjoitukset ovat vapaaehtoisia ohjaustilaisuuksia, joissa voi kysyä neuvoa ongelmatilanteissa. Lisäksi kurssin suoritukseen kuuluu ryhmänä tehtävä ohjelmointiprojekti.

Viikkoharjoitustehtävät sekä opettajan kokeet tehdään Viope-opetusympäristössä, tehtävien palautuspäivämäärien ollessa pääsääntöisesti viikon välein. Ryhmässä tehtävän projektin ohjelmointiympäristö on valinnainen, mutta palautus tulee tehdä SVN-version-hallintaympäristöön (Subversion control) samalla varmistaen ohjelman toimivuus myös Linux-ympäristössä. Kurssin alkuosalla viikkoharjoitukset on pidetty niin kutsutussa Windows-luokassa ja jälkimmäisellä periodilla, harjoitustyön julkistamisen yhteydessä, on siirrytty Linux-luokkaan.

Viikkotehtävien, opettajan kokeiden ja harjoitustyön osalta on todettava, että ne vaihtuvat vuosittain eivätkä ole suoraan vertailukelpoisia, tosin vaikeustaso tehtävissä on jotakuinkin sama. Vuonna 2010 Käytännön ohjelmointi -kurssilla luentokertoja oli 14, seitsemän sekä 3. että 4. periodilla. Periodin 3 luentojen ja 4. periodin ensimmäisen luennon aiheina olivat C-kieli ja yleiset ohjelmointimetodit. Seuraavat luennot olivat aiheiltaan: ohjelman suunnittelu, -toteutus, virheiden hallinta, ohjelman testaus, versionhallinta ja lopuksi kurssin päätös. Kurssin päättävällä luentokerralla kerrattiin lyhyehkösti kurssilla käsitellyt aiheet valmistautumisena tenttiin. Kurssin sisältö oli noudattanut jo useamman vuoden samaa sisältörakennetta.

Keväällä 2011 kurssi noudatti sisällöllisesti vastaavaa rakennetta 3. periodilla kuin edellisenä vuotena, tosin dynaaminen muistinhallinta käsiteltiin jo samassa asiakokonaisuudessa periodilla 3, samoin eri aihealueiden painotukset hieman muuttuivat luennoitsijan vaihduttua. Periodilla 4 siirrettiin painopiste Linuxiin ohjelmointiympäristönä. Aihevalinnat etenivät poiketen aiemmasta seuraavasti; suunnittelu ja toteutus, moduuli-testaus, tarkastus, testauksen hallinta ja kertaus. Viimeisellä luentokerralla oli myös demonstraatio regressiotestauksesta Python-ympäristössä.

Käytännön ohjelmointi -kurssien 2010 ja 2011 vertailu pelkän teoriaosuuden perusteella ei ole kovin kattava, koska luennoilla opetetaan opiskelijoille teoriapohjaa harjoituksia ja ryhmätyötä varten. Kattavamman vertailun tulisi pohjautua kaikkeen kurssilla käytössä olleeseen materiaaliin, mitä nyt ei kuitenkaan ollut käytössä, viitaten eritoten Viopen

viikkokokeisiin ja opettajan kokeisiin. Viopen käytöstä oppimisympäristönä on olemassa tutkimuksia, esimerkiksi Carver ja Henderson (2006) toteavat opiskelijoiden olevan halukkaita käyttämään Viopea tai vastaavaa oppimisympäristöä.

Merkittävä käytännön muutos vuoden 2011 kurssin suorituksessa oli SVN-versionhallinnan käyttöönottaminen. Opiskelijat käyttivät versionhallintaa projektina tehdyssä harjoitustyössä, saaden näin konkreettisen käsityksen käytännön ohjelmointi-työstä. Versionhallinnan käyttöä helpotti se, että luennoilla annettiin ohjeet myös Windows-pohjaisen TortoiseSVN:n käyttöönottoon, jolloin kynnys versionhallinnan käytölle oli matalampi. Tämä edellä mainittu seikka todennäköisesti johti Linuxin pienempään käyttöön itse ohjelmointiympäristönä.

Opiskelijoiden siirtyminen Ohjelmoinnin perusteet-kurssilta Käytännön ohjelmointi -kurssille oli sikäli helpompaa, että molemmilla kursseilla on käytössä Viope-oppimisympäristö. Jatkossa on täten mahdollista tehdä tarkempaa seurantatutkimusta myös Käytännön ohjelmointi -kurssista, kun Viope-ympäristöön kertyy tilastoitavaa dataa opiskelijasuorituksista ja ajankäytöstä.

5 Konstruktio

Konstruktiivinen prosessi käynnistyi tarpeesta saada selkeä oppimateriaali luennoitsijan ja opiskelijoiden käyttöön Käytännön ohjelmointi -kurssille. Kurssilla oli jo käytössä päivitetty luentomoniste, joka sisälsi C-ohjelmoinnin perusteet (Kasurinen 2011), mutta oli tarve materiaalille jossa käsiteltäisiin käytännön tasolla C-ohjelmointiin liittyviä asioita kuten ohjelmistokehitysympäristöä, versionhallintaa ja ohjelmistotestausmenetelmiä sekä testaustyökaluja.

Konstruktio käynnistyi tekemällä runko opetusaiheista seitsemälle 90 minuutin luentokerralle, ja näiden luentokokonaisuuksien hyväksyttämisellä tämän diplomityön ohjaajalla. Kun hyväksyttävä luentokokonaisuus oli muodostettu, alkoi materiaalin koostaminen ja työstäminen. Konstruktion kokoaminen olikin tämän diplomityön työllistävin ja keskeisin vaihe konstruktiivisen tutkimusmetodin mukaisesti. Lopputuotoksena oli luentomoniste, jonka rakenne noudatti Ohjelmoinnin perusteet-kurssin päivitettyä Python 3 – ohjelmointiopasta. (Vanhala 2010)

Liitteenä 1 oleva konstruktio muodostaa alustavasti seitsemän teorialuennon kokonaisuuden, minkä lisäksi lopussa on kappale Yhteenveto sisältäen lisämateriaalia kokonaisuuteen liittyen. Konstruktio rakennettiin tukemaan ohjelmistoprojektin etenemistä sekä siirtymistä Windows-ympäristöstä Ubuntu Linux-ohjelmointiympäristöön. Ensimmäisellä luennolla käsiteltävinä teemoina ovat muun muassa ohjelmistokehitysympäristöt, Ubuntu ja Windows-käyttöjärjestelmien eroavaisuudet, Ubuntun asennus sekä ohjelman kääntäminen Unix-ympäristössä. Aiheiden valinta painottui ensimmäisen ohjaavan luennon jälkeen *toteuttamiseen (luku 1), työkaluihin (luku 2), virheiden etsintään (luku 3), testaukseen (luku 4), integrointiin (luku 5)* ja *muutosten hallintaan (luku 6)*. *Tyyliopas (LIITE 1)*, opas jonka tarkoituksena on ohjata opiskelijoita yhtenäiseen ja selkeään tapaan kirjoittaa koodia, on yksi konstruktion keskeisin osa. Tyyliopas on irrotettu erilliseksi liitteeksi, jolloin siihen on helpompi viitata sekä tarvittaessa tehdä lisäyksiä.

5.1 Kurssi 2011 vs konstruktio

Konstruktion rakenteeseen pyrittiin saamaan mahdollisimman paljon käytännönläheisiä aiheita varaamalla ohjelmiston testaukseen kaksi luentokertaa, tämä tosin toteutui jo kurssilla 2011. Tehdyn konstruktion ja kevään 2011 kurssin sisältö käsittää samat asiakokonaisuudet, tosin konstruktiossa hieman eri tavoin ryhmiteltyinä. Jatkossa luennoilla on tarkoitus konstruktion mukaisesti esitellä myös testaustyökaluja ja näin totuttaa opiskelijat käyttämään esimerkiksi lausekattavuuden testaavia työkaluohjelmia. Myös konstruktiossa painotetaan versionhallinnan käyttöä osana harjoitustyönä tehtävää ohjelmistoprojektia.

5.2 Konstruktion muodostuminen

Konstruktion tavoitteena on antaa luennoitsijalle perusmateriaali jota käyttää luentojen tukena sekä toimia oppaana opiskelijoille heidän tehdessään sekä harjoituksia että projektityötä. Tähän edellä mainittuun pyrittiin ottamalla materiaaliin toimivia ohjelmaesimerkkejä sekä komentojonoja muun muassa versionhallinnan yhteydessä.

Konstruktio, *C-kieli ja käytännön ohjelmointi – osa 2*, ei kata vielä ensimmäisessä versiossaan kaikkia luennoilla mahdollisesti käsiteltäviä aihealueita, mutta konstruktion ja siis myös luentomonisteen versio 1.0 tulee toimimaan päivitettävänä kurssimonisteena, kehittyen näin askel askeleelta.

Luentomateriaalin tavoitteena on vähentää luennoitsijan ja assistentin työmäärää kun on olemassa opas, johon voidaan viitata ja opiskelija voi käyttää sitä lähdemateriaalina rinnan toisen oppaan, *C-kieli ja käytännön ohjelmointi osa 1* (Kasurinen 2011) kanssa. Kun yliopistolla on käytössään omia julkaisuja, voidaan niihin tehdä suhteellisen pienillä viiveillä ja kustannuksilla sekä päivityksiä että muutoksia.

Ensimmäinen konstruktio rakentui seitsemälle luentokokonaisuudelle laadittuihin erillisiin aihekokonaisuuksiin: projektin aloitus, toteutus, moduulitestaus, katselmointi, testaus

laajemmin, muutosten hallinta ja yhteenveto. Konstruktio muodostui seitsemästä erillisestä Word-dokumentista, mikä ei ollut käytännöllistä luettavuuden kannalta. Ensimmäisestä konstruktiosta palautteena saatu kritiikki voidaan jakaa seuraaviin luokkiin: asiat joita ei käsitellä kurssilla, lainausten määrä, asioiden järjestys, sovitus kurssin tasoon, siirtymät kappaleiden välillä – johdannot, esimerkkipohjat (testauksen raportointi ja testauspäiväkirja), vain yhden menettelytavan esittäminen (kooditarkastus), tyyliopas sekä yleinen layout tarkoittaen laitoksen Python-oppaan mallia.

Suurin vaikeus ilmeni ohjelmistoarkkitehtuurin käsittelemisessä, sillä kurssilla käytetään perinteistä C-ohjelmointikieltä, jolloin UML-kuvantamismenetelmiä ei voida soveltaa. Näin päädyttiin lyhyehköön SA- menetelmän (Structured Analyses) käsittelyyn kuvantamismenetelmänä. Toinen ongelmallinen kohta oli ohjelman suunnittelu omana aihekokonaisuutenaan. Tietovirtakaaviot esiteltiin (Data Flow Diagram) osana SA-menetelmää. Nämä molemmat edellä mainitut menetelmät karsiutuivat kuitenkin pois materiaalista, koska ne ovat myöhempien kurssien aiheita.

Lainausten määrä sekä vain yhden menettelytavan esittäminen asiakokonaisuudessa korjaantuivat käytännössä runsaahkon sisällön karsimisella, jota oli kertynyt tavoitteena saada oheismateriaalia käytettäväksi jo kevään 2011 luennoilla. Tyyliopas oli ensimmäisessä konstruktiossa suppea, joten sitä laajennettiin käyttäen pohjana GNU:n ohjelmointiopasta (GNU Coding Standards), samalla lisäten kokonaisia toimivia ohjelmaesimerkkejä.

Toisessa konstruktion versiossa layout oli yhteneväinen Python-oppaan kanssa, mutta tiettyjä pienempiä puutteita vielä oli. Muutamien kuvien termit olivat kieleltään englanniksi jotka suomennettiin. Lisäksi tarkennettiin muutamia yksityiskohtia kuten C-kääntäjän parametrien merkitystä, testikoodin osuus testiajurin ja tyngän yhteydessä sekä selvennettiin laskennallisella esimerkillä testikattavuutta. Unixin komentorivit samoin kuin koodiesimerkit muutettiin fontille courier new, jolloin ne erottuivat selvemmin omana kokonaisuutenaan itse leipätekstistä. Konstruktioon kirjoitettiin myös luku Loppusanat, joka päättää luentomonisteen.

5.3 Konstruktion kuvaus

Oppimateriaalissa käydään läpi käytännön ohjelmoinnin perusteita C-kielellä. Opas toimii jatkona julkaisulle C-kieli ja käytännön ohjelmointi osa 1. Tässä jatko-osassa käsitellään esimerkkien, ohjeiden ja teorian avulla termejä, joita jokainen aloitteleva ohjelmoija tulee käyttämään työelämässä. Lukukappaleita on seitsemän, jotka on jaettu alustavasti luentokertojen mukaisiin asiakokonaisuuksiin. Lisänä on vielä lukukappale, joka sisältää keskeisiä ohjelmoinnin aihealueita, jotka tosin näillä näkymin eivät kuulu luennoilla käsiteltäviin asioihin. Oppimateriaalin lukukappaleet ovat seuraavat:

- *Ohjelmistokehitysympäristöt*
- *Versionhallinta*
- *Arkkitehtuuri*
- *Tyyliopas*
- *Moduulitestaus*
- *Kooditarkastus*
- *Testaus laajemmin*
- *Yhteenveto.*

5.4 Konstruktion arviointi

Konstruktiivisen tutkimuksen tuloksille on olemassa useita arviointikriteerejä eri näkökulmista: käsitteistö, malli, metodi ja toteutus (Järvinen 2004, 123). Koska konstruktio on kokonaisuudessaan käytössä vasta seuraavalla alkavalla ohjelmointikurssilla yhtenäisenä luentomonisteena, myös sen varsinainen opiskelija-arviointi esimerkiksi luettavuuden, loogisuuden, käytettävyyden ja ymmärrettävyyden osalta tulee arvioiduksi laajemmin vasta myöhemmässä vaiheessa, eikä siitä saatavaa opiskelijapalautetta voida tässä työssä raportoida. Korjattu ohjaajan oikolukema konstruktio on tämän diplomityön liitteenä (ks. liite 1). Konstruktio erotetaan myös omaksi erilliseksi luentomonisteeksi. Näin luentomoniste, omalla ISBN-numerollaan, on otettavissa luentokäyttöön seuraavalla Käytännön ohjelmointi -kurssilla keväällä 2012.

5.5 Hypoteesin validointi opiskelijapalautteen pohjalta

Kurssin loppukysely toteutettiin 18-kohtaisena Webropol-kyselynä (ks. liite 2) joista 11 kohtaa olivat monivalintakysymyksiä, kuudessa (6) pyydettiin sanallista palautetta ja yhdessä (1) kahta prosenttilukuarvoa. Webropol-loppukysely toteutetaan pääsääntöisesti kaikilla tietotekniikan kursseilla, jotta saadaan oppilaspalautetta mahdollisista kurssin kehityskohdista. Käytännön ohjelmointi -kurssille keväällä 2011 ilmoittautui 123 opiskelijaa.

Harjoitustehtävät jakautuivat kolmeen eri ryhmään (suluissa hyväksytyt suoritukset):

- Viope-kurssin opintolukujen tehtävät (85)
- kokeet (51)
- harjoitustehtävät (53).

Kaikki pakolliset tehtävät teki hyväksyttävästi 45 opiskelijaa. Kurssikyselyyn vastanneita oli 40 joten oletettavasti lähes kaikki, jotka olivat suorittaneet vaadittavat tehtävät, vastasivat kurssikyselyyn. Kirjoitushetkellä 37 opiskelijaa 123:sta oli saanut kurssista loppuarvosanan jolloin hyväksyntäprosentti oli 30,1. Mikäli kaikki 45 kaikki pakolliset tehtävät tehnyttä opiskelijaa suorittavat tentin hyväksytysti, nousee hyväksyntäprosentti 36,6:een. Verrattaessa saatua lopullista kurssin suoritusprosenttia, joka tulee siis olemaan välillä 30,1-36,6, vastaaviin Ohjelmoinnin perusteet-kurssin suoritusprosentteihin viisivuotisessa seurantatutkimuksessa (Nikula 2011): 2005 (36 %), 2006 (54 %), 2007 (65 %), 2008 (61 %) ja 2009 (68 %), voitaneen olettaa että kehitys tulee olemaan samansuuntainen.

Seuraavissa luvuissa (5.5.1 – 5.5.8) käsitellään kymmentä kysymystä Webropol-lomakkeesta (kysymysten numerointi on sama kuin Webropol-lomakkeella):

2. *Kokonaisarvio opintojaksosta*
3. *Jos kurssin vaatimukset ja sisältö pysyvät nykyisellä tasolla, niin kuinka näet kurssin opintopistemäärän?*
7. *Miten arvioit kurssia yleisten ominaisuuksien osalta?*
8. *C-kielen osaaminen*
9. *Käytännön ohjelmoinnin osaaminen*
10. *Käytetyt käyttöjärjestelmät ja ohjelmat / osaaminen ja käyttö*
11. *Luentojen ja videoiden seuraaminen*

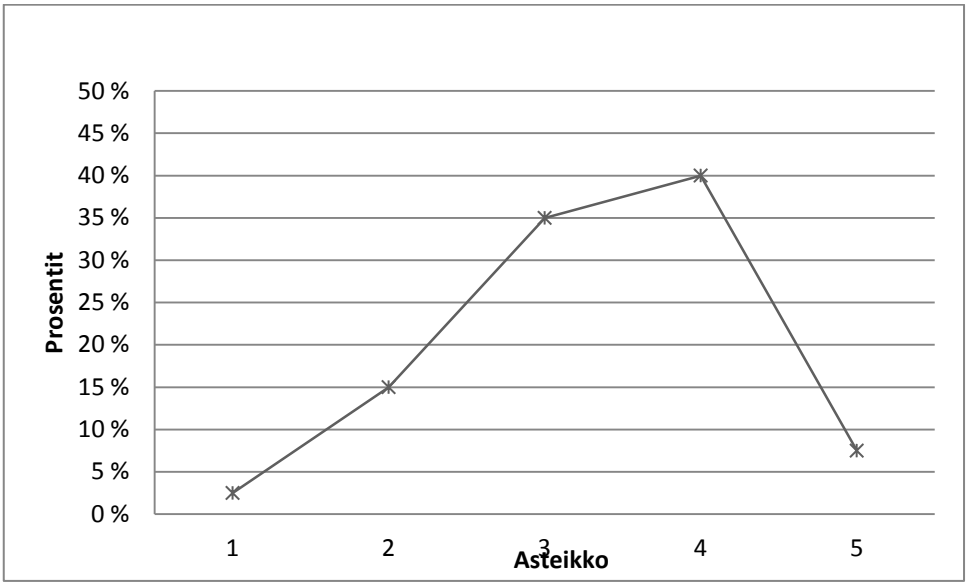
Yleinen palaute (sanallinen)

16. *Oliko kurssilla jotain hyvää, mitä sinusta ei pitäisi muuttaa jatkossa?*
17. *Oliko kurssilla jotain huonoa, jota sinusta pitäisi muuttaa jatkossa?*
18. *Vapaa palaute opintojaksosta, esimerkiksi kehittämisehdotukset, vahvuudet, heikkoudet.*

Ensimmäiset seitsemän kysymystä esitetään graafisesti ja kolme viimeistä mainitsemalla yleisimmät aiheet.

5.5.1 Kokonaisarvio kurssista

Arvio opintojaksosta asteikolla 1-5 oli keskiarvo = 3,35 hajonnalla = 0,91 (ks. kuva 4). Jatkossa keskiarvoa merkitään symbolilla μ ja hajontaa symbolilla σ (Std = Standard deviation).



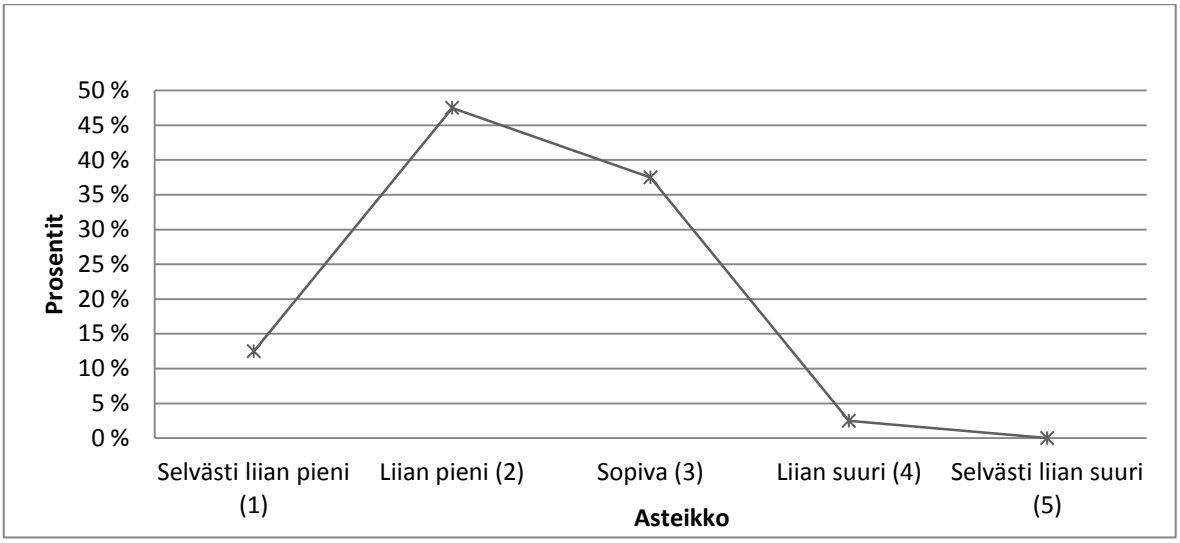
Kuva 4. Kokonaisarvio kurssista.

5.5.2 Kurssin vaatimukset ja sisältö

Kysymyksessä:

> *Jos kurssin vaatimukset ja sisältö pysyvät nykyisellä tasolla, niin kuinka näet kurssin opintopistemäärän?*

jakaantuivat opiskelijoiden mielipiteet kurssia vastaavasta opintopistemäärästä seuraavasti asteikolla 1-5 (1 = selvästi liian pieni, 5 = selvästi liian suuri): $\mu = 2{,}3$ ($\sigma = 0{,}71$) (ks. kuva 5).



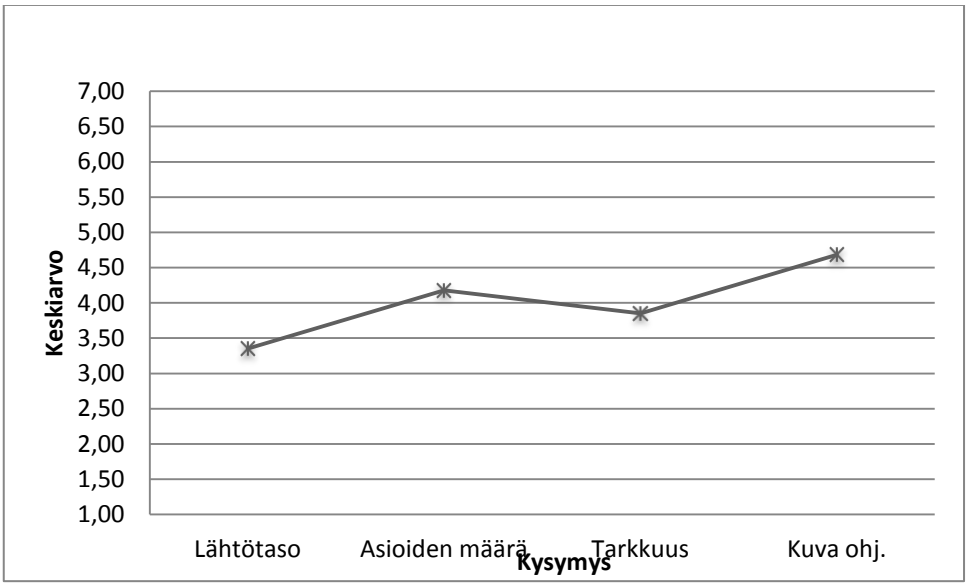
Kuva 5. Opintopistemäärien vastaavuus kurssista.

5.5.3 Arvio kurssista

Seuraavassa kysymyksessä opiskelijat arvioivat kurssia neljän kriteerin pohjalta asteikolla 1…7 (ks. kuva 6):

Miten arvioit kurssia seuraavien yleisten ominaisuuksien pohjalta?

1. Lähtötaso: kurssi lähti liikkeelle (1) liian perusasioista, (7) liian edistyneistä asioista, $\mu = 3{,}35$ ($\sigma = 1{,}11$)
2. Määrä: kurssilla käsiteltyjen asioiden määrä oli (1) liian pieni, (7) liian suuri, $\mu = 4{,}18$ ($\sigma = 0{,}92$)
3. Tarkkuus: asiat käsiteltiin kurssilla (1) liian pinnallisesti, (7) liian tarkasti, $\mu = 3{,}85$ ($\sigma = 0{,}91$)
4. Kurssilla muodostunut kuva käytännön ohjelmoinnista on (1) epämääräinen, (7) selkeä, $\mu = 4{,}68$ ($\sigma = 1{,}51$).



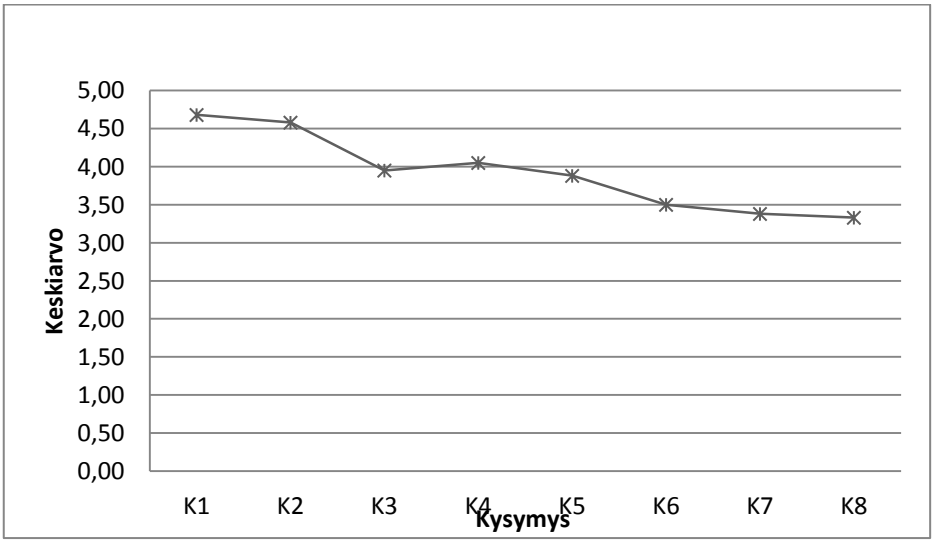
Kuva 6. Arvio kurssista.

5.5.4 C-kielen osaaminen

C-kielen osaamistaan opiskelijat arvioivat kahdeksalla eri osa-alueella asteikolla (1) en ole varma mistä on kysymys, (5) pystyn käyttämään / tekemään projektissa omatoimisesti (ks. kuva 7):

1. Yksinkertaiset tietorakenteet ja tietotyypit, muuttujat, $\mu = 4{,}68$, ($\sigma = 0{,}61$)
2. Peruskäskyt kuten syöttö, tulostus, valinta-, toisto- ja hyppyrakenteet, virheenkäsittely, $\mu = 4{,}58$, ($\sigma = 0{,}74$)
3. Tiedostonkäsittely, teksti ja binaaritiedostot, tietovirrat, $\mu = 3{,}95$, ($\sigma = 0{,}89$)
4. Funktiot, parametrit, paluuarvot, rekursio, kirjastofunktiot, $\mu = 4{,}05$, ($\sigma = 0{,}96$)
5. Ohjelman ja tiedoston rakenne, useat tiedostot, komentoriviparametrit, $\mu = 3{,}88$, ($\sigma = 1{,}00$)
6. Dynaaminen muistinhallinta, osoittimet ja rakenteiset tietorakenteet, $\mu = 3{,}5$, ($\sigma = 1{,}18$)
7. Linkitetty lista, $\mu = 3{,}38$ ($\sigma = 1{,}28$)
8. Esikääntäjä, kääntäminen ja linkkaus, $\mu = 3{,}33$ ($\sigma = 1{,}26$).

Vastaukset jakautuivat seuraavasti asteikolla 1-5 (ks. kuva 7):

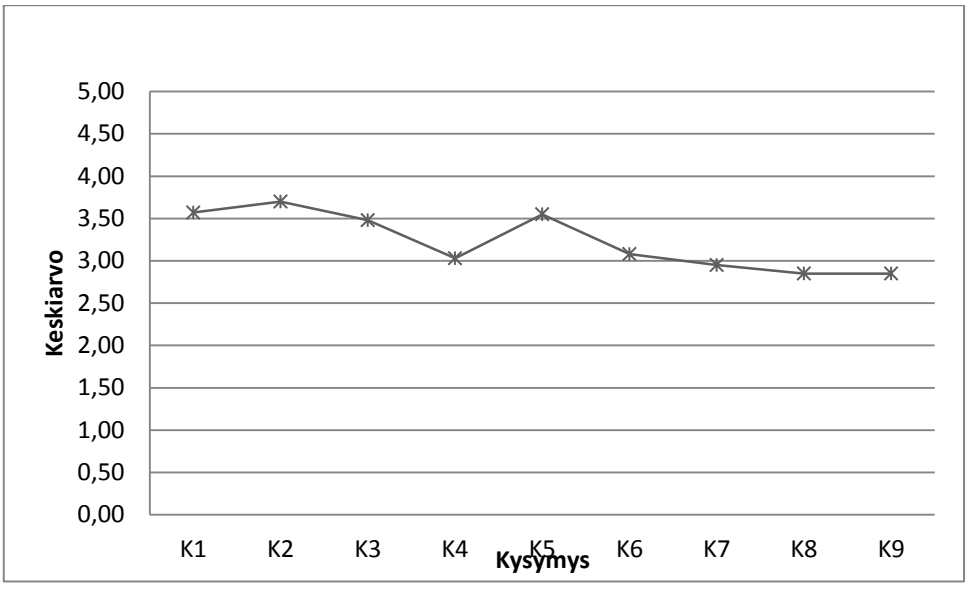


Kuva 7. C-kielen osaaminen.

5.5.5 Käytännön ohjelmoinnin osaaminen

Käytännön ohjelmoinnin osaamistaan opiskelijat arvioivat yhdeksällä eri osa-alueella asteikolla (1) en ole varma mistä on kysymys … (5) pystyn tekemään projektissa omatoimisesti (ks. kuva 8), seuraavilla keskiarvoilla (μ) ja keskihajonnoilla (σ):

1. Arkkitehtuurisuunnittelu, μ = 3,57 (σ = 1,03)
2. Moduulisuunnittelu, μ = 3,70 (σ = 0,84)
3. Toteutus, tyyliopas, make, versionhallinta, μ = 3,48 (σ = 0,97)
4. Debuggaus, μ = 3,03 (σ = 0,99)
5. Kooditarkastukset, μ = 3,55 (σ = 1,05)
6. Moduulitestaus, testiajurit ja tynkämoduulit, μ = 3,08 (σ = 1,12)
7. Integrointitestaus, μ = 2,95 (σ = 1,07)
8. Systeemitestaus, μ = 2,85 (σ = 1,09)
9. Testauksen hallinta, kattavuus ja automatisointi, μ = 2,85 (σ = 0,85).



Kuva 8. Käytännön ohjelmoinnin osaaminen.

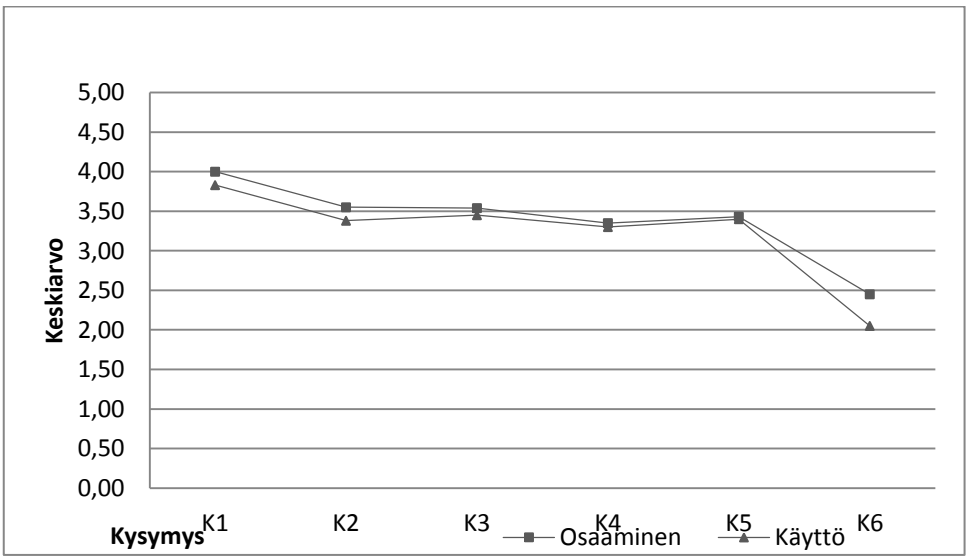
5.5.6 Käytetyt käyttöjärjestelmät ja ohjelmat – osaaminen ja käyttö

Käyttöjärjestelmien osaamista opiskelijat arvioivat kuudella eri osa-alueella asteikolla (1) en ole varma mistä on kysymys … (5) pystyn tekemään projektissa omatoimisesti ja käyttöä asteikolla (1) en ole käyttänyt … (5) luennoijan ohjeiden ja esimerkin mukaisesti (ks. kuva 9). Keskiarvot (μ) ja keskihajonnat (σ) olivat:

1. Codeblocks kehitysympäristö, μ = 4,00 (σ = 0,74)
2. Linux-käyttöjärjestelmä, μ = 3,55 (σ = 1,07)
3. gcc ja gedit ohjelmointityökalut, μ = 3,54 (σ = 1,19)
4. make ja Makefile, μ = 3,35 (σ = 1,19)
5. Versionhallinta ja svn, μ = 3,43 (σ = 1,09)
6. Debuggeri, esimerkiksi DDD, μ = 2,45 (σ = 1,09).

ja käyttö:

1. Codeblocks kehitysympäristö, μ = 3,83 (σ = 1,28)
2. Linux-käyttöjärjestelmä, μ = 3,38 (σ = 1,30)
3. gcc ja gedit ohjelmointityökalut, μ = 3,45 (σ = 1,40)
4. make ja Makefile, μ = 3,30 (σ = 1,27)
5. Versionhallinta ja svn, μ = 3,40 (σ = 1,24)
6. Debuggeri, esimerkiksi DDD, μ = 2,05 (σ = 1,24).



Kuva 9. Käytetyt järjestelmät ja ohjelmat – osaaminen ja käyttö.

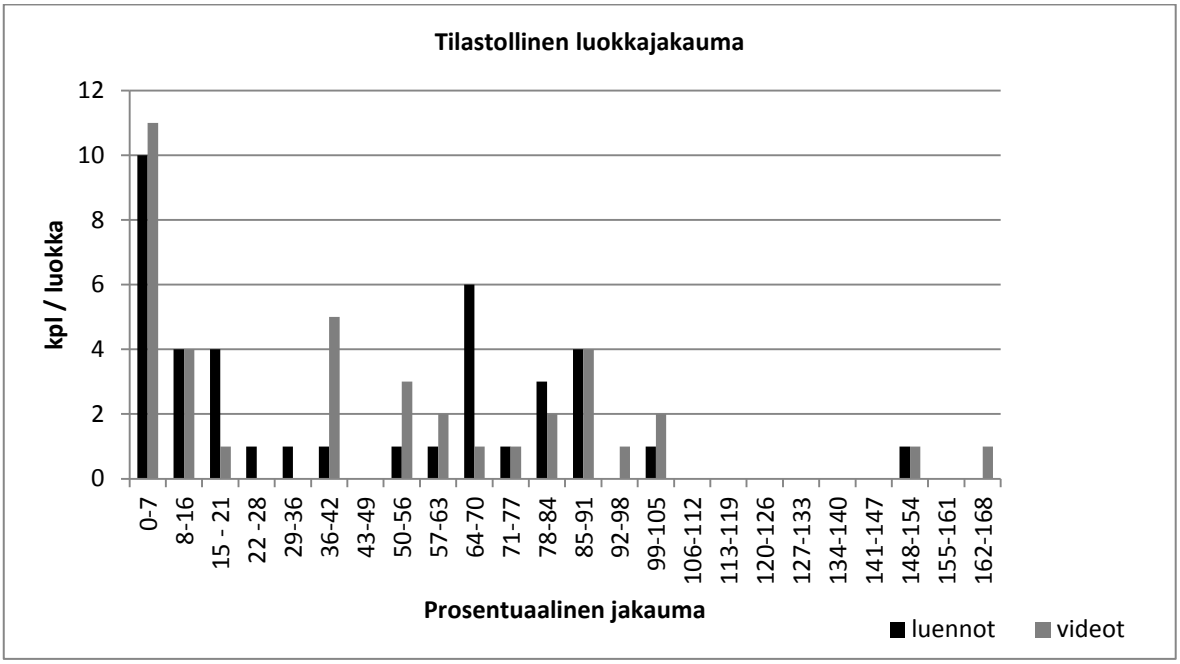
Kuvassa 9 käyrät korreloivat hyvin selvästi (ks. s. 37) mikä olikin odotettua ajatellen Käytännön ohjelmointi -kurssin sisältöä. Tekemällä annettuja harjoitustehtäviä myös metodit ja työkalut tulevat tutuiksi.

5.5.7 Luentojen ja videoiden seuraaminen

Luentojen ja videoiden seuraamista kartoitettiin seuraavalla tavalla ohjeistetuilla kahdella kysymyksellä (vastaajia 39, luentokertoja 14);

> Vastaa antamalla prosenttiosuus kokonaislukuna. Yksi luentokerta, 90 min, vastaa 7 % kurssin luennoista ja 7 kertaa vastaa 49 %:ia. Jos katsoit videoita useamman kerran, laske ne myös mukaan eli jos katsoit kaikki luennot kahteen kertaa, vastaa 200.
>
> *Missä määrin osallistuit luennoille?*
>
> *Missä määrin katsoit luentovideoita?*

Opiskelijoiden ilmoittamat luennoille osallistumisprosentit ja luentovideoiden seuraamisprosentit jaettiin 24 luokkaan 7 %:n jaotuksella ja saadun luokkajakauman pohjalta piirrettiin kuva (ks. kuva 10).



Kuva 10. Tilastollinen luokkajakauma luennoille osallistumisesta ja luentovideoiden katsomisesta (prosentuaalisena jakaumana välillä 0 - 168 %:ia).

Keskiarvo (μ) luennoille osallistumisessa oli 43,90 prosenttia ja luentovideoiden katsomisessa 47,79 prosenttia. Kuvasta voidaan havaita että luentojen ja videoiden seuraaminen painottuvat välille 0-91 %:ia.

5.5.8 Yleinen palaute

Yleinen palaute oli muotoiltu kolmena kysymyksenä:

(1) Oliko kurssilla jotain hyvää, mitä sinusta ei pitäisi muuttaa jatkossa?

Kysymykseen vastasi 28 opiskelijaa. Yksitoista (11) opiskelijaa piti luentovideoita tärkeinä ja kurssin suorittamista tukevina, neljä (4) opiskelijaa koki Unix-ympäristön käytön tärkeänä

(2) Oliko kurssilla jotain huonoa, jota sinusta pitäisi muuttaa jatkossa?

Vastauksia oli 25. Seitsemän (7) opiskelijaa kritisoi tehtävien palautus aikarajoja, neljä (4) opiskelijaa usean käyttöjärjestelmän käyttöä ja neljä (4) opiskelijaa kritisoi versionhallinnan käyttöä.

(3) Vapaa palaute opintojaksosta, esimerkiksi kehittämisehdotukset, vahvuudet, heikkoudet.

Vapaaseen palautteeseen opintojaksosta vastasi 18 opiskelijaa. Neljän (4) opiskelijan antama palaute liittyi harjoitustyöhön, sen vaikeuteen / saamatta jääneeseen palautteeseen. Kolme (3) opiskelijaa piti kurssin kokonaisuutta hyvänä. Muutamat opiskelijat antoivat sekä suullista että kirjallista palautetta vaikeustasoon, ohjeistukseen ja materiaaliin liittyen. Ottamalla tämä edellä mainittu kritiikki (ks. luku 5.7) paremmin huomioon seuraavalla kurssilla, palautteen määrä samoin kuin kurssin hyväksytysti suorittaneiden opiskelijoiden prosentuaalinen osuus, tullee nousemaan selvästi. (Nikula 2011)

5.6 Yhteenveto kyselystä ja sanallisesta palautteesta

Sekä sanallisen että numeerisen palautteen pohjalta voidaan selvästi päätellä, että opiskelijat pitävät luentovideoita sangen hyödyllisinä ja tarpeellisina. Kyselyissä korostui tässä suhteessa kaksi seikkaa; päällekkäiset luennot ja poissaolot. Varsin monet opiskelijat halusivat painottaa luentovideoiden merkitystä oppimisessa. 39 prosenttia myönteisen palautteen antaneista opiskelijoista korosti luentovideoiden osuutta oppimisessa. Seuraavassa käsittelen tilastoitua materiaalia kahdelta kannalta: opiskelijoiden arviona kurssista ja opituista asioista.

Tilastoidun palautteen mukaan kokonaisarvio kurssista oli $\mu = 3{,}35$ asteikolla 1…5, mitä voidaan pitää kohtuullisen hyvänä, tosin hajonta oli suhteellisen iso $\sigma = 0{,}91$. Kurssista saatavia opintopisteiden määrää pidettiin jonkin verran liian pienenä keskiarvolla 2,30 ($\sigma = 0{,}71$).

Kurssin katsottiin lähtevän liikkeelle hieman liian perusasioista (asteikolla 1..7) $\mu = 3{,}35$, toisaalta käsiteltyjen asioiden määrää pidettiin riittävänä keskiarvolla $\mu = 4{,}18$. Kurssin tavoitteena ollut kuva käytännön ohjelmoinnista sai kurssilaisilla hyvän keskiarvon $\mu = 4{,}68$ ($\sigma = 0{,}91$).

C-kielen osaamista luokittelevat asiat (ks. kuva 8) jakautuivat välille $\mu = 3{,}33$ - $\mu = 4{,}68$ (asteikolla 1-5), mitä voidaan pitää hyvänä tuloksena. Esikääntäjä, kääntäminen ja linkkaus ($\mu = 3{,}33$) sekä linkitetty lista ($\mu = 3{,}38$) osoittautuivat vaikeimmiksi, mikä toisaalta näyttää korreloivat aiheiden käsittelyn määrään luennoilla.

Käytännön ohjelmoimista mittaavat tekijät jakautuivat välille $\mu = 2{,}85$ - $\mu = 3{,}70$, missä hankalimmat aihepiirit olivat systeemitestaus ($\mu = 2{,}85$) ja testauksen hallinta, kattavuus ja automatisointi ($\mu = 2{,}85$).

Käytettyjen käyttöjärjestelmien osaamisessa ja käytössä (ks. kuva 10) jäivät arvot välille μ = 2,05 (debuggerin käyttö) ja μ = 4,00 (Codeblocks kehitysympäristön osaaminen).

Kuvasta 10 voidaan päätellä että suurin osa opiskelijoista käytti pääsääntöisesti Codeblocks-ympäristöä, keskiarvon ollessa 4,00 ja hajonta oli Codeblocks-kehitys-ympäristön osaamisessa alle yhden (σ = 0,74), toisaalta Linux-käyttöjärjestelmän kohdalla keskiarvo oli 3,55 ja hajonta yli yhden (σ = 1,14). Myös Linuxin perustyökalujen osalta esimerkiksi gcc-kääntäjä (σ = 1,19) ja gedit-editori sekä Makefile (σ = 1,19) ovat hajonnat suurehkoja. Tarkasteltaessa kuvan 10 käyriä jotka kuvaavat eri kehitysympäristöjen ja työkalujen käytön osaamista sekä käyttöä, voidaan havaita niiden korreloivan hyvin selvästi.

Otettaessa huomioon opiskelijoiden antaman sanallisen palautteen, voitaisiin pitää perus-teltuna käyttää vain yhtä kehitysympäristöä koko kurssin ajan, siis sekä periodilla kolme että neljä. Tällä varmistettaisiin että opiskelijat käyttävät Linuxin perustyökaluja, kuten debuggereita ja versionhallintaa, jolloin myös projektityön tallennuksessa SVN-versionhallintaan ilmenevistä ongelmista pääsääntöisesti vältyttäisiin. Opiskelijoilta saadun palautteen mukaan versionhallinnan käyttäminen Ubuntussa oli helpompaa kuin Windows-ympäristössä. Tämä seikka puoltaa Ubuntun valintaa ohjelmointiympäristöksi, sillä kurssin tärkeimpänä aihekokonaisuutena oli ohjelmistoprojektin suunnittelu ja toteuttaminen. Rajoittamalla Viope-oppimisympäristön harjoitusten palautuskertojen määrää, voitaisiin myös ohjata opiskelijoita käyttämään haluttua ohjelmointiympäristöä.

5.7 Aiemmat tutkimukset LTY:ssä

Lappeenrannan teknillisessä yliopistossa on tehty useampia tutkimuksia Ohjelmoinnin perusteet-kurssiin liittyen. Nikula, Gotel ja Kasurinen (Nikula 2011, 34) pitivät viiden vuoden pitkittäistutkimuksessaan tärkeimpinä tekijöinä pyrittäessä mahdollisimman hyviin oppimistuloksiin ja pieniin kurssien keskeyttämisprosentteihin:

- kurssin hygieniatekijöitä (opetusmateriaalit, opetustilat)
- sisäistä motivointia (kurssin sisällön kiinnostavuus ja hyödyllisyys)

- ulkoisten motivointitekijöiden käyttöönottoa jolloin parannetaan opiskelijoiden käyttäytymisen ennakointia.

Käytännön ohjelmointi -kurssin opiskelijapalautteessa keväällä 2011 korostui *luentovideoiden* merkitys opetusmateriaalina sekä harjoitustyöstä toivottava *palaute*. Nämä kaksi edellä mainittua tekijää liittyvät aiemman tutkimuksen (Nikula 2011) hygieniatekijöihin ja sisäiseen motivointiin, jotka siis tulivat tässäkin seurannassa esiin. Käytännön ohjelmointi -kurssin Webrobol-kyselyyn 2011 (liite 2) tässä työssä verrattavat Ala-Mutkan (Ala-Mutka 2005) ja Milnen (Milne 2002) tutkimukset pohjautuvat yliopistoissa 1-2 ohjelmointikurssia suorittaneiden opiskelijoiden seurantaan.

Jenkins'n tutkimuksen mukaan sisäinen motivointi (49,6 %) muodosti hieman suuremman osuuden opiskelijoiden opiskeluasenteesta kuin ulkoiset motivointitekijät (47,1 %). Prosenttiosuuksia voidaan pitää yllättävän korkeina, mutta tutkimuksen asettelu ja kysymykset johtivat suhteellisen suuriin prosenttiosuuksiin. (Jenkins 2001)

6 Keskustelu

Vertailen seuraavassa Ala-Mutkan (2005) ja Milnen (2002) tutkimuksia tämän diplomityön tutkimustuloksiin. Näiden kolmen tutkimuksen kysymysten eroavaisuuksien tähden oli vaikea tehdä suoria vertailuja, mutta seuraavia alla esiteltyjä yhtäläisyyksiä kuitenkin on havaittavissa.

Verrattaessa tämän Webropol-kyselyn tuloksia aiempiin opiskelijatutkimuksiin, voidaan pyrkiä löytämään yhteisiä hankaliksi osoittautuneita aihealueita joita tulisi opetuksessa painottaa. Ala-Mutkan (2005) tutkimusta, johon osallistui kuudesta yliopistosta 559 opiskelijaa, jotka olivat ennen kyselyä suorittaneet 1-2 ohjelmointikurssia, voidaan pitää sangen kattavana ja vertailuun sopivana materiaalina. Tässä tutkimuksessa ongelmallisimmiksi käsitteiksi erottuivat: *ohjelmiston suunnittelu, toimintojen jakaminen prosesseihin ja virheiden löytäminen omista ohjelmista*. Ohjelmoinnissa hankalimpia olivat *osoitteet ja viitteet, abstraktit tietotyypit, virheiden käsittely ja ohjelmakirjastojen käyttö*. Parhaana opiskelumateriaalina opiskelijat pitivät tarjolla olleita ohjelmaesimerkkejä.

Käytännön ohjelmointi -kurssin Webropol-kyselyn mukaan opiskelijat mielestään hallitsivat arkkitehtuurisuunnittelun ja moduulisuunnittelun keskimääräistä paremmin. Arkkitehtuurisuunnittelu sai keskiarvon 3,57 (σ = 1.03) ja moduulisuunnittelu μ = 3,70 (σ = 0,84) ollen arvojoukon ylälaidassa. Tässä tulee huomauttaa että Webropol-kyselyn ja Ala-Mutkan tutkimuksen asteikot olivat käänteiset. Webropol-lomakkeella kysyttiin asioiden osaamista asteikolla (1) en ole varma mistä on kysymys ... (5) hyvä osaaminen ja Ala-Mutkan tutkimuksessa vaikeutta oppia asiaa asteikolla (1) helppo oppia ..(5) vaikea oppia. Käytännön ohjelmointi -kurssilla ohjelmoinnin peruskäskyt kuten syöttö, tulostus, valinta-, toisto- ja hyppyrakenteet, virheenkäsittely oli niputettu yhteen kysymykseen ja vastausten keskiarvo oli 4,58 (σ = 0,74), eli opiskelijat pitivät osaamistaan näillä aihealueilla hyvänä. Ala-Mutkan tutkimuksessa kysymys oli jaettu useampaan osioon seuraavasti (hakasulkeissa keskiarvo käännettynä samalle skaalalle kuin Webropol-kyselyssä); syöttö / tulostus μ = 2,96 (σ = 1,04) [3,04], virheiden käsittely μ = 3,33 (σ =

1,01) [2,67], valintarakenteet μ = 1,98 (σ = 0,90) [4,02] ja toistorakenteet μ = 2,09 (σ = 0,97) [3,91]. Ala-Mutkan tutkimuksen ja Webrobol-kyselyn tuloksien kesken havaittiin siis suurehko eroavaisuus.

Dundeen yliopistossa tehdyn tutkimuksen, Difficulties in Learning and Teaching Programming (Milne 2002), kattavuus oli noin 70 opiskelijaa. Sen mukaan aloittelevat ohjelmoijat pitävät vaikeimpina aihealueina *osoittimia ja dynaamiseen muistinkäsittelyyn liittyviä operaatioita*. Milnen tutkimuksen tulokset ovat yhtenevät Käytännön ohjelmointi -kurssin kyselyn tulosten kanssa. Lappeenrannan teknillisessä yliopistossa tehdyn kyselyn *mukaan linkitetty lista, osoittimet, dynaaminen muistinhallinta ja rakenteiset tietorakenteet* olivat vaikeimmat aihealueet.

Mistä suuret eroavaisuudet Ala-Mutkan (2005) ja Lappeenrannassa tehdyn tutkimusten välillä voivat johtua? Merkittävimmät syyt ovat varmaankin vertailtujen kurssien sisältöjen poikkeavuudet. Lisäksi, jos kysymykset olisi esitetty samansisältöisinä, olisi vastausten analysointi helpompaa. Webrobol-kysely lähetettiin opiskelijoille (53) jotka olivat tehneet kaikki pakolliset tehtävät, ja näistä opiskelijoista 40 (75,5 %:ia) palautti kyselyn. Ala-Mutkan tutkimukseen osallistui opiskelijoita kuudesta yliopistosta otannan ollessa suuruudeltaan 559.

Milne’n (2002) ja tämän tutkimuksen tulokset ovat puolestaan yhteneväiset niiden aihealueiden suhteen joita opiskelijat pitivät vaikeimpina oppia.

Ala-Mutkan (2005) tutkimukseen osallistuneista opiskelijoista 58,6 %:lla oli ohjelmointikokemusta ennen yliopisto-opiskelua ja näistä opiskelijoista 40,6 % pitivät ohjelmointiosaamistaan kohtuullisen hyvinä. Opiskelijoiden osaamistason suurehko vaihtelu lienee yleistä yliopisto-opiskelijoiden keskuudessa, jolloin se asettaa haasteita kurssien järjestäjille, jotta kaikkien opiskelijoiden motivointi onnistuu.

Kun luentomateriaalit ja harjoitustehtävät ovat kurssia varten valmiiksi laadittuina, ei muutoksia tarvitse tehdä kuin projektityön aiheenvalinnassa. Tällä tavoin menetellen päästään kohti samaa rakennetta kuin muillakin Lappeenrannan teknillisen yliopiston ohjelmistotekniikan kursseilla on käytössä. Tutkimusartikkelissa, joka käsittelee Käytännön ohjelmointi -kurssin teknisen rakenteen parantamista Lappeenrannan teknillisessä yliopistossa (Nikula et al. 2009), todetaan että käytettävät opetustekniikat saattavat hidastaa opiskelijoiden oppimista. Artikkelissa päädytäänkin lopputulokseen, että on kaksi vaihtoehtoa parantaa oppimistuloksia: joko löytää paremmin kurssia tukevia opettamisteknologioita tai panna enemmän painoa pedagogiselle puolelle. Seurantatutkimuksessa, joka tehtiin Ohjelmoinnin perusteet-kurssille vuosina 2005-2006, havaittiin että ohjelmointikielen (Python) valinnalla ja muiden opetusmateriaalien kehittämisellä pienennettiin ohjelmointia pakollisena oppiaineena opiskelevien keskeyttämismäärää 20 %:lla. (Kasurinen 2008)

Opiskelijapalautteen mukaan Noppa-portaalin käyttöä keskitettynä kurssimateriaalin talletuspaikkana tulisi vahvistaa. Lisäksi kurssilla tulee jatkossa olemaan käytössä kaksi erillistä luentomonistetta; C-kieli ja käytännön ohjelmointi osa 1 (Kasurinen 2011) sekä uusi jatko-osa edellä mainitulle luentomonisteelle, C-kieli ja käytännön ohjelmointi osa 2. Kurssimateriaalin yhtenäisempi ja harkittu rakenne tullee jatkossa nostamaan kurssin läpäisyprosenttia sekä kurssiarvosanoja. Voidaan myös olettaa, että käytettäessä kurssin alusta alkaen Linux-ympäristöä, ei opiskelijoille tulisi ongelmia päättää kumpaa kehitysympäristöä käyttää, jolloin myös luennoitsijan ja assistentin työkenttä selkenisi keskityttäessä yhteen kehitysympäristöön.

Jatkotutkimuksen aiheena voidaan pitää seuraavalla käytännön ohjelmointi -kurssilla saatavaa opiskelijapalautetta ja hypoteesin lopullista testausta (ks. luku 3.2) uudesta kurssimateriaalista (ks. liite 1).

7 Yhteenveto

Tämän tutkimuksen tarkoituksena oli koostaa luentomateriaalia käytettäväksi Käytännön ohjelmointi -kurssilla Lappeenrannan teknillisessä yliopistossa. C-ohjelmointikieli on jossain määrin menettänyt asemiaan opetuskielenä verrattaessa Suomen yliopistojen kurssitarjontaa, kun on siirrytty käyttämään oliopohjaisia kieliä ja suunnittelumenetelmiä. Työelämässä on kuitenkin vielä selvää tarvetta myös perinteisten ohjelmointikielten taitajille.

Käytännön ohjelmointi -kurssin C-kielen perusteisiin oli jo oma oppaansa ja tehdyn konstruktion tuloksena rakentui luentomoniste C-kieli ja käytännön ohjelmointi – osa 2. Luentomoniste käsittelee käytännön ohjelmointiin liittyviä aihekokonaisuuksia ollen samalla myös opas ja muistilista opiskelijoille keskeisistä Unix-komennoista esimerkiksi versionhallinnan yhteydessä. Muutenkin oppaassa pyritään antamaan sekä ohjelmaesimerkein että komentojonoin opiskelijoille selkeä kuva käsiteltävistä aihekokonaisuuksista. Käytännön ohjelmointi -kurssin harjoitustyötä ajatellen luentomoniste sisältää myös ohjeet testauksen raportointiin ja esimerkki Excel-pohjat testauspöytäkirjasta ja testauspäiväkirjasta, joita opiskelijat voivat hyödyntää.

Liitteenä oleva luentomoniste koostuu seitsemästä yhtenäisestä lukukappaleesta, joissa on alussa lyhyt johdanto ja lopussa yhteenveto, jossa mainitaan aihekokonaisuuteen liittyvät www-linkit. Kahdeksas luku on yhteenveto sisältäen aihekokonaisuuksia joita ei kurssilla käsitellä, mutta joista uskotaan olevan opiskelijoille vastaisuudessa hyötyä. Luentomonisteen liitteenä on suhteellisen lyhyt tyyliopas sisältäen kuusi sivua, mutta se tullee ohjaamaan opiskelijoita yhtenäisempään ohjelmointitapaan, joka puolestaan johtaa koodin parempaan luettavuuteen ja helpompaan tarkastettavuuteen.

Lähteet

Aalto-yliopisto, 2010, *Tietotekniikan koulutusohjelman opinto-opas,* [viitattu 31.05.2011],<URL:https://into.aalto.fi/display/fitik/Opinto-oppaat>.

Åbo Akademi, 2010, *Undervisningsprogrammet 2009-2010 och 2010-2011,* [viitattu 01.06.2011], <URL:http://www.abo.fi/student/it_undervisningsprogram>.

Ala-Mutka, K., Järvinen, H. & Lahtinen Essi. 2005, June, *A Study of the Difficulties of Novice Programmer. Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education* , Monte de Caparica p. 14-18.

Anderson, M.S. 2001, "The complex relations between the academy and industry - Views from the literature", *The Journal of Higher Education,* vol. 72, no. 2, pp. 226-246.

Carver, J. & Henderson, L. 2006, "Viope as a Tool for Teaching Introductory Programming: An Empirical Investigation", *CSEET '06: Proceedings of the 19th Conference on Software Engineering Education & Training (CSEET'06)* .

Cheng, H.H. 2009, "C for the Course", *Mechanical Engineering,* vol. 131, no. 9, pp. 50-52.

Cheng, H.H. *Ten Reasons to Teach and Learn Computer Programming in C,* [viitattu 17.08.2011] ,<URL:http://iel.ucdavis.edu/publication/WhyC.html>.

CodeBlocks, *Code::Blocks-sivusto,* [viitattu 25.04.2011], <URL:http://www.codeblocks.org/downloads/binaries#windows>.

GNU Coding Standards, [viitattu 28.3.2011], <URL:http://www.gnu.org/prep/standards/html_node/index.html>.

Gorgone, J.T., Davis, G.B., Valacich, J.S., Topi, H., Feinstein, D.L. & Longenecker, H.E. 2002, *Model Curriculum and Guidelines for Undergraduate Degree Programs in Information Systems,* ACM, New York, NY, USA, [viitattu 19.09.2011], <URL:http://www.is.informatik.unikiel.de/~thalheim/Wirtschaftsinformatik/StudienprofilWirtschaftsinformatik/LinkedDocuments/IS%202002.pdf>.

Haikala, I. & Järvinen, H. 2004., *Käyttöjärjestelmät,* Talentum, Helsinki.

Helsingin Yliopisto, 2010, *Matemaattis-luonnontieteellinen tiedekunta, opinto-opas,* [viitattu 01.06.2011], <URL:http://www.helsinki.fi/ml/opinto-opas/index.html>.

Itä-Suomen yliopisto, 2010, *Luonnontieteiden ja metsätieteiden tiedekunta, Opinto-opas,* [viitattu 01.06.2011], <URL:http://www.uef.fi/c/document_library/get_file?uuid=fc37d285-1bd2-404e-bac5-16f3621f2ed7&groupId=78409&p_l_id=78564>.

Järvinen, P. & Järvinen, A. 2004, *Tutkimustyön metodeista*, Opinpaja, Tampere.

Jenkins, T. 2001, "The motivation of students of programming", *ACM SIGCSE Bulletin*, vol. 33, no. 3, pp. 53-56.

Jokela, T. 2001, *Assessment of user-centred design processes as a basis for improvement action An experimental study in industrial settings.* Acta Universitatis Ouluensis. Series A, Oulun yliopisto, Oulu.

Jyväskylän yliopisto, 2010, *Informaatioteknologian tiedekunnan opinto-opas*, [viitattu 01.06.2011], <URL:http://opinto-opas.jyu.fi/it/2010/opas/pdf/index.php>.

Kasurinen, J. 2011, *C-kieli ja käytännön ohjelmointi osa 1*, Lappeenrannan teknillinen yliopisto.

Kasurinen, J. & Nikula, U. 2008, "Lower dropout rates and better grades through revised course infrastructure", *Proceedings of the 10th IASTED International Conference on Computers and Advanced Technology in Education* , pp. 152-157.

Kernighan, B.W. & Pike, R. 1999, *The practice of programming*, Addison-Wesley, Indianapolis.

Kernighan, B.W. & Ritchie, D.M. 1988, *The C programming language*, Prentice-Hall, Englewood Cliffs (NJ).

Koffman, E.B., Miller, P.L. & Wardle, C.E. 1984, "Recommended curriculum for CS1, 1984", *Commun.ACM*, vol. 27, no. 10, pp. 998-1001.

Koffman, E.B., Stemple, D. & Wardle, C.E. 1985, "Recommended curriculum for CS2, 1984: a report of the ACM curriculum task force for CS2", *Commun.ACM*, vol. 28, no. 8, pp. 815-818.

Koskimies, K. & Mikkonen, T. 2005, *Ohjelmistoarkkitehtuurit*, Talentum, Helsinki.

Lapin yliopisto, 2010, *Menetelmätieteiden opinto-opas 2010-2011*, [viitattu 01.06.2011], <URL:http://www.ulapland.fi/Suomeksi/Yksikot/Yhteiskuntatieteiden_tiedekunta/Opiskelu/Opiskeluinfoa/Opinto-opas.iw3>.

Lappeenrannan teknillinen yliopisto 2010, *Tekniikan opinto-opas 2010-2011*, [viitattu 25.04.11], <URL:http://www.lut.fi/fi/lut/studies/tools/studyguide/Documents/Tekniikan%20opinto-opas%202010-2011.pdf>.

Lappeenrannan teknillinen yliopisto 2006, *Tekniikan opinto-opas 2006-2007*, [viitattu 20.08.11], <URL:http://www.lut.fi/fi/lut/studies/tools/studyguide/Documents/DIopas0607.pdf> .

Lester, R.K. & Sotarauta, M. 2007, "*Yliopistot, innovaatio ja alueiden kilpailukyky: Huomioita ja johtopäätöksiä Local Innovation Systems -projektista*" , Technology review 214/2007, Tekes,Helsinki, pp. 31-42.

March, S.T. & Smith, G. 1995, "Design And Natural-Science Research On Information Technology", *Decision Support Systems,* vol. 15, no. 4, pp. 251-266.

Milne, I. & Rowe, G. 2002, "Difficulties in Learning and Teaching Programming—Views of Students and Tutors", *Department of Applied Computing, University of Dundee, Dundee* , pp. 55-66.

Nikula, U., Alaoutinen, S., Kasurinen, J. & Pirinen, T. 2009, "Improving the technical infrastructure of a programming course", *Proceedings - 2009 9th IEEE International Conference on Advanced Learning Technologies, ICALT 2009*, pp. 374-376.

Nikula, U., Gotel, O. & Kasurinen, J. 2011, "A Motivation Guided Holistic Rehabilitation of the First Programming Course", *ACM Transactions on Computing Education,* Accepted for publication.

Oulun yliopisto, 2010, *Opinto-opas 2010-2011 Teknillinen tiedekunta Oulun yliopisto,* [viitattu 01.06.2011], <URL:http://www.lutk.oulu.fi/tiedostot2007/opinto_opas/LuTKOpintoopas2010_2011.pdf>.

Slaughter, S. & Leslie, L.L. 1997, *Academic capitalism : politics, policies, and the entrepreneurial university,* The Johns Hopkins University Press, Baltimore.

Tampereen teknillinen yliopisto, 2010, *Opinto-opas 2010-2011,* [viitattu 01.06.2011], <URL:http://www.tut.fi/wwwoppaat/opas2010-2011/perus/laitokset/Ohjelmistotekniikka/index.html>.

Tampereen yliopisto, 2010, *Opinto-oppaat 2010-2011,* [viitattu 01.06.2011], <URL:https://www10.uta.fi/opas/tutkintoOhjelma.htm?rid=4380&uiLang=fi&lang=fi&lvv=2010>.

The Joint Task Force on Computing Curricula 2001, "Computing Curricula 2001 Computer Science", vol. 1, no. 3.

Trauth, E.M., Farwell, D.W. & Lee, D. 1993, "The IS expectation gap: Industry expectations versus academic preparation", *MIS Quarterly,* vol. 17, no. 3, pp. 293.

Turun yliopisto, 2010, *Opinto-opas Matemaattis-luonnontieteellinen tiedekunta 2010-2011,* [viitattu 01.06.2011], <URL:https://nettiopsu.utu.fi/opas/tutkintoOhjelma.htm?rid=4108&uiLang=fi&lang=fi&lvv=2010>.

Vaasan yliopisto, 2010, *Opinto-opas 2010-2011,* [viitattu 01.06.2011], <URL:http://www.uwasa.fi/tekniikka/oppaat/opintooppaat/opas/>.

Vanhala, E. & Nikula, U. 2010, *Python 3 – ohjelmointiopas, versio 1.0, Käsikirjat 13*, Lappeenrannan teknillinen yliopisto.

C-KIELI JA KÄYTÄNNÖN OHJELMOINTI – OSA 2

Lauri Kyttälä

(jatkuu)

Sisällysluettelo

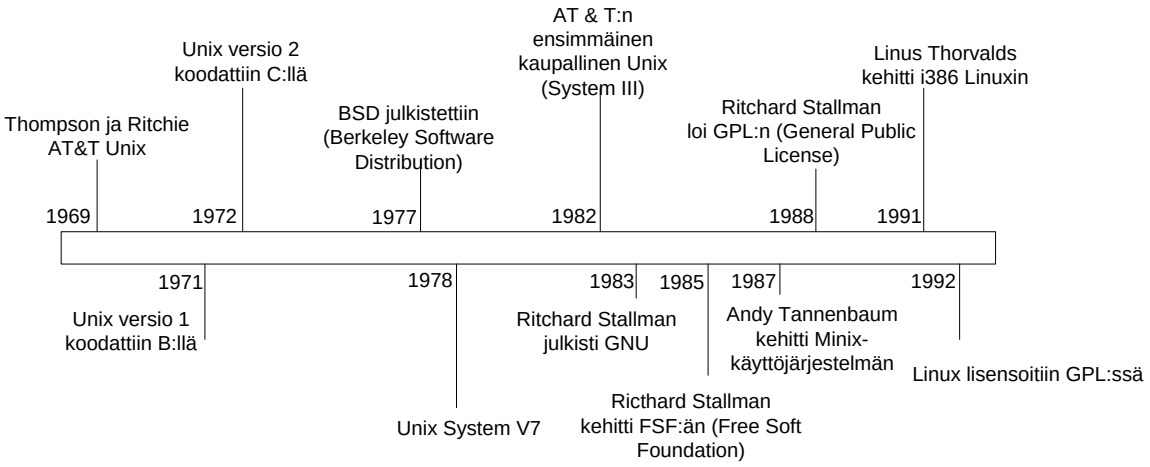
Luku 1: Ohjelmistokehitysympäristöt

Unix- ja Windows- käyttöjärjestelmät eroavat toisistaan sekä ohjelmisto- että yleiskäyttöisinä ympäristöinä. Seuraavassa esitellään käyttöjärjestelmien perusteita lyhyesti vertaillen niitä toisiinsa.

1.1.Unix

Ohjelmointi C-kielellä Unix / Linux ympäristössä on sikäli luonnollista, että Unix on kirjoitettu C-kielellä, näin ollen esim. perus C-kääntäjät tulevat oletuksena mukana asennuksessa, tai ainakin ne ovat helppoja asentaa sekä Linuxissa ilmaisia. Lisäksi kääntäjien käyttö on yleensä varsin ongelmatonta ja suoraviivaista. Unixista on olemassa useita kaupallisia versioita mm. Oracle Solaris. Tunnetuin Unix-klooneista on nykyään Linux, joka on GNU (GNU's Not Unix) GPL-lisenssin (General Public Licence) mukaista koodia. GNU/Linux- jakelupaketteja on useita jotka kuitenkin perustuvat Debian, Red Hat, Fedora, Slackware ja SuSe paketteihin. Red Hat on kaupallinen jakelupaketti, johon samanniminen pörssiyhtiö tarjoaa mm. ylläpitopalveluja.

Linuxissa on nykyään varsin kehittynyt graafinen käyttöliittymä, joka sallii ohjelmistokehityksen halutussa kehitysympäristössä merkkipohjaisen (komentorivi) käyttöliittymän sijaan. Gnome-työpöydälle on saatavissa mm. varsin monipuolinen Anjuta kehitystyöympäristö, IDE (Integrated Development Environment). Anjuta mahdollistaa projektinhallinnan sekä sisältää debuggerin, kääntäjän sekä monipuolisen editorin. Linux-jakelun mukana tulee myös ilmainen OpenOffice-ohjelmisto. Kuvassa 1 esitetään Unix – GNU/Linux kehityskaari alkaen vuodesta 1969 päättyen Linuxin julkistamiseen.



Kuva 1. UNIX GNU/Linux aikajana (Robot Wisdom).

1.2.Windows

Windows käyttöjärjestelmänä edustaa puhtaasti kaupallista linjaa ollen työasemamarkkinoilla johtavassa asemassa 92 prosentin osuudellaan (Keizer 2010). Käyttöjärjestelmään on saatavissa myös freeware ja shareware ohjelmia. Windowsin tuettuja työasemaversioita on käytössä tällä hetkellä kolme; Windows XP, Vista ja Windows 7.

Nykyään useimmat kehitysympäristöt tukevat Windows- käyttöjärjestelmää sen johtavan markkina-aseman tähden, mutta monet kehitysympäristöt tukevat myös GNU/Linuxia kuten esim. Eclipse ja CodeBlocks.

1.3.Ubuntu Linux

Ubuntu on avoimesta lähdekoodista koostuva Linux-käyttöjärjestelmä. Ubuntu sisältää kaikki peruskäyttöön tarvittavat ohjelmat, kuten esim. tekstinkäsittelyn, taulukkolaskennan, sähköpostiohjelman ja nettiselaimen. Ohjelmia on helppo asentaa lisää joko komentoriviltä tai synaptic asennustyökalulla. Graafisen asennusohjelman avulla käyttöjärjestelmän asennus on ongelmatonta ja sujuu nopeasti. Ubuntu-projekti on sitoutunut noudattamaan vapaan ohjelmistokehityksen periaatteita.

1.4.Unix / Linux työpöytäympäristöt

Työpöytää käytetään sovellusten hallintaan ja käyttöön. X11 ikkunointijärjestelmälle on kehitetty useita yhtenäisen käyttöliittymän antavia työpöytäympäristöjä. Kaupallisella puolella Sun:lla ja DEC:llä (Digital Equipment Corporation) oli omat tuotteensa, sittemmin lähes kaikki ovat siirtyneet käyttämään The Open Groupin kehittämää Common Desktop Environment:ia (CDE). Linux-puolella 1998 julkaistiin KDE 1.0, joka käyttää Qt-käyttöliittymäkirjastoa. Vuonna 1997 käynnistyi GNOME-projekti (GNU Network Object Model Environment), joka on nykyisin GNU:n virallinen työpöytäympäristö. (Wikipedia X Window System)

Gnome-projektin tavoitteena on tarjota helppokäyttöinen tietokoneen graafinen käyttöliittymä ja kattava kehitysalusta sovelluksille. Gnome on saatavilla useimpiin Linux- ja BSD-käyttöjärjestelmiin. Ubuntun oletustyöpöytäympäristö Gnome on käytetyin UNIX- ja Linux-työpöytä ja kehitysalusta. Vaihtoehtoisesti on mahdollista asentaa jälkikäteen KDE- tai Xfce-työpöytäympäristöt Ubuntun Gnome-työpöydän sovellustenhallinnan kautta (Sovellukset - Ubuntu sovellusvalikoima). Ohjeet työpöytien asennukseen löytyvät esim. linkistä: http://www.psychocats.net/ubuntu/kde.

1.5.Ubuntun editorit

Ubuntuun löytyy monia hyviä ja helppokäyttöisiäkin editoreja kuten esim. `vi`, `nano`, `gedit` (kuuluvat perusasennukseen) ja `emacs` (asennettava erikseen). Emacs kehittyneempänä editorina tarjoaa monipuolisia mahdollisuuksia myös koodin kääntämiseen suoraan alasvetovalikon kautta.

1.6.Gnome-sovelluksia

Muutamia käytetyimpiä Gnomen mukana toimitettavista sovellusohjelmista ovat:

- Evince – PDF- ja PostScript-dokumenttien katselin
- Evolution – sähköposti- ja kalenteriohjelma
- Empathy – pikaviestiohjelma
- Screenshot – kuvankaappausohjelma.

1.7.Ubuntun asennus

Linux voidaan asentaa joko Windowsin rinnalle nk. dual-boot:ina tai virtuaalikoneeseen. Virtuaalikone löytyy esim. linkistä: http://www.vmware.com/products/player/, samoin ohjeet sen asentamiseen. Dual-boot asennus hoituu sekin suoraviivaisesti asennusohjelman ohjeitten mukaisesti. Valittaessa asennustavaksi dual-boot, GRUP-loader ohjelma latautuu koneelle hoitaen jatkossa alkukäynnistyksen. Ubuntu Linux-käyttöjärjestelmä on ladattavissa osoitteesta http://www.ubuntu-fi.org/ iso- tiedostoformaatissa.

1.8.Ohjelman kääntäminen Linuxissa

Ohjelma voidaan kirjoittaa esim. gedit-editointiohjelmalla ja tallettaa haluttuun hakemistorakenteeseen. Tämän jälkeen ohjelma käännetään komennolla:

```
gcc –o ohjelma ohjelma.c
```

Mikäli käännös onnistuu ilman virheitä, tallentuu hakemistoon suoritettava ohjelmatiedosto, joka on ajettavissa komennolla: `./ohjelma` (./:llä kohdistetaan suorituskomento työhakemiston ajettavaan binääritiedostoon `ohjelma`). On suositeltavaa käyttää käännöskomentoa parametreilla; `gcc –o ohjelma ohjelma.c -std=c99 –Wall`, määreellä `-o` käännetään lähdekoodi `ohjelma.c` binääriseksi suoritettavaksi tiedostoksi, määreellä `-std=c99` lähdekoodin tulee täyttää ANSI C99 standardin mukaiset vaatimukset ja `Wall` aktivoi kaikki varoitukset (warnings).

1.9.Makefile

GNU Make on työkalu joka hyväksyy syötteekseen Makefile-tiedoston, ks. esimerkki 1 alla. Työhakemistossa on siis kolme lähdekooditiedostoa `project.c`, `read.c` ja `write.c` jotka ensi käännetään kukin erikseen ja lopuksi yhdistetään yhdeksi suoritettavaksi tiedostoksi `project`. Makefilen tarkoituksena on yksinkertaistaa ja helpottaa ohjelman käännöstä sen koostuessa useammasta komponentista. Komennolla `make –f <tiedosto>` voidaan komennon parametrina antaa tiedoston nimi joka suoritetaan.

Esimerkki 1.

```
# Yksinkertainen Makefile
project:project.o read.o write.o
          gcc –std=c99 -Wall project.o read.o write.o -o project
```

Ajettuna esimerkin 1 Makefile suorittaa seuraavat komennot:

```
cc    -c -o project.o project.c
cc    -c -o read.o read.c
cc    -c -o write.o write.c
gcc project.o read.o write.o -o project
```

jolloin syntyy suoritettava binääritiedosto `project`.

(jatkuu)

Käännettävän ohjelman koostuessa useammasta komponentista on hyödyllistä tehdä käännös- ja linkkausvaihe erikseen, jolloin aikasäästöt saattavat olla huomattavat.

Kuvassa 2 laatikko kuvaa aina yhden Makefilen lohkon suoritusta:

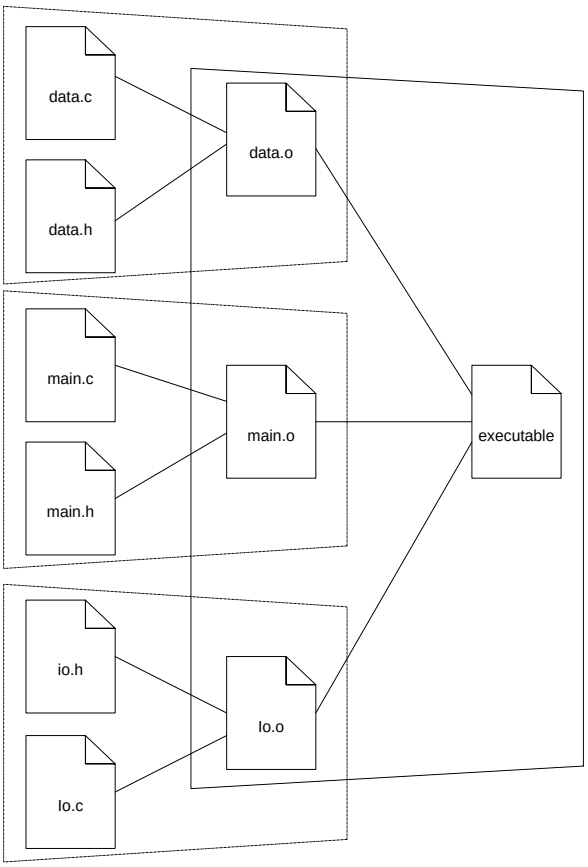
```
all: data.o io.o main.o
          gcc data.o io.o main.o –o executable

data.o: data.c data.h
          gcc –c –std=c99 –Wall data.c

io.o: io.c io.h
          gcc –c –std=c99 –Wall io.c

main.c: main.c data.h io.h main.h
          gcc –c –std=c99 –Wall main.c
```

Makefile vaatii rivinvaihdon yhteyteen tabulaattorin toimiakseen.



Kuva 2. Makefilen hierarkinen lohkorakenne.

(jatkuu)

(liite 1 jatkoa)

Alla vielä toinen esimerkki Makefilestä jossa on käytetty parametrejä.

Esimerkki 2.

```
# Kommentti
# make komento täydentää all: rivin vaatimukset
# make täydentää clean: rivin joka poistaa (rm = remove) rekursiivisesti
(-r) mainitut tiedostot
# Voit käyttää muuttujia kirjoittaessasi Makefile:n. Tämä on käytännöl-
listä kun
# haluat esim. vaihtaa kääntäjää tai kääntäjän parametreja.
# muuttuja CC ilmoittaa kääntäjän
# muuttuja CFLAGS välittää parametrit kääntäjälle

CC = gcc
CFLAGS = -c -Wall -std=c99

all: koe
koe: main.o my_stdio.o
          $(CC) main.o my_stdio.o -o koe

main.o: main.c
          $(CC) $(CFLAGS) -c main.c

my_stdio.o: my_stdio.c my_stdio.h
          $(CC) $(CFLAGS) -c my_stdio.c
# make clean
clean:
          rm -r *.o koe *.*~ *~
```


(jatkuu)

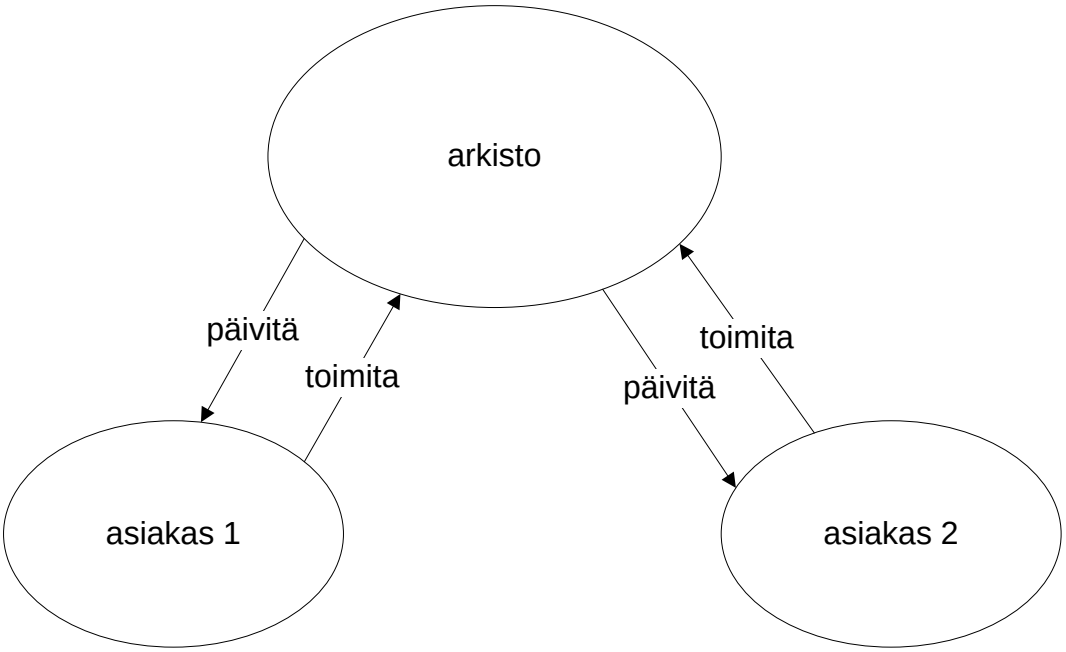
1.10.Lopuksi

Ohessa on muutamia hyödyllisiä linkkejä liittyen luvun aiheeseen:
Ohje gcc-kääntäjästä:
http://www.network-theory.co.uk/gcc/intro/gccintro-sample.pdf
Unix-harjoituksia aloittelijoille:
http://www.ee.surrey.ac.uk/Teaching/Unix/index.html
Ubuntun asennus:
Ubuntu: http://www.ubuntu-fi.org/
virtuaalikone: http://www.vmware.com/products/player/
Makefile:
http://linux.fi/wiki/Makefile
http://oreilly.com/catalog/make3/book/index.csp

Luku 2: Versionhallinta

Yksi tärkeimmistä asioista ohjelmistoprojekteissa on versionhallinta. Versionhallintajärjestelmät rakentuvat asiakas-palvelin periaatteelle yksinkertaistetun kuvan 3 mukaan. Tällä rakenteella varmistetaan se, että ohjelmamoduulista säilyy aina alkuperäinen kappale arkistossa (repository) ja vain kopiota editoidaan.

Ohjelmistokehityksen aikana tulee taata stabiili työympäristö, jossa projektiin osallistuvat henkilöt eivät tee samanaikaisia muutoksia moduuleihin, siis ylikirjoita toistensa tekemiä muutoksia. Lisäksi muutokset tulee systemaattisesti kommentoida. Kun ohjelmistoa kehitetään sen moduuleiden versionumerot normaalisti kasvavat, mutta saattaa tulla tarve jonkin moduulin osalta palata edelliseen versioon. Versionhallintaohjelmistot mahdollistavat nämä edellä mainitut, minkä lisäksi ne tallentavat tiedostoja editoineen henkilön tiedot ja aikaleiman.



Kuva 3. Versionhallinnan yksinkertaistettu asiakas-palvelin periaate.

2.1.SVN (SubVersioN)

Tigris SVN (http://subversion.tigris.org/) on avoimen lähdekoodin versionhallintaohjelmisto GNU/Linux- ympäristöön. Versionhallinnan arkisto sisältää kaikki ohjelmamoduulien versiot joista muodostuu ohjelman versiohistoria. Päivitä-komennolla haetaan työversio (working copy) arkistosta. Ideana versionhallinnassa on että työversio olisi editoitavana vain yhdellä henkilöllä kerrallaan. Jos tilanne kuitenkin muuta vaatii, on siihen olevissa menetelmiä joilla työversioon tehdyt muutokset voidaan tehdä hallitusti, SVN:ssä kopioi/muuta/yhdistä -malli.

Toimita-komennolla (commit) tallennetaan editoitu tiedosto ja siihen liittyvät kommentit arkistoon. Versionhallintatyökalun avulla projektissa useamman henkilön on mahdollista tehdä hallitusti muutoksia ohjelmistoon.

2.2.SVN:n tärkeimmät komennot

`svn checkout` [polku] [svn_hakemisto] : noutaa työkopion

[polku] esim. muotoa [https://www2.it.lut.fi/svn/courses/CT60A0210]

`svn update` : päivittää työkopion

`svn ci -m` : tallentaa työkopion arkistoon kommentoituna

esim. **`svn ci –m`** ”Tuotu tiedosto nimi foo.c”

`svn add` : lisää työhakemistosta projektiin tiedoston/tiedostoja

`svn status` : tarkistaa tiedostojen tilanteen edelliseen työkopion päivitykseen (update)

tai

`svn diff` : tarkistaa tiedostojen tilanteen edelliseen työkopion päivitykseen

`svn revert` : peruuttaa tehdyt muutokset

`svn help` : näyttää komentolistauksen

SVN:ään lähetettävissä tiedostonimissä ei tule käyttää skandinaavisia merkkejä (ä, ö) tai välilyöntejä sillä eri käyttöjärjestelmien välillä operoituina nämä voivat aiheuttaa ongelmia. Samoin isot ja pienet kirjaimet erotellaan toisistaan unixin tapaan.

2.3.Lyhyt SVN ohje

Seuraavassa on lyhyt esimerkki SVN-versionhallinnan käyttöönotosta Linux-työasemalla.

Aluksi siirrytään kotihakemistoon, luodaan projektihakemisto ja siirrytään sinne:

```
cd ~ [kotihakemisto]
mkdir CT60A0210
cd CT60A0210
```

Suoritetaan SVN checkout, joka luo paikallisen version SVN-palvelimen tämän hetkisestä sisällöstä:

```
:~/CT60A0210$ svn checkout https://www2.it.lut.fi/svn/courses/CT60A0210
```

(Syötetään tarvittaessa käyttäjätunnus ja salasana.)

Siirretään jokin tiedosto versionhallinnan piiriin:

```
:~/CT60A0210$ cd ryhmaxx
:~/CT60A0210/ryhmaxx$ cp polku/tiedosto
:~/CT60A0210/ryhmaxx$ svn add tiedosto
```

Päivitetään tehty muutos palvelimelle commit-komennolla:

```
:~/CT60A0210/ryhmaxx$ svn ci -m "Tuotiin uusi tiedosto 'tiedoston nimi'."
```

Tarkistetaan, onko palvelimelle tullut muita muutoksia sitten checkoutin.

```
:~/CT60A0210$ svn update
```

(Paikallinen sisältö päivittyy tarvittaessa. Käytetään svn updatea aina tästä eteenpäin tuomaan päivitetty tilanne palvelimelta, checkoutia ei enää tarvitse tehdä.)

Tehdään muutoksia jo versionhallinnan piirissä oleviin tiedostoihin:

```
:~/CT60A0210/teamxx$ pico tiedosto
```

Voidaan halutessa tarkistaa tiedostojen tilanne suhteessa edelliseen päivitykseen (update).

```
:~/CT60A0210/teamxx$ svn status
tai
:~/CT60A0210/teamxx$ svn diff
```

Kun tiedostoa on editoitu, voidaan tehdä taas uusi commit. SVN add-komentoa ei enää tarvita uudestaan tälle tiedostolle, sillä SVN tietää jo pitää kirjaa sen muutoksista.

2.4.Tortoise SVN

Tortoise SVN on avoimen lähdekoodin asiakasohjelma Windows-käyttöjärjestelmiin, joka integroituu resurssienhallintaan (http://tortoisesvn.net/). Sen voi linkittää samaan laitoksen svn-palvelimeen kuin linux-kone.

Lisää TortoiseSVN:än käytöstä linkissä: http://tortoisesvn.net/support.html, josta löytyy myös suomenkielinen käyttöopas.

2.5.Lopuksi

Ohessa on muutamia hyödyllisiä linkkejä liittyen luvun aiheeseen:

Linux:

http://linux.fi/wiki/Ohjelmien_asentaminen

SVN:

http://linux.fi/wiki/Subversion

http://svnbook.red-bean.com/

http://subversion.tigris.org/

TortoiseSVN:

http://tortoisesvn.net/support.html (myös suomenkielinen käyttöopas)

Luku 3: Arkkitehtuuri

Ohjelman arkkitehtuuri muodostuu yleensä useammista ohjelmakomponenteista, jolloin isomman kokonaisuuden hallitseminen ja hahmottaminen on helpompaa. Jaettaessa ohjelma osakokonaisuuksiin voidaan vastaavaa jakoa toteuttaa ohjelmistoprojektissa.

3.1.Ohjelmistoarkkitehtuuri yleisesti

Ohjelmistoarkkitehtuuria voidaan kuvailla mm. seuraavasti (Koskimies 2005, 27):

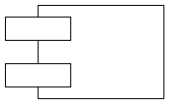
- Ohjelmistoarkkitehtuuri on osa ohjelmistosuunnittelua, kaikki suunnitteluasiat eivät kuitenkaan ole arkkitehtuuria.
- Komponenttien väliset suhteet kuuluvat arkkitehtuuriin, mutta niiden sisäiset asiat eivät.
- Arkkitehtuuriin sisältyvät myös ratkaisujen perustelut.
- Ohjelmistoarkkitehtuuri voidaan nähdä rakennettavan järjestelmän perustana joka antaa rajat yksityiskohtaiseen suunnitteluun ja toteutukseen.
- Arkkitehtuuriperustainen ohjelmistokehitys korostaa arkkitehtuurin roolia kehitysprosessia ohjaavana suunnitteluartefaktina.
- Ohjelmistoarkkitehtuuri voidaan kuvata eri asioita korostavista näkökulmista käsin ja eri abstraktiotasoilla.

Ohjelmiston arkkitehtuuri on tärkein ohjelmistoa luonnehtiva informaatio. Tästä syystä ohjelmiston eri sidosryhmien välinen kommunikointi koskee usein arkkitehtuuriin liittyviä kysymyksiä. Jotta kaikilla olisi sama käsitys ohjelmiston arkkitehtuurista, tulisi sen olla mahdollisimman kattava ja yksiselitteinen kuvaus. Arkkitehtuurilla on tärkeä merkitys järkevän kommunikaation mahdollistavana ohjelmistoartefaktina, joka antaa paitsi keskeiset ohjelmistoratkaisut myös käsitteistön ja sanaston, jolla järjestelmästä voidaan puhua. Nykyisin teollisuudessakin ymmärretään arkkitehtuurikuvausten merkitys, ja siitä on tullut keskeinen dokumentti ohjelmistokehitysprosesseissa. (Koskimies 2005, 29)

Mallinnustyökaluja on olemassa useitakin mutta GPL-lisenssillä oleva Dia (http://live.gnome.org/Dia) on ilmainen ja monipuolinen ohjelma. Toinen hyvä vaihtoehto on kaupallinen MS Visio.

3.2.Komponentin määritelmä

Ohjelmistokomponentti (ks. kuva 4) on itsenäinen ohjelmistoyksikkö, joka tarjoaa palvelujaan hyvin määriteltyjen rajapintojen kautta parametrien avulla. Ohjelmat yleensä koostuvat useammista komponenteista (moduuleista) jolloin puhutaan modulaarisesta ohjelmoinnista.



Kuva 4. Ohjelmistokomponentin piirrosmerkki.

3.3.Komponentit ohjelmistokehityksen yksikköinä

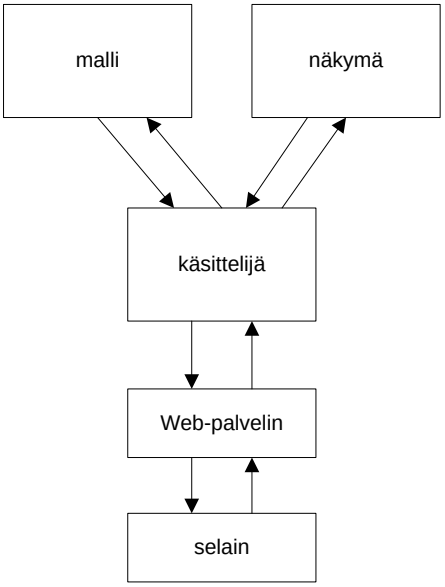
Koska ohjelmiston arkkitehtuuri vaikuttaa keskeisesti ohjelmistokehitysprosessiin ja kehittävään organisaatioon, myös komponenteille arkkitehtuurin perusyksikköinä tulee tärkeä rooli. Komponentti annetaan tyypillisesti tietyn henkilön kehitettäväksi ja vastuulle jolloin se on myös työnjaon perusyksikkö. Komponentin kehittämisellä on tällöin oma aikataulutuksensa ja resursointinsa. Organisaatioon kuuluva ryhmä kehittää toisiinsa liittyviä komponentteja, esim. jonkin alijärjestelmän osia. (Koskimies 2005, 29)

Ohjelmiston suunnittelun perustana, jaossa komponentteihin, on tavoitteena minimoida eri komponenttien väliset kytkennät (coupling) ja maksimoida niiden sisäinen yhteneväisyys (cohesion). Minimoimalla komponenttien väliset kytkennät voidaan komponentteja muuttaa mahdollisimman paljon toisistaan riippumatta.

3.4.Komponentit ja rajapinnat

Yksi tärkeimmistä ohjelmistotekniikan periaatteista on pyrkiä erottamaan toisistaan se, mitä halutaan saada aikaan, ja miten tämä tapahtuu. Ohjelmistokomponenttien tapauksessa tämä tarkoittaa sitä, että palvelun toteutus on erotettava palvelusta abstraktiona. Palvelun käyttäjän ei tulisi riippua tietystä palvelun tuottajasta, komponentista, vaan ainoastaan palvelusta itsestään abstraktina käsitteenä. Tämä abstrakti käsite esitetään rajapintana, jonka yksi tai useampi konkreettinen komponentti toteuttaa. (Koskimies 2005, 29)

Rajapinnat ovat keskeinen osa ohjelmistoarkkitehtuuria. Ne määräävät tavat, joilla komponentit kommunikoivat keskenään. Myös monet arkkitehtuurityylit ja suunnittelumallit kuten esim. kuvan 5 malli-näkymä-käsittelijä (Model-View-Controller) perustuvat rajapintojen käyttöön. Rajapintojen huolellinen suunnittelu on edellytyksenä paitsi kehitystyön järkevälle jakamiselle myös monille tärkeille ohjelmiston laatuominaisuuksille, kuten testattavuudelle, ylläpidettävyydelle ja joustavuudelle. Rajapintojen suunnittelu alkaa kun keskeisimmät arkkitehtuuriratkaisut on tehty.



Kuva 5. Malli-näkymä-käsittelijä suunnittelumalli (BetterExplained).

3.5.Lopuksi

Ohessa on muutamia hyödyllisiä linkkejä liittyen luvun aiheeseen:

http://live.gnome.org/Dia

http://office.microsoft.com/fi-fi/visio/

Luku 4: Tyyliopas

Ohjelmoinnin tyylioppaalla pyritään yhtenäiseen ja selkeään tapaan kirjoittaa koodia jolloin sen luettavuus ja ymmärrettävyys paranevat. Kirjoittamalla koodi lohkoihin yhtenäisellä tavalla, esittelemällä ja nimeämällä muuttujat loogisesti sekä kommentoimalla koodi parannetaan ohjelman luettavuutta jolloin myös koodin ylläpidettävyys helpottuu. Samoin on koodin siirrettävyys ja yhteneväisyyssyistä suositeltavaa kirjoittaa ANSI C standardin mukaista koodia (ISO/IEC 2007) ja kääntää ohjelma parametrilla -std=c99. Yleensä ohjelmistoalan yrityksillä on omat tyylioppaat kuten dokumenttipohjatkin.

Liitteenä oleva tyyliopas (ks. liite 1) on tarkoitettu orientoimaan opiskelijoita käyttämään yhtenäisiä rakenteita ja samaa tyyliä kirjoittaessaan ohjelmia. Tämä tyyliopas rakentuu pääosin GNU:n tyylioppaan pohjalle (GNU Coding Standards).

4.1.Lopuksi

Ohessa on muutamia hyödyllisiä linkkejä liittyen luvun aiheeseen:

Tyylioppaita:

http://www.gnu.org/prep/standards/html_node/index.html

http://en.wikipedia.org/wiki/Programming_style

C-kielen standardi:

http://www.open-std.org/JTC1/SC22/WG14/www/docs/n1256.pdf

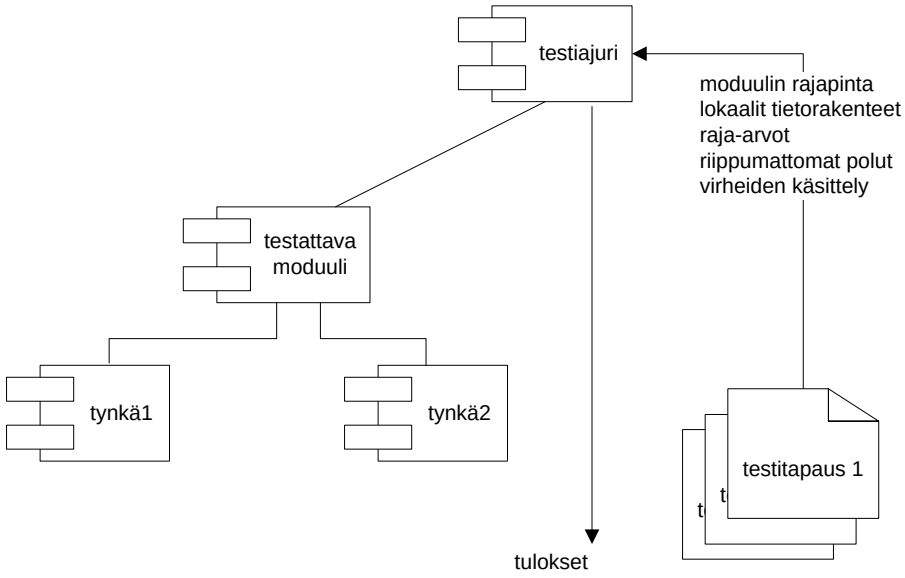
Luku 5: Moduulitestaus

Ohjelman arkkitehtuurin koostuessa useista ohjelmistokomponenteista tämä rakenne ohjaa testauksen suunnittelua ja toteutusta. Tässä luvussa käsitellään ohjelmistotestauksen perusmetodeja sekä virheiden etsimiseen olevia työkaluja.

Moduulitestauksessa (yksikkötestauksessa) ohjelmistokomponentteja testataan ennen niiden liittämistä yhteen yhdeksi kokonaisuudeksi integrointivaiheessa. Moduulitestauksen osa-alueita ovat mm. yksikön käyttämät tietorakenteet, suorituspolut, silmukkarakenteet, virhetilanteiden käsittelyt ja raja-arvot. Moduulitestauksessa on kaksi mallia top-down (ylhäältä alas) ja bottom-up (alhaalta-ylös) metodit.

Top-down metodissa testataan pääohjelma jonka jälkeen aliohjelmat. Bottom-up metodissa testataan ensin aliohjelmat ja siirrytään ylös pääohjelmaan päin kun kaikki vastaavan tason aliohjelmat on testattu.

Moduulitestauksessa käytetään työkaluina testiajureita ja tynkiä (ks. kuva 6).



Kuva 6. Moduulitestauksen peruskomponentit (Pressman 2005, 397).

(liite 1 jatkoa)

5.1.Testiajuri

Testiajuri (test driver) on ohjelmistomoduuli, jolla kutsutaan testattavaa moduulia ja usein välitetään testisyötteet, kontrolloidaan ja monitoroidaan suoritusta sekä raportoidaan testitulokset (IEEE 1990).

Esimerkissä 3 pääohjelma kuvaa testiajuria, jossa muuttujalle `luku` asetetaan arvo, kutsutaan aliohjelmaa ja lopuksi tulostetaan muutettu arvo. Funktio `lueLuku()` on siis testattava osuus, osa joka menee tuotantoon. Pääohjelma on osa testwarea (testikoodi, -tapaukset, testaussuunnitelmat, -raportit) on vain testaustarkoitukseen tehty koodinpätkä.

Esimerkki 3.

```
#include <stdio.h>

int lueLuku(int luku);

int main(void) {
    int luku;
    printf("Anna luku: ");
    scanf("%d",&luku);
    luku = lueLuku(luku);
    printf("Luku on nyt: %d\n",luku);
    return 0;
}

int lueLuku(int luku) {
    int apu = luku;
    printf("Annoit luvun: %d\n", luku);
    apu = apu*2;
    return apu;
}
```


(jatkuu)

5.2.Tynkä

Tynkä (stub) on ohjelmistomoduulin testausta varten tehty toteutus, jota käytetään kehitettäessä tai testattaessa moduulia joka kutsuu sitä tai on muuten siitä riippuvainen (IEEE 1990).

Esimerkissä 4 funktio `laskeKertoma()` on korvattu tyngällä joka vain palauttaa syötteenään saamansa parametrin. Funktio `tynka()` on osa testwarea. Pääohjelma on testattava, tuotantoon menevä osa. Tynkämoduulin palauttama arvo pitää miettiä tapauskohtaisesti, esim. vakio tai x kertaa sisään mennyt arvo. Oleellista on, ettei tynkämoduulissa ole muita toimintoja esim. printf(" ").

Esimerkki 4.

```
#include <stdio.h>
#define MAX 3650000

/* int laskeKertoma(int n); */
int tynka(int n);

int main(void) {
    int n, kertoma;
    printf("Anna luku: ");
    scanf("%d", &n);
    /*  kertoma = laskeKertoma(n); */
    kertoma = tynka(n);
    printf("Antamasi luvun kertoma: %d\n", n);
    return 0;
}

/* int laskeKertoma(int n) */
int tynka(int n) {
    return n;
}
```

5.3.Testikattavuus

Kattavuusmitoilla pyritään todentamaan testattavan ohjelman riittävä läpikäynti eli suorittaminen.

Lausekattavuus (statement coverage) on laskennallinen arvo siitä kuinka suuri osa ohjelmakoodin lauseista on suoritettu. Gradyn mittaukset osoittavat, että käytännössä lausekattavuus jää yleensä selvästi alle 90 prosentin (Grady 1993).

Päätöskattavuudessa (decision coverage) ehtorakenteen päätöksen tulee saada vähintään kerran kaikki arvonsa.

Ehtokattavuudessa (condition coverage) valintilannepäätösten on saatava kaikki mahdolliset arvonsa.

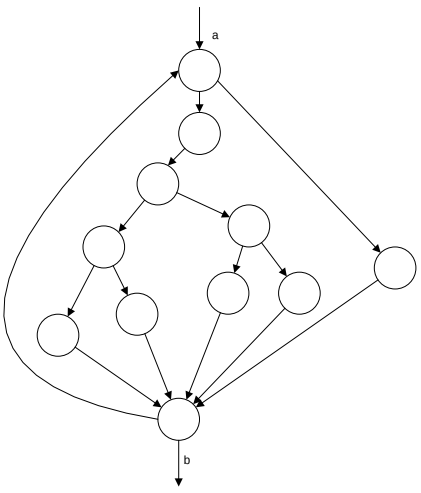
Polkutestauksessa (path coverage) mitataan kaikkien haarojen yhdistelmät ja kaikki suoritettavat polut.

Kattavuusmitat ilmoitetaan aina prosentteina,

esim.

$$lausekattavuus = \frac{läpikäydyt\ lauseet}{kaikki\ lauseet} * 100\%$$

Moduulitestauksessa testitapauksissa on tärkeätä ottaa huomioon erityisesti ehtolauseiden kriittiset raja-arvot, funktiorajapinnat ja testauspolut. Tarkasteltaessa alla olevaa esitystä erään ohjelmamoduulin kontrollirakenteesta (ks. kuva 7) havaitaan, että suorituspolkuja saattaa olla varsin monia jolloin suoritusaikakin voi kasvaa yllättävän suureksi. (Myers et al. 2004)

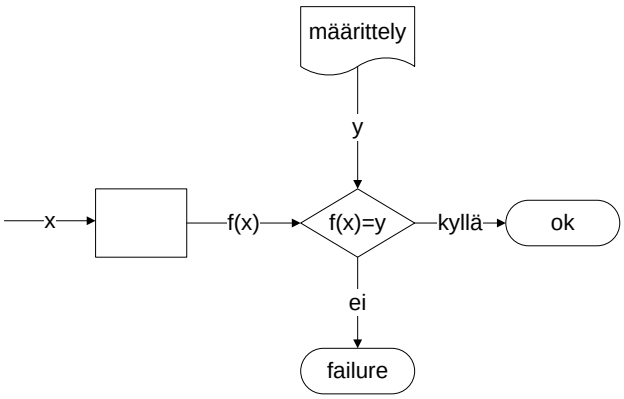


Kuva 7. Suorituspolut (Myers et al. 2004, 12).

Kuva 7 esittää do- silmukkaa jonka sisällä on viisi vaihtoehtoista polkua jotka suoritetaan 20 kertaa. Vaihtoehtoisia polkuja a:n ja b:n välillä on täten $5^{20} + 5^{19} + \ldots + 5^{1} = 10^{14}$. (Myers et al. 2004, 12)

5.4.Lasilaatikkotestaus

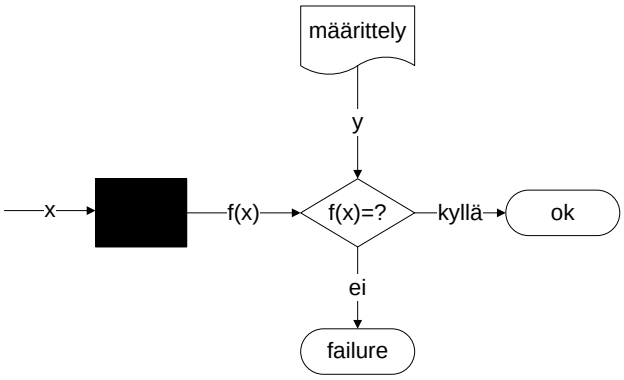
Lasilaatikko (white-box) testausmenetelmällä (ks. kuva 8) testataan yksittäisten algoritmien, ohjelmamoduulien toimintaa niin suunnittelu- kuin toteutusvaiheissa, tarkoituksena löytää virheitä koodin harvemmin suoritettavista osista. Algoritmeista rakennetaan kaavio, josta testataan sen toimintaa kaikilla suorituspoluilla. Lasilaatikkotestaus perustuu ohjelman toteutuksen eli sisäisen rakenteen tuntemiseen.



Kuva 8. Lasilaatikkotestaus mukailtu (Paakki 2000, 22).

5.5.Mustalaatikkotestaus

Mustalaatikko (black-box) testausmenetelmällä (ks. kuva 9) ohjelman toimintaa testataan aineistolla, josta saatavat tulokset ovat jo tiedossa, siis testitapaukset valitaan määrittelyjen avulla tuntematta itse ohjelman toteutusta. On siis tärkeää, että oikeat tulokset ovat etukäteen tiedossa, jotta kaikki virheet voidaan jäljittää itse järjestelmään.



Kuva 9. Mustalaatikkotestaus (Paakki 2000, 22).

(jatkuu)

5.6. Virheiden etsintä

Virheenjäljittimet jaetaan staattisiin kuten **gcc**-kääntäjä tai **Lint** (staattinen koodin analysaattori) ja dynaamisiin kuten debuggerit. Kääntäjä ja Lint käytännössä havaitsevat kielen syntaksivirheet kun taas debuggereilla voidaan suorittaa ohjelmakoodia seuraten muuttujien ja muistin sisältöä.

Linux-ympäristössä ohjelmointivirheiden etsintään on perustyökaluja kuten **gdb**- ja **ddd**-debuggerit, jotka mahdollistavat ohjelman suorittamisen askeltamalla lause kerrallaan. Gdb-työkalu toimii merkkipohjaisesti, mutta toiminta on sujuvaa tottuneelle käyttäjälle. Ddd-degubberissa on graafinen käyttöliittymä, ja siksi monet mieluummin käyttävät sitä ajatellen sen olevan helppokäyttöisempi ja havainnollisempi kuin gdb.

Assert on apuohjelma joka ilmoittaa virheestä jos se saa arvokseen nollan. Alla olevassa esimerkissä 5 ohjelman suoritus päättyy kun muuttuja `varastosaldo` saa arvokseen nollan.

Esimerkki 5.

```
#include <stdio.h>
#include <assert.h>

int main(void) {
    int varastosaldo = 5;
    int tilattu = 4;
    varastosaldo -= tilattu;
    tilattu++;
    varastosaldo--;
    assert(varastosaldo > 0);
    printf("Varastossa: %d kpl\n",varastosaldo);
    return 0;
}
```

Esimerkissä 6 ohjelman suoritus päättyy kun muuttuja `numero` saa arvokseen nollan.

Esimerkki 6.

```
#include <stdio.h>
#include <assert.h>

void tulosta(int* numero) {
    assert(numero!=0);
    printf("%d\n",*numero);
}

int main () {
    int a=10,b=5,*c=0;
    int * d = &a;
    int * e = &b;
    int * f = c;
    tulosta(d);
    tulosta(e);
    tulosta(f);
    return 0;
}
```

`Valgrind` on työkalu jolla voidaan tarkastaa ohjelman muistialueen käyttöä. Valgrindiä käytettäessä ohjelman kääntäminen tehdään esim. seuraavaan tapaan (parametri W on Valgrindiä varten):

```
gcc -W -Wall -g –std=c99 -o ohjelma koodi.c
```

Valgrind käynnistetään vastaavasti `valgrind ./ohjelma`.

5.7.Lopuksi

Ohessa on muutamia hyödyllisiä linkkejä liittyen luvun aiheeseen:

http://www.network-theory.co.uk/valgrind/manual/valgrind-sample.pdf

http://valgrind.org/docs/manual/index.html

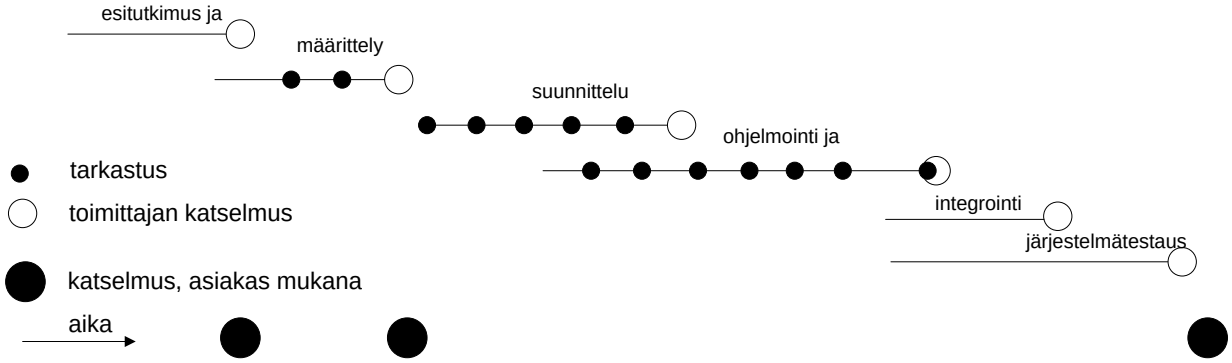
http://www.gnu.org/software/ddd/

http://www.gnu.org/software/gdb/documentation/

Luku 6: Kooditarkastus

Ohjelmiston kooditarkastus (Code Review) on tehokas keino löytää ja paikallistaa virheitä sekä puutteita koodista onhan se tekstiä ja sopii hyvin tarkastettavaksi. Tämä luku käsittelee tavallisimpia metodeja suorittaa kooditarkastus, sekä toimivien ohjelmamoduulien yhteen liittämistä eli integrointia.

Kooditarkastuksessa tarkistetaan että kaikki tarvittavat dokumentit sekä kriteerit on täytetty ja todetaan tarkastusvaiheen päättyminen kuvan 10 mukaisesti.



Kuva 10. Projektin katselmukset ja tarkastukset (Haikala 2004, 52).

Kooditarkastuksella ei voida korvata testausta mutta se vähentää eri testausvaiheissa löytyvien virheiden määrää. Tarkastuksella on testaukseen verrattuna Haikalan mukaan myös seuraavia etuja (Haikala 2004, 277):

- Kommenteissa ja vakiomerkkijonoissa olevat virheet löytyvät usein vain tarkastuksilla.
- Tyylivirheet eivät löydy testaamalla.
- Reaaliaikajärjestelmien rinnakkaisuuteen liittyvät epädeterministiset virheet löytyvät usein vain tarkastamalla.
- Turha tai epätarkoituksenmukainen koodi ei löydy testaamalla.
- Epähavainnolliset muuttujien nimet eivät löydy testaamalla.
- Erilaiset "aikapommit", kuten alustamattomat muuttujat, löytyvät usein vain tarkastamalla.

Koodin tarkastuksen hyödyistä on tutkimustuloksia esim. Bell-Northern Research:in kooditarkastusten käyttöönottoraportissa. Raportin mukaan vikojen löytyminen tarkastuksilla oli 2-4 kertaa nopeampaa kuin virheiden etsiminen ja korjaaminen testaamalla, koska tällöin virhe tulee samalla paikannettua ja aikaa säästyy. Ero koodin testauksen ja katselmoinnin välillä on vielä suurempi, jos kyseessä on asiakkaan raportoima vika. (Russell 1991)

Tarkastuksia pidetään yleisesti tehokkaimpana tunnettuna laadunvarmistuskeinona ohjelmistokehityksen tukitoiminnoissa. Alkuperäinen ohjelmistojen tarkastusmenetelmä kehitettiin IBM:n organisaatiossa 1970-luvulla, josta kirjallisuudessa puhutaan yleisesti Faganin tarkastuksena (Kollanus 2006). Toinen keskeinen lähde tarkastusmenetelmistä puhuttaessa on Gilbin ja Grahamin (Gilb 1993) kirjoittama kirja Software Inspection.

6.1.Tarkastus Faganin tapaan

Faganin alkuperäinen tarkastusmenetelmä (Fagan 1976) jakaantuu seuraaviin vaiheisiin:

- Aloituspalaverissa, jossa koko ryhmä on koossa, suunnittelija kuvailee kokonaisuuden.
- Valmistautuminen, jolloin osallistujat opiskelevat ja tutkivat dokumentaation.
- Tarkastuspalaveri jossa kaikki ovat jälleen koolla ”lukija” kuvaa ohjelman toteutuksen kattaen kaikki logiikan tasot ja haarat. Kaikki ohjelmalistaukset tulee olla paikalla.
- Virheiden korjauksessa suunnittelija ja ohjelmoija ratkaisevat kaikki virheet ja ongelmat jotka tulivat esiin tarkastuspalaverissa.
- Seurantavaihe, jolloin kaikkien virheiden tulisi olla ratkaistu sillä muuten aiheutuneet kustannukset nousevat huomattavasti.

6.2.Integrointitestaus

Komponenttien integrointitestaukseen on olemassa kuusi strategiaa; kriittinen moduulitestaus, kerrosvoileipä, säie, suuri pamaus, alhaalta ylös ja ylhäältä alas (Eickelmann 1996). Seuraavassa tarkastellaan vain viimeistä näistä em. metodeista sen tarjoamien kiistattomien etujen tähden.

Ylhäältä alas etenevän integroinnin etuja on mahdollisuus varmistaa kontrollirakenteiden toiminta jo testauksen alkuvaiheessa. Lisäksi testaus ja integrointi voidaan aloittaa hyvin aikaisessa vaiheessa. Komponentteja voidaan kehittää rinnakkain ja tarvittaessa voidaan lykätä alemman tason laiteriippuvien komponenttien valmistamista. Tämä on erityisen hyödyllistä, mikäli laiteympäristö on vasta kehitysvaiheessa tai sen muuttaminen ennen kehitystyön alkua on mahdollista. (Satukangas 2003, 10)

Käytännössä ylhäältä alas (top-down) -integroinnissa testaus aloitetaan kontrollihierarkian ylimmästä yksiköstä, jonka kutsumat yksiköt korvataan tyngillä ja yksikkö testataan,

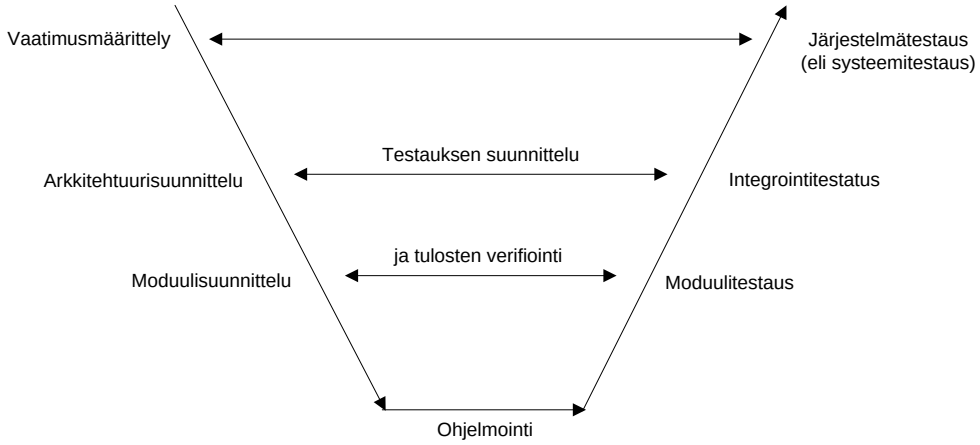
onnistuneen testauksen jälkeen tyngät korvataan alkuperäisillä ohjelmayksiköillä. Vastaavasti edetään askel askeleelta ohjelman arkkitehtuurissa alaspäin ja kukin syntynyt osakokonaisuus testataan. Tämän lähestymistavan käyttöä hankaloittaa se, ettei yleensä ole olemassa vain yhtä yksittäistä ylhäältä alas johtavaa polkua, jonka pohjalta integroinnin voisi suorittaa. Tyngät saattavat myös sisältää monimutkaisia rutiineja, minkä johdosta ne voivat olla hankalia toteuttaa. Myös tyngät tulisi testata niiden oikean toiminnallisuuden varmistamiseksi. (Beizer 1990, 115)

Integrointitestauksessa yksikkötestauksen läpäisseet ohjelmistokomponentit kootaan systemaattisesti alijärjestelmiksi ja näin taas kokonaisiksi järjestelmiksi. Integrointitestaus alkaa silloin, kun kaksi tai useampia yksiköitä on valmiina integroitaviksi ja päättyy, kun järjestelmä on valmis järjestelmätestaukseen. Integrointitestauksessa varmistetaan komponenttirajapintojen yhteensopivuus sekä se, että tietojen ja kontrollin välitys rajapintojen yli toimii oikein (Eickelmann 1996). Löydetyt virheet poikkeavat yksikkötestausvaiheessa löydetyistä ja liittyvät testattavien yksiköiden väliseen yhteistoimintaan ja riippuvuuksiin (Satukangas 2003, 10).

Luku 7: Testaus laajemmin

Ohjelmiston testaus on monivaiheinen prosessi alkaen suunnittelusta päättyen järjestelmätestaukseen. Tämä kappale käsittelee mm. testauksen dokumentointia ja raportointia, järjestelmä- ja regressiotestausta. Lopuksi on lyhyt yhteenveto testaustyökaluista.

Hyvin yleinen testaustasoja havainnollistava esitys on nk. V-malli (ks. kuva 11) jossa esitetään ohjelmistokehityksen vaiheet toisiinsa sidoksissa olevana polkuna.



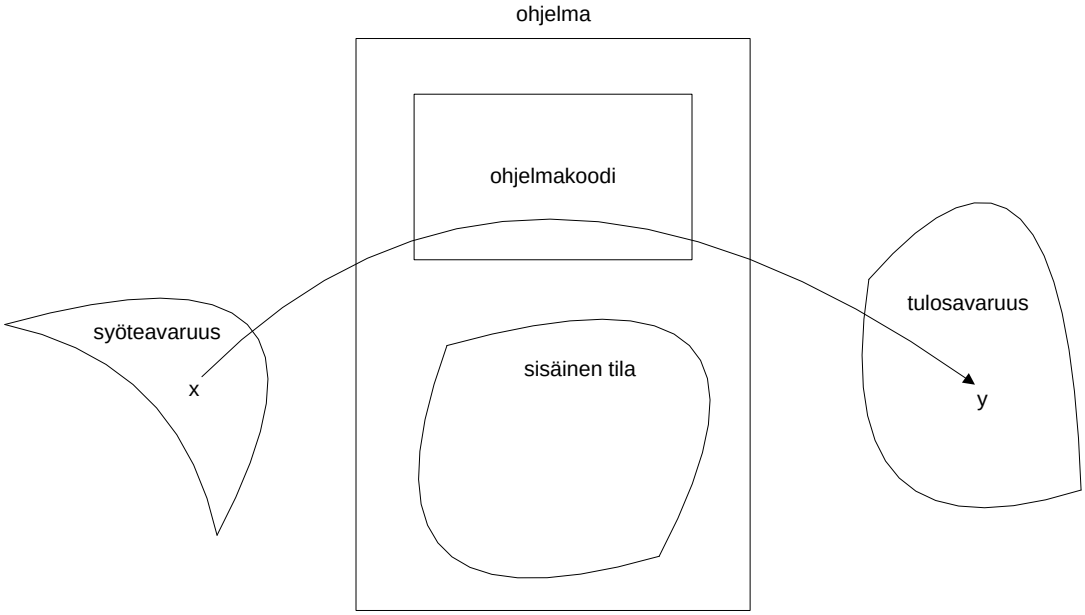
Kuva 11. Testauksen V-malli (Haikala 2004, 289).

7.1.Testitapaukset

Tämä alaluku perustuu Juha Holopaisen Diplomityöhön; Regressiotestaus ja testien valintatekniikat. (Holopainen 2005, 8)

Ohjelman testaamiseksi luodaan testitapauksia, joiden tiedot kirjataan määrittelydokumentteihin. Testitapaus sisältää sarjan ohjelmalle annettavia syötteitä, ohjelman odotetun tuloksen annetuilla syötteillä, esiehdot testin suorittamiseksi sekä jälkiehdot ohjelman tilan oikeellisuuden varmentamiseksi testin jälkeen (ks. kuva 12). Esiehdoissa kerrotaan, missä tilassa ohjelman ja ympäristön pitää olla testitapauksen suorittamiseksi. Jälkiehdot kertovat, missä tilassa ohjelman pitää olla suorituksen jälkeen. Näiden lisäksi testitapauksen määrittelydokumentissa selitetään lyhyesti testitapauksen testaamat ohjelman toiminnot ja osat sekä kerrotaan viitteet muihin asiaan liittyviin dokumentteihin. Jos testitapaus on riippuvainen toisista testitapauksista, ilmoitetaan mitkä testitapaukset pitää ajaa ennen kyseistä testitapausta tai muulla tavoin selvennetään suoritusjärjestys.

(jatkuu)



Kuva 12. Ohjelmiston testaus (Haikala 2004, 284).

Määritellyistä testitapauksista kootaan testijoukkoja ohjelman tai sen osien testaamiseksi. Testijoukko on joukko testitapauksia, jotka on kerätty yhteen tiettyä spesifistä tarkoitusta varten kuten esim. ohjelman eri osien, komponenttien, testaamiseen.

Testausprosessissa ohjelmaa testataan jokaisella ohjelmalle luodun testijoukon sisältämällä testitapauksella. Ensin ohjelman tila ja suoritusympäristö alustetaan testitapauksessa kerrottujen esiehtojen mukaiseksi. Seuraavaksi testattavalle ohjelmalle annetaan testitapauksessa määritellyt syötteet. Lopuksi ohjelman antamaa tulosta verrataan testitapauksessa määriteltyyn odotettuun tulokseen ja tarkastetaan, että jälkiehdot ovat voimassa. Mikäli saatu tulos eroaa odotetusta, on ohjelmassa virhe, olettaen että testitapaus on laadittu huolellisesti.

(jatkuu)

7.2.Testauksen dokumentointi

Testausdokumentaatiota voi syntyä runsaasti mm. järjestelmätestaus-suunnitelma, integrointitestaussuunnitelma jokaisesta integrointitestistä, moduulitestaussuunnitelma jokaisesta moduulitestistä ja vastaavasti raportti jokaisesta suoritetusta testistä. Ohjeessa ISO 9000-3 testaussuunnitelma ja -raportit suositellaan tehtäviksi sekä järjestelmätestauksesta että integrointitestauksesta. Moduulitestaustasolla testaussuunnitelman korvaa laatukäsikirjasta löytyvä ohjeistus. Ohjeistus sisältää suuntaviivat moduulitestauksesta ja halutun testikattavuuden määritelmän (Haikala 2004, 299). Pienehköissä projekteissa riittää yleensä yksi testaussuunnitelma, joka kattaa moduuli- ja integrointitestauksen.

Testaussuunnitelma sisältää tiedot mm. suoritettavista testeistä, testausjärjestelyistä ja odotetuista lopputuloksista. Testauksen lopettamiskriteerit, jotka tulee määritellä erikseen, täyttyvät pienemmissä projekteissa esim. kun testaukselle varattu aika päättyy, kaikki testitapaukset on ajettu ja analysoitu, viat on korjattu jonka jälkeen on suoritettu integrointitestaus.

7.3.Testauksen hallinta

Järjestelmällisen testauksen perusedellytyksiä Haikalaa mukaillen ovat:

- suunnittelu
- testiympäristön luonti
- testien suorittaminen
- tulosten arviointi
- dokumentointi.

Testaussuunnitelman tulisi käsittää ainakin seuraavat osat:

- testauksen tavoitteiden määrittely
- testausympäristö
- testauksen organisointi, integrointisuunnitelma ja raportointi
- testattavat ja ei-testattavat toiminnot
- toimintojen testitapaukset ja hyväksymiskriteerit
- ei-toiminnallisten ominaisuuksien testaus.

7.4.Testauksen raportointi

Testauksen raportointi voidaan tehdä esim. seuraavan dokumenttipohjan (ks. kuva 13) mukaisesti (STR template 2011).

1 Testin nimeäminen, paikka, aikataulu ja osallistujat
  1.1 Testattava ohjelmisto (nimi, versio)
  1.2 Testausdokumentit (nimet, versiot)
  1.3 Testiympäristö
  1.4 Kunkin testin alkamis- ja päättymisajat
  1.5 Testausryhmän jäsenet
2 Testausympäristö
  2.1 Ohjelmisto- ja laitteistokonfiguraatiot
3 Testaustulokset
  3.1 Testien tunnistetiedot
  3.2 Testitapauksien tulokset
    3.2.2 Testitapauksen tunniste
    3.2.3 Tulos OK / NOK
    3.2.4 Jos NOK, yksityiskohtainen kuvaus tuloksista ja ongelmista
4 Taulukko kokonaisvirhemääristä, jakautumisesta ja tyypeistä
5 Erityiset tapahtumat ja testaajien ehdotukset
  5.1 Erityiset tapahtumat ja odottamattomat testausvasteet
  5.2 Testauksen aikana esiintyneet ongelmat
  5.4 Muutosehdotukset testausmenetelmiin ja testiaineistoihin

Kuva 13. Dokumenttipohja testiraportiksi (STR template 2011).

Eri testaustasoilla löytyneet virheet tulisi raportoida ja analysoida. Kirjattavia tietoja ovat mm. virheen kuvaus, millainen virhe (esim. toiminnallinen, suorituskyky, rasitus), miten vakavasta virheestä on kysymys, milloin se löydettiin, miten se olisi voitu löytää aikaisemmin, milloin virhe oli tehty ja miten se olisi voitu estää (ks. kuva 14). Asiakkaalta tulevia virheilmoituksia varten käytetään yleensä erityistä lomaketta. Järjestelmätestauksessa voidaan käyttää samaa lomaketta tai erityistä virhepäiväkirjaa, johon em. tiedot kirjataan. Ilman raportteja on mahdotonta saada selville testaukseen käytettyä aikaa ja laatujärjestelmän kehityskohteita. Käytännössä virheiden dokumentointi raporttien muotoon jää usein järjestelmätestausraporttiin sekä asiakasreklamaatioiden seurauksena tehtyjen virheiden ja reklamaatioiden käsittelyn kirjaamiseen (ISO 9001 -vaatimus). (Haikala 2004, 300)

Testitapaukset							
Testi ID	Testin tarkoitus	Testin kuvaus	Suoritettu	Ok/Fail	Vakavuus	Tyyppi	pvm.

Kuva 14. Esimerkki testauspöytäkirjasta.

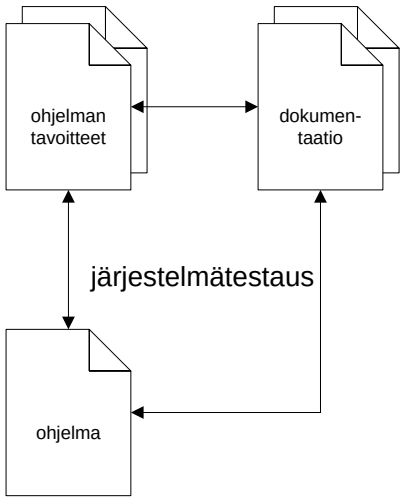
Suositeltava yksinkertainen tekniikka on testauspäiväkirjan (ks. kuva 15) käyttö johon kirjataan tiedot jokaisesta löytyneestä virheestä.

Löydetyt virheet						
Virhe ID	Virheen kuvaus	Missä vaiheessa virhe on tehty	Milloin löydetty	Miten korjattiin	Korjaukseen käytetty aika	pvm.

Kuva 15. Esimerkki testauspäiväkirjasta.

7.5.Järjestelmätestaus

Järjestelmätestauksessa kuva 16 mukaan (system testing) tarkastellaan koko ohjelmistoa suhteessa määrittely- ja asiakasdokumentaatioon.



Kuva 16. Järjestelmätestaus (Myers et al 2004, 131).

Järjestelmätestaukseen sisältyy esimerkiksi: toiminnallisuus, suorituskyky, käyttöliittymä, käyttäjädokumentit, kuormitustestaus, turvallisuus, liitettävyys, toipumiskyky virhetilanteista sekä asennettavuus.

7.6.Regressiotestaus

Ohjelman regressiotestaus on uudelleentestausta, jolla varmistetaan tehtyjen muutosten oikeellisuus ja se ettei ohjelman toiminnallisuuteen ole tullut muutoksia.

Ohjelmaan tehdyn muutoksen laadusta riippuen voidaan ohjelman regressiotestauksessa käyttää uudelleen vaihteleva määrä vanhoja, edellisellä testauskierroksella ohjelman onnistuneesti läpäisseitä testitapauksia. Näiden lisäksi joudutaan joskus tekemään uusia regressiotestitapauksia vanhan toiminnallisuuden oikeellisuuden tarkistamiseksi. Jos ohjelmaan tehty muutos on toteutettu pyrkimyksenä korjata ohjelmasta löydetty virhe, regressiotestataan muutettua ohjelmaa virheen löytäneillä testitapauksilla. Jos ohjelman vanhasta, aiemmin testatusta toiminnallisuudesta löytyy ohjelmaan tehdyn muutoksen jälkeisessä regressiotestauksessa virhe, sanotaan että ohjelma on regressoitunut eli taantunut. (Holopainen 2005, 9)

7.7.Regressiotestauksen automatisointi

Regressiotestauksen tulisi olla mahdollisimman pitkälle automatisoitua sillä testaus suoritetaan useamman kerran ohjelman elinkaaren aikana. Testauksen automatisointiin käytettäviä testaustyökaluja ovat mm. testipetigeneraattorit, testitapausgeneraattorit, vertailijat ja testikattavuustyökalut. Testipetigeneraattori generoi testattavalle ohjelmamoduulille testipetin, jolle voidaan kuvata testikuvauskielellä ajettava testi. Kielellä voidaan myös kuvata halutut testitulokset, jolloin testin tulosten tarkastelu on automatisoitavissa. Vertailijaohjelmilla voidaan verrata ohjelman tulosteita aikaisempien testauskertojen tulosteisiin. (Haikala 2004, 297)

7.8.Regressiotestaus käytännössä

Onoman (Onoma et al. 1998) esittämän tutkimuksen mukaan ohjelmistojen käytännön regressiotestauksessa suurin osa yrityksistä noudatti seuraavia menetelmiä regressiotestauksessa:

- Muutospyyntö tehdään kun ohjelmasta on löytynyt virhe tai ohjelman toiminnalliseen tai tekniseen määrittelyyn on tehty muutos.
- Tehdään muutos joko lähdekoodiin tai myös määrittely- ja suunnitteludokumentteihin.
- Valitaan regressiotestaukseen mukaan otettavat testitapaukset uudelleen, tavoitteena valita oikeat sopivat testit.
- Valittu testijoukko ajetaan, yleensä automatisoituna testien suuren määrän tähden. Testien suoritushistoria voidaan myös tallentaa myöhempää käyttöä varten.
- Virheet löydetään vertaamalla saatuja testituloksia odotettuihin tuloksiin. Mikäli tulokset eroavat toisistaan selvitetään onko virhe koodissa vai testitapauksessa. Testitapaukset pitää tarvittaessa hyväksyä uudestaan vielä tässä vaiheessa.
- Virheen paikannus voi olla vaikeaa johtuen monista ohjelmamoduuleista ja -versioista. Jos lähdekoodin epäillään olevan virheellinen, on välttämätöntä että ohjelmoijan paikantaa virheen alkusyyn.
- Kun virheen alkusyy on paikannettu, täytyy se käsitellä tekemällä ohjelmalle uusi muutospyyntö virheen korjaamiseksi tai poistamalla regressiovirheen syntymiseen johtanut muutos.

(jatkuu)

7.9.Testauksen työkaluja

Ohjelmistotestaukseen kuuluvat työkalut voidaan jakaa useampaan ryhmään niiden käyttötarkoituksen mukaan. Kiinnostavimmat ovat toimintojen ja suorituskyvyn testaukseen tarkoitetut ohjelmat joiden lisäksi löytyy testihallintaohjelmia. Kaupallisia testaustyökaluja löytyy useita myös C/C++ -ympäristöön;

- Testwell++ (http://www.testwell.fi/availabi.html)
- TestCocoon (http://www.testcocoon.org/)
- BullseyeCoverage (http://www.bullseye.com/download.html).

Ei-kaupallisilla lisensseillä C-testaustyökaluja on vaikeampi löytää mutta ainakin seuraavat pari löytyy;

- xCover (http://www.xcover.org/documentation/0.2.1/)
- Trucov (http://code.google.com/p/trucov/).

Testaustyökaluiksi voidaan laskea myös testauksessa yleisesti käytettävät skriptikielet AWK, Perl ja Python. Aina on myös mahdollista kirjoittaa itse testaustyökaluja esim. strcmp() funktiota hyväksikäyttäen. Esimerkiksi Viopen testit perustuvat pitkälti saadun ja odotetun vasteen vertailuun merkkijonoina.

7.10.Lopuksi

Ei-kaupallisilla lisensseillä löytyviä testaustyökaluja löytyy helpommin muille kielille kuin perinteinen C. JUnit-kehys on Eclipse kehitysalustaan integroituva testaustyökalu, jonka avulla voidaan kehittää projektin yhteydessä toimivia testiohjelmia Java-ympäristössä. Toinen työkalu on GCT (http://www.exampler.com/testing-com/tools.html).

Ohessa on muutamia hyödyllisiä linkkejä liittyen luvun aiheeseen:

Testauskirjan www-sivut:

http://www.testauskirja.com/

Testausohjelmia:

http://sourceforge.net/

http://www.opensourcetesting.org/

Luku 8: Yhteenveto

Tämä viimeinen kappale sisältää irrallisia aiheita liittyen C-ohjelmoinnin perustyökaluihin kuten esim. scriptit ja makrot.

8.1.Esikääntäjä

Esikääntäjä (C precompiler) on ohjelma, jonka läpi lähdekoodi viedään kääntäjälle. Esikääntäjää ohjataan lähdekoodissa #-merkillä alkavilla direktiiveillä kuten #include, #define, #if, #elif, #else, #endif. Esikääntäjän toiminta tapahtuu normaalisti huomaamattomasti käännöksen yhteydessä, mutta halutessaan sen voi myös pakottaa tapahtumaan erikseen. GCC-paketissa oleva esikääntäjä on nimeltään CPP. Esikääntäjä muun muassa yhdistelee koodirivejä, poistaa koodista kommentit ja korvaa sanoja toisilla. Sen avulla voi kirjoittaa myös pieniä funktioita muistuttavia makroja. Esikäännös voidaan tehdä myös gcc-kääntäjän parametrilla –E.

8.2.Makro

Parametriton makro on rinnastettavissa symboliseen vakioon, ts. makron nimi korvataan esikäännöksessä korvaustekstillä.

Makron määrittelyssä on kolme osaa, jotka ovat: esikääntäjän ohjaus: *#define*, makron nimi ja korvausteksti.

Makrot tuottavat välitöntä koodia, jossa ei tarvita hyppyä funktioon ja palaamista takaisin kutsukohtaan, jolloin myös prosessoriaikaa säästyy. Makrot eivät välitä parametrien tyypeistä. Esimerkiksi alla olevaa SUURIN-makroa (ks. esimerkki 7) voidaan käyttää sekä int- että float-tyyppisille parametreille. Sama ei päde funktiolle SUURIN().

Esimerkki 7.

```
#define SUURIN(a,b) ((a) > (b) ? (a) : (b))

x = SUURIN (z, y);
```

esikääntäjä tuottaa ohjelmaan rivin

```
x = ((z) > (y) ? (z) : (y));
```

8.3.Scriptit

Termi komentosarjakieli eli skripti on yleistetty tarkoittamaan kaikkia kieliä, joilla on mahdollista suorittaa käyttöjärjestelmällä erityyppisiä toimintoja ilman ohjelman kääntämistä. Tällaisia uusia komentosarjakieliä ovat mm. AWK, Perl, PHP ja Python.

Unix- ympäristössä esim. Bourne-shell ympäristössä on syytä opetella käyttämään komentojonotiedostoja, sillä niitä voi hyödyntää varsin monella tapaa (ks. esimerkki 8). Luotaessa komentotiedosto tulee sille antaa suoritusoikeudet jotta se voidaan ajaa, `chmod 544 script_esimerkki.sh`. Loppuliitteekseen bash-komentotiedosto vaatii .sh loppuliitteen.

Esimerkki 8.

```
# script_esimerkki.sh
# listaa hakemiston tiedostokoot kasvavassa järjestyksessä
#!/bin/bash

ls –l | awk  ‘{print $5}’ | sort –g
```


(jatkuu)

8.4.Funktio-osoitin

Funktio-osoittimella voidaan selventää koodin rakennetta mahdollistamalla funktion suoritus ajonaikaisilla arvoilla (ks. esimerkki 9).

Esimerkki 9.

```
#include<stdio.h>

void nuori(int);
void vanha(int);
void viesti(void (*)(int), int);
int main(void) {
    int ika;
    printf("Anna ikäsi vuosina? ");
    scanf("%d", &ika);
    if (ika > 30) {
        viesti(vanha, ika);
    }
    else {
        viesti(nuori, ika);
    }
    return 0;
}

void viesti(void (*fp)(int), int k) {
    fp(k);
}

void nuori(int n) {
    printf("Ai vasta %d v, oletpa nuori!\n", n);
}

void vanha(int m) {
    printf("Oho %d v, alat olla keski-iässä.\n", m);
}
```


(jatkuu)

8.5.Errno

Errno käyttää standardi kirjastokutsuja näyttääkseen C errno- arvot. Linuxin komentorivillä suoritettava komento `man errno` antaa listauksen errnon symbolisista arvoista. Alla olevan ohjelmaesimerkin mukaisesti on kuitenkin mahdollista muuttaa errno:n arvoa, kuten esimerkissä asettaa ernon arvoksi EIO, eli I/O- failure. Ernnon arvon voit asettaa joko kirjainlyhenteenä tai numeerisena-arvona (ks. esimerkki 10).

Esimerkki 10.

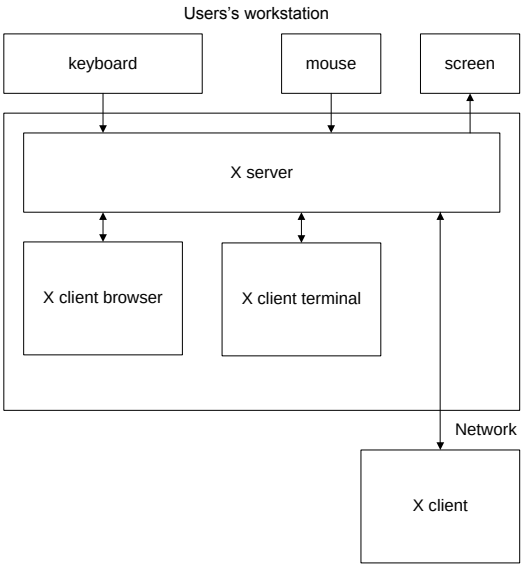
```
#include <stdio.h>
#include <string.h>
#include <errno.h>

int main(void)
{
    FILE *fp;
    if ((fp = fopen("foobar","r")))
    printf("File opened successfully\n");
    else {
        errno = EIO;
        printf("Report: %s\n", strerror(errno));
    }
    return 0;
}
```


(jatkuu)

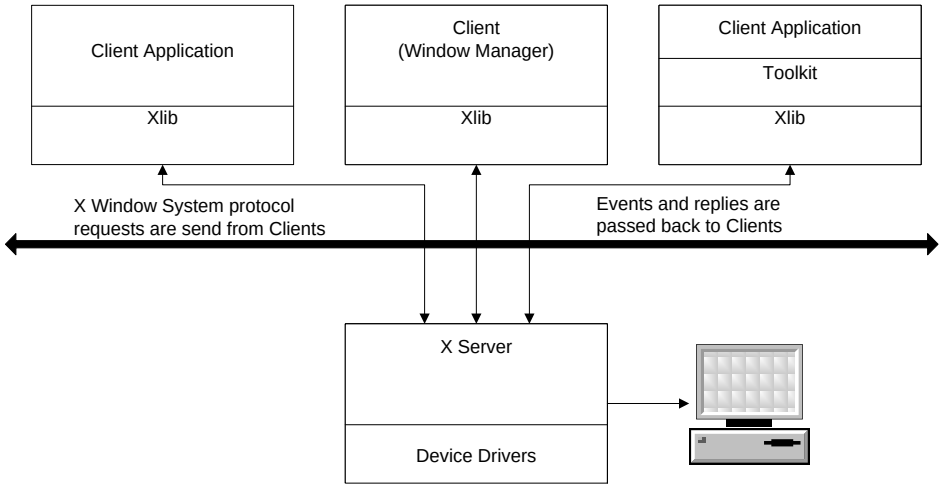
8.6.X11 ikkunointijärjestelmä

Syyskuussa 1987 Massachusetts Institute of Technology julkaisi X Window System ikkunointijärjestelmästä version 11, mikä oli yksi aikansa merkittävimmistä ohjelmistoteknologioista (kutsutaan yleisesti X11:sta). X11 on bittikarttanäytölle kehitetty graafinen asiakas-palvelin toteutuksella toimiva ikkunointijärjestelmä (ks. kuva 17) jonka päällä toimiva ikkunanhallintaohjelma (window manager) piirtää kehykset ikkunoille, mahdollistaen niiden siirtämisen ruudulla ja koon muuttamisen. X11 osaa piirtää suorakulmion muotoisia, päällekkäisiä, hierarkkisia ikkunoita, joihin voidaan piirtää tekstiä ja grafiikkaa. X11 ei siis määrittele käyttöliittymää vaan sen oheen käynnistetään yleensä työpöytäympäristö (esim. Gnome tai Kde) tai pelkkä ikkunanhallintaohjelma. (Tronche) Linuxin X-ikkunassa komennolla **`xcalc`** käynnistyy funktiolaskin ja komennolla **`xclock`** graafinen kello, kunhan tarvittavat kirjastot on asennettuna.



Kuva 17. X11 ikkkunointijärjestelmä (Wikipedia X Window System).

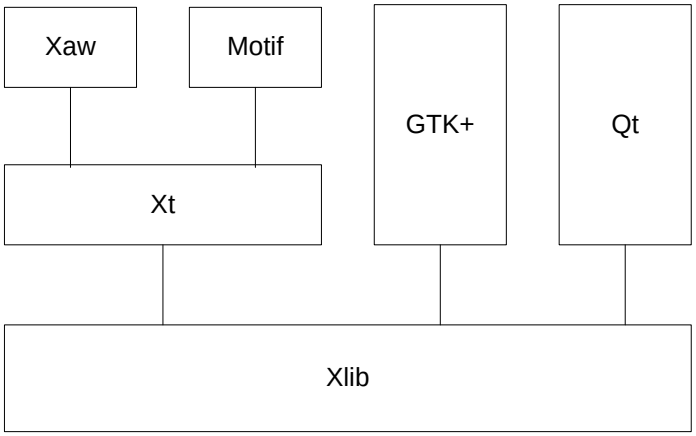
Kun X11 ajetaan yhdellä koneella asiakas ja palvelin käyttävät nopeampia kommunikointitapoja, etäkäytössä verkkoprotokollia (ks. kuva 18). Komennolla `ssh -l <username> @<IP address>` voidaan kytkeytyä etäistunnolla koneelle johon on pääsyoikeudet. Ohjelma voidaan suorittaa koneessa johon etäistunnolla on kytkeydytty jolloin näyttöpäätteenä toimii oman koneen näyttö.



Kuva 18. X-server/client (Tronche, 13).

8.7.Xlib

Motif on The Open Group:in (alkujaan Open Software Foundation) GUI- kehitysympäristö X window järjestelmään. Motif käyttää Xlib-ohjelmakirjastoa (ks. kuva 19) minkä avulla voidaan luoda valikkoja, painikkeita, liukusäätimiä, tekstikehyksiä yms. Xlib Programming Manual (Tronche) on kattava ohjesivusto Xlib-ohjelmoinnista kiinnostuneille.



Kuva 19. Xlib ja sitä höydyntäviä kirjastoja (Wikipedia Xlib).

Alla olevaa esimerkkiohjelmaa (esimerkki 11) kokeiltaessa tarvitaan xlib.h tiedosto joka on asennettavissa Ubuntuun komennolla:

```
> sudo apt-get install libx11-dev
```

Samoin käännöksessä tulee muistaa sisällyttää kirjasto mukaan:

```
> gcc –o esim1_x11 esim1_x11.c –lX11
```

Esimerkki 11.

```
/*
http://www.xmission.com/~georgeps/documentation/tutorials/Xlib_Beginner.h
tml */
#include <stdio.h>
#include <stdlib.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xos.h>
#include <X11/Xatom.h>
#include <X11/keysym.h>


int main() {
    Display *display;
    Window window;
    display = XOpenDisplay(NULL);
    window = XCreateSimpleWindow(display, RootWindow(display, 0), 1, 1,
500, 500, 0, BlackPixel (display, 0), BlackPixel(display, 0));
    XMapWindow(display, window);
    XFlush(display);
    sleep(5);
    return(0);
}
```

Loppusanat

8.8.Huomioita

Tässä oppaassa käydään läpi käytännön ohjelmoinnin perusteita C-kielellä. Opas toimii jatkona julkaisulle C-kieli ja käytännön ohjelmointi osa 1. Tässä jatko-osassa käsitellään esimerkkien, ohjeiden ja teorian avulla termejä joita jokainen aloitteleva ohjelmoija tulee käyttämään työelämässä. Vaikka opas ei olekaan tämän laajempi, toivon lukijan löytävän sisällöstä kiinnostavia aiheita ja pohjatietoa jota voi jatkossa syventää sekä ohjelmistotekniikan muilla kursseilla että työelämässä.

8.9.Lisäluettavaa

Käytännön ohjelmoinnista ja testauksesta on edelleen aika rajoitetusti materiaalia suomenkielisenä. Ohjelmiston arkkitehtuurista kiinnostuneille Koskimiehen ja Mikkosen Ohjelmistoarkkitehtuurit on varsin kattava teos (Koskimies 2005). Haikalan ja Märijärven teos Ohjelmistotuotanto (Haikala 2004), puolestaan kattaa varsin laaja-alaisesti ja syventävästi alan teemoja. Erityisesti ohjelmistojen testauksesta kiinnostuneet joutuvat vielä jonkin aikaa odottamaan ensimmäistä suomenkielistä alan teosta, tosin www-sivut ovat jo olemassa: http://www.testauskirja.com/.

Lähteet

Beizer, B. 1990, *Software testing techniques,* 2. ed., Van Nostrand Reinhold, New York (NY).

BetterExplained, [viitattu 01.04.2011], <URL:http://betterexplained.com/articles/intermediate-rails-understanding-models-views-and-controllers/>.

Eickelmann, N.S. & Richardson, D.J. 1996, October, *What makes one software architecture more testable than another? Joint proceedings of the second international software architecture workshop (ISAW-2) and international workshop on multiple perspectives in software development (Viewpoints '96) on SIGSOFT '96 workshop,* , San Francisco p. 65-67.

Fagan, M.E. 1976, *Design and code inspection to reduce errors in program development,* IBM Systems Journal 15(3), 182-211.

Gilb, T. & Graham, D. 1993, *Software inspection,* Addison-Wesley, Wokingham, England.

GNU Coding Standards, [viitattu 28.3.2011], <URL:http://www.gnu.org/prep/standards/html_node/index.html>.

Grady, R.p. 1993, November, *Practical Results from Measuring Software Quality*, Cmm ACM, Vol. 36, No. 11 p. 62-68.

Haikala, I. & Märijärvi, I. 2004, *Ohjelmistotuotanto,* Talentum, Helsinki.

Holopainen, J. 2005, *Regressiotestaus ja testien valintatekniikat,* Kuopion yliopisto, Kuopio.

IEEE 1990, *IEEE Standard Glossary of Software Engineering Terminology,* 610.12-1990.

ISO/IEC 2007, *International Standard Programming languages - C, WG14/N1256 Committee Draft, ISO/IEC 9899:TC3,* [viitattu 10.5.2011], <URL:http://www.open-std.org/JTC1/SC22/WG14/www/docs/n1256.pdf>.

Keizer, G. 2010, *Windows market share slide resumes,* Computerworld, [viitattu 22.03.2011], <URL:http://www.computerworld.com/s/article/9142978/Windows_market_share_slide_resumes >.

Kollanus, S. 2006, *Ohjelmistojen tarkastuskäytänteiden puutteet ja ongelmat teoriassa ja käytännössä,* Jyväskylän Yliopisto.

Koskimies, K. & Mikkonen, T. 2005, *Ohjelmistoarkkitehtuurit,* Talentum, Helsinki.

Myers, G.J., Badgett, T., Thomas, T.M. & Sandler, C. 2004, *The art of software testing*, 2nd edn, John Wiley & Sons, Hoboken, N.J.

Onoma, A., K., Tsai, W., Poonawala, M., H. & Suganuma, H. *Regression testing in an industrial environment*, Communications of the ACM, Vol. 41, No. 5, 1998, p. 81-86.

Paakki, J. 2000, *Ohjelmistojen testaus (software testing) lecture notes, autumn 2000*, Helsingin yliopisto, tietojenkäsittelytieteen laitos, Helsinki.

Pressman, R.S. 2005, *Software engineering: A Practitioner's Approach*, McGraw-Hill, Singapore.

Robot Wisdom [viitattu 10.5.2011], <URL:http://www.robotwisdom.com/linux/timeline.html>.

Russell, G.B. 1991, Jan, *Experience with inspection in ultralarge-scale development*, Software IEEE, Vol. 8, Issue 1 p. 25-51.

Satukangas, A. 2003, *Yksikkö- ja integrointitestauksen menetelmät ja mittarit*, Helsingin yliopisto, Helsinki.

STR template [viitattu 31.03.11], <URL:http://www.docstoc.com/docs/19773875/Frame-104-Software-test-report-%28STR%29---template>.

Tronche, C. *Xlib Programming Manual*, Inc.,[viitattu 03.04.2011] ,<URL:http://www.sbin.org/doc/Xlib/index_contents.html> O'Reilly & Associates.

Wikipedia Programming Style, [viitattu 28.3.2011], <URL:http://en.wikipedia.org/wiki/Programming_style>.

Wikipedia X Window System [viitattu 01.04.2011], <URL:http://fi.wikipedia.org/wiki/X_Window_System>.

Wikipedia Xlib, [viitattu 01.04.2011], <URL:http://fi.wikipedia.org/wiki/Xlib>.

Liite 1: Tyyliopas

Tyylioppaan tarkoituksena on orientoida opiskelijoita käyttämään yhtenäisiä rakenteita ja samaa ohjelmointityyliä.

Sisennys

Lohkojen sisentämisessä on suositeltavaa käyttää vähintään neljää (4) välilyöntiä. Lohkon aloittava aaltosulku on suositeltavaa laittaa lohkon aloituksen määräävälle riville.

Rivien pituus

Lähdekoodirivien pituus olisi suositeltavaa rajoittaa 80 merkkiin, jolloin tulosteissa ja näytöllä koodin luettavuus pysyy parempana. Jos kuitenkin on tarvetta kirjoittaa pidempiä koodirivejä, voidaan riviä jatkaa seuraavalle ”\”-merkin avulla.

Tyhjät rivit

Funktiot ja tietorakenteet erotetaan toisistaan tyhjällä rivillä. Funktion sisällä loogiset kokonaisuudet voidaan erotella toisistaan tyhjällä rivillä.

Välilyönnit

Unäärioperaattorit kirjoitetaan ilman välilyöntiä operaattorin ja operandin välillä:

```
!arvo
~bitit
i++
*ptr
```

Binäärioperaattorit erotetaan operandeista välilyönnillä:

```
n += 2
n > 0 ? n : -n
a < b
a <= b
```

Välilyönti kunkin puolipisteen jälkeen ja avainsanat if, else, while, for, switch, return erotetaan välilyönnillä:

```
for (i=0; i < 5; i++)
if (a < b)
while (a > b)
```

Suluissa olevien lausekkeiden alkava ja päättävä sulku ilman välilyöntiä:

```
x = (a + b) + c;
```

Kommentointi

Kommenttien tulisi olla sisällöltään selkeitä, selittäviä ja ajan tasalla. Hyvä kommentointi kuuluu hyvään ohjelmointitapaan ja tyyliin. Kielenä tulisi käyttää yhtenäisesti joko suomea tai englantia.

Koodilohkon kommentointi sisennetään samalle tasolle koodiosion kanssa. Kommenttirivin jatkuessa useammalle riville on uuden rivin aloittavana merkkinä ”*”-merkki.

Koodinsisäinen kommentointi (jossa kommenttirivi kirjoitetaan koodirivin perään) erotetaan vähintään kahdella välilyönnillä. Jos koodinsisäistä kommentointia on ohjelmalohkossa enemmän, tulee sisennyksen olla kaikissa sama. C++ tyylin mukaisia kommentteja (alkaa ”//”-merkeillä) ei suositella käytettäväksi vaikka kääntäjät ne yleensä hyväksyvätkin.

```
/* Tämä on C-tyylin mukainen kommentti */

/* Tämä on C-tyylin mukainen
    useamman rivin pituinen kommentti. */
```

Nimeämiskäytännöt

Käytä loogista ja selittävää nimeämistapaa. Muuttujanimien tulisi olla kuvaavia ja lyhyitä, eikä järjestelmän varattuja sanoja saa käyttää. Muuttujien esittelyt tehdään omilla riveillään vakio sisennyksellä mikäli mahdollista tai vaihtoehtoisesti samantyyppiset muuttujat yhdistettyinä omille riveilleen. Tällöin kuitenkin muuttujien mahdollinen alustaminen on tehtävä selkeästi. Kielenä käytetään yhtenäisesti joko suomea tai englantia sekä nimeämisessä että kommentoinnissa.

Lokaalit muuttujat kirjoitetaan pienillä kirjaimilla, sanat erotettuina alaviivalla:

```
int suurin_luku
```

Globaalit ja tiedoston muuttujat kirjoitetaan isoilla alkukirjaimilla:

```
ClkTimeStamp tai
Clk_TimeStamp (selkeämpi)
```

Vakiomäärittelyt ja makrot tulisi aina kirjoittaa isoilla kirjaimilla:

```
#define MAX 10
#define SUURIN_LUKU 45
```

Paikalliset ja globaalit funktiot kirjoitetaan isoilla alkukirjaimilla erottaen sanat toisistaan alaviivalla.

```
Clk_Set_Time()
```

Tietorakenteen (struct) jäsenet kirjoitetaan aloittaen kukin sana isolla kirjaimella.

```
CLK_TIME.DayOfWeek
```

Huomioita

Tässä on lyhyesti kuvattu C-kielen keskeisimmät tyyliohjeet. Lisää tietoa ohjelmoinnin tyylisäännöistä löytyy esimerkiksi GNU:n sivuilta: http://www.gnu.org/prep/standards/html_node/index.html.

Esimerkkirakenteita

Valintarakenne

/* Kysytään käyttäjältä tietoja, tulostetaan vastausten mukaan */

```
#include <stdio.h>

int main() {
    char fuksi;
    int ika;

    printf("Oletko fuksi (k/e)? ");
    fuksi = getchar();
    getchar();

    if (fuksi == 'k' || fuksi == 'K') {
        printf("Anna ikäsi: ");
        scanf("%d",&ika);
        if (ika >= 0 && ika <= 20)
            printf("Olet parhaassa iässä!\n");
        else if (ika >= 21 && ika <= 100)
            printf("Olet jo elämää nähnyt fuksi!\n");
        else
            printf("Ohjelmassa tapahtunut virhe!\n");
    }
    return 0;
}
```

Toistorakenteet

/* For silmukka jonka toistojen määrä on rajattu */

```
#include <stdio.h>

int main(void) {
    int i, kokonaisluku;
    printf("Anna kokonaisluku: ");
    scanf("%d", &kokonaisluku);
    for (i=1; i <= kokonaisluku; i++) {
        printf("%d\n", i);
    }
    return 0;
}
```

/* For silmukka jonka ei tulosta mitään */

```
#include <stdio.h>

int main(void) {
    int i, kokonaisluku;
    printf("Anna kokonaisluku: ");
    scanf("%d", &kokonaisluku);
    for (i=1; i <= kokonaisluku; i++) {
        /* ei komentoa */
        /* jos suoritusosa on tyhjä tulisi se kommentoida*/
    }
    return 0;
}
```

/* While lause */

```
#include <stdio.h>

int main(void) {
    int kokonaisluku, kertoma = 1, aloitus = 1;
    printf("Anna kokonaisluku: ");
    scanf("%d", &kokonaisluku);
    while(aloitus <= kokonaisluku) {
        kertoma = kertoma * aloitus;
        aloitus++;
    }
    printf("Luvun %d kertoma on %d\n",kokonaisluku,kertoma);
    return 0;
}
```

/* Do-while rakenne */

```
#include <stdio.h>

int main(void) {
    float kokonaisluku, keskiarvo, summa=0;
    int arvosanojen_lukumaara=0;
    printf("Ohjelma laskee syötettyjen arvosanojen keskiarvon.\nLopetus
negatiivisella kokonaisluvulla.\n");
    do {
        printf("Anna arvosana (4-10): ");
        scanf("%d", &kokonaisluku);
        if (kokonaisluku > 0) {
            summa += kokonaisluku;
            arvosanojen_lukumaara++;
        }

    } while(kokonaisluku > 0);
    arvosanojen_lukumaara = arvosanojen_lukumaara-1;
    /* silmukassa arvo kasvaa yhden yli halutun -> dekrementointi */
    keskiarvo = summa/(arvosanojen_lukumaara);
    /*  printf("summa = %.0f arvosanojen maara = %.0f
",summa,arvosanojen_lukumaara);*/
    printf("Ohjelmaan syötetty %.0d arvosanaa.\n",arvosanojen_lukumaara);
    printf("Arvosanojen keskiarvo: %.2f\n",keskiarvo);
    return 0;
}
```

/* Switch case rakenne */

```
#include <stdio.h>

int main(void) {
    int valinta;
    printf ("Anna valintasi (kokonaisluku): ");
    scanf("%d",&valinta);
    if ((valinta > 0) && (valinta <= 3)) {
        switch (valinta){
            case 1:
                printf("Tapaus 1\n");
                break;
            case 2:
                /* ei komentoa */
                break;
            case 3:
                printf("Tapaus 3\n");
                break;
            default:
                printf("Virheellinen valinta\n");
            break;
            }
    }
    else printf("Virheelinen valinta\n");
    return 0;
}
```

CT60A0210 Käytännön ohjelmointi 2011

LAPPEENRANNAN TEKNILLINEN YLIOPISTO

1) Käytetyt työmuodot soveltuivat opintojaksolle hyvin ja ne tukivat oppimistani opintojaksolla

1=täysin eri mieltä, 5=täysin samaa mieltä

1	2	3	4	5
○	○	○	○	○

2) Kokonaisarvioni opintojaksosta (arvosana asteikolla 1-5)

1	2	3	4	5
○	○	○	○	○

3) Jos kurssin vaatimukset ja sisältö pysyvät nykyisellä tasolla, niin kuinka näet kurssin opintopistemäärän

- ○ 1 Selvästi liian pieni (väh. 50% lisää opintopisteitä)
- ○ 2 Liian pieni
- ○ 3 Sopiva
- ○ 4 Liian suuri
- ○ 5 Selvästi liian suuri (väh. 50% liikaa opintopisteitä)

4) Mieti työpanostasi kurssin osa-alueisiin suhteessa kurssin sivuilla esitettyihin työmääräarvioihin. Kuinka näet tekemäsi työmäärän

	1 Paljon vähemmän	2 Vähemmän	3 Saman verran	4 Enemmän	5 Paljon enemmän
Luennot	○	○	○	○	○
Harjoitukset	○	○	○	○	○
Seminaarit	○	○	○	○	○
Kotitehtävä (t)	○	○	○	○	○
Harjoitustyö (t)	○	○	○	○	○
Luettava materiaali	○	○	○	○	○

5) Olettaen että kurssin opintopistemäärä pysyy samana, kuinka muuttaisit nykyisiä painotuksia tuntimäärissä

	1 Vähennä reilusti	2 Vähennä	3 Sopiva	4 Lisää	5 Lisää reilusti
Luennot	○	○	○	○	○
Harjoitukset	○	○	○	○	○
Seminaarit	○	○	○	○	○
Kotitehtävä(t)	○	○	○	○	○
Harjoitustyö(t)	○	○	○	○	○
Luettava materiaali	○	○	○	○	○

6) Laske kurssin osasuorituksiin käyttämäsi aika ja suhteuta se koko kurssin työmääräarvioon. Millaisen työmäärän koet tehneesi

- ○ 1 alle 50%
- ○ 2 50-75%
- ○ 3 75-100%
- ○ 4 100-125%
- ○ 5 125-150%
- ○ 6 yli 150%

KURSSIKOHTAISET KYSYMYKSET

7) Miten arvioit kurssia seuraavien yleisten ominaisuuksien osalta?

	1	2	3	4	5	6	7
Lähtötaso: kurssi lähti liikkeelle 1-liian perusasioista... 7-liian edistyneistä asioista	○	○	○	○	○	○	○
Määrä: kurssilla käsiteltyjen asioiden määrä oli 1-liian pieni... 7-liian suuri	○	○	○	○	○	○	○
Tarkkuus: asiat käsiteltiin kurssilla 1-liian pinnallisesti... 7-liian tarkasti	○	○	○	○	○	○	○
Kurssilla muodostunut kuva käytännön ohjelmoinnista on 1-epämääräinen... 7-selkeä	○	○	○	○	○	○	○

8) C-kielen osaaminen

Asteikko: 1-en ole varma mistä on kysymys ... 5-pystyn käyttämään/tekemään projektissa omatoimisesti

	1	2	3	4	5
Yksinkertaiset tietorakenteet ja tietotyypit, muuttujat	○	○	○	○	○
Peruskäskyt kuten syöttö, tulostus, valinta-, toisto- ja hyppyrakenteet, virheenkäsittely	○	○	○	○	○
Tiedostonkäsittely, teksti- ja binaaritiedostot, tietovirrat	○	○	○	○	○
Funktiot, parametrit, paluuarvot, rekursio, kirjastofunktiot	○	○	○	○	○
Ohjelman ja tiedoston rakenne, useat tiedostot, komentoriviparametrit	○	○	○	○	○
Dynaaminen muistinhallinta, osoittimet ja rakenteiset tietorakenteet	○	○	○	○	○
Linkitetty lista	○	○	○	○	○

Esikääntäjä, kääntäminen ja linkkaus ○ ○ ○ ○ ○

9) Käytännön ohjelmoinnin osaaminen

Asteikko: 1-en ole varma mistä on kysymys ... 5-pystyn tekemään projektissa omatoimisesti

	1	2	3	4	5
Arkkitehtuurisuunnittelu	○	○	○	○	○
Moduulisuunnittelu	○	○	○	○	○
Toteutus, tyyliopas, make, versionhallinta	○	○	○	○	○
Debuggaus	○	○	○	○	○
Kooditarkastukset	○	○	○	○	○
Moduulitestaus, testiajurit ja tynkämoduulit	○	○	○	○	○
Integrointitestaus	○	○	○	○	○
Systeemitestaus	○	○	○	○	○
Testauksen hallinta, kattavuus ja automatisointi	○	○	○	○	○

10) Käytetyt käyttöjärjestelmät ja ohjelmat

Osaaminen: 1-en ole varma mistä on kysymys ... 5-pystyn käyttämään projektissa omatoimisesti
Käyttö: 1-en ole käyttänyt ... 5-luennoijan ohjeiden ja esimerkin mukaisesti

	Osaaminen					Käyttö				
	1	2	3	4	5	1	2	3	4	5
Codeblocks-kehitysympäristö	○	○	○	○	○	○	○	○	○	○
Linux-käyttöjärjestelmä	○	○	○	○	○	○	○	○	○	○
gcc ja gedit ohjelmointityökalut	○	○	○	○	○	○	○	○	○	○
make ja Makefile	○	○	○	○	○	○	○	○	○	○
Versionhallinta ja svn	○	○	○	○	○	○	○	○	○	○
Debuggeri, esim. DDD	○	○	○	○	○	○	○	○	○	○

LUENTOVIDEOT

11) Luentojen ja videoiden seuraaminen

Vastaa antamalla prosenttiosuus kokonaislukuna. Yksi luentokerta, 90 min, vastaa 7% kurssin luennoista ja 7 kertaa vastaa 49%:ia. Jos katsoit videoita useamman kerran, laske ne myös mukaan eli jos katsoit kaikki luennot kahteen kertaa, vastaa 200.

Missä määrin osallistuit luennoille

Missä määrin katsoit luentovideoita

12) Missä määrin seuraavat väitteet vastaavat tilannettasi kurssilla

	Täysin eri mieltä	Eri mieltä	En osaa sanoa	Samaa mieltä	Täysin samaa mieltä	Ei koske minua

Olisin saanut kurssin suoritettua ilman luentovideoita	○	○	○	○	○	○
Ymmärsin luennoilla esitetyt asiat hyvin ollessani luentosalissa läsnä	○	○	○	○	○	○
Ymmärsin luennoilla esitetyt asiat hyvin videolta kuultuna	○	○	○	○	○	○
Minulla ei ollut teknisiä ongelmia videoihin liittyen	○	○	○	○	○	○

13) Listaa ja selitä lyhyesti videoiden tärkeimmät höydyt sinulle

14) Listaa ja selitä lyhyesti videoiden kanssa kokemasi ongelmat, jos niitä oli

15) Luentopoissaolojen syyt

Mikäli et osallistunut kaikille luennoille, listaa tärkeimmät syyt poissaoloihisi siten, että ne kattavat 80% poissaoloistasi. Käytä yleisiä mutta ymmärrettäviä syitä ja selitä ne tarpeen tullen lyhyesti, esim. päällekkäiset luennot/harjoitukset (+kurssinumero), päätoiminen työssäolo, sairaus, jne.

YLEISTÄ PALAUTETTA

Vastaa alla oleviin kysymyksiin nimeämällä pari-kolme tärkeintä asiaa ja selitä vastauksesi lyhyesti.

16) Oliko kurssilla jotain hyvää, mitä sinusta ei pitäisi muuttaa jatkossa?

(liite 2 jatkoa)

17) Oliko kurssilla jotain huonoa, jota sinusta pitäisi muuttaa jatkossa?

18) Vapaa palaute opintojaksosta, esim. kehittämisehdotukset, vahvuudet, heikkoudet.

☐Haluan lähettää vastaukset

Lähetä