AUTOMAATTINEN OHJELMAKOODIN TARKASTUS

Boss- ja CourseMaster -järjestelmien soveltuvuus ohjelmoinnin virtuaaliopintoihin

Taina Lehtonen

29.10.2002

Joensuun yliopisto

Tietojenkäsittelytiede

Pro gradu -tutkielma

TIIVISTELMÄ

Arviointiprosessin automatisointia tarvitaan tietojenkäsittelyn opetuksessa lähinnä opiskelijamäärien kasvamisen takia. Ohjelmoinnin opettaminen vaatii käytännön ohjelmointiharjoitusten valmistelua ja arviointia, mikä vie paljon aikaa. Automatisointi vähentää opettajan tehtävien arviointiin käyttämää aikaa ja tekee arvioinnista objektiivisempaa. Opiskelijoiden kannalta katsottuna etuna on nopeampi tehtävien arviointi ja niistä saatu palaute. Tässä pro gradu -tutkielmassa on käsitelty automaattisia koodintarkastus -järjestelmiä ja niistä erityisesti Boss- ja CourseMaster -järjestelmiä, joita on myös käytännössä testattu. Testattaessa päämääränä oli selvittää Bossin ja CourseMasterin soveltuvuutta Joensuun yliopiston virtuaaliopintoihin. Testauksessa ongelmia aiheuttivat CourseMasterin asennus ja Boss-järjestelmässä tehtävien soveltuvuus järjestelmään. Käyttöönotossa haittapuolina olisivat asiakas-sovelluksen asentaminen jokaiselle koneelle erikseen ja palomuuriin tehtävät reiät. Lisäksi järjestelmien saaminen käyttökuntoon vaatisi huolellista valmistautumista ja huomattavan paljon aikaa.

Esipuhe

Haluan kiittää tutkielmani ohjaajia Jarkko Suhosta ja professori Erkki Sutista. Lisäksi haluan esittää kiitokset professori Mike Joylle ja laboratorio insinööri Juha Hakkaraiselle heidän avustaan järjestelmien testaamisessa.

SISÄLLYSLUETTELO

1. JOHDANTO

Joensuun yliopiston virtuaalisissa ohjelmoinnin opinnoissa tehtävien tarkastaminen ja palautteen antaminen vie paljon aikaa yhdessä muiden kurssiin liittyvien tehtävien kanssa ja vaatii useiden henkilöiden työpanoksen. Tarkastusprosessin nopeuttamiseksi ja helpottamiseksi onkin harkittu automaattisen tai puoliautomaattisen koodintarkastusjärjestelmän käyttöönottoa ohjelmointikurs-seilla.

Tämän pro gradu -tutkielman tehtävänä on osaltaan selvittää, voisiko ohjelmoinnin virtuaaliopetuksessa käyttää Boss- tai Ceilidh-järjestelmiä. Nämä järjestelmät on valittu tutkielman pääasiallisiksi tutkimuskohteiksi, koska ne ovat hyvin tunnettuja ja laajalti käytettyjä koodintarkastusjärjestelmiä. Lisäksi kuvailen, millaisia muita automaattisia koodintarkastus-järjestelmiä on kehitetty maailmalla.

Virtuaaliopinnoilla tarkoitan Joensuun yliopiston tietojenkäsittelytieteen laitoksen ja Pohjois-Karjalan lukion opettajien yhdessä luomaa projektia, jossa opetetaan yliopistotasoisia tietojenkäsittelytieteen opintoja. Virtuaaliapprossa lukio-opiskelijat opiskelevat 15 opintoviikkoa tietojenkäsittelytieteen opintoja Internetin välityksellä. Opinnot antavat opiskelijoille perustiedot kolmesta osa-alueesta: tietojenkäsittelytieteestä, ohjelmoinnista Javalla ja tietokoneista. Kun opiskelija on suorittanut opintoviikot, hän on selvittänyt tietojenkäsittelytieteen ensimmäisen vuoden opinnot. Jos hän läpäisee ohjelman arvosanalla 2/3, hän pääsee suoraan opiskelemaan Joensuun yliopistoon tietojenkäsittelytiedettä. (Haataja et al. 2001)

Boss ja Ceilidh- järjestelmät on molemmat alun perin kehitetty UNIX-käyttöjärjestelmään, pääasiassa kurssin hallinnointiin ja opiskelijoiden ohjelmointikurssilla tekemien ohjelmakoodien automaattiseen tarkastukseen. Ceilidh otettiin käyttöön Nottinghamin yliopistossa vuonna 1988 ja Boss Warwickin yliopistossa vuonna 1993. Molemmat järjestelmät on myöhemmin toteutettu uudelleen alusta alkaen Javalla, Ceilidh-järjestelmän uudelleentoteutuksen yhteydessä järjestelmän nimi vaihdettiin CourseMasteriksi.

Automaattinen koodin tarkastaminen on yksi osa tietokoneavusteisesta arvioinnista. Ensimmäisessä luvussa käsittelenkin arviointia ja tietokoneavusteista arviointia yleensä, sekä palautteen antamista oppimisprosessista ja plagioinnin tarkistamista opiskelijoiden vastauksista. Luvun lopussa olen kertonut automaattisesta koodin tarkastamisesta yleisellä tasolla.

Toisessa luvussa esittelen automaattiseen koodin tarkastamiseen suunniteltuja järjestelmiä. Tässä luvussa esittelen Boss- ja Ceilidh -järjestelmien lisäksi muutamia muita vastaavia järjestelmiä, kuten Pilot-, WebAssign-, RoboProf- ja SchemeRobo-järjestelmiä, sekä prototyyppiä järjestelmästä, jossa ohjelmatehtävien tarkastaminen suoritetaan kielestä riippumattomasti. Luvun loppuun olen koonnut yhteenvedon automaattisille koodintarkastusjärjestelmille ominaisista piirteistä.

Kolmannessa luvussa käsittelen automaattisen koodin tarkastamisen toteutustekniikoita, joista ohjelman testaaminen on ohjelman tärkein oikeellisuuden mittaamisen metodi. Lisäksi käsittelen ohjelmille asetettuja vaatimuksia, kuten laatua, tehokkuutta ja luettavuutta.

Neljäs luku pitää sisällään Boss- ja Ceilidh -järjestelmien empiiristä testaamista. Testaamisessa hyödynsin Joensuun yliopiston virtuaaliopinnoissa käytettyjä ohjelmointitehtäviä, jotka lähetin CourseMaster- ja Boss-järjestelmään opiskelijana, ja tarkastelin eri vastauksista automaattisesta arvioinnista saatuja tuloksia, sekä punnitsin järjestelmien soveltuvuutta etäopintoihin. Lopun yhteenvedossa pohdin mahdollisia jatkotoimenpiteitä.

2. TIETOKONEAVUSTEINEN ARVIOINTI

Tietokoneavusteinen arviointi ja automaattinen koodin tarkastaminen liittyvät toisiinsa siten, että automaattinen koodin tarkastaminen on yksi tietokoneavusteisen arvioinnin osa-alueista. Luvussa käsittelen tietokoneavusteisen arvioinnin merkitystä oppimisessa sekä erilaisia arvioinnin muotoja. Lisäksi kuvaan palautteen merkitystä opiskelijalle ja opettajalle sekä tehtävien plagiointia yleisellä tasolla.

2.1 Arviointi oppimisessa

Choate ja Evans (1992) esittävät, että arviointi on prosessi, jossa kerätään systemaattisesti opettamisen apuna käytettävää tietoa opiskelijoiden kyvyistä, käyttäytymisestä ja ympäristöstä. Prestonin ja Shackelfordin (1999) mukaan tarkka ja merkityksellinen oppimisen arviointi on tärkeää, koska se antaa opiskelijalle ja opettajalle palautetta oppimisesta, luo varmuutta opiskelijan suorituksen mittaamiseen ja tarjoaa järkevän välineen opiskelijan tietojen ja taitojen mittaamiseksi. Lisäksi arviointi luo opettajalle ja johdolle keinon kontrolloida laatua sekä antaa uudenlaisia mahdollisuuksia tutkia opetusta.

McAlpine (2002) pitää arviointia eräänä vuorovaikutuksen muotona, joka voi olla monensuuntaista; esimerkiksi opiskelijalle annettavaa palautetta oppimisesta, opiskelijan antamaa palautetta luennoitsijalle hänen opetuksestaan tai palautetta opetussuunnitelmasta sen tekijälle. Arvioinnin strategian suunnittelussa on otettava huomioon useita seikkoja, jotta kommunikointi olisi merkityksellistä, käyttökelpoista ja totuudenmukaista.

2.1.1 Prosessin ja tuotteen arviointi

McAlpinen (2002) mukaan arviointi voi kohdistua joko *oppimisprosessiin* tai *-tuotteeseen*. Oppimistuotteen arviointi voi kohdistua esimerkiksi opiskelijan tekemään esseetehtävään. Prosessipohjainen arviointi voi olla vaikkapa uuden alueen tutkimistehtävän arviointia. Opettajan onkin hyvä miettiä arvioidaanko opiskelijan oppimisen tuotetta vai suoritettua prosessia.

Arvioinnin painopiste voidaan asettaa joko hankittuihin taitoihin tai saatuun tietämykseen. Yleensä arviointi on sekoitus molempia.

2.1.2 Formatiivinen ja summatiivinen arviointi

Formatiivinen arviointi edesauttaa oppimisprosessia tarjoamalla oppijalle palautetta, jota voidaan käyttää paljastamaan lisäopiskelua tarvitsevia alueita. Näin pyritään parantamaan opiskelijan suoritusta tulevaisuudessa. Itsearviointi ja diagnostinen arviointi ovat formatiivisen arvioinnin muotoja, joilla on erityiset tarkoitukset. Formatiivinen arviointi auttaa opiskelijaa ja muita osapuolia muodostamaan yksityiskohtaisen kuvan opiskelijan kyvyistä. Tätä kuvaa voidaan käyttää parantamään opiskelijan yleistä suoritustasoa, keskittämällä hänen mielenkiintoaan sopiville alueille. Formatiivinen arviointi on asianmukaisin arvioinnin tapa, kun tuloksia käyttävät oppimisprosessissa mukana olleet eli opiskelijat ja opettajat. (McAlpine 2002)

Summatiivinen arviointi on suunniteltu mittaamaan opiskelijan yleistä suorituskykyä ja se tehdään kurssin lopussa. Tällainen arviointi on hyödyllisintä niille ihmisille, jotka ovat opetusprosessin ulkopuolella, kuten esimerkiksi työnantajat ja jatko-opiskelupaikat, ja jotka tekevät päätöksiä opiskelijoista kerätyn informaation perusteella. Summatiivinen arviointi tuottaa yleensä suppean yhteenvedon opiskelijan kyvyistä, esimerkiksi arvosanan. (McAlpine 2002)

2.1.3 Formaali- ja informaali -arviointi

Choate ja Evans (1992) sanovat arvioinnin jakaantuneen perinteisesti *formaaliin* ja *informaaliin* arviointiin. Formaaleissa arviointimenetelmissä käytetään standardisoituja kokeita tai testejä, jossa opiskelijan suoritusta verrataan normatiiviseen vertaisryhmään. McAlpinen (2002) mukaan formaalissa arvioinnissa opiskelijat tietävät tekevänsä harjoitukset arviointia varten. Formaali arviointi on monesti informaalia arviointia tarkempaa ja puolueettomampaa. Formaalilla arvioinnilla voi olla motivoiva vaikutus opiskelijoihin ja alkuvaiheessa tehty formaali arviointi saattaa rohkaista heitä parempiin suorituksiin, jos opiskelijat ovat huonosti motivoituneita.

Toisaalta formaali arviointi voi aiheuttaa stressiä ja saada jotkut opiskelijat suoriutumaan edellytyksiään huonommin.

McAlpine (2002) sanoo, että *informaalissa arvioinnissa* arviointi sisällytetään muihin tehtäviin. Informaalia arviointia ovat esimerkiksi luentomuistiinpanojen tekeminen käytännön harjoituksen aikana, luokan ranskan kielen keskustelusta otettu nauhoitus, tietokonepohjaisen oppimisen käytöstä otetut nauhoitukset ja itsearviointi. Informaali arviointi saattaa vähentää stressiä ja antaa pätevämmän kuvan opiskelijan kyvyistä, kuitenkin jotkut opiskelijat voivat tuntea itsensä huijatuksi ilman mahdollisuutta loistamiseen. Ongelmia voi tulla myös, koska piilotetut ennakkoasenteet ja stereotypiat saattavat vaikuttaa tarkastajan arviointiin.

2.1.4 Loppuarviointi ja jatkuva arviointi

Arviointi voidaan suorittaa *loppuarviointina* tai *jatkuvana arviointina*. Loppuarviointi tapahtuu kurssin lopussa, kun taas jatkuva arviointi hajoaa nimensä mukaisesti koko kurssin ajalle. Loppuarviointia on esimerkiksi perinteiset lopputentit. Jatkuva arviointi voi olla esimerkiksi modulaarista arviointia, jossa arviointi tehdään jokaisen tehdyn alan harjoitelman lopussa. (McAlpine 2002)

McAlpinen (2002) mukaan loppuarvioinnin suurin hyöty on, että se on helppo organisoida ja sen avulla arviointiprosessi tiivistyy lyhyelle aikavälille. Tästä johtuen ajoituksen merkitys on tärkeä ja esimerkiksi sairaus saattaa vaikuttaa lopputulokseen. Loppuarviointia ei myöskään voida käyttää formatiivisen arvioinnin kanssa. Loppuarviointi soveltuu hyvin sellaiseen oppimiseen, jossa jokainen uusi opiskeltava asia auttaa muiden osa-alueiden ymmärtämistä ja siksi oppiminen voidaan arvioida vain kokonaisuutena osien sijaan.

Jatkuvan arvioinnin hyöty on, että opiskelijat ja opettajat saavat palautetta prosessista, jota voidaan käyttää opettamisen tai oppimisen apuna. Lisäksi lopulliset tulokset perustuvat koko opiskelujakson ajalle. Haittapuolena on työtaakan kasvaminen. Jatkuva arviointi soveltuu parhaiten sellaiseen opiskeluun, jossa opiskelijan palautetta tarvitaan ja osatietojen sarjoja vaaditaan koko kurssin läpi, jotta kokonaiskuva opiskelijan taidoista voidaan rakentaa.

Tietokoneavusteinen arviointi voi tarjota tehokkaat keinot opiskelijoiden jatkuvaa arviointia varten. Lisäksi se välittää nopean ja yksityiskohtaisen palautteen opiskelijoille oppimisprosessista. (McAlpine 2002)

2.1.5 Konvergentti ja divergentti arviointi

Konvergentti arviointi tarkastaa kysymyksiä, joihin on olemassa vain yksi ainoa oikea vastaus. Konvergentti arviointi kattaa opintosuunnitelman laajasti ja sitä voidaan käyttää opiskelijan tiedoissa olevien puutteiden löytämiseen. Konvergenttien vastausten arviointi on yleensä suhteellisen helppoa ja nopeaa, niin automaattisesti kuin käsin tehtynä. Konvergentti arviointi on käytännöllinen tapa silloin, kun kurssin päätarkoituksena on tietämyksen hankkiminen. Huonona puolena on kuitenkin, että ne voivat olla rajoittuneita tietylle alalle tai jäädä pelkkiä yksittäisiä faktatietoja kysyväksi visailuksi. (McAlpine 2002)

Tietokoneavusteinen arviointi on yhä yleisemmin käytetty konvergentin arvioinnin muoto. Tietokoneavusteista arviointia käytetään usein opiskelijoiden tietojen laaja-alaiseen ja nopeaan arviointiin, sillä se löytää tehokkaasti mahdolliset puutteet opiskelijoiden tiedoissa käyttämällä tilastollisia analyysejä ja raportteja. Tietokoneet tarjoavat myös tietynlaisia etuja alan laajentamisessa ja konvergentin arvioinnin oikeellisuudessa. Hyvät kysymykset ja testit vaativat kuitenkin taidokasta suunnittelua ja kunnollisen lauserakenteen. (McAlpine 2002)

Divergentti arviointi hyväksyy oikeaksi monenlaisia vastauksia, jotka perustuvat opiskelijan tietoisesti muodostettuihin näkemyksiin ja mielipiteisiin. Divergentti arviointi on konvergenttia arviointia autenttisempaa ja se tekee kognitiivisten taitojen arvioinnin helpommaksi, mutta toisaalta se on hitaampi toteuttaa. Lisäksi divergentti arviointi vaatii enemmän arviointitaitoja kuin konvergentti arviointi, jonka vuoksi arvioijia voi joutua kouluttamaan ja suunnittelemaan yksityiskohtaisia arviointikriteereitä. (McAlpine 2002)

2.1.6 Arvioinnin laatu

Arvioinnin on tuotettava tarvittava informaatio. Huonosti toteutettuna testin tuottama informaatio voi olla osittaista tai jopa harhaanjohtava. Testin kysymykset ovat laadukkaita, kun ne ovat sopivan vaikeita ja niiden avulla kyetään erottelemaan opiskelijoita. (McAlpine 2002)

Validiteetti (validity) ja *luotettavuus* (reliability) ovat McAlpinen (2002) näkemyksen mukaan tärkeitä osa-alueita arvioinnissa, koska ne muodostavat arvioinnin yleisen laadun. Validi arviointi tarkoittaa sitä, että arviointi mittaa oikeita asioita. Validiteettivaatimus kohdistuu moneen eri osa-alueeseen: opetussuunnitelman pitää olla validi, eli sen päämäärien ja sisällön pitää vastata sitä, mitä opiskelijoiden tulee tietää. Arvioinnin pitää liittyä läheisesti arvioitavaan aihealueeseen, ja sen avulla pitäisi pystyä ennustamaan opiskelijan tulevaa opintomenestystä samalla aihealueella. Tällä niin kutsutulla ennustavalla validiteetilla on merkitystä lähinnä, kun arvioinnin tarkoituksena on valita opiskelijoita.

Luotettavuus mittaa sitä, ovatko arvioinnin tulokset samoissa olosuhteissa jokaisella kerralla samat. Arvioinnista saatujen tuloksien tulisi olla jatkuvia, esimerkiksi jos opiskelija tekee saman testin uudestaan, hänen pitäisi saada samankaltainen tulos myös toisella kerralla. Näin varmistetaan, että kyse ei ole pelkästä sattumasta. (McAlpine 2002)

2.1.7 Arvosanojen antaminen

Arvosanojen antaminen (grading) sisältää opiskelijoiden suoritusten vertaamisen joihinkin edeltä määriteltyihin standardeihin. Kaksi yleisimmin käytettyä vertaamistapaa arvioinnissa ovat *normiin vertaava arviointi* (norm referenced grading) ja *asian hallinnan oppiminen* (mastery learning). Normiin verratessa opiskelijan suoritusta verrataan toisten henkilöiden suorituksiin, joiden ajatellaan määritelleen jonkin tietyn tasoisen standardisuorituksen. Normiin vertaavaa arviointia käytetään esimerkiksi älykkyystesteissä, joissa testit on normitettu edeltä määritellyn ryhmän mukaan ja myöhempien testiin osallistujien suoritusta verrataan näihin normeihin. Koulutuksessa tämän tyyppisestä arvostelusta on tullut harvinaista, vaikka se vaikuttaa vieläkin

monissa tiedostamattomissa arviointiin liittyvissä uskomuksissa. Normiin vertaava arviointi vaikuttaa olevan ajan myötä muuttumaton, mutta opetusohjelman ja opiskelijaryhmien vaihtuessa tämä ei välttämättä pidä paikkaansa. Asian hallitsemisen oppimisessa taas opiskelijan suoritusta verrataan tiettyihin ennalta määrättyihin oppimistavoitteisiin. Käytännössä arvioinnissa käytetään usein molempien tapojen yhdistelmää. (McAlpine 2002)

2.2 Tietokoneavusteinen arviointi

O'Leary (2002) määrittelee *tietokoneavusteisen arvioinnin* (englanniksi Computer Aided Assessment, lyhenne CAA) sisältävän tietokoneen käyttämisen tehtävien tai tenttien jakamiseen, arviointiin ja analysointiin sekä *optisten lukijoiden* (optical mark reader) tietojen tarkistamiseen ja analysointiin. Termi viittaa tietokoneen hyödyntämiseen arvioinnin apuna. Tietokoneavusteista opetusta käytetään muun muassa kokeissa (esimerkiksi QuestionMark for Windows), rutiinitarkastuksissa, etäopetuksessa ja itsearvioinnissa.

Tehtävien automaattinen tarkastaminen on Malmin (2002) mukaan menettely, jossa tietokoneohjelma arvioi opiskelijan palauttamaa vastausta ja antaa siitä arvion, joko pisteinä tai muunlaisena palautteena. Automaattisella tarkastamisella voidaan kuitenkin tarkoittaa monenlaisia asioita. Tarkastus voi olla puoliautomaattista, jolloin tietokone toimii pelkkänä opettajan apuvälineenä, joka helpottaa normaalia tarkastamisen tekemistä. Toisaalta tarkastaminen voi tapahtua täysin automaattisesti, jolloin se hoidetaan kokonaan koneen avulla. Tässä pro gradu -tutkielmassa tietokoneavusteisella arvioinnilla tarkoitetaan tietokoneohjelmaa, joka tarkastaa opiskelijoiden tekemiä harjoitustöitä tai tenttejä, joko kokonaan tai osittain ihmisen avustuksella. Harjoitustyöt voivat olla monenlaisia: monivalintatehtäviä, lukuarvotehtäviä, sanallisia vastauksia, kuvan avulla vastattavia tehtäviä, essee-tehtäviä tai ohjelmointitehtäviä.

Dalziel ja Gazzard (1999) näkevät tietokoneavusteisten arviointiohjelmien roolin opetuksessa merkittävänä 1900-luvun loppupuolella. Tietokoneavusteiset tarkastusohjelmat ovat sisältäneet muun muassa *ohjelmiston testauspaketteja* (software testing package), *CAA-moduuleja* ja *optisia merkin luku-sovelluksia* (optical character recognition tai OCR).

Tietokoneavusteiseen arviointiin tarkoitettuja ohjelmia ovat muun muassa etupäässä opettamiseen suunnitellut CASTLE, Examine, GeoData Quiz ja WebAssign -järjestelmät, sekä kaupalliset järjestelmät, kuten markkinajohtaja Question Mark, WebMCQ ja WinAsks Professional (Alexander 1999). Myös automaattiseen kurssin hallinnointiin (administration) ja arviointiin on kehitetty useita järjestelmiä, kuten esimerkiksi HANDIN, AUTOMARK ja CENVIRON (Benford et al. 1994).

2.2.1 WWW-pohjaiset järjestelmät

Dalzielin ja Gazzardin (1999) mielestä web-pohjainen arviointi antaa joustavuutta tietokoneavusteiseen arviointiin, erityisesti ajan, paikan ja työvauhdin suhteen, ja sen merkitys onkin kasvanut opetuksessa. Web-pohjaisia järjestelmiä käytettäessä voidaan välttää monia ei-web-pohjaisten tietokoneavusteisten tarkastusjärjestelmien rajoituksia, kuten paikan ja alustan rajoitukset, sekä erillinen asennus ja päivitys jokaiselle koneelle. Lisäksi ei-web-pohjaiset järjestelmät ovat olleet monesti vaikeita käyttää tai muokata ja niitä on ollut vaikea integroida muuhun opetusmateriaaliin.

Web-pohjaiset järjestelmät tulevat todennäköisesti syrjäyttämään niitä edeltävät järjestelmät monessa yhteydessä, koska niiden avulla on mahdollista luoda järjestelmä, jossa ei tarvita lataamista, verkkoon kirjautumista, tai asennusta. Web-pohjaiset arviointiohjelmat voivat sisältää suuren joukon kyselymuotoja, suuremman joustavuuden materiaalin esittämisessä ja kehittyneemmän opiskelijoiden seurannan. (Dalziel, Gazzard 1999)

WWW-pohjaisia opiskelijan ohjelmakooditehtävien tarkastukseen käytettyjä järjestelmiä ovat muun muassa Pilot-, RoboProf- ja WebAssign-järjestelmät sekä myös CourseMaster-järjestelmän Singaporessa kehitetty versio. Web-pohjaisia tarkastusjärjestelmiä voidaan käyttää muun muassa virtuaaliyliopistoissa, etäopiskelussa ja online-luokissa. Näin on tehty esimerkiksi PILOT- ja WebAssign -järjestelmissä. (Bridgeman et al. 2000)

2.2.2 Eri tehtävätyypit

Tietokoneavusteinen arviointi käsitetään monesti pelkkinä summatiiviseen arviointiin soveltuvina monivalintakysymystehtävinä. Tämä ei Nicholsonin (2001) mukaan kuitenkaan ole koko totuus, sillä monivalintatehtävien lisäksi tietokoneavusteisesti voidaan arvioida muun muassa lukuarvotehtäviä, joissa opiskelija antaa vastauksensa lukuarvona, sanallisia vastauksia, joissa vastauksena on yleensä yksi sana, kuvan avulla vastattavia tehtäviä, essee-tehtäviä ja ohjelmointitehtäviä.

Tietokoneavusteista arviointia käytetään yleensä *objektiivisissa testeissä* (objective testing), kuten tosi/epätosi-tehtävissä ja monivalintatehtävissä, joissa opiskelija valitsee oikean vastauksen hänelle tarjotuista vaihtoehdoista. Objektiivisilla testeillä on yksiselitteinen oikea vastaus tai usean vastauksen sarja. Monivalintatehtävät ovat käytetyin objektiivisen testin kysymystyyppi. Syitä siihen ovat muun muassa, että ne voidaan arvioida nopeasti ja niillä voidaan kattaa laaja osa kurssin sisällöstä. Lisäksi objektiiviset testit ovat erittäin sopivia itsearviointiin, koska ne tarjoavat opiskelijoille jatkuvaa palautetta heidän edistymisestään. (O'Leary 2002)

2.2.3 Tietokoneavusteisen arvioinnin edut ja ongelmat

Tietokoneavusteisella arvioinnilla on useita merkittäviä etuja. Opiskelijalle voidaan tarjota tarkastamisesta välitöntä palautetta ja arviointi on jatkuvampaa. Arvioinnin yleinen tehokkuus kasvaa, kun tarkastaminen nopeutuu ja arviointikulut vähentyvät. Suurin hyöty automatisoinnilla saavutetaankin arvioinnin tehokkuudessa ja opettajan tarkastamiseen käyttämän ajan säästämisessä muuhun opettamiseen liittyvään toimintaan, josta on hyötyä erityisesti suuria opiskelijamääriä käsittävillä kursseilla (Dalziel, Gazzard 1999).

O'Learyn (2002) mukaan myös arvioinnin objektiivisuus paranee, koska tarkastajan henkilökohtaiset ominaisuudet, kuten joidenkin opiskelijoiden suosiminen, väsyminen ja ennakkoasenteet, eivät vaikuta arviointiin. Tietokoneet mahdollistavat lisäksi nopean ja tarkan

automaattisen arvioinnin analysoinnin, jonka avulla opettaja voi poimia esiin ongelmalliset alueet arvioinnista.

O'Leary (2002) näkee tietokoneavusteisella arvioinnilla myös ongelmia: monissa järjestelmissä kysymystyypit on rajoitettu vain monivalintatehtäviin. Tällöin tuloksien luotettavuus voi kärsiä, koska testaus saattaa keskittyä vain alemman tason oppimisen testaamiseen ja oikeat vastaukset voidaan saada myös arvaamalla. Tarvittava tietokonelaitteisto vaatii myös paljon rahaa ja kuulusteluiden valvonta voi olla vaikeaa.

Dowsing ja Long (1999) jakavat arvioinnin karkeasti neljään eri vaiheeseen, joista ensimmäinen on arvioinnin päämäärien ja vaatimusten määritteleminen. Tässä vaiheessa asetetaan tavoitetaidot ja kriteerit, jotka spesifioivat suorituksen tai tiedon tason. Toinen vaihe on opiskelijoiden näytön kerääminen. Viimeiset vaiheet ovat näytön sovittaminen arviointitavoitteisiin, eli saavutettujen tavoitteiden mittaaminen, sekä tulosten arviointi.

Riittävien, yksikäsitteisten ja virheettömien arviointisääntöjen luominen tietokoneavusteisessa arvioinnissa on vaikea ja virhealtis tehtävä. Sääntöjen tekeminen ihmistä varten on kuitenkin yksinkertaista, koska heidän voidaan olettaa käyttävän omaa harkintaansa tapauksissa, joissa kriteerit ovat puutteellisia. Ongelmaksi jää kuitenkin arvioinnin jatkuvuus, kun kaikki tarkastajat eivät ajattele samalla tavalla. Tietokoneavusteisessa arvioinnissa tällaista harkintaa ei ole ja sen rakentaminen tekoälytekniikoilla on vaikeaa. Automatisoitua arviointia käytettäessä arviointikriteerit on siis määriteltävä paljon yksityiskohtaisemmin kuin manuaalisessa arvioinnissa. (Dowsing, Long 1999)

Automatisoidussa arvioinnissa arvioitava materiaali kerätään elektronisesti. Dowsingin ja Longin (1999) mukaan se vaikuttaa arviointiprosessiin usealla tavalla, esimerkiksi esseetä tai tietokoneohjelmaa tarkastettaessa samanlainen tuloste voidaan tuottaa erilaisilla käyttäjän tekemillä toiminnoilla. Jotkin paperitulosteen ominaisuudet eivät välttämättä ole saatavissa elektronisessa kopiossa ja jotkin ominaisuudet, jotka ovat vaikeita tai jopa mahdottomia mitata paperilta, ovat helppoja arvioida elektronisessa muodossa. Samojen arviointikriteerien

käyttäminen ihmisen ja tietokoneen tekemälle arvioinnille on edellä luetelluista syistä ongelmallista.

Opiskelijan saavuttamien taitojen mittaamisessa on kaksi tehtävää. Ensimmäinen niistä on mahdollisten virheiden löytäminen vastauksista ja toinen on yhdistää opiskelijan tekemät virheet yhteen tai useampaan arviointikriteeriin eli virheiden luokittelu. Ihmiset ovat hyviä virheiden luokittelussa, mutta huonoja niiden löytämisessä. Tietokoneavusteinen arviointi taas on huono virheiden luokittelussa, koska siltä puuttuu kyseessä olevan alan ja semantiikan tietämys. Tietokoneet ovat kuitenkin hyviä virheiden etsimisessä. Monesti onkin järkevää yhdistää ihmisen ja tietokoneen tekemät arvioinnit. Lisäksi on hyvä muistaa, että mitä korkeamman tasoisia tietoja tai taitoja mitataan, sitä tärkeämmäksi ihmisen tekemä arviointi tulee. (Dowsing, Long 1999)

2.3 Automaattinen koodin tarkastus

Ohjelmoinnin opettaminen vaatii käytännön ohjelmointiharjoitusten valmistelua ja arviointia. Opettajien työmäärä ohjelmointikursseilla on tavallisesti huomattava ja varsinkin suuria opiskelijamääriä käsittävillä kursseilla se on voinut vaatia kohtuuttomasti aikaa ja vaivaa. Arvioijien välinen luotettavuus ja objektiivisuus on ongelma, joka heikentää arvioinnin jatkuvuutta ja laatua. Näin on varsinkin suurilla kursseilla, joilla voi olla esimerkiksi satoja opiskelijoita ja yli sata apulaisopettajaa, jotka tarkastavat yli 4000 tehtävää joka viikko. (Preston, Shackelford 1999) Helpotusta ongelmaan tuo ohjelmointitehtävien lähetyksen ja arvioinnin automatisoiminen joko kokonaan tai osittain. (Joy, Luck 1999a)

Automaattisessa ohjelmakoodin tarkastamisessa pitää ottaa huomioon useita seikkoja. Ensinnäkin harjoitukset pitää spesifioida hyvin ja ne on ratkaistava etukäteen, jotta voidaan varmistaa, että automaattinen arviointi on yleensäkään mahdollista. Toiseksi virheilmoituksien merkitys on muistettava ja niistä on tehtävä mahdollisimman informatiivisia. (Korhonen, Malmi, Saikkonen 2001)

Jackson ja Usher (1997) ovat sitä mieltä, että ohjelman oikeellisuuden virheetön arviointi ei ole mahdollista pelkästään tutkimalla lähdekoodia, varsinkaan jos kyseessä on suuri ohjelma. Kaikki

arvioinnin osat eivät edes sovellu tietokoneella tehtäviksi, kuten vaikkapa kommentoimisen ymmärtäminen. Joyn ja Luckin (1999a) mukaan testidatalla ajaminen on käytännöllinen tapa ohjelman oikeellisuuden arvioimiseksi, mutta usealla testidatalla ajaminen vie paljon aikaa käsin tehtynä. Opiskelijoita voidaan vaatia tekemään testaus itse, mutta käytännössä heillä ei välttämättä ole riittävästi taitoa tehdä sitä. Lisäksi opiskelijoiden on helppo muokata testituloksia haluamakseen ja harhauttaa opettajaa. Automatisoimalla ohjelmatehtävien lähetys ja arviointi edellä mainitut ongelmat voidaan osittain kiertää.

Ennen automaattisia arviointijärjestelmiä opiskelijoiden tekemät tehtävät tulostettiin paperille, josta ne arvioitiin käsin. Tässä tavassa oli useita ongelmia: opiskelijat pystyivät räätälöimään ohjelmansa testidatalle sopiviksi yleisemmin toimivien ohjelmien kehittämisen sijaan, koska testausta vaadittiin vain pienellä tietomäärällä. Opiskelijat saattoivat myös unohtaa testaustulosten palauttamisen (Joy, Luck 1998a). Lisäksi suurilla opiskelijamäärillä tarkastajan käsittelemä paperimäärä oli epämiellyttävän suuri, osa virheistä jäi huomaamatta, koska ohjelmia ei oikeasti ajettu ja tarkastajan aikaa kului prosessissa kohtuuttoman paljon. Arvioinnin yhtenäisyys saattoi myös kärsiä useiden tarkastajien takia. (Foxley, Zin 1994).

Joy ja Luck (1995) määrittelevät hyvälle automaattiselle ohjelmointitehtävien tarkastusohjelmalle tiettyjä kriteerejä. Ensinnäkin automaattisen tarkastusohjelman pitäisi huolehtia riittävästä tietoturvallisuudesta minimoimalla mahdolliset riskit. Järjestelmän tulisi myös kopioida opiskelijan lähdekoodi lähetysajalla ja -päivämäärällä varustettuna paikkaan, johon vain tarkastajalla on oikeus päästä käsiksi (Joy, Luck 1999a).

Toiseksi, koska opiskelijat käyttävät paljon aikaa ja vaivaa ohjelmointitehtävien tekemiseen, he ansaitsevat perinpohjaisen ja täsmällisen tehtävien arvioinnin, joka annetaan nopeasti ja joka tarjoaa hyödyllistä palautetta. Järjestelmän tulisi olla myös helppokäyttöinen ja riittävän joustava, jotta sitä voitaisiin käyttää erilaisilla kursseilla ja useilla eri ohjelmointikielillä. (Joy, Luck 1995)

Automaattisilla ohjelmakoodin tarkastusjärjestelmillä on useita hyötyjä, joista osa on samoja kuin jo edellä mainitun tietokoneavusteisen arvioinnin edut. Joyn ja Luckin (1995) mukaan suuria opiskelijamääriä voidaan käsitellä tehokkaasti automaattisen tarkastusjärjestelmän avulla. Lisäksi

tehtävien lähetyksen turvallisuus voidaan varmistaa, lähetettävien ohjelmien laiton kopioiminen voidaan estää erilaisilla suojauksilla ja lähetysten häviämisen riski tai vahingossa poistaminen on pienempi.

Ohjelmien kokoaminen arviointia varten voidaan tehdä tehokkaasti ja papereiden käsittelyä ei tarvita. Arvioinnin ja testauksen tarkkuus kasvaa, jonka seurauksena opiskelijoiden luottamus arviointia kohden paranee. Arvioinnin jatkuvuus paranee ja sen lisäksi voidaan kerätä monenlaisia tilastotietoja ja tehdä erillisiä tarkistuksia, kuten plagioinnin tarkistus (Joy, Luck 1995). Automaattinen ohjelmien tarkastus parantaa myös opiskelijoille annettavaa palautetta ja mahdollistaa yksilöllisten tehtävien antamisen yksittäisille opiskelijoille. (Jones 2001)

2.4 Palaute

Petren ja Pricen (1997) mukaan opettajan opiskelijoille antama palaute on yksi ratkaiseva vuorovaikutuksen aspekti. Selkeän palautteen antaminen opiskelijoille muuttaa tarkastusmekanismin opetustyökaluksi. Mitä vähemmän opiskelijalla on mahdollisuutta osallistua kurssimateriaalista käytävään keskusteluun, sitä tärkeämpää hänelle on yksityiskohtainen palaute tekemistään töistä.

Ohjelmointikursseilla opettajan antama palaute opiskelijan tekemistä kurssitöistä voi olla esimerkiksi merkintätavan korjaamista, vaihtoehtoisen koodin esittämistä tai kommentteja ohjelman rakenteesta. Välitön palaute rohkaisee opiskelijaa yrittämään ongelman ratkaisua ja saa hänet tuntemaan tyytyväisyyttä onnistumisen jälkeen. Jokainen ongelma rakentuu yleensä aikaisempien tehtävien päälle ja opiskelija voi näin vähitellen kasvattaa luottamustaan ohjelmointitaitoihinsa. Käsin arvosteltuna olisi miltei mahdotonta arvioida yhtä paljon tehtäviä ja antaa kaikista palautetta kuin automaattista järjestelmää käytettäessä. (Daly 1999)

2.4.1 Palautteen tarkoitus opiskelun kannalta

Kasvokkain annettu yksilöllinen palaute on mahdollista korvata ainakin osittain järjestelmän automaattisesti antamalla palautteella. Odekirk-Hash ja Zachary kehittivät Utahin yliopistossa tutorointijärjestelmän, jonka avulla he tutkivat, voiko automaattinen järjestelmä tarjota tehokasta ja välitöntä palautetta aloittelevalle ohjelmoinnin opiskelijalle. Tutorointijärjestelmän nimeksi tuli *InStep* (Independent Student Tutoring by Examing Programs). Saatujen tuloksien mukaan opiskelijat, jotka saavat palautetta suoraan järjestelmältä, tarvitsevat vähemmän opettajan antamaa palautetta kuin ne opiskelijat, jotka eivät saa järjestelmän palautetta ohjelmistaan. InStep-järjestelmästä palautetta saaneet opiskelijat tarvitsivat vain noin kolmasosan muiden opiskelijoiden vaatimasta palautteesta. (Odekirk-Hash, Zachary 2001)

Tehtävät ovat opiskelijalle arvokkaita tilaisuuksia erityiseen vuorovaikutukseen opettajan kanssa varsinkin etäopiskelussa, jossa vuorovaikutus opettajan ja muiden opiskelijoiden kanssa jää yleensä vähäisemmäksi kuin lähiopetuksessa. Etäopetuksen kulttuuri painottaa rohkaisemista ja opettajat antavatkin yleensä yksityiskohtaisempaa palautetta etäopiskelijoille. Elektroninen tehtävien käsittely antaa lisää mahdollisuuksia havaintoesityksille ja toteuttamiselle, joilla on erityistä arvoa ohjelmoinnin opetuksessa. (Petre, Price 1997)

2.4.2 Opiskelijoiden antama palaute

Haganin ja Lowderin (1999) mukaan palaute voi kulkea myös opiskelijoilta opettajille ja takaisin. Monashin yliopistossa tietojenkäsittelytieteen laitoksella on käytössä web-pohjainen anonyymi palautejärjestelmä, jota on käytetty antamaan tietojenkäsittelytieteen laitoksen opiskelijoille tapa saada äänensä kuuluviin ja mahdollisuus vaikuttaa omaan oppimiseensa. Palautejärjestelmä kehitettiin vuonna 1996 tietojenkäsittelytieteen ja kasvatustieteen laitosten yhteisessä projektissa, jonka päämääränä oli parantaa ensimmäisen vuoden tietojenkäsittelytieteen opetuksen laatua kartoitustutkimuksien avulla.

Alussa palautejärjestelmä oli yksinkertainen ja sitä oli tarkoitus käyttää pelkästään opiskelijoiden mieliä painavien asioiden havaitsemiseksi. Kun järjestelmän välityksellä saatiin paljon arvokasta tietoa, se otettiin lopullisesti yliopiston käyttöön. Nykyään palautejärjestelmää on kehitetty edelleen ja se muistuttaa uutisryhmää, jossa opiskelijat voivat halutessaan osallistua keskusteluun myös anonyymisti. Järjestelmä sisältää välineet lähetyksien poistamiseen, henkilökunnan vastauksien lähettämiseen ja tilastotietojen keräämiseen. (Hagan, Lowder 1999)

Palautejärjestelmässä keskustelua on käyty opetettavien aineiden sisällöstä, kuten esimerkiksi ohjelmointiongelmista ja yleensäkin opiskelijoiden kohtaamista opiskeluun liittyvistä ongelmista. Vaikka nämä yleiset opiskeluun liittyvät kysymykset eivät suoranaisesti liitykään opetettavaan sisältöön, ne askarruttavat opiskelijoita ja niistä keskusteleminen auttaa heitä keskittymään itse aiheeseen. Lisäksi nämä kysymykset kasvattavat mielenkiintoa järjestelmässä käytyä keskustelua kohtaan ja näyttävät opiskelijoille, että opettajat ottavat vakavasti opiskelijoiden ongelmat ja ovat valmiita keskustelemaan kaikesta. Tämä saa opiskelijat tuntemaan aiheen omakseen ja lisää heidän opiskelumotivaatiotaan. (Hagan, Lowder 1999)

Opettajan kritisoiminen voi olla vaikeaa monille opiskelijoille, varsinkin jos heidät voidaan tunnistaa. Kuitenkin laadukkaan opetuksen tarjoamiseksi on tärkeää, että opiskelijat voivat vaikuttaa omaan oppimiseensa ja että he tuntevat aineen omakseen. Opiskelijoita askarruttavien asioiden käsitteleminen julkisesti ja opettajien nopea vastaaminen kysymyksiin auttavat näissä pyrkimyksissä. (Hagan, Lowder 1999)

Hagan ja Lowder (1999) painottavat, että myös opettajille on hyötyä opiskelijoiden antamasta palautteesta. Opettajat tarvitsevat jatkuvaa palautetta siitä, kuinka opiskelijat pärjäävät, siitä mitä ongelmia he kohtaavat ja siitä, mitä ongelmia heillä on oppimisympäristön kanssa. Palaute-järjestelmän avulla Monashin yliopistossa on huomattu monia ongelmia ja sen välityksellä on kysytty monia kysymyksiä, jotka muuten olisivat jääneet kysymättä. Näin järjestelmä on rohkaissut opiskelijoita myös oppimaan lisää.

2.5 Plagiointi

Foxley (1997) määrittelee plagioinnin tarkoittamaan jonkun toisen työn kopioimista ja sen esittämistä itse tehtynä, toisin sanoen se on dokumentin tai ohjelman tunnustamatonta kopiointia. Plagiointina pidetään tekstikirjan kappaleiden, kohtien tai ohjelmien kopioimista, toisen tekemän työn kopioimista toisen luvalla tai ilman lupaa ja ohjelman tai esseen ryhmässä tekemistä niin, että kukin ryhmän jäsen lähettää tuotoksen omana työnään.

Monissa automaattisissa koodin tarkastusjärjestelmässä plagioinnin tarkastaminen on oleellinen osa opiskelijoiden töiden tarkastamista. Wisen (1992) mukaan plagiointi on todellinen ongelma yliopistoissa, sillä se vähentää luottamusta akateemiseen rehellisyyteen ja kotitehtävien käyttämistä arvioinnin osana. Lisäksi tehtävien kopiointi pilaa niiden opetukselliset tarkoitukset. Toisaalta tietokoneavusteinen plagioinnin tarkistaminen voi myös palauttaa luottamuksen tietokonepohjaisiin tehtäviin, koska tietokoneita voidaan käyttää plagioinnin tehokkaampaan etsimiseen. Plagioinnin tarkistamisesta opiskelijoiden palauttamista töistä onkin tulossa rutiiniosa arviointia.

Plagiointia tapahtuu monessa eri kontekstissa, kuten yrityksissä ja yliopistoissa. Myös opiskelijat voivat yrittää parantaa arvosanojaan plagioimalla toisten töitä. Yleensä he eivät kuitenkaan saa siitä mitään taloudellista hyötyä tai saavuta suurta ajansäästöä. Kopioinnin salaamiseen käytetyt metodit ovat tavallisesti yksinkertaisia ja plagioinnin havaitsemiseen riittääkin monesti melko vaatimaton ohjelmisto. (Joy, Luck 1999b)

2.5.1 Koodin plagiointi

Opiskelijoiden töiden plagiointiin on monia syitä. Ensinnäkin, heikko opiskelija voi tehdä työnsä yhteistyössä toisen kanssa siinä uskossa, että se on hyväksyttävää. Tässä tapauksessa opiskelija toimii kuitenkin harmaalla alueella, hyväksyttävyyden rajamailla. Toisaalta opiskelija voi kopioida ja editoida toisen opiskelijan työtä joko toisen tietämättä tai toisen luvalla. Huonona

puolena opiskelijan kannalta on, että hänelle saa todennäköisesti vain heikon ymmärryksen ohjelmasta. Lisäksi palautetut ohjelmat ovat melko samanlaisia. (Joy, Luck 1999b)

Kolmannessa tapauksessa heikosti motivoitunut, muttei välttämättä heikko opiskelija, kopioi ja editoi toisen opiskelijan ohjelman, vähentääkseen omaa työmääräänsä. Tässä tapauksessa opiskelijalla voi olla hyvä tietämys aiheesta ja hän voi osata tehdä hienostuneita muutoksia ohjelmaan, jonka vuoksi plagiointi voi olla vaikeasti havaittavissa. Onkin hyvä muistaa, että plagiointi voi aina jäädä huomaamatta, vaikka käytössä olisi kuinka hyvät välineet. (Joy, Luck 1999b)

2.5.2 Koodin plagiointitekniikoita

Ohjelmointikielien määrä kasvaa jatkuvasti ja siksi Joyn ja Luckin (1999b) mukaan ei olekaan mahdollista luokitella kaikkia mahdollisia metodeja, joilla ohjelma voidaan muokata erilaiseksi samalla tavalla toimivaksi ohjelmaksi. Kuitenkin joitakin yleisesti käytettyjä tekniikoita voidaan tunnistaa. *Sanastomuutokset* (lexical changes) tarkoittavat muutoksia, jotka voitaisiin tehdä myös tekstieditorilla. Niiden tekeminen ei vaadi ohjelman jäsentämisen vaatimaa kielen tuntemusta. Sanastomuutoksia ovat muun muassa kommenttien editointi, kuten lisääminen, muuttaminen ja pois jättäminen, muotoilun muuttaminen, muuttujien nimien vaihtaminen tai rivinumeroiden muuttaminen sellaisissa kielissä, joissa niitä käytetään (esimerkiksi FORTRAN).

Rakenteelliset muutokset (structural changes) vaativat sellaista tietämystä ohjelmasta, jota tarvitaan sen jäsentämiseen. Nämä muutokset ovat hyvin kieliriippuvaisia. Rakenteellisia muutoksia ovat muun muassa silmukoiden korvaaminen toisenlaisella silmukalla, esimerkiksi Pascal-kielessä while-silmukka voidaan korvata until-silmukalla. Sisäkkäiset if-lauseet voidaan korvata case-lausekkeilla tai toisin päin. Lauseiden järjestystä voidaan myös muuttaa, kun ottaa huomioon etteivät muutokset vaikuta ohjelman toimintaan. Proseduurin ja funktion kutsut voidaan myös korvata toisillaan. Proseduurin kutsut voidaan myös korvata proseduurin rungon kopiolla. Lisäksi operandien järjestystä voidaan vaihtaa. (Joy, Luck 1999b)

2.5.3 Plagioinnin estäminen ja tarkastaminen

Plagioinnin estäminen ennalta on parempi vaihtoehto kuin sen tarkistaminen jälkikäteen, sanovat Culwin ja Lancaster (2001b). Opettaja voi suunnitella tehtävät siten, että ne ovat vähemmän alttiita plagioinnille, vaikka niin ei voidakaan poistaa plagioinnin tarkastusjärjestelmän tarvetta. Plagiointia voidaan estää esimerkiksi välttämällä tehtävien uudelleenkäyttämistä tai antamalla tehtäväksi niin harvinainen tai uusi aihe, että siitä ei todennäköisesti löydy valmista materiaalia. Lisäksi ennalta valmisteltujen arvioitavien tehtävien tekeminen luokassa tekee plagioinnista vaikeampaa.

Opiskelijamäärien kasvaessa plagioinnin havaitsemisesta on tullut vaikeampaa ja opiskelijoiden on hyvin helppoa kopioida ohjelmakoodia toisiltaan. Opiskelijoille on muistutettava plagioinnin rangaistavuudesta ja heille on ilmoitettava, että heidän palauttamansa työt tarkistetaan. Plagioinnin tarkastajan on pystyttävä myös todistamaan, etteivät ohjelmien samanlaisuudet ole pelkkää sattumaa. Suurin osa opiskelijoista, jotka plagioivat toisten töitä tekevät sen, koska he eivät täysin ymmärrä miten ohjelmoidaan. Heidän käyttämänsä plagiointimenetelmät ovat niin selkeitä, että he myöntävät tekonsa heti, kun asia on huomattu. (Joy, Luck 1999b)

Plagioinnin tarkastus-ohjelmia on ollut olemassa jo yli parin kymmenen vuoden ajan. Ensimmäiset järjestelmät perustuivat *attribuuttien laskentaan* (attribute counting) ja ne laskivat tarkistettavan ohjelman lähdekoodista useita pinnallisia metriikoita. Tähän metodiin perustuva järjestelmä on esimerkiksi *Halstedin ohjelmistometriikka* (Halsted's software science metrics). (Culwin, MacLeod, Lancaster 2001)

Toinen, parempi tapa etsiä plagiointia ohjelmakoodista, on tutkia ohjelman rakennetta, jossa jokainen lähetys pelkistetään *merkkien* (token) tai tunnisteiden sarjoiksi. Merkit tai tunnisteet edustavat esimerkiksi funktion kutsua tai muuttujan määrittelyä (Culwin, MacLeod, Lancaster 2001). Tämäntyyppisiä järjestelmiä ovat muun muassa Plague ja Yap. (Wise, 1992)

2.5.4 Plagiointi esseevastauksissa

Plagiointia voidaan tarkastaa ohjelmakoodin lisäksi esseevastauksista. Tällöin yleensä tutkitaan, onko materiaali tai osa siitä kopioitu Internetistä (Culwin, Lancaster 2001b). Turnitin.com on yksi käytetyimmistä kaupallisista plagioinnin etsimispalveluista. Se skannaa opiskelijan paperin tietokoneelle, jotta opettaja voi tarkastaa, onko materiaali kopioitu Internetistä tai muista palvelun tietokannassa olevista papereista. (Young, 2001). Muita vastaavia maksullisia järjestelmiä ovat muun muassa Plagiarism.org, Paperbin ja Copycatch.com. Ilmaisia palveluja tarjoavat muun muassa FindSame, HowOriginal ja Plagiserve, joka vaatii rekisteröinnin ennen käyttöä. (Culwin, Lancaster 2001b)

Viime aikoina on alettu kehittää myös *laitoksen sisäistä plagiointia* (intra-corpal plagiarism) tarkastavia järjestelmiä. Laitoksen sisäisellä plagioinnilla tarkoitetaan esimerkiksi luokan oppilaiden välillä tapahtuvaa plagiointia tai luvatonta yhteistyötä. *Samanlaisuuden visualisointi ja analysointi -järjestelmä* VAST (Visualisation and Analysis of Similarity Tool) on järjestelmä, joka etsii plagiointia analysoimalla esseiden samankaltaisuutta. Järjestelmä tekee tuloksista *graafiset kuvat* (similarity visualisations), joista tutorit tarkastavat mahdolliset plagiointitapaukset manuaalisesti. (Culwin, Lancaster 2001a)

2.5.5 Plagioinnin tarkastamisen ongelmia

Plagiointitarkistuksen tekeminen on erityisen tärkeää silloin, kun käytetään järjestelmää, joka tarkastaa opiskelijoiden tehtävät täysin automaattisesti. Varsinkin ohjelmoinnin peruskursseilla opiskelijoiden tekemät ohjelmat ovat kuitenkin niin lyhyitä ja samanlaisia, ettei niistä ole helppoa tarkastaa plagiointia automaattisesti. Lisäksi tehtävän runko voidaan antaa opiskelijoille valmiina joillakin kursseilla. Tällöin on hyödyllistä, että järjestelmä vertaa kaikkia palautettuja tehtäviä ja etsii niistä sellaisia opiskelijapareja, joiden ratkaisut ovat eniten samanlaisia. Opettaja voi sitten tarkistaa epäilyttävät parit käsin. (Korhonen, Malmi, Saikkonen 2001)

Joidenkin professoreiden mielestä rutiininomaiset plagiointitarkastukset ruokkivat epäluottamuksen ilmapiiriä yliopistoissa. Youngin (2001) mukaan joissakin korkeakouluissa on myös luovuttu sopimuksista Turnitin.comin kanssa, koska niissä ei ole ollut riittävästi kiinnostusta plagioinnin tarkastamiseen. Osassa korkeakouluja ollaan huolestuneita siitä, että Turnitin.com rohkaisee professoreita lähettämään jokaisen opiskelijan paperit sen palvelun kautta.

3. AUTOMAATTISEEN KOODIN TARKASTUKSEEN SUUNNITELTUJA VÄLINEITÄ

Luvussa esitellään automaattista koodin tarkastusta ja koodin tarkastuksessa käytettäviä järjestelmiä. Erityisesti keskitytään laajasti käytössä olevien Boss- ja Ceilidh -järjestelmien kuvailuun. Luvussa kerrotaan myös muista käytössä olevista järjestelmistä. Lopussa vertailen toisiinsa joitakin automaattisille koodin tarkastusjärjestelmille ominaisia piirteitä.

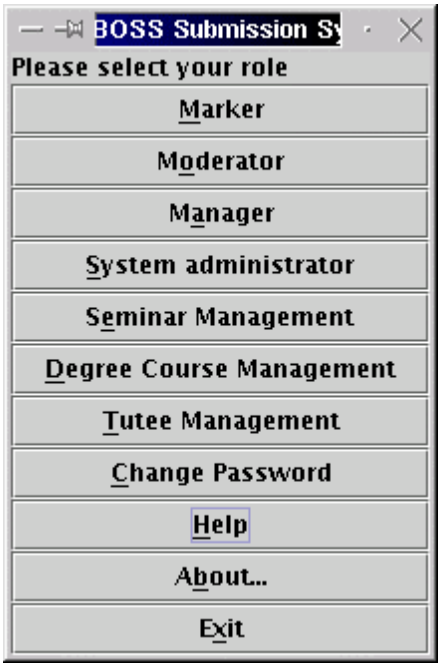
Automaattiset lähetys- ja tarkastus-järjestelmät voidaan jakaa *puoli- ja täysin automaattisiin järjestelmiin*. Puoliautomaattisella arviointijärjestelmällä tarkoitetaan sellaista tehtävien tarkastamiseen käytettävää järjestelmää, jossa tehtävien tarkastaminen ja arvosanojen antaminen ei tapahdu kokonaan järjestelmän toimesta, vaan apuna tarvitaan myös ihmistä. Täysin automaattinen järjestelmä tarkastaa ja antaa arvosanat opiskelijoiden palauttamista töistä täysin ilman ihmistä. Myös täysin automaattisessa järjestelmässä tarvitaan ihmistä määrittelemään tarkastusprosessia.

3.1 Boss

Boss-järjestelmä on opiskelijoiden ohjelmointitehtävien lähetys- ja on-line tarkastusjärjestelmä, jonka tavoitteena on helpottaa kurssin hallintaa. Bossin uudemmat versiot sisältävät elektronisia tarkastusohjelmia, jotka yhdistävät ohjelmatehtävien automaattisen testauksen ja tarkastuksen, sekä käsin tehtävän tarkastuksen turvallisessa ympäristössä. Boss-järjestelmä ei sisällä kurssin opetusmateriaalia, vaan se pidetään järjestelmästä täysin erillisenä. (Joy, Luck 1998a)

Boss toteutettiin tekstipohjaisena Sun Solaris -käyttöjärjestelmälle vuonna 1993. Myöhemmin järjestelmään tehtiin tuki graafiselle käyttöliittymälle ja suora yhteys yliopiston opiskelijatietojen tietokantaan. Vuonna 1999 Boss toteutettiin kokonaan uudelleen. Ohjelmistoon oli tehty lisäyksiä eri ohjelmointikielillä (C-, Perl- ja Tcl/Tk -kielillä), mistä seurasi ongelmia ohjelmiston ylläpidossa ja kehittämisessä. Lisäksi sen siirrettävyys oli huono ja mahdollisuus verkkotyöskentelyyn puuttui. Myös vuonna 2000 ohjelmasta tuli uusi versio, jota on testattu suurilla opiskelijaryhmillä. Nyt käytössä oleva versio Boss 2 valmistui vuonna 2001 avoimen

lähdekoodin periaatteella. (Boss 2002) Järjestelmästä on valmistumassa taas uudempi versio syyskuussa 2002 (Joy 2002).



Kuva 1: Boss-järjestelmän henkilökunnan pääikkuna.

Kuvassa 1 on Boss-järjestelmän henkilökunnan pääikkuna, josta käyttäjät pääsevät omiin ikkunoihinsa. Kuvassa näkyvät myös järjestelmään lisätyt *seminaarin varaus* -moduuli (Seminar Management), jossa opettaja voi varata ajan tietylle ryhmälle tapaamista tai seminaaria varten ja *kurssin hallinta -moduuli* (Degree Course Management), jossa käyttäjä voi esimerkiksi nähdä kaikki tietyn moduulin arvosanat. *Tutorointi-moduuli* (Tutee Management) on edellisen moduulin rajoitetumpi versio (Boss 2002). Edellä mainittuja moduuleita ei käsitellä tässä tutkielmassa, eivätkä ne olleet kokeiltavissa testattavassa Boss-järjestelmän versiossa.

3.1.1 Toteutus

Boss 2-järjestelmä on toteutettu Javalla ja se noudattaa täysin Java-standardia. Boss-ohjelmisto on alustasta riippumaton ja sitä on testattu Sun Solariksella, Linuxilla ja Windows 95/98/NT/2000 -käyttöjärjestelmillä. Boss on niin sanottu asiakas-palvelin -järjestelmä, jossa

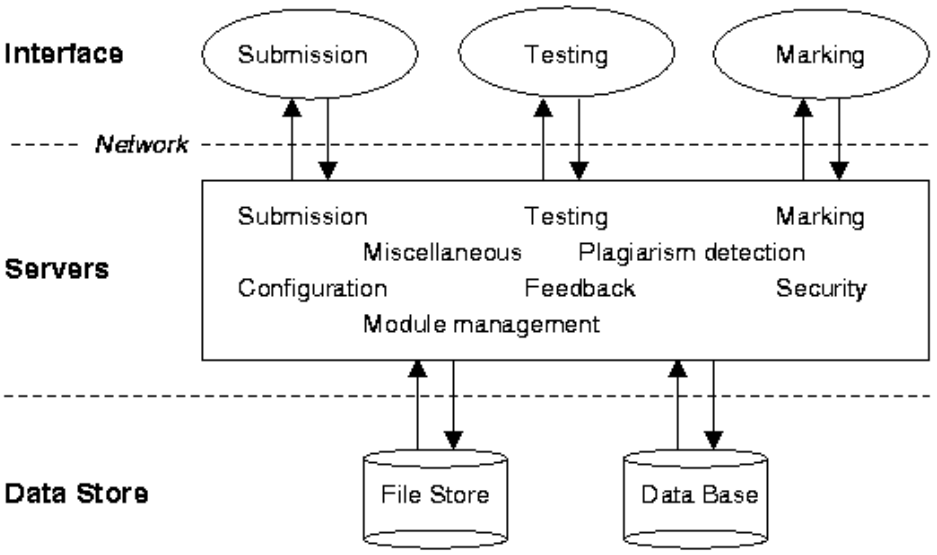
asiakaskoneelle asennetaan pieni esilatausohjelma, joka lataa asiakasohjelmiston keskuspalvelimelta ja tekee siten ohjelmiston päivityksen helpommaksi. (Boss 2002)

Tietoturva on tärkeää, jotta tiedostoja voivat lukea vain ne henkilöt, joilla on siihen oikeus; toisaalta se on myös keino suojautua viruksilta (Joy, Luck 1998a). Boss-ohjelmistossa tietoturva hoidetaan käyttämällä *SSL-salausta* (Secure Sockets Layer), joka hoitaa kommunikoinnin asiakkaiden ja palvelimien välillä. Käytössä ovat myös salasanat ja rajoitetut oikeudet. Boss-ohjelmistoon pääsee kirjautumaan sisään vain salasanalla, joka on tallennettu omaan salasanatietokantaan erilleen muista järjestelmistä. Alussa käyttäjälle arvotaan satunnainen salasana, joka lähetetään hänelle sähköpostin välityksellä. (Boss 2002)

3.1.2 Arkkitehtuuri

Boss on kolmikerrosarkkitehtuuri-järjestelmä, jonka osat ovat käyttöliittymä, palvelimet ja tietovarasto. Käyttöliittymä koostuu käyttäjien liittymistä, joita ovat 1) lähetysliittymä (submission interface), jota käyttävät opiskelijat, 2) tarkastusliittymä (marking interface), jota käyttää henkilöstö ja 3) testausliittymä (testing interface), jota ajetaan erillisenä. Nämä liittymät kommunikoivat kolmen palvelimen kanssa, joilla on pääsy opiskelijoiden lähettämiin tiedostoihin ja tietokantatauluihin, esimerkiksi arvosanoihin. Järjestelmässä tarvittava tieto, kuten esimerkiksi käyttäjien tiedot, on tallennettu boss-nimiseen MySql-tietokantaan ja opiskelijoiden lähettämät tehtävät arkistoidaan ZIP-muodossa. (Boss 2002)

Lähetys ja arviointi-asiakkaat on toteutettu esilatausohjelmina, jotka lataavat varsinaisen ohjelman palvelimelta ja suorittavat sen. Kommunikointi liittymien ja palvelimien välillä tapahtuu *RMI*:ä (Remote Method Invocation) käyttäen TCP/IP-verkossa. Slave-palvelimen kanssa kommunikoivan testausliittymän tarkoitus on ajaa automaattiset testit mahdollisimman turvallisessa ympäristössä, jotta vältytään mahdollisilta riskeiltä, kuten tiedon vahingoittumiselta. Client- ja Staff -palvelimet kommunikoivat lähetys- ja arviointi -asiakkaiden kanssa. Palvelimet tarjoavat Java-luokat ja -metodit, joita liittymät tarvitsevat, mutta sisäisesti ne jakavat koodin. Kuva 2 esittää Boss-järjestelmän arkkitehtuuria. (Boss 2002)



Kuva 2: Boss-järjestelmän arkkitehtuuri. (Boss 2002)

3.1.3 Tiedonhallinta

Suuri osa järjestelmästä liittyy jollakin tavalla tiedonhallintaan. Opiskelijoiden lähettämät tehtävät ja dokumentit on tallennettava, jotta niitä voidaan myöhemmin käsitellä, kääntää ja testata. Tallennus on tehtävä turvallisesti, jotta tiedot on saatavissa vain arviointia varten. Boss-järjestelmä sisältää myös auditointitiedoston, johon on tallennettu jokainen lähetys tai lähetyksen yritys, jotta tapahtumat voidaan jäljittää ongelmatilanteissa. (Joy, Luck 1999a)

Tallennettu data on rakennettu käyttäjäkeskeisten käsitteiden ympärille. Boss-järjestelmän sisällä *kurssilla* (course) tarkoitetaan koulutusohjelmaan kuuluvaa opintojaksoa, kuten esimerkiksi johdatuskurssia ohjelmointiin. *Harjoitus* (assignment) on työ, joka muodostaa osan kurssin saavutuksista, esimerkiksi tentti. Harjoitukselle on yleensä annettu jokin määräaika ja se sisältää yhden tai useamman *tehtävän* (problem). (Joy, Luck 1999a)

3.1.4 Käyttäjät

Boss-järjestelmän sisällä käyttäjät jaetaan viiteen eri kategoriaan. Kaikkien käyttäjien täytyy olla kyseessä olevan laitoksen jäseniä eli heillä pitää olla nimen lisäksi henkilökohtainen *tunniste-numero* (university ID). *Opiskelija* on henkilö, joka normaalisti on rekisteröitynyt kurssille ja yhdelle tai useammalle moduulille. Opiskelija voi lähettää tiedostoja ja ajaa hänelle osoitettuja automaattisia testejä. (Boss 2002)

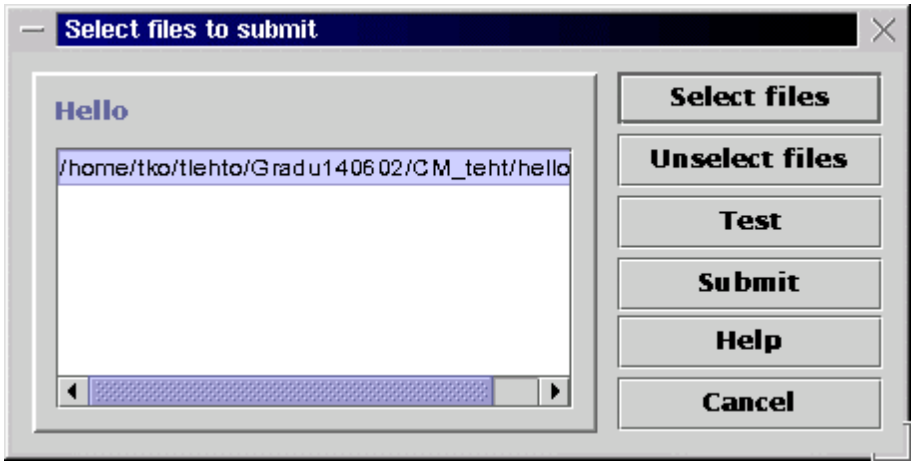
Pääkäyttäjä (administrator) on henkilökunnan jäsen, joka on vastuussa Boss-järjestelmän hallinnoimisesta. Pääkäyttäjällä on globaalit oikeudet koko järjestelmään ja hänen erikoistehtävänään on konfiguroida uudet Boss-järjestelmän moduulit, ennen kuin moduulin hoitaja lisää niihin dataa. (Boss 2002)

Varsinaisesta tarkastusprosessista vastaavia rooleja ovat *moduulin hoitaja, tarkastaja* ja *moderaattori*. *Moduulin hoitaja* (manager) on henkilökunnan jäsen, yleensä luennoitsija, joka on vastuussa yksittäisestä moduulista. Moduulin hoitajalla on oikeudet kaikkeen moduulin tietoon ja monia hallinnollisia työkaluja, hän esimerkiksi valitsee tehtävien määräajat ja ajaa plagiointitestaukset. Harjoitusten tarkastamisesta on vastuussa *tarkastaja* (marker). Hän antaa arvosanat moduulin hoitajan määrittelemien kriteerien mukaan eikä tiedä opiskelijan identiteettiä tai tarkastuskriteerien suhteellisia painotuksia. *Moderaattori* (moderator) on henkilö, joka on vastuussa arvosanojen viimeistelystä. Moderaattori voi olla sama henkilö kuin moduulin hoitaja. (Boss 2002)

3.1.5 Opiskelijaliittymä

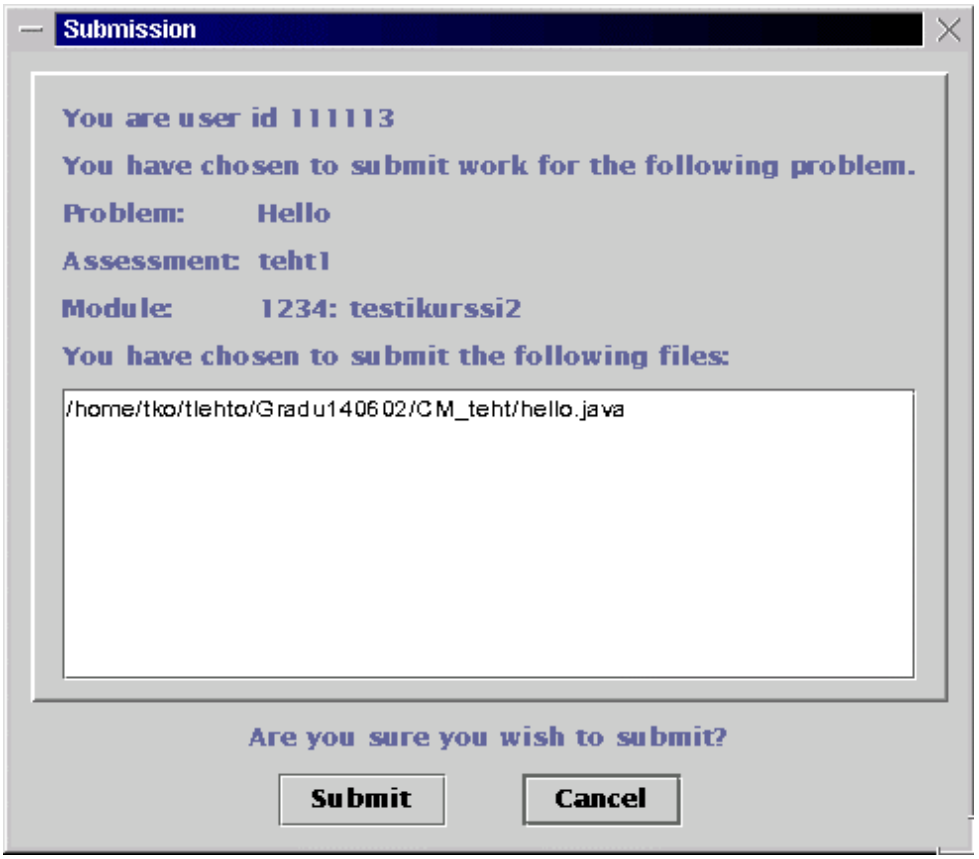
Opiskelijan käyttöliittymä on helppokäyttöinen ja yksinkertaisen näköinen. Sisäänkirjautumis-ikkunan *New Password* -painikkeesta opiskelija kirjautuu sisään ensimmäisen kerran tai saa uuden salasanan, jos hän on unohtanut entisen. Boss-järjestelmä lähettää uuden salasanan opiskelijalle automaattisesti sähköpostissa ja opiskelija voi vaihtaa sen myöhemmin haluamakseen (katso liite 6).

Opiskelijat voivat lähettää ohjelmatehtävänsä verkon välityksellä valitsemalla oikean kysymyksen *Select a problem* -ikkunassa ja lähetettävät tiedostot *Select files to submit* -ikkunassa. Opiskelijat voivat myös testata ohjelmaansa yhdellä testidatalla, jota myös järjestelmä käyttää ongelman tarkastamisessa. Näin varmistetaan, että ohjelma toimii testattaessa samoin kuin opiskelijalle ja opiskelija saa varmuuden siitä, että hänen lähettämänsä ohjelma täyttää asetetut minimivaatimukset (Joy, Luck 1998a). Kuvassa 3 on ikkuna, jossa opiskelija voi testata valitsemansa tehtävän painamalla *Test*-painiketta tai lähettää tehtävän napsauttamalla *Submit*-painiketta.



Kuva 3: Opiskelijan tehtävän lähetys -ikkuna.

Lähetyksen jälkeen opiskelija Boss näyttää opiskelijalle ikkunan (kuvassa 4), jossa varmistetaan vastauksen lähettäminen ja jossa kerrotaan lähetyksen tiedot. Opiskelija voi tässä vaiheessa tarkastaa tiedot oikeaksi ennen lähetystä. Lisäksi opiskelija saa lähetyksensä vastaanotosta *vastaanottotodistuksen* (receipt) sähköpostilla, muussa tapauksessa heitä on pyydetty ottamaan yhteyttä tarkastajaan. Todistus vastaanotosta sisältää koodin, jolla tiedosto voidaan tunnistaa (katso liite 6).



Kuva 4: Lähetystiedot-ikkuna.

3.1.6 Tehtävien tarkastus

Boss ei ole täysin automatisoitu tarkastusjärjestelmä, vaan tarkastajan vastuulla on määritellä niin sanottu *arvosanakaavio* (marking sheet), joka ottaa huomioon Boss-järjestelmän antamat arvosanat ja muut tärkeät osatekijät. Tarkastajan vastuulla on myös määritellä mitä tehdään, jos opiskelijan ohjelma ei läpäise yhtä tai useampia testejä, koska Boss ei osaa arvioida osittain toimivaa ohjelmaa. Ohjelman tarkoitus on vain avustaa tarkastajaa ja tehdä arvioinnista nopeampaa, tarkempaa ja yhtenäisempää. (Joy, Luck 1998a)

Elektronisia tarkastuslomakkeita käytetään Boss-järjestelmässä automaattisen tarkastuksen määrittelemiseen. Tarkastaja määrittelee elektroniselle lomakkeelle kategoriat ja niiden painotukset, joista järjestelmä antaa arvosanat automaattisesti. Tarkastaja määrittelee itse jäljelle

jäävät kategoriat liukukytkimen avulla. Kategorioita ovat kommentointi, algoritmit, sisennyksen jatkuvuus, yleinen tyyli, aliohjelmien käyttö ja testidatat. (Joy, Luck 1998a)

Elektroninen tarkastuslomake yhdistää arvosanat, jotka saadaan ohjelman testauksen ja ajamisen tuloksena, sekä arvosanat, jotka liittyvät ohjelman eri aspekteihin, kuten tyyliin. Boss-järjestelmä yhdistää automaattisten testien ja ohjelman tulosten perusteella saamansa arvosanat suoraan tarkastuslomakkeelle. Jos tulokset ovat oikein ja ohjelma läpäisee testin, se saa kyseessä olevasta kategoriasta täyden arvosanan. Jos ohjelma ei läpäise automaattista testiä, sille ei anneta mitään arvosanaa, mutta tarkastaja voi muuttaa automaattisesti annettua arvosanaa. Ongelmallisissa tapauksissa tarkastaja voi tutkia testauksen tuloksia tai ajaa testit manuaalisesti. Lopullinen arvosana muodostuu automaattisesti annettujen arvosanojen lisäksi tarkastajan käsin antamista arvosanoista ja niiden painotuksista, joten tarkastaja ei voi muuttaa lopullista arvosanaa käsin. (Joy, Luck 1998a)

Tarkastamisen jälkeen moderaattori viimeistelee tarkastajan arvioinnin. Moderointi-lomakkeella moderaattori näkee tarkastajien käyttäjätunnukset ja heidän antamansa arvosanat, sekä järjestelmän antamat arvosanat ja edellisistä arvosanoista automaattisesti muodostetun lopullisen arvosanan. Moderaattori voi säätää lopullista arvosanaa, ennen kuin hän vahvistaa sen. Lisäksi moderaattori voi lukea tarkastajan hänelle lisäämän palautteen ja editoida opiskelijalle lähetettävää palautetta. Kaikki kommentit ja arvosanat tallennetaan, jotta ne on helppo tarkistaa uudelleen ongelmatilanteissa. (Joy, Luck 1998a)

Tehtävän tarkastaminen tapahtuu anonyymisti, opiskelijan nimen sijasta käytetään hänen tunnistenumeroaan. Boss-järjestelmässä on myös lisäohjelma, joka vie arvosanat automaattisesti yliopiston tietokantaan, jossa lopulliset arvosanat yhdistetään opiskelijan nimeen. (Joy, Luck 1998a)

Palautteen antamista varten Boss-järjestelmässä on oma ohjelmansa, joka käynnistyy tarkastuslomakkeella olevasta painikkeesta. Sen avulla jokainen tarkastaja voi kommentoida lähetettyä ohjelmaa, joko moderaattorille tai palautteena opiskelijalle. Lopullista arvosanaa annettaessa moderaattori voi muokata opiskelijalle annettavaa palautetta, jonka järjestelmä laittaa

sähköpostilla suoraan opiskelijalle. Tarkastajan moderaattorille lähettämä palaute ei näy opiskelijalle. (Joy, Luck 1998a)

3.1.7 Plagiointi Boss-järjestelmässä

Boss-järjestelmä sisältää myös plagioinnin tarkastusohjelman. Tähän tarkoitukseen käytetään Sherlock-ohjelmaa. Ohjelma käyttää niin sanottua *muutoskopioinnin lähestymistapaa* (incremental comparison), joka perustuu ohjelmointitehtävien vertaamiseen pareittain. Ohjelmointitehtävät on lähetetty yhtenä tiedostona ja ne on tehty tietyllä ennalta määrätyllä kielellä. (Joy, Luck 1999a, Joy, Luck 1999b)

Kaikkia opiskelijoiden samaan harjoitukseen lähettämiä ohjelmia verrataan pareittain siten, että kahden eri opiskelijan lähettämää ohjelmointitehtävää verrataan viisi kertaa toisiinsa. Ohjelmia verrataan aluksi toisiinsa niiden alkuperäisessä muodossa, sitten maksimimäärä tyhjää tilaa poistettuna, kolmannella kerralla kaikki kommentit poistettuna, neljännellä kerralla tyhjät tilat ja kommentit poistettuna ja lopuksi vielä muunnettuna merkkitiedostoksi. (Joy, Luck 1999a)

Merkkitiedosto koostuu primitiivisistä kielen komponenteista, kuten nimistä, operaattoreista tai silmukoista, jotka soveltuvat käytettävään ohjelmointikieleen. Plagioinnin selvittämiseen käytettävät komponentit eivät ole välttämättä samoja kuin ne, joita on käytetty kielen oikean toteutuksen jäsentelyssä. Ohjelmaa ei myöskään tarvitse jäsentää niin tarkasti kuin kääntäjä tekee, järjestelmä toimii jopa yksinkertaisilla merkkivaihtoehdoilla, alkeellisella jäsentäjällä ja se on helposti päivitettävissä uudelle ohjelmointikielelle. (Joy, Luck 1999a, Joy, Luck 1999b)

Jos ohjelmaparit sisältävät samanlaisuuksia, on todennäköistä, että yksi tai useampi vertailuista osoittaa sen. Opettaja voi katsoa kokoelmaa lähetetyistä ohjelmista. Samankaltaisuuksia ja vastaavia koodin kohtia tutkimalla hänen pitäisi päästä alustavaan päätelmään siitä, ovatko samanlaisuudet sattumaa vai eivät. (Joy, Luck 1999a)

Sherlockin käyttö on Joyn ja Luckin (1999a) mukaan vähentänyt ohjelmointitehtävien plagiointitapauksia huomattavasti Warwickin yliopiston tietojenkäsittelytieteen laitoksella. Osa

plagiointitapauksista on mahdollisesti osattu piilottaa, mutta se on melko vaikeaa. Sherlock-ohjelmalla tarkastettuja ohjelmointitehtäviä oli vuonna 1994 yli 550 kappaletta, joista varmoja plagiointitapauksia oli noin 6,5 prosenttia. Kahdessa vuodessa plagiointitapaukset vähentyivät alle yhteen prosenttiin vuodessa. Vääriä plagiointisyytöksiä on ollut vain vähän.

3.2 Ceilidh

Ceilidh on kurssin hallintaan ja ohjelmointikurssien harjoitustöiden automaattiseen tarkastamiseen luotu järjestelmä. Sen on kehittänyt Learning Technology Research Group Nottinghamin yliopistossa. (Ceilidh 1999) Ceilidhin ensimmäinen versio julkaistiin vuonna 1988 ja siitä lähtien sitä on käytetty Nottinghamin yliopistossa kurssinhallintajärjestelmänä. Alussa Ceilihiä käytettiin C- ja C++ -kielien kursseilla, myöhemmin sille suunniteltiin kursseja viidelletoista eri ohjelmointikielelle, joita olivat muun muassa Pascal, Fortran, Modula2, Ada ja Prolog. (Foxley, Higgins, Gibbon 1996). Malmin mukaan (2002) yksi Ceilidhin suurimmista eduista olikin, että sillä voitiin tarkastaa periaatteessa millä tahansa käännettävällä kielellä kirjoitettuja ohjelmia, koska tarkastusprosessi ei ota kantaa käytettyyn ohjelmointikieleen.

Alkuperäinen Ceilidh-versio kirjoitettiin UNIX-käyttöjärjestelmälle C-kielellä ja sitä käytettiin pääasiassa arvosanojen merkintään (Foxley, Higgins, Gibbon 1996). Sen jälkeen Ceilidhiä laajennettiin eri ohjelmien ja työkalujen kokoelmalla, jotka oli kirjoitettu monilla eri kielillä, kuten esimerkiksi kuori- (shell), C- ja awk-kielillä. Ceilidhin versio 2 ilmestyi vuonna 1993, jolloin järjestelmä oli kehittynyt ja kasvanut ja siitä oli tullut vaikeasti päivitettävä ja laajennettava. Lopulta vuonna 1997 siitä tehtiin kokonaan uusi toteutus ja samalla Ceilidhin nimi muutettiin CourseMasteriksi (Foxley et al. 2001). Järjestelmästä on tehty myös WWW-pohjainen toteutus. (Foxley, 2000)

3.2.1 Tietoturva

Tietoturva on Foxleyn, Higginsin ja Tsintsifasin (2002) mukaan tärkeä osa Ceilidh-järjestelmää ja se liittyy useaan eri alueeseen järjestelmässä. Ensimmäinen tietoturvaan liittyvä alue on

tallennettava data. Ceilidh-järjestelmä säilyttää kopion opiskelijoiden lähettämästä materiaalista, kuten ohjelmatehtävistä. Jokaisesta tehtävästä viimeisin kopio ylikirjoittaa edellisen version, mutta tarkastuksen yksityiskohdat lisätään jo olemassa olevaan dataan. Tietojen säilyttämisestä on hyötyä esimerkiksi silloin, kun opiskelija kyseenalaistaa saamansa arvosanat.

Toinen alue on käyttäjien tiedon saanti. Jokaisella opiskelijalla täytyy olla pääsy julkiseen tietoon, jota ovat esimerkiksi kurssin muistiinpanot ja kysymykset. Lisäksi hänen pitää voida nähdä omat arvosanansa ja lähettämänsä ratkaisut. Toisaalta opiskelijat eivät saa päästä käsiksi kaikkeen tietoon, kuten esimerkiksi malliratkaisuihin, opettajan ja tutorin väliseen keskusteluun tai toisten opiskelijoiden lähettämiin tehtävän ratkaisuihin. (Foxley, Higgins, Tsintsifas 2002)

Kolmas tietoturvan alue on *jäljitysketjut* (audit trails). Järjestelmä pitää tallessa pääkäyttäjän määrittelemät jäljitysketjut. Jäljitysketjut voivat liittyä tiettyyn opiskelijaan, jonka kaikki toiminnot tallennetaan ajalla varustettuna. Vaihtoehtoisesti voidaan pitää kirjaa jostakin tietystä Ceilidhin toiminnosta, kuten opiskelijan ohjelman uudelleenkääntämisestä, muistiinpanojen tulostamisesta tai arvosanojen katsomisesta. Jäljitysketjut ovat vain henkilökunnan käytettävissä. (Foxley, Higgins, Tsintsifas 2002)

Neljäs tietoturvaan liittyvä seikka on opiskelijan tunnistaminen. Ceilidhissä opiskelija kirjautuu järjestelmään käyttämällä standardia UNIX-järjestelmän salasanaa. Jokaista komentoa suoritettaessa järjestelmä tunnistaa käyttäjän ja tarjoaa hänelle hänen oikeuksiensa mukaisia valikkoja, komentoja ja näppäimiä. Järjestelmä identifioi käyttäjät kurssikohtaisten käyttäjälistojen avulla. Listoissa on käyttäjien käyttäjätunnukset ja mahdollisesti muuta tietoa, kuten käyttäjän koko nimi, osoite ja sähköpostiosoite. (Foxley, Higgins, Tsintsifas 2002)

3.2.2 Käyttäjät

Ceilidh jakaa järjestelmän käyttäjät viiteen eri kategoriaan, joita ovat *opiskelija* (student), *tutor* (tutor), *opettaja* (teacher), *kurssin kehittäjä* (developer) ja *pääkäyttäjä* (system administrator). Käyttäjäryhmien oikeudet lisääntyvät ryhmissä siten, että seuraavalla ryhmällä on, omien oikeuksiensa lisäksi, oikeus edeltävien ryhmien toimintoihin. Järjestelmä kontrolloi ryhmien

oikeuksia standardeilla UNIX-käyttöjärjestelmän tiedosto-oikeuksilla. Opiskelijat voivat selata muistiinpanoja, lukea kysymyksiä, katsoa päivän viestin, lähettää valmiin tehtävän ratkaisun tai ladata runko-ohjelman. (Foxley et al. 2001)

Toinen käyttäjäryhmä, eli tutorit, voivat edellisten toimintojen lisäksi katsoa malliratkaisuja, lukea tutorin aputiedostoja ja monitoroida opiskelijoiden edistymistä. Kolmas ryhmä - opettajat - valitsee mitä harjoitustehtäviä käytetään, milloin ne avataan ja suljetaan, sekä milloin harjoituksien palautusten määräajat ovat (Foxley et al. 2001). Lisäksi opettajan vastuulla on painotusten ja erilaisten rajoitusten, kuten tehtävän maksimilähetysmäärän, asettaminen. Kurssin kehittäjän vastuulla on kirjoittaa uutta kurssimateriaalia, kuten kurssin muistiinpanot (notes) ja tehtävien kysymykset sekä kehittää tietyllä kurssilla mahdollisesti vaadittavat uudet työkalut (Foxley, Higgins, Tsintsifas 2002). Pääkäyttäjä asentaa järjestelmän, hoitaa sen ylläpidon ja lisää kursseille uusia käyttäjiä. (Foxley et al. 2001)

3.2.3 Tehtävien lähetys ja tarkastus

Opiskelijat lukevat verkossa viikon kysymyksen, tekevät ohjelmasta luonnostelman rungoksi, kehittävät ratkaisun ja lähettävät ohjelman arvosteltavaksi. Kaksi viimeistä askelta voidaan toistaa, niin että opiskelijat voivat tehdä useita yrityksiä, joista jokaisesta järjestelmä antaa palautetta ohjelman heikkouksista. (Foxley, Higgins, Gibbon 1996)

Ensimmäisessä Ceilidhin versiossa opiskelijoiden ohjelmien tarkastus tehtiin staattisten ja dynaamisten metriikoiden perusteella. Staattisia ominaisuuksia olivat ohjelman layout, sisennykset, tunnisteiden valinta, luettavuus, ohjelman rakenne, kommenttien käyttö, monimutkaisuus, sekä lähdekoodin tarkastajan eli lintin antamat varoitukset ja epäilyttävät rakenteet. Dynaamisia ominaisuuksia olivat testi- ja tietosarja-ajot, kuoriskriptien käyttö, ohjelman validi tulos ja tuloksien oikeellisuuden tarkistavan oraakkelin käyttö. (Foxley, Higgins, Gibbon 1996)

Myös ohjelman tehokkuutta voitiin monitoroida. Alun perin harjoitustehtävien tarkastus tehtiin yön aikana ja maksimimäärä lähetyksille oli yksi päivässä. Nykyään harjoitustehtävien tarkastus

on vuorovaikutteista, siitä annetaan opiskelijalle välitöntä palautetta, arvosanat tallennetaan tietokantaan ja kopio opiskelijan ohjelmasta tallennetaan järjestelmään. (Foxley, Higgins, Gibbon 1996)

3.2.4 Kurssin hallinnointi

Kurssin hallinnointiin kuuluu kolme pääaluetta, joita ovat kurssin hallintotehtävät, opiskelijoiden saavutusten arvioiminen ja tiedon esittäminen opiskelijoille. Kurssin hallintotehtävät sisältävät yksittäisen opiskelijan ja koko kurssin edistymisen seuraamisen, tutoreiden informoimisen asiaankuuluvalla tiedolla, poisjäävien opiskelijoiden havaitsemisen, kopioiden säilyttämisen opiskelijoiden osallistumisesta kurssille ja plagioinnin havaitsemisen opiskelijoiden töistä. (Foxley et al. 1999)

Opiskelijoiden saavutusten arvioiminen voi tapahtua monenlaisissa eri muodoissa, joita ovat useilla eri kielillä tehdyt tietokoneohjelmat, monivalintakysymykset, kysymys/vastaus harjoitukset, yhden sanan tai lauseen vastaukset, ja esseet tai raportit. Edelliset tehtävät voivat liittyä kurssin harjoituksiin tai olla osana koetta. (Foxley et al. 1999)

Tiedon esittäminen opiskelijoille vastaa luentoa tavanomaisilla kursseilla. Perinteinen tietokoneavusteinen oppiminen on ollut tiedon esittämistä oppilaille, opiskelijan määräämässä tahdissa, hänen valitsemiaan reittejä pitkin. Järjestelmän suorittama arviointi on ollut opiskelijoiden edistymisen arviointia. Ceilidh ei kattanut vuonna 1996 tiedon esittämisen aluetta, se lisättiin järjestelmään vasta myöhemmin. Opiskelijoiden saavutusten arvioiminen ja kurssin hallintotehtävät vievät paljon työvoimaa, joten tietokoneen apu on niissä tärkeintä. Järjestelmä ja itse kurssit pitää erottaa toisistaan; järjestelmä itsessään on yleinen kokoelma ohjelmistoja ja sisältää yleisiin tarkoitukseen tarkoitettuja arviointityökaluja. Kurssit ovat saatavissa erikseen ja niitä voi olla samanaikaisesti useita, joista jokainen voidaan asentaa siten kuin tarvitaan. Jokainen kurssi sisältää omat erityiset arviointityökalut ja kokoelman harjoitustehtäviä, sekä niiden arvioinnin yksityiskohdat. (Foxley et al. 1999)

3.2.5 Arvosanojen antaminen

Opettaja ja muu henkilökunta voi muuttaa tai antaa arvosanoja käsin. Opiskelijat voivat lähettää verkon kautta myös esseet, jotka voidaan tallentaa järjestelmään ja arvostella käsin tai ilman konetta paperitulosteista. Opettaja voi tämän jälkeen arvioida luokan, opiskelijan tai tehtävien arvosanoja tilastollisesti ja nähdä, keiden lähetykset puuttuvat ja tehdä plagiointitestauksen. Tutor voi seurata opiskelijoiden edistymistä ja heidän lähetyksiään. (Foxley et al. 1999)

Ceilidhin typografian tarkastusmenetelmä yrittää arvioida kuinka hyvä on ohjelman ulkoasu. Hyvä layout tekee ohjelmasta luettavan, parantaen näin koodin uudelleenkäytettävyyttä ja ylläpidettävyyttä. Ulkoasun testaamiseen on kehitetty erilaisia heuristiikkoja, joita kutsutaan Ceilidh-järjestelmässä typografiaksi. Hyvän ohjelman typografisiin piirteisiin kuuluu monimutkaisten koodin osien kommentointi, riittävän tyhjän tilan jättäminen koodin väliin ja johdonmukainen kielen syntaksin käyttö, esimerkiksi C/C++ -ohjelmissa aaltosulkujen pitää olla kohdillaan ja koodissa on käytettävä sisennyksiä. Lisäksi muuttujien nimien pitää olla merkityksellisiä ja sopivan mittaisia. (Gibbon 2002)

Ceilidhin antamia arvosanoja voidaan käyttää eri tavoilla. Opettaja voi määritellä, että Ceilidh antaa 60% arvosanoista ja tentti 40%, tai Ceilidhiä voidaan käyttää vain dynaamisten testien arviointiin, jolloin opettaja arvioi ohjelman lähdekoodin käsin. Ceilidh voi myös arvioida koko kurssin tehtävät, mutta rajatapaukset käsitellään manuaalisesti. (Foxley, Higgins, Gibbon 1996)

Hyvänä puolena esseiden käytössä on se, että opiskelijat voivat esseissä selventää ohjelmien suunnittelun ja testauksen takana olevaa ajatteluaan. Ceilidh antaa esseille valmiin rungon, joka varmistaa, että esseet käsittelevät oikeita asioita. Esseet tallennetaan Ceilidhin ratkaisu-hakemistoon ja oikoluetaan. Esseet voidaan arvostella verkossa, mutta yleensä opettajat tulostavat ja tarkastavat ne käsin. (Foxley, Higgins, Gibbon 1996)

Kysymyksen tekeminen on vaikeaa, sillä sen täytyy olla täydellinen, yksiselitteinen ja johdonmukainen. Mitä täsmällisempi kysymys on, sitä helpompi valmiit vastaukset on arvioida. Opiskelijalle voidaan lisäksi antaa runko ohjelmasta tai esseetehtävästä, testidataa, kopio

kysymyksestä tai muita auttavia tiedostoja. Opiskelijan tekemän harjoitustehtävän lähetyskertojen maksimimäärää voidaan säätää siten, että se vaikuttaa arvosanaan. (Foxley, Higgins, Gibbon 1996)

Oikeudellisista ja moraalisista syistä opiskelijan täytyy saada tietää miten arvosanojen antaminen toimii ja mitä tietoa heistä on tallennettu järjestelmään. Akateemisista syistä koko kurssien sisällöistä pitää olla arkistoidut kopiot. Ceilidh pitää tallessa jäljitysketjut ja sen, mitä opiskelijalle on annettu tehtäväksi ja kuka käyttää tiettyjä Ceilidhin sovelluksia. (Foxley, Higgins, Gibbon 1996)

3.2.6 Käyttökokemuksia

Foxleyllä, Higginssillä ja Gibbonilla (1996) on pitkä kokemus Ceilidhin käytöstä. Sinä aikana arviointityyliin on tullut useita muutoksia. Kurssien muokkaus ja uusien piirteiden lisääminen on tehty entistä helpommaksi. Lisäksi arvioinnista on tehty yksinkertaisempaa. Alussa tehtävien tarkastus suunniteltiin turhan monimutkaiseksi. Arvioitavia osia oli liian paljon, koska ihmisen suorittama tarkastus yritettiin minimoida. Nykyään on keskitytty kolmeen arviointityökaluun, joita ovat dynaaminen testaus, typografinen ulkoasu ja ohjelman ominaisuudet. Uusi tapa on vaikuttanut siten, että opettajan on helpompi perustella opiskelijoille tehtävien arviointiperusteita ja opiskelijoilla on selkeämmät tavoitteet tehtäviä tehdessään. Arviointityökaluja käsitellään tarkemmin seuraavassa luvussa.

Ceilidh-järjestelmällä on myös useita huonoja puolia; se vaatii osaavaa henkilökuntaa järjestelmän asennukseen ja ylläpitoon, koska se perustuu UNIX-käyttöjärjestelmään. Järjestelmän käyttöliittymä pohjautuu ASCII-merkkipäätteeseen ja vaikka Ceilidhin arviointimekanismi onkin tehokas, sen antama palaute opiskelijoille on rajattua. (Foxley et al. 2001) Lisäksi testien kirjoittaminen vaatii huolellisuutta, jos jokin erikoistapaus jää huomiotta, opiskelija saattaa jäädä ilman ansaitsemiaan pisteitä. (Malmi 2002)

3.2.7 Ceilidhin plagiointitestaus ja myöhässä palauttaminen

Ceilidh-järjestelmässä on automaattinen plagioinnin ilmaisuohjelma, joka vertaa toisiinsa kaikkien opiskelijoiden palauttamat ohjelmat koko kurssin laajuudelta. Ohjelma pystyy löytämään kaikki ohjelmat, jotka ovat identtisiä, melkein identtisiä tai vain samanlaisia. Plagioinnin ilmaisuohjelmaa on hyödynnetty jokaisella kurssilla Nottinghamin yliopiston tietojenkäsittelytieteen laitoksella. Ongelmia plagiointitestauksessa on aiheuttanut se, että yleensä alkeiskursseilla tehtävien ratkaisut ovat lyhyitä ja samanlaisia vastauksia tulee varsinkin, kun tehtävän runko annetaan valmiina. (Foxley 1997)

Foxleyn (1997) kokemuksen mukaan työt, joihin liittyy plagiointia, lähetetään juuri ennen palautuksen määräaikaa. Myöhässä palautetusta työstä voidaan Ceilidh-järjestelmässä vähentää 5 % arvosanasta per päivä. Ceilidh pitää kopioita kaikista lähetetyistä kurssitöistä, jonka vuoksi tietyn harjoitustyön eri lähetyksiä on mahdollista verrata toisiinsa. Harjoitustehtävän testaus plagioinnin suhteen tehdään sen jälkeen kun tehtävä on suljettu eli kun kaikki opiskelijat ovat palauttaneet vastauksensa. Plagiointitestaus tuottaa listan käyttäjätunnuspareista, joiden ohjelmat ovat lähimpänä toisiaan. Kurssin edetessä voidaan julkistaa lista käyttäjistä, joiden ohjelmat sisältävät samanlaisuuksia tai, jos joillakin opiskelijoilla on samanlaisia vastauksia toistuvasti, heidän arvosanaansa voidaan laskea tai jakaa se puoliksi.

3.3 CourseMaster

CourseMaster on täydellinen uudelleen toteutus Ceilidh-järjestelmästä. Sen kehityksessä on hyödynnetty Ceilidh-järjestelmästä saatua maailmanlaajuista kokemusta kymmenen vuoden ajalta (Foxley et al. 1999). CourseMaster-järjestelmää voidaan käyttää tarkastamaan automaattisesti Java- ja C++ -ohjelmia, olioperustaista suunnittelua, vuokaavioita ja loogisia piirejä (Higgins, Symeonidis, Tsintsifas 2002). CourseMaster on toteutettu tiukkojen ohjelmistostandardien mukaisesti käyttäen olio-ohjelmointia. Päämääränä oli parempi suorituskyky, käytettävyys ja laajennettavuus sekä helpompi ylläpito. (Foxley et al. 2001)

CourseMaster on asiakas/palvelin -pohjainen järjestelmä, joka on toteutettu Javalla (Rawles 2001). Java-kieli valittiin, koska se tukee alustariippumattomuutta ja tarjoaa sopivan jakelumenetelmän RMI-mekanismin avulla. RMI on huomattavasti tehokkaampi kuin muut asiakas-palvelin -järjestelmät ja sitä on suhteellisen helppo käyttää. Lisäksi se tekee verkosta läpinäkyvän ohjelmoijalle. (Foxley et al. 2001)

CourseMaster säilytti suurimman osan Ceilidhin toiminnallisuudesta jonka lisäksi siinä on uusia toimintoja. Tärkeimpiä uusista toiminnoista ovat analyyttisen palautepuun näyttäminen harjoituksesta sekä ominaisuus, joka antaa opiskelijan määritellä työympäristönsä haluamakseen. Palautepuu sisältää laajan palautteen tarkastuksen eri osa-alueista saaduista tuloksista. (Foxley et al. 2001)

Uudelleenkehityksen aikana tehtiin monia parannuksia, joista tärkeimpiä ovat graafisen käyttöliittymän parantaminen niin, että opiskelijoiden on helpompi käyttää sitä, ja parannettu palautteenantojärjestelmä, joka edistää oppimiskokemusta. Lisäksi järjestelmässä on uusi tehokas tarkastusalijärjestelmä, joka on helposti laajennettavissa. Teoriassa tarkastusjärjestelmä voidaan asettaa tarkastamaan minkä tyyppinen dokumentti tahansa, jos vain sille sopivat oikeat metriikat voidaan tunnistaa ja kehittää. (Foxley et al. 2001)

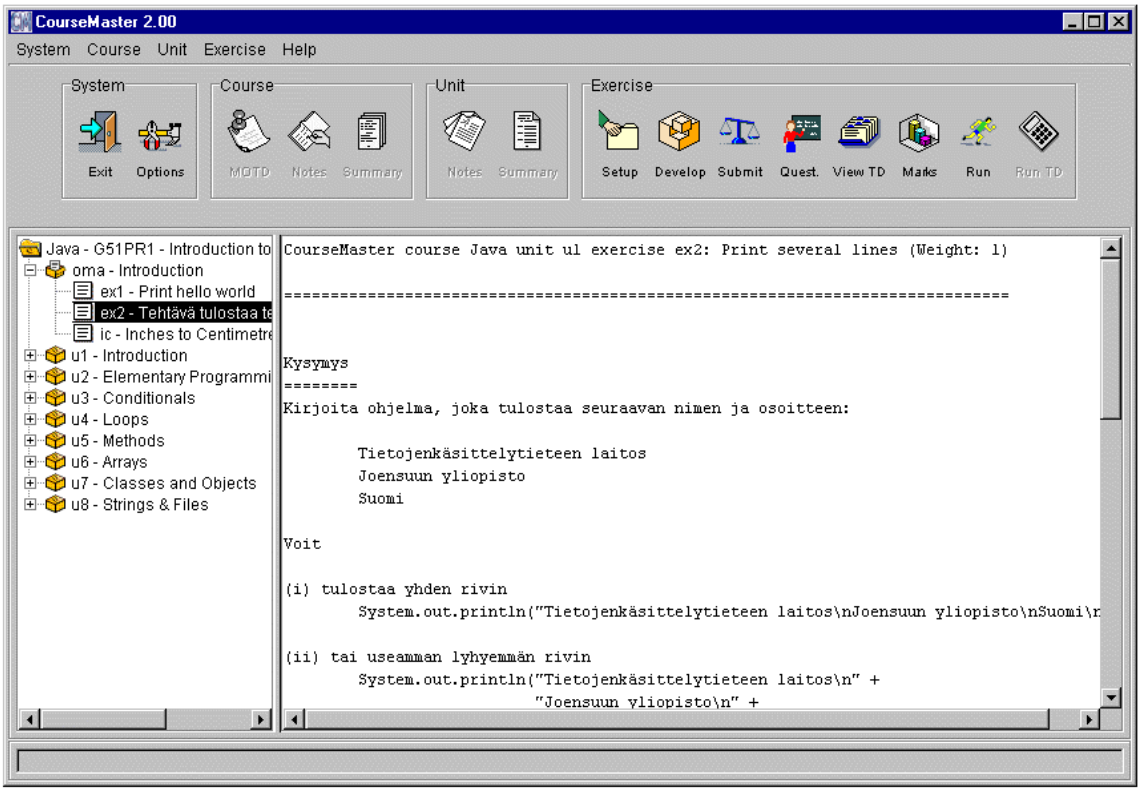
CourseMaster-järjestelmä on kaupallinen tuote, joka on suunniteltu kurssin tehtävien arviointiin, eikä sitä käytetä summatiivisena työkaluna. Järjestelmää hyödynnetään pääasiassa formatiivisessa testaamisessa ja itsearvioinnissa, sekä jossakin määrin diagnostisessa testaamisessa. (Rawles 2001)

3.3.1 Opiskelijanäkymä

Opiskelija käyttää CourseMasteria avaamalla sen asiakas-sovelluksen ja kirjautumalla siihen omilla tunnuksillaan. Aloitusikkuna avautuu ja opiskelija näkee päivän viestin. Luettuaan viestin opiskelija napsauttaa siinä olevaa *Ok*-painiketta. Vasemmanpuoleisesta ikkunasta opiskelija näkee valittavana olevat *kurssit* (course). Kurssien alta napsauttamalla löytyvät myös *moduulit*

(unit) ja niiden alla olevat yksittäiset *tehtävät* (exercise). Oikeassa ikkunassa on järjestelmän yleinen viesti. (Foxley et al. 1999)

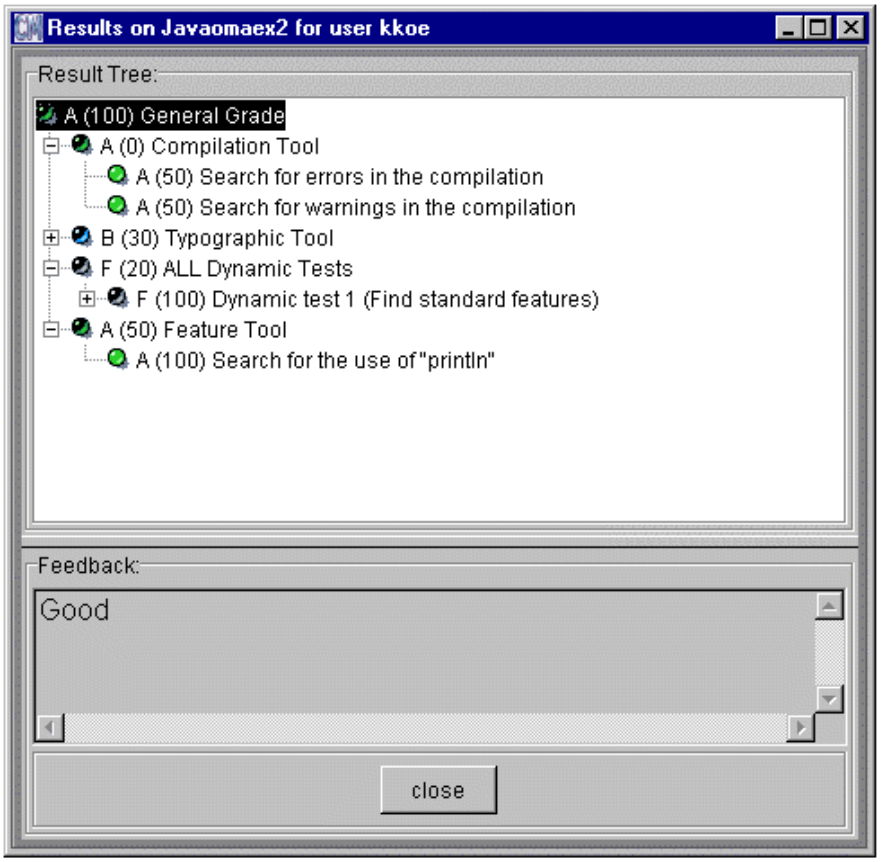
Opiskelija valitsee tietyn kurssin, moduulin ja tehtävän vasemmasta ikkunasta. Kurssin opettaja on kertonut opiskelijalle, mihin mennessä tietyt tehtävät pitää palauttaa. Oikeanpuoleisessa ikkunassa näkyy kurssin jäsentely, silloin kun kurssitaso on valittuna. Kun moduulitaso on valittuna, ikkunassa näkyvät moduulin muistiinpanot, ja kun jokin yksittäinen tehtävä on valittu, oikeanpuoleisessa ikkunassa näkyy tehtävän tehtävänanto. (Foxley et al. 1999) Kuvassa 5 on CourseMasterin asiakassovellus.



Kuva 5: CourseMaster-järjestelmän aloitusikkuna.

Aluksi opiskelija lukee tehtävän kysymyksen, joka kertoo tarkasti mitä ohjelman pitää tehdä. Opiskelija napsauttaa *Setup*-painiketta, joka asentaa tarvittavat tiedostot opiskelijan työympäristöön ja *Develop*-painiketta, joka aukaisee opiskelijalle määritellyn kehitysympäristön. Kehitysympäristössä opiskelija tekee ohjelmointitehtävän ja testaa, että ohjelma toimii oikein. Kehitysympäristönä voi toimia esimerkiksi Emacs-editori tai Visual C++. Lopuksi opiskelija

lähettää tekemänsä ohjelman napsauttamalla *Submit*-painiketta. CourseMaster lähettää ohjelman ja yhteenvedon ohjelman tuloksista laajennettavan puun muodossa tarkastuspalvelimelle. Kuvassa 6 on palautepuu, jossa näkyy opiskelijan arvosana tehtävästä. Yleisarvosana (General Grade) A, koostuu kääntämisestä (Compilation Tool), typografiasta (Typographic Tool), dynaamisista testeistä (Dynamic Tests) ja ohjelman ominaisuuksista (Feature Tool) saaduista arvosanoista. Opiskelija voi vielä parantaa ohjelmaa ja lähettää sen uudelleen, jos hän ei ole tyytyväinen saamiinsa tuloksiin. (Foxley et al. 1999)



Kuva 6: CourseMasterin opiskelijalle antama palautepuu.

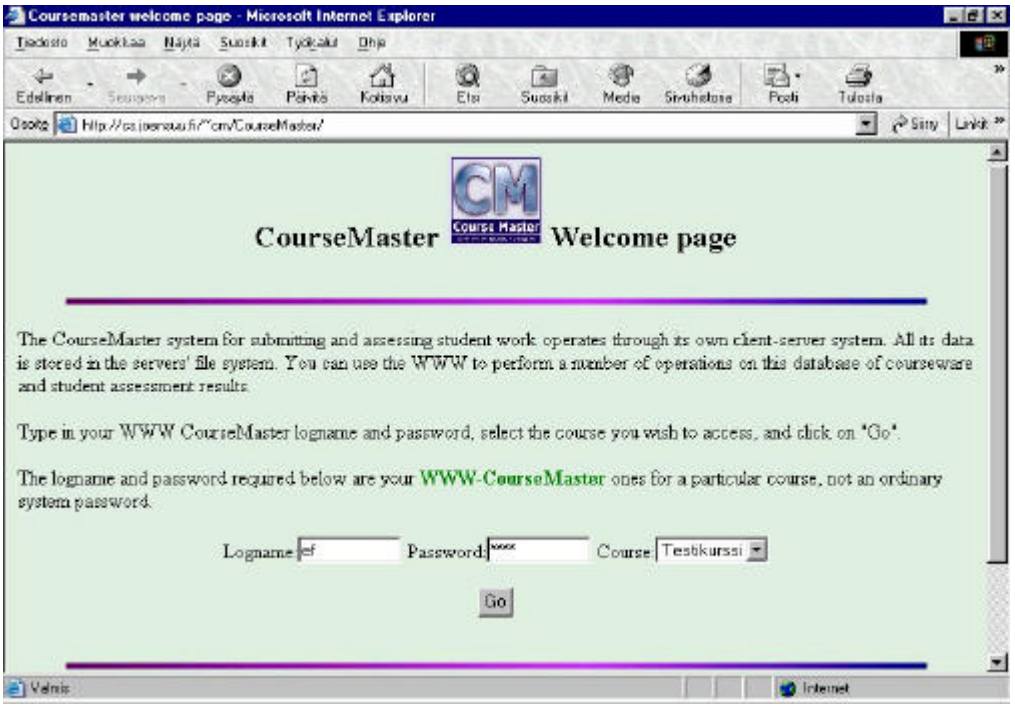
3.3.2 Tutorin näkymä

Kaikki opiskelijoiden lähettämät ohjelmat ja arvosanat on tallennettu CourseMasterin tietokantaan. Tutor pääsee käsiksi tähän tietoon käyttämällä CourseMasterin asiakas-

sovelluksesta täysin erillään olevaa WWW-pohjaista järjestelmää. Järjestelmä koostuu HTML-sivuista ja CGI-komennoista, jotka analysoivat tietoja ja tekevät niistä monenlaisia yhteenvetoja. Tutor voi nähdä koko luokan tai yhden oppilaan arvosanojen yhteenvedon, tehtävän arvosanojen hajonnan tai tehtävän kysymyksen ja ratkaisun. (Foxley et al. 1999)

3.3.3 Opettajan näkymä

Tutorin toimintojen lisäksi opettaja voi avata tai sulkea tehtäviä, editoida päivän viestiä ja tehtävien kysymystiedostoja, aloittaa plagioinnin tarkastusprosessin tehtävän sulkemisen jälkeen ja katsoa jäljitysketjuja. Opettaja voi myös selata tehtäviä ja valita niistä haluamansa kullekin viikolle. (Foxley et al. 1999) Kuvassa 7 on WWW-pohjainen opettajan käyttöliittymä.



Kuva 7: CourseMasterin WWW-pohjainen opettajan käyttöliittymä.

3.4 Muita koodintarkastus-järjestelmiä

Automaattisia koodin tarkastamiseen käytettyjä järjestelmiä on kehitetty vuosien aikana useita ja niissä on myös monia eroavaisuuksia verrattuna edellä tarkasteltuihin Boss- ja Ceilidh/CourseMaster-järjestelmiin. Seuraavassa aliluvussa käsittelen joitakin toisistaan poikkeavia automaattisia koodin tarkastamiseen käytettyjä järjestelmiä, kuten Pilot-, SchemeRobo, WebAssign- ja RoboProf -järjestelmiä, sekä prototyyppiä järjestelmästä, joka tarkastaa ohjelmia ohjelmointikielestä riippumattomasti niin sanotun standardialgoritmin avulla. Tarkastelun ulkopuolelle jäävät muun muassa ASSYST- (Jackson, Usher 1997), TRY- (Reek 1996), TRAKLA- (Korhonen, Malmi, Saikkonen 2002), PSGE- (Jones 2001) ja VIOPE-järjestelmät (Ageenko, Vihtonen 2002).

3.4.1 Pilot

Interaktiivisuus ja animaatio tarjoavat mahdollisuuden sitoa oppilaat aktiivisesti oppimisprosessiin. Pilot on USA:ssa kehitetty interaktiivinen ja alustariippumaton työkalu, joka on saatavilla maailmanlaajuisesti. Käsitteiden visualisoinnin etuja opetuskäytössä on kuvattu paljon kirjallisuudessa. Pilot on suunniteltu testaamaan tietojenkäsittelytieteen käsitteitä ja sitä käytetään algoritmien visualisointityökaluna, kuten esimerkiksi pienimmän virittävän puun ongelman ratkaisussa Primin algoritmilla tai lyhyimmän polun algoritmissa. Algoritmien animaatiota on käytetty menestyksekkäästi muun muassa graafisiin kaavioihin, lajitteluun ja etsimiseen. Lisäksi ohjelmakoodin animointi parantaa uuden ohjelmointikielen oppimista ja vaikeiden käsitteiden ymmärtämistä. (Bridgeman et al. 2000).

Bridgemanin et al. (2000) mukaan Pilot-järjestelmää suunniteltaessa on ollut useita eri tavoitteita. Ensinnäkin on haluttu kehittää järjestelmä, jota voi käyttää luokkahuoneessa apuna käsitteiden selittämisessä. Toiseksi kehittäjät ovat halunneet suunnitella työkalun, joka osaa luoda satunnaisia ilmentymiä ohjelmointiongelmasta, koska monesti opiskelijat oppivat parhaiten esimerkkien kautta. Pilot-järjestelmä testaa opiskelijan kykyjä muodostaa vastaukssia pelkän

oikean vastauksen valitsemisen sijaan. Kolmantena tavoitteena on ollut luoda automaattinen tarkastusjärjestelmä, jossa opiskelija saa välittömän palautteen työstään.

Yksi järjestelmän eduista on, että se on alustariippumaton appletti, jota voi ajaa selaimella. Toinen, vielä tärkeämpi piirre on sen kyky kommunikoida opiskelijan kanssa yksityiskohtaisen palautteen avulla. Jos esimerkiksi opiskelija on valinnut puusta väärän kaaren, Pilot korostaa sen ja ehdottaa kuinka sen voisi korjata. Järjestelmä osaa arvostella myös osittain oikeita ratkaisuja. (Bridgeman et al. 2000)

Bridgemanin et al. (2000) mukaan Pilot-järjestelmää voi käyttää tenttien tai tehtävien arviointiin, jos apuna käytetään jotain tietoturvallisuutta lisäävää metodia, esimerkiksi salasanoja tai salausta. Tarkastajien täytyy syöttää ongelmat ja opiskelijoiden vastaukset järjestelmään, koska nykyistä järjestelmää voi käyttää vain välittömästi generoitujen vastauksien tarkastamiseen. Tällä tavalla opiskelijat eivät voi syöttää järjestelmään omia kotitehtäviään ja käyttää apuna Pilot-järjestelmän ongelmanratkaisukykyä.

Järjestelmä perustuu asiakas/palvelin -arkkitehtuurille ja se on rakennettu GeomNetin päälle. GeomNet -mallissa asiakas on vastuussa käyttöliittymän ylläpidosta ja kaikki algoritmiin liittyvä laskenta suoritetaan palvelimella. Asiakas on Pilotissa toteutettu Java-applettina ja palvelin sisältää myös Javalla toteutetut *ongelman tuottajat* (problem generators), *tarkastajat* (checkers) ja *ratkaisijat* (solvers). Järjestelmän tekijät valitsivat asiakas/palvelin -arkkitehtuurin lähinnä sen joustavuuden vuoksi. Palvelin voi ajaa Java-kielisiä ohjelmia, eikä sillä ole rajoituksia Java-applettien suhteen. Lisäksi GeomNetin modulaarisuus tekee uusien komponenttien lisäämisen helpoksi Pilot-järjestelmään. (Bridgeman et al. 2000).

3.4.2 WebAssign

WebAssign on elektronisten tehtävien lähetys-, arviointi- ja tallennusjärjestelmä, joka kehitettiin Pohjois-Carolinan osavaltion yliopistossa alun perin fysiikkaa ja muita luonnontieteitä varten. (WebAssign 2002) Sen WWW-pohjainen käyttöliittymä tukee opiskelijoiden kurssille kuuluvia tehtäviä, kurssin hallintaa ja kaikkia tarkastusprosessin vaiheita.

WebAssign-järjestelmän kehitystyö aloitettiin vuonna 1996 ja se kaupallistettiin vuonna 1998. Järjestelmää käytetään nykyään sadoissa eri opetuslaitoksissa (WebAssign, 2002). Niistä yksi on Fernin etäyliopisto Hagenissa, Saksassa, jossa WebAssignin käyttö on nopeuttanut tehtävien tarkastusprosessia yli viidestä viikosta alle kahteen viikkoon. WebAssign -järjestelmästä on saatavilla Internetistä demoversio. (Brunsmann et al. 2000)

Brunsmannin et al. (2000) mukaan WebAssign-järjestelmän päätavoite on tarjota tehokas web-pohjainen tuki monen tyyppisille harjoitustehtäville ja arviointitavoille, alkaen monivalintatehtävistä konstruktiivisuutta vaativiin essee-vastauksiin, matemaattisiin todistustehtäviin, elektronisiin piireihin ja ohjelmointitehtäviin. Lisäksi järjestelmän tavoitteita ovat tehtäväprosessin kaikkien toimintojen tukeminen ja helppo integroitavuus muiden virtuaaliyliopistossa käytettyjen komponenttien kanssa sekä tarkastusprosessin nopeuttaminen.

Järjestelmässä voidaan luoda satunnaisia ilmentymiä numero-, sana-, fraasi-, grafiikka-, ääni- tai videokysymyksistä. Näin on mahdollista tehdä samasta kysymyksestä eri versioita jokaiselle opiskelijalle. WebAssign-järjestelmän käyttäjät voivat myös hyödyntää valmiita tietokannasta löytyviä tekstikirjan kysymyksiä, koska järjestelmällä on sopimus joidenkin oppikirjojen julkaisijoiden kanssa. (WebAssign 2002)

WebAssign-järjestelmässä käyttäjät jaetaan neljään eri rooliin, joilla on kaikilla omat toimintonsa. Nämä roolit ovat *opiskelija, valvoja* (supervisor), *tarkastaja* (corrector) ja *pääkäyttäjä* (administrator). Opiskelijalla on pääsy tehtävänantoihin, saamiinsa pisteisiin ja palautteeseen. Lisäksi hän voi lähettää tekemiään tehtäviä. Valvoja konfiguroi tehtävät, suunnittelee tehtävä materiaalin, kirjaa järjestelmään opiskelijat ja tarkastajat, sekä valvoo tarkastajia. Tarkastaja arvostelee tehtävät ja antaa niistä palautetta. Pääkäyttäjä asentaa kurssin alustavasti. (Brunsmann et al. 2000)

Pääkomponentti WebAssign-järjestelmässä on keskuspalvelinprosessi (central server process), johon käyttäjät pääsevät Internet-selaimen kautta. Palvelinprosessi tallentaa tietokantaan kaiken tarvittavan tiedon ja dokumentit, kuten esimerkiksi tehtäväsivut, opiskelijoiden lähetykset,

tarkastamisen, pisteet ja malliratkaisut. Tietokannan tiedot on suojattu salasanojen avulla (WebAssign 2002). Lisäksi järjestelmään on lisätty Java-appletteja, joilla on visualisoitu matemaattisia kaavoja ja graafisia kuvioita. Näin on pyritty kiertämään HTML-kielen lomakkeiden rajoituksia. (Brunsmann et al. 2000)

Arviointi WebAssign-järjestelmässä voi Brunsmannin et al. (2000) mukaan tapahtua eri tavoin. Ensinnäkin opiskelijoiden ratkaisuja voidaan arvioida kahdessa eri ajankohdassa, joko automaattisesti heti lähetyksen jälkeen tai loppuarviointina jakson jälkeen. Tavat voidaan myös yhdistää, eli opiskelijan lähetykset arvioidaan automaattisesti heti lähetyksen jälkeen ja tarkempi arviointi ja palaute annetaan vasta lopussa.

WebAssign-järjestelmässä tarkastetaan automaattisesti standardit tehtävätyypit, joita ovat kyllä/ei- kysymykset, monivalintatehtävät ja numeerisen vastauksen antavat tehtävät. Käyttäjä voi myös luoda ja ladata järjestelmään omia automaattisia tarkastusmoduuleja Java-kielellä, jos kurssilla tarvitaan kehittyneempää arviointia. Brunsmann et al. (2000) kertovat, että itse tehtyjä automaattisia tarkastusmoduuleja on käytetty Fernin etäyliopistossa Hagenissa ohjelmointitehtävien esitestaukseen. Opiskelija on lähettänyt ohjelman ja järjestelmä on kääntänyt ja testannut sen. Tapa sopii monenlaisiin eri ohjelmointitehtäviin ja sitä on käytetty useilla eri ohjelmointikielillä.

3.4.3 RoboProf

RoboProf-järjestelmä on WWW-pohjainen ohjelman tarkastusjärjestelmä, joka noudattaa elektronisen kirjan rakennetta. Järjestelmä esittää opiskelijalle tietoa tarkasti määritellystä aiheesta ja sen jälkeen tarkastaa tähän aiheeseen liittyvät tehtävät. Kun opiskelijan saamat tulokset ovat riittäviä, hän siirtyy toiseen, edistyneempään aiheeseen. Tämä tapa tekee opiskelusta haastavaa ja pitää yllä opiskelijan mielenkiintoa. (Daly 1999)

Daly (1999) kertoo, että Dublinin yliopistossa RoboProf-järjestelmää on käytetty ohjelmoinnin johdantokurssilla. Kurssilla järjestelmää käytettiin kuuden viikon ajan helpottamaan joidenkin C++-kielen lauseiden, syntaksin ja semantiikan opettamista. Järjestelmän tehtävät olivat osa

jatkuvaa arviointia ja ne muodostivat neljännesosan moduulin arvosanasta. Ainoa määräaika tehtävien palauttamiselle oli lukukauden loppu, siitä huolimatta kaikki opiskelijat läpäisivät kurssin RoboProf-osuuden kaksi kuukautta ennen määräaikaa.

Opiskelijat voivat kirjautua sisään RoboProf-järjestelmään WWW-selaimen avulla. Opiskelija voi tehdä tehtäviä omaan tahtiinsa mihin aikaan ja missä paikassa tahansa, jossa vain on Internet-yhteys. Alussa opiskelija kirjautuu sisään järjestelmään, lukee aiheeseen liittyvän oppimateriaalin ja vastaa sen jälkeen annettuihin tehtäviin. Ohjelmoinnin johdanto -kurssilla miltei kaikki tehtävät olivat pieniä ohjelmointiongelmia, kuten esimerkiksi for-silmukka, joka tulosti numerot 1 - 10. Opiskelijan tehtävänä oli muuttaa ohjelmaa tulostamaan numerot 5 - 20. (Daly 1999)

Daly (1999) sanoo, että järjestelmä tarkastaa lähetetyt ohjelmat melkein täysin niiden antamien tulosten perusteella. Tarkastusosa on erillinen ja se voidaan korvata kehittyneemmillä testeillä. Kaikki ohjelmalähetykset tallennetaan ja luennoitsija voi tarkastaa ne myöhemmin. RoboProf-järjestelmä antaa välittömästi oikeat vastaukset ja palautetta vastausten oikeellisuudesta. Opiskelija voi sitten korjata ohjelmansa ja lähettää sen uudelleen. Vääristä vastauksista ei rangaista, vaan opiskelija joutuu vastaamaan toiseen samanlaiseen tehtävään. Vähitellen opiskelija suorittaa koko kurssin ja tehtävät tällä tavalla edeten.

Dalyn (1999) mukaan RoboProf helpottaa tehtävien hallinnoimista, mutta se ei vähennä ihmisen tarvetta tehtävien tekemisen apuna. Dublinin yliopistossa tutoreita on käytetty avustamaan opiskelijoita. Tutorit voivat myös antaa lähetetyistä tehtävistä yksityiskohtaisempaa palautetta myöhemmin sähköpostin välityksellä, sen jälkeen kun he saavat raportin opiskelijoiden kehityksestä.

RoboProf sisältää opiskelija- ja kurssitietokannat sekä palvelimen ajaman Java-servletin, joka esittää moduulin muistiinpanot ja ongelmat. Java-appletti käsittelee ohjelmien lähetykset, kääntää ohjelmat ja testaa ne. Ohjelmaa ajetaan testidatasarjoilla ja opiskelijalle näytetään ohjelman antamat tulokset ja testidatojen odotetut tulokset. RoboProfin tarkastusjärjestelmä voidaan määritellä jokaiselle ongelmalle erikseen. Luennoitsija voi lisätä järjestelmään moduulin, joka analysoi lähetetyn ohjelman ja antaa arvosanat sen perusteella. Järjestelmässä on myös valmiita

tarkastuskaavioita, jotka tarkastavat ohjelman sen antamien tulosten perusteella. Uutta ongelmaa lisätessä luennoitsija voi käyttää valmiita tarkastuskaavioita tai laittaa niiden tilalle oman tarkastusohjelmansa. Testidata voi olla sama jokaiselle lähetykselle, tai sitten se voidaan muodostaa satunnaisesti. Ohjelmoinnin johdatuskurssilla Dublinissa opiskelijat saattoivat lähettää ohjelman, jonka he tiesivät toimivan väärin, vain saadakseen mallin syötteestä ja tuloksista. (Daly 1999)

Java-kielen etu on, että se antaa ladata ohjelmamoduuleja dynaamisesti. Toisin sanoen kone voi ladata tietyn ongelman luonti- ja testaus-moduulit ilman, että laitetta itseään muutetaan. Jokainen ongelma voidaan luoda ajon aikana ja muokata opiskelijan kykyjä vastaavaksi. Tarkastus-ohjelma voidaan ohjelmoida itse sellaiseksi kuin halutaan erikseen jokaiselle ohjelmalle. Oletusarvoisesti järjestelmä esittää muuttumattoman määrittelyn ja arvosana perustuu ohjelman tuloksiin ja testidatan sisältävän tiedoston lukemiseen. (Daly 1999)

Dublinissa RoboProf-järjestelmää on kehitetty edelleen, esimerkiksi plagioinnin ilmaisuohjelma on jo lisätty ja opiskelijoiden vastauksien analysointia kehitetään. Jos järjestelmä tunnistaa tietyn vääränlaisen vastauksen, se esittää opiskelijalle polun siihen oppimateriaalin kohtaan, jossa kyseessä olevaa aihetta käsitellään. Opiskelijoiden lähetyksistä etsitään yleisiä virheitä ja järjestelmä esittää virheiden perusteella lisäkysymyksiä, jonka perusteella opiskelija voi korjata virheitä. Lisäksi järjestelmä analysoi tehtyjen yritysten määrää, ongelmien ratkaisuun käytettyä aikaa ja niin edelleen. Kerätty data auttaa tutoreita löytämään opiskelijat, joilla on vaikeuksia tehtävien tekemisessä. (Daly 1999)

3.4.4 Scheme-robo -järjestelmä

Scheme-robo -järjestelmä ei ole web-pohjainen ja se poikkeaa muista esitellyistä koodintarkastusjärjestelmistä siten, että se tarkastaa funktionaalisella Scheme-kielellä kirjoitettuja proseduureja, kokonaisten ohjelmien sijaan. Korhosen, Malmin ja Saikkosen (2001) mukaan Scheme-robo tarkastaa proseduurit täysin automaattisesti, joten opettajan ei tarvitse monitoroida tarkastusprosessia, eikä tehdä sen osia käsin. Opettajalle aiheutuu kuitenkin lisätyötä niin

sanottujen konfiguraatiotiedostojen kirjoittamisesta jokaiselle tehtävälle. Järjestelmä analysoi oikeellisuuden lisäksi myös muita ohjelman piirteitä kuten ohjelman rakennetta ja ajoaikaa.

Korhosen, Malmin ja Saikkosen (2001) mukaan Scheme-kieltä käytetään Helsingin Teknillisessä korkeakoulussa ohjelmoinnin alkeiskurssilla tietojenkäsittelytiedettä pääaineenaan opiskeleville. Kurssilla on käytetty viikoittaisia pakollisia ohjelmointitehtäviä, jotta opiskelijat pysyisivät aktiivisina koko kurssin ajan. Scheme-kieltä ei juurikaan käytetä käytännön ohjelmoinnissa ja joillakin opiskelijoilla on ollut vaikeuksia motivoitua opiskelemaan akateemisesti painottuvaa ohjelmointikieltä. Scheme-kieltä on kuitenkin käytetty, koska se tarjoaa Java- tai C-kieltä paremman mahdollisuuden opettaa monia tärkeitä ja monimutkaisia tietojenkäsittelyn käsitteitä. Tämä lähestymistapa on sopinut hyvin kursseille, koska opiskelijoiden tehtävät koostuvat yleensä yksittäisistä Scheme-proseduureista kokonaisten ohjelmien sijaan.

Scheme-robo -järjestelmällä on useita tärkeitä piirteitä. Arviointi tapahtuu verkossa ja opiskelijat saavat palautteen tehtävistään lähes välittömästi. Näin opiskelijat voivat korjata vastaustaan ja lähettää sen uudelleen nopeasti. Rajoitettu lähetysten lukumäärä pakottaa opiskelijoita miettimään virheiden syitä tarkasti. Scheme-proseduurit palauttavat monesti pelkän arvon sen tulostuksen sijaan, jonka vuoksi ohjelmien poikkeavien tuloksien tarkastaminen ei tuota järjestelmässä ongelmia. Lisp-kielen kaltainen syntaksi tekee helpoksi opiskelijoiden koodin rakenteen arvioinnin ja plagioinnin tarkastamisen. Opettaja voi rajoittaa kussakin tehtävässä sallittuja kielen rakenteita. Scheme-robo-järjestelmässä voidaan käyttää myös satunnaisia testejä ja se antaa ajaa koodia täysin turvallisessa ympäristössä, eikä esimerkiksi päättymätön koodi vahingoita arviointia. (Korhonen, Malmi, Saikkonen 2001)

Tehtävien ja palautteen lähetys tapahtuvat järjestelmässä sähköpostin välityksellä. Opiskelija lähettää tekemänsä tehtävät Scheme-robo-järjestelmään sähköpostilla. Kymmenen minuutin kuluttua järjestelmä lähettää opiskelijalle automaattisen vastauksen, joka sisältää kopion ja kommentteja lähetetystä tehtävästä. Uudet harjoitustehtävät lähetetään opiskelijoille sähköpostia käyttäen, opettaja määrää kullekin tehtäväsarjalle määräajan ja opiskelija voi lähettää kunkin tehtävän enintään 20 kertaa. Opiskelija voi myös pyytää raporttia saamistaan pisteistä tai kopioita ratkaisuistaan. (Korhonen, Malmi, Saikkonen 2001)

Opettaja lisää uusia tehtäviä järjestelmään tekemällä *konfiguraatiotiedoston* (configuration file), joka kuvaa kuinka ratkaisu tarkastetaan. Yleensä nämä tiedostot ovat noin 20 - 100 riviä pitkiä ja ne sisältävät testiajoja, malliratkaisuja ja muuta informaatiota. Korhosen, Malmin ja Saikkosen (2001) mukaan tarkastaminen voi tapahtua usealla eri tavalla, pääasiassa käytetty metodi on suorittaa koodi käyttäen testiajoja ja tutkia palautusarvoja. Koodin rakennetta voidaan myös analysoida eli tarkistaa noudattaako koodi opettajan valmiiksi antamaa pakollista runkoa. Lisäksi tehtävästä voidaan etsiä avainsanoja. Jos kyseessä ei ole ohjelmointi-tehtävä, vaan siinä kysytään esimerkiksi numeroita, tarkastamiseen voidaan käyttää yksinkertaista *täsmäystä* (pattern matching). Jokainen edellä mainittu metodi vaatii omanlaisensa konfiguraatiotiedoston ja eri tarkastusmetodeja voidaan yhdistellä kirjoittamalla niitä useamman sisäkkäin.

Scheme-robo - järjestelmässä on myös plagioinnin tarkastusohjelma. Koodia esikäsitellään ennen plagioinnin tarkastamista. Tehtävien ratkaisut pelkistetään abstrakteiksi syntaksipuiksi, muuttujien nimet poistetaan ja esimerkiksi määrittelyt järjestetään. Helsingin teknillisessä korkeakoulussa ohjelmoinnin peruskursseilla tehdyt ohjelmat olivat käytännössä niin lyhyitä ja samanlaisia, ettei niistä voinut tarkastaa plagiointia täysin automaattisesti, vaan opettaja joutui tarkistamaan epäilyttävät parit käsin. (Korhonen, Malmi, Saikkonen 2001)

Scheme-robo hyödyntää TRAKLA-järjestelmän ydintä kurssin hallinnointitehtävien käsittelemisessä. Hallinnointitehtäviä ovat esimerkiksi lähetettyjen tehtävien käsittely, opiskelijoiden saamien pisteiden laskeminen ja tilastotietojen laskeminen. (Korhonen, Malmi, Saikkonen 2001)

3.4.5 Kielestä riippumaton ohjelman tarkastaminen

Japanissa ShiZuokan yliopistossa on tehty prototyyppi järjestelmästä, joka arvioi ohjelman kielestä riippumattoman algoritmin kuvauksen eli *standardialgoritmin* (standard algorithm) avulla. Järjestelmä luokittelee ohjelmat sellaisiin, joita opettajan ei tarvitse tarkastaa, ja sellaisiin, jotka vaativat tarkastamisen. Näin opettajan ei tarvitse tarkastaa kaikkia palautettuja ohjelmia ja hänen aikaansa säästyy. (Itoh et al. 2001)

Itoh et al. (2001) sanovat järjestelmän arvioinnin perustan muodostuvan niin sanotusta *standardialgoritmista*, eli algoritmin kuvauksesta, jonka opettaja määrittelee. Prototyypissä algoritmin kuvaus pitää kirjoittaa tietyn ohjelmointikielen operaatioilla. Standardialgoritmia ja opiskelijan tarkastettavaa ohjelmaa verrataan toisiinsa tarkastusvaiheessa siten, että järjestelmä vertaa sopivatko ohjelmien operaatiot toisiinsa, eli onko niillä sama toiminta. Jotta kaikkia mahdollisia ohjelman operaatioita ei tarvitsisi verrata toisiinsa, järjestelmä käyttää apuna Pascal-kielen joukkorakenteita ja *yhteensopivuusluokittelun* (matching rate) arvioituja arvoja.

Järjestelmässä opettajan pedagogiset päämäärät on jaettu kahteen eri luokkaan, päämääriin, jotka ovat ohjelmointikielestä riippuvia ja niihin, jotka eivät riipu kielestä. Käytännön ohjelmoinnin opetuksessa on tavallista, että opettaja pyrkii opettamaan opiskelijoille kielestä riippumattomia algoritmeja, tietorakenteita ja niiden operaatioita. Ohjelmia pitäisi Itohin et al. (2001) mukaan arvioida siitä näkökulmasta, onko abstraktit tietorakenteet toteutettu oikein, riippumatta toteutuksen yksityiskohdista. Tämän vuoksi valmiin järjestelmän pitäisi pystyä hyväksymään kielestä riippumattomilla abstraktien tietorakenteiden operaatioilla toteutettuja ohjelmia. Lisäksi sen pitäisi kyetä vertaamaan abstraktia algoritmia konkreettiseen, jollakin tietyllä kielellä toteutettuun ohjelmaan.

3.5 Vertailu

Tässä kohdassa vertailen Boss-, CourseMaster-, WebAssign- ja RoboProf-järjestelmien keskeisiä piirteitä. Ominaisuudet on aluksi lueteltu alla olevassa taulukossa, jonka jälkeen taulukon pohjalta on tehty lyhyt yhteenveto. Pari tietoa on jäänyt puuttumaan, koska kaikkia tietoja ei ole käynyt ilmi lähteistä.

Taulukko 1: Vertailu Boss-, CourseMaster-, WebAssign- ja RoboProf-järjestelmistä.

	Boss	**CourseMaster**	**WebAssign**	**RoboProf**
Kurssin hallinta	On	On	On	On
Tarkastuksen automaattisuus	Puoliautomaattinen	Automaattinen, tarkastaja voi muokata.	Automaattinen	Automaattinen
Käyttöjärjestelmä	Alustariippumaton	Unix/Windows WWW-pohjainen versio.	Web-pohjainen	Web-pohjainen, Sun Solaris-palvelin.
Tietoturva	SSL-salaus, salasanat, rajoitetut oikeudet.	Salasanat, rajoitetut oikeudet.	Salasanat	Salasanat
Opetusmateriaali verkossa	Ei	On	Ei	On
Arvioitavat tehtävätyypit	Ohjelmointitehtävät ja esseet.	Monivalintakysymykset, esseet, tietokoneohjelmat jne.	Monivalintakysymykset, esseet, tietokoneohjelmat jne.	Ohjelmointitehtävät
Plagioinnin tarkastus	On	On	-	On
Tehtävien kielet	Useita (miltei mikä vain).	Java, C++	Useita	C++
Palaute opiskelijalle	Tarkastajan palaute tarkastamisen yhteydessä.	Välitön (palautepuu).	Välitön	Välitön, mutta niukka.
Arviointikriteerit	Automaattiset testit ja ohjelman tulokset yhdessä. Opettaja tai tutor voi myös muuttaa arvosanoja käsin.	Dynaaminen oikeellisuus, typografia ja ohjelman ominaisuudet.	Automaattiset tarkastusmoduulit	Ohjelman tulokset
Toteutuskieli	Java	Java	-	Java
Etäopiskelu mahdollista	On	On	On	On

Arviointikriteerit tarkoittavat tässä yhteydessä sitä, mitä ominaisuuksia ohjelmasta tarkastetaan ja minkä perusteella numerot annetaan. Yleisin arviointitapa on ajaa ohjelmaa ja arvioida sen palauttamia tuloksia. Kaikissa vertailluissa järjestelmissä arviointi suoritetaan ainakin osittain automaattisesti. Jos tarkastaminen tapahtuu täysin automaattisesti heti lähetyksen jälkeen, järjestelmä antaa tehtävästä välitöntä palautetta opiskelijalle. Yleensä tarkastaja tai tutor kuitenkin kirjoittaa tarkempaa palautetta opiskelijalle joko tarkastaessaan tehtäviä tai tutkiessaan annettuja arvosanoja.

Boss-järjestelmä on alustariippumaton. Sitä on testattu niin Windows- kuin UNIX-puolellakin. CourseMaster toimii UNIX-käyttöjärjestelmässä, mutta siitä on tehty myös WWW-versio. Muut kaksi järjestelmää ovat WWW-pohjaisia, eivätkä ne vaadi käyttäjältä muuta kuin Javaa tukevan WWW-selaimen toimiakseen. WWW-pohjaisuus on helpottanut myös opetusmateriaalin jakelua verkon kautta. Verkossa oleva opetusmateriaali on helposti opiskelijoiden saatavilla ja opettajan päivitettävissä.

Kurssinhallinta vie suurilla opiskelijamäärillä paljon aikaa. Automaattinen kurssin hallinnointi onkin tärkeä keino helpottaa kurssista vastuussa olevan henkilön työtaakkaa ja se kuuluu olennaisena osana jokaiseen edellä mainittuun järjestelmään. Myös tietoturva on tärkeä osa automaattista tarkastamista, sen avulla voidaan estää asiattomien pääsy luottamukselliseen tietoon. Kaikissa järjestelmissä on käytössä ainakin salasanat ja rajoitetut oikeudet eri käyttäjäryhmille.

Arvioitavat tehtävätyypit vaihtelevat automaattisesti tarkastettavista monivalintakysymyksistä käsin korjattaviin esseisiin ja tietokoneohjelmiin. CourseMaster- ja WebAssign -järjestelmissä on tuettu useanlaisia eri tehtävätyyppejä, kun taas RoboProf-järjestelmä on tarkoitettu pelkästään ohjelmointitehtäviä varten. Boss-järjestelmässä ohjelmointitehtävien lisäksi on arvioitu essee-tehtäviä, etupäässä siksi, että niille on voitu suorittaa plagiointitarkastus (Joy 2002).

Eri ohjelmointikieliä on automaattisissa koodin tarkastusjärjestelmissä paljon, esimerkiksi CourseMaster-järjestelmässä kursseja on tehty useille eri kielille, kuten Java- ja C++ -kielille. Boss-järjestelmässä kursseja voitaisiin tehdä käytännössä mille tahansa kielelle, mutta Boss-

järjestelmän uudempi, tarkastuksessa käytetty, paradigma soveltuu vain Java-tehtävien automaattiseen tarkastamiseen (Joy 2002).

Nykyään automaattiset arviointiohjelmat toteutetaan yleensä Java-kielellä ja esimerkiksi CourseMaster- ja Boss-järjestelmät on tehty kokonaan uudelleen Javalla, koska entinen useilla eri kielillä toteutettu järjestelmä oli käynyt vaikeasti ylläpidettäväksi. Java-kielellä on monia, jo edellä mainittuja etuja, kuten alustariippumattomuuden tukeminen ja sopivan jakelumenetelmän tarjoaminen RMI-mekanismin avulla (Foxley et al. 2001), sekä mahdollisuus ladata ohjelmamoduuleja dynaamisesti (Daly 1999).

Useat automaattiset järjestelmät sisältävät plagioinnin tarkastuksen ja sitä yleensä myös käytetään, esimerkiksi Nottinghamin yliopistossa plagioinnin tarkastus on ollut rutiiniosa arviointia. Plagioinnin tarkastaminen ei tapahdu kuitenkaan täysin automaattisesti, vaan tehtävien tarkastaja joutuu vertailemaan testauksen löytämiä epäilyttäviä opiskelijapareja käsin (Foxley 1997). Lisäksi plagioinnin tarkastus ei sellaisenaan sovellu kaikille kursseille, koska varsinkin alkeiskursseilla ohjelmointitehtävät voivat olla hyvin lyhyitä ja opettaja on voinut antaa niille valmiin rungon. Plagioinnin tarkastamisesta aiheutuukin tarkastajalle lisää työtä, mutta toisaalta se lisää arvioinnin luotettavuutta ja toimii pelotteena opiskelijoille (Wise 1992).

Boss ja CourseMaster soveltuvat myös etäopiskeluun, mutta se vaatii tiettyjä toimenpiteitä onnistuakseen. WebAssign-järjestelmää on käytetty etäopiskelussa Fernin etäyliopistossa Hagenissa (Brunsmann et al. 2000). Myös RoboProf-järjestelmä soveltuu hyvin etäopiskeluun, koska sen käyttämiseen riittää pelkkä Internet-selain ja -yhteys (Daly 1999).

4. AUTOMAATTISEN KOODIN TARKASTAMISEN TOTEUTUSTEKNIIKOITA

Koodin automaattisessa tarkastamisessa pyritään mittaamaan tarkastettavan ohjelman yleistä laatua. Ohjelman laadun mittaamisessa useimmiten käytettyjä tekniikoita ovat mallivastauksen ja opiskelijan työn antamien tuloksien vertaaminen toisiinsa tai tiettyjen ohjelmistometriikoiden soveltaminen (Rowe 2001). Käsittelen luvussa ensin koodin automaattista tarkastamista yleisellä tasolla, jonka jälkeen siirryn Ceilidh/CourseMaster-järjestelmän ja Boss-järjestelmän tapoihin tarkastaa koodia.

4.1 Ohjelman laadun arvioiminen

Foxley ja Zin (1994) sanovat ohjelman laadun arvioimisen olevan tärkeä, mutta vaikea tehtävä. Laatua voidaan arvioida joko mittaamalla ohjelman yleistä laatua, tai etsimällä ohjelmasta kohtia, joissa on korjaamista. Yleisin tapa mitata ohjelman laatua on tarkastella sellaisia ohjelman osia, jotka vaikuttavat ohjelman laatuun ja yhdistää niistä saadut arviot. Nämä laatuun vaikuttavat tekijät voidaan jakaa karkeasti kolmeen eri kategoriaan, joita ovat *ohjelman operaatiot* (product operations), *ohjelman revisiot* (product revision) ja *ohjelman muutokset* (product transition). Ohjelman operaatioihin kuuluvat oikeellisuus, luotettavuus, tehokkuus, yhtenäisyys ja käytettävyys. Ohjelman revisioihin sisältyvät ylläpidettävyys, joustavuus ja testattavuus. Ohjelman muutokseen lukeutuvat siirrettävyys, uudelleenkäytettävyys ja yhteistoimivuus (interoperability). Eri tekijöiden suhteellinen painoarvo vaihtelee eri ympäristöissä.

4.2 Ohjelman oikeellisuuden ja tehokkuuden mittaaminen

Foxleyn ja Zinin (1994) mukaan käytetyimmän ohjelman oikeellisuuden mittaamisen metodi on ohjelman testaaminen. Ohjelman oikeellisuuden testaaminen sisältää ohjelman suorittamisen ja sitä voi käyttää myös tehokkuuden mittaamisessa. Ohjelman testaaminen voidaan jakaa kahteen tyyppiin, joita ovat staattinen analyysi ja dynaaminen testaus.

4.2.1 Ohjelman testaaminen

Foxley ja Zin (1994) määrittelevät *staattisen analyysin* ohjelman lähdekoodin tutkimiseksi ilman sen suorittamista. Ohjelman lähdekoodin rakenne ja syntaksi sekä staattiset virheet tarkastetaan ja ohjelmasta tuotetaan tilastolliset tiedot. Myös kääntäminen on staattisen analyysin muoto, mutta termiä käytetään yleensä niistä toiminnoista, joiden avulla etsitään erilaisia virheitä tai mahdollisia virheen tiloja. Näitä toimintoja ovat muun muassa ikuiset silmukat, lauseet joita ei käsitellä tai tilat, jotka aiheuttavat ristiriitoja, sekä väärin muodostetut sisäkkäiset silmukat ja käyttämättömät muuttujat.

Dynaaminen testaus tarkoittaa ohjelman ajamista testidataa käyttäen. Sen päätarkoitus on löytää ajonaikaiset virheet. Ideaalitapaus olisi testata ohjelmaa kaikilla mahdollisilla syötteillä, mutta se on mahdotonta jokaisessa oikeassa tapauksessa. Käytännössä testaus voidaan suorittaa vain pienellä syötteiden osajoukolla ja siksi testidata on valittava harkiten. Testidatan valitsemiseen on olemassa erityisiä tekniikoita. (Foxley, Zin 1994)

Ohjelman ajaminen tuottaa *tulosteita.* Yleensä testausprosessit perustuvat *oraakkelin* käyttämiseen. Oraakkeli on mekanismi, joka tarkastaa tuloksien oikeellisuuden. Oraakkelin on ymmärrettävät kaikki oikeat tulokset, joiden muoto on erilainen. Siksi oraakkelin tekeminen ohjelmille, joissa ohjelman antamien tuloksien muoto ei ole tarkasti spesifioitu, ei ole triviaalia. Oraakkelin tekemistä voi helpottaa esimerkiksi testaamalla pelkkiä proseduureja koko ohjelman testaamisen sijaan. (Foxley, Zin 1994)

Oraakkeli tunnistaa, sisältääkö annettu teksti jonkin nimenomaan vaaditun merkityksen. Ceilidhissä ja CourseMaster-järjestelmässä oraakkelia käytetään moneen tarkoitukseen, sen avulla tarkistetaan dynaamisten testien antamien tulosten oikeellisuus tai joidenkin tiettyjen ominaisuuksien sisältyminen arvioitavaan ohjelmaan, sekä tarkastetaan monivalintakysymyksiä. (Foxley, Zin 1996) Ohjelman dynaamisesta testaamisesta saadaan ohjelman suorituksesta tietoa, jota kutsutaan *ohjelman profiiliksi* (program profile). Ohjelman profiili on hyödyllinen ohjelmoijalle, sitä voidaan käyttää suoritettujen koodin segmenttien optimoimiseen, löytämään dynaamisesti kuollut koodi tai laskemaan silmukan toistojen määrä. (Foxley, Zin 1994)

Ohjelman testausprosessi sisältää ohjelman kääntämisen, staattisen analyysin ja ohjelman suorittamisen testidataa vasten erikseen jokaiselle testidatasarjalle, tulosten vertaamisen odotettuihin tuloksiin ja tulosten analysoinnin ohjelman profiilista. (Foxley, Zin 1994)

4.2.2 Ohjelman oikeellisuuden mittaaminen

Ohjelman testaamisen päätarkoitus on paljastaa ohjelmassa olevat virheet. Yksi tapa mitata oikeellisuutta on käyttää verifiointia. Verifiointi voidaan jakaa kahdeksaan eri tasoon. Ohjelman laatu on sitä parempi, mitä korkeammalla tasolle ohjelma yltää. Verifioinnin tasot näkyvät taulukossa 2. Toinen tapa mitata ohjelman oikeellisuutta on asettaa jokaiselle testausprosessin toiminnalle sopiva painotus ja oma arvosana. (Foxley, Zin 1994)

Taulukko 2: Verifioinnin tasot.

Taso 1	Ohjelma ei sisällä syntaksivirheitä
Taso 2	Ohjelma ei sisällä käännösvirheitä, tai järjestelmän löytämiä virheitä ajon aikana.
Taso 3	Ohjelma antaa oikean tuloksen jollakin testidatasarjalla.
Taso 4	Ohjelma antaa oikeat tulokset usealla testidatasarjalla ajettaessa.
Taso 5	Ohjelma antaa oikeat vastaukset harkiten valituilla, vaikeilla testidatasarjoilla ajettaessa.
Taso 6	Ohjelma antaa oikeat vastaukset kaikilla mahdollisilla testidatasarjoilla, jotka ovat valideja ja vastaavat ohjelman määrittelyjä.
Taso 7	Ohjelma antaa oikeat vastaukset kaikilla valideilla testidatasarjoilla kaikissa todennäköisissä tapauksissa väärällä syöttöarvoilla annettuna.
Taso 8	Ohjelma antaa oikeat vastaukset kaikilla mahdollisilla syöttöarvoilla.

4.3 Ylläpidettävyyden mittaaminen

Foxleyn ja Zinin (1994) mukaan ohjelmiston ylläpidettävyys on laaja-alainen toiminto, joka edellyttää ylläpidettävän ohjelman ymmärtämistä. Tästä johtuen yleisin tapa mitata ohjelman ylläpidettävyyttä on mitata sen ymmärrettävyyttä. Ohjelman ymmärrettävyyttä mittaamaan on

kehitetty erilaisia malleja, kuten *monimutkaisuusmallit* (complexity model), *vaikeusmallit* (program difficulty model) ja *ohjelmointityylimallit* (programming style model). Monimutkaisuusmalli perustuu Van Verthin ajatukseen, jonka mukaan ohjelman ymmärrettävyys on käänteisessä suhteessa sen monimutkaisuuteen. Monimutkaisuusmalleja ovat muun muassa M. Halstead'in mitat ja T. McCabe'n *syklomaattinen luku* (cyclomatic number).

Vaikeusmalli yrittää analysoida miten ohjelman dynaaminen osa kontrolloi staattisia elementtejä. Jokaisen ohjelman elementin vaikeus voidaan esittää määräämällä painotus jokaiselle syntaktiselle elementille, joita ovat muun muassa parametrit, muuttujat ja taulukot. Mallin esittäjä on G. M. Bern, jonka vaikeusmalli on nimeltään *ylläpidon analysointityökalu* (The Maintenance Analysis Tool). (Foxley, Zin 1994)

Ohjelmointityylimalli pyrkii arvioimaan ohjelman ymmärrettävyyttä ohjelman tyylin perusteella. Paremman tyylin omaavan ohjelman ajatellaan mallissa merkitsevän luettavampaa ohjelmaa. M. J. Rees oli ensimmäinen, joka kuvasi Pascalin lähdekoodin tyyliluokittelua kymmenen eri toimijan avulla. Näitä toimijoita olivat rivin pituuden keskiarvo, kommentointi, sisennykset, muuttujien pituus, nimiöiden ja hyppy-lauseiden käyttö, tyhjät rivit, välilyönnit, modulaarisuus, varatut sanat ja muuttujien nimet. Toinen ohjelmointityylimalli on Redishin ja Smythin kehittämä malli, jossa ohjelman tarkastaminen perustuu niin sanottuun malliohjelmaan. (Foxley, Zin 1994)

4.4 Ceilidh-järjestelmä

Ceilidh- ja CourseMaster -järjestelmissä automaattinen koodin tarkastaminen tapahtuu kolmen työkalun avulla. Näitä ovat *dynaaminen testaus* (Dynamic Testing), *ominaisuuksien tarkastus* (Feature Tool) ja *typografinen analyysi* (Typographic Tool) (Rawles 2001). Näiden lisäksi CourseMasterissa on omat työkalunsa vuokaavioiden, olioperustaisten suunnittelukaavioiden ja virtapiirien tarkastamiseen (Higgins, Symeonidis, Tsintsifas 2002).

4.4.1 Ceilidhin ja CourseMasterin arviointityökalut

Ohjelman ominaisuuksien mittaaminen on yksi Ceilidh-järjestelmän automaattisen arvioinnin osa. Ohjelman ominaisuuksia mittaava työkalu antaa pisteitä, kun tietty merkkijono löytyy tarkistettavasta ohjelmasta. Lisäksi se antaa palautetta merkkijonojen löytymisen perusteella. (Rawles 2001)

Dynaaminen testaus tarkoittaa opiskelijan lähettämän ajettavan ohjelman ajamista opettajan määrittelemillä testidatasarjoilla (Foxley, Zin 1994). Dynaaminen testaus ottaa talteen tarkastettavan ohjelman tulokset ja käyttää testejä ja oraakkelissa määriteltyä dataa tarkastaessaan ohjelman oikeellisuutta. Jokaiseen testiin voidaan liittää useita merkkijonoja odotetuille tuloksille painotuksella ja kuvauksella varustettuina. (Rawles 2001)

Typografia on kolmas Ceilidh-järjestelmän automaattisen arvioinnin osa. Ceilidhissä käytetään seuraavaksi esiteltyjä ominaisuuksia hyvän ohjelman perusolemuksen löytämiseksi. *Merkkien lukumäärän keskiarvo per rivi.* Rivillä ei saa olla liikaa tai liian vähän merkkejä, Ceilidhissä 15 - 30 merkkiä on arvioitu kohtuulliseksi määräksi yhdellä rivillä. (Gibbon 2002)

Tyhjät rivit koodissa mahdollistavat sen luettavuuden, mutta liiallisesta koristelusta rangaistaan. 15 - 30 % tyhjiä rivejä ohjelman koodista on hyvä määrä. Myös riveillä olevat *välit* huomioidaan arvioinnissa. Välilyöntejä pitäisi sijoittaa operaattoreiden ja pisteiden ympärille. Suositeltava määrä on, että 10 - 30 % merkeistä keskeltä laskettuna pitäisi olla merkitöntä tilaa, kun edessä ja lopussa olevaa tilaa ei oteta huomioon. (Gibbon 2002)

Muuttujien nimillä on merkitystä koodin ymmärrettävyyden kannalta. *Muuttujien pituuden keskiarvo* on yksi typografiaa mittaava suure. Muuttujien pitäisi olla informatiivisia ja kuvaavia ja niiden keskimääräisen pituuden pitäisi olla 5 - 10 merkkiä. Toinen muuttujista mitattava asia on *hyvän mittaisten muuttujien nimien prosenttiarvo,* jonka mukaan 70 % muuttujien nimistä pitäisi olla sopivan mittaisia. (Gibbon 2002)

Ohjelman riveistä 10 - 60 % pitäisi olla kommenttirivejä. Tätä mitataan suureella *kommenttirivien % -määrä koodista,. Kommentteina oleva prosenttiosuus merkeistä* taas mittaa kommentissa olevaa tekstiä. Kaikista merkeistä 10 - 60 % pitäisi olla kommentteja. Lisäksi s*isennykset* olisi oltava oikealla kohdalla, esimerkiksi aaltosulkujen pitää olla täsmätty. (Gibbon 2002)

CourseMaster-järjestelmässä voidaan tarkastaa ohjelmakoodin lisäksi myös vuokaavioita, olioperustaisia kaavioita ja virtapiirejä. Vuokaavioiden automaattinen tarkastaminen tapahtuu *vuokaaviotyökalun* (Flowchart Tool) avulla. Vuokaaviotyökalu muuntaa opiskelijan tekemän vuokaavion BASIC-lähdekoodiksi ja syöttää sen dynaamiseen testaukseen, joka tarkastaa vuokaavion. *Olioperustainen suunnittelukaavio* (OO Tool) testaa opiskelijan suunnittelukaavion täydellisyyden ja oikeellisuuden kaavion luokkien ja suhteiden osalta. *Virtapiirin simulointi* (Circuit Simulation Tool) testaa opiskelijan loogista virtapiiriä simuloimalla sitä. (Higgins, Symeonidis, Tsintsifas 2002)

4.5 Boss-järjestelmä

Boss-järjestelmässä palautettujen tehtävien tarkastamiseen käytetään erilaisia metriikoita ja ohjelman antamien tulosten vertaamista oikeisiin tuloksiin. Boss-järjestelmässä automaattinen tarkastaminen on mahdollista tehdä kahdella eri paradigmalla, joista uudempaa voi soveltaa vain Java-ohjelmia tarkastettaessa.

4.5.1 Automaattiset testit

Boss antaa ajaa ohjelmamuodossa olevia lähetyksiä ja testata tiedostoja automaattisesti arviointi-tarkoitukseen. Myös opiskelijat voivat testata ohjelmakoodiaan yhdellä testidatalla ennen lähetystä. (Boss 2002) Boss-järjestelmää on käytetty muun muassa Pascal-, C++ ja UNIX-kuoriohjelmointi-kursseilla (Joy, Luck 1998b), mutta sitä voidaan ainakin teoriassa käyttää lähes mihin tahansa ohjelmointikielen tarkastamiseen (Joy 2002).

Boss-järjestelmässä on automaattiselle tarkastamiselle käytössä kaksi eri paradigmaa, joita ovat *tekstisyöte ja -tuloste -paradigma* (Text Input and Output Paradigm) sekä s*yöte ja tuloste objekteina -paradigma* (Input and Output as Objects Paradigm). Tekstisyöte ja -tuloste -paradigma on näistä vanhempi.

Boss kirjoitettiin alun perin ohjelmointimoduulien hallintaan ja tarkastamiseen. Alkuperäinen Boss toimi UNIXissa. Automaattiset testaukset määriteltiin tekstinä ja tekstejä verrattiin odotettuun tulosteeseen käyttämällä UNIXin hyötyohjelmia (esimerkiksi diff), tai käsin koodattuja kuori-ohjelmia. Tästä menettelystä seurasi ongelmia muun muassa tyhjän tilan, kontrollimerkkien ja oikeinkirjoituksen kanssa. BOSS 2 -järjestelmässä vanhempaa paradigmaa käytetään vielä muiden kuin ohjelmointitehtävien tarkastamisessa (Boss 2002). Automaattisten testien tekemisestä vanhemmalla tavalla on kerrottu tarkemmin Boss-järjestelmän testaamisen yhteydessä seuraavassa luvussa.

Syöte ja tuloste objekteina -paradigma käsittelee syötettä ja tulostetta objekteina ja sitä voidaan käyttää vain Javalla tehtyjen ohjelmien automaattiseen tarkastamiseen. Paradigmassa käytetään metodia equals() verrattaessa opiskelijan toteuttamaa metodia oikeaan vastaukseen, esimerkiksi Java-ohjelmointitehtävälle voidaan määritellä syöte, tulos ja metodi. Ohjelman syöte määritellään luokkana C_{input}, ja ohjelman tulos määritellään vastaavasti luokkana C_{output}. Metodiin $_{method}$ lisätään määrittely C_{output} method(C_{input}). (Boss 2002)

Tämän jälkeen määritellään automaattinen testi: objekti O_{input}, joka on luokan C_{input} ilmentymä ja objekti O_{output}, joka on luokan C_{output} ilmentymä niin, että m(O_{input}) palauttaa arvon O_{output}. Opiskelijan toteutusta metodista method voidaan testata lauseella (m(O_{Input}).equals(O_{output}). (Boss 2002)

Boss-järjestelmä käyttää ToString()-metodia näyttämään dialogia sisältävät testin tulokset. Tarvittaessa equals() -metodi voidaan ylikirjoittaa. Paradigma ei määrittele kuinka syöte ja tuloste pitäisi esittää, ja siksi opettaja voi tarjota I/O -koodin valmiina mallina, tai antaa opiskelijoiden luoda oma I/O -koodinsa. (Boss 2002)

Testauksessa ohjelmaa ajetaan useilla eri testidatoilla ja verrataan vastaavatko saadut tulokset määrittelyjä kokonaan tai osittain. Ohjelmakoodia voidaan myös mitata metriikoilla, joita ovat esimerkiksi modulaarisuus, kommentointi, sisennykset ja niin edelleen. Metriikoiden avulla kerätään konkreettista tietoa ohjelman yleisestä ohjelmointityylistä, vaikka ne eivät tuotakaan täysin eksaktia tietoa. Edellinen testaustapa muodostaa yleensä suurimman osan testausprosessista ja toista tapaa käytetään vain tarkentamaan arvosanaa. Kyseessä olevan kurssin tavoitteista riippuu, kuinka paljon tarkastuksessa painotetaan oikeellisuutta ja kuinka paljon ohjelmointityyliä. (Joy, Luck 1998a)

5. BOSS- JA COURSEMASTER-JÄRJESTELMIEN TESTAUS

Boss- ja CourseMaster -järjestelmien testaamisessa on tarkoituksena tutkia järjestelmien soveltuvuutta virtuaaliopintojen ohjelmointikursseille. Keskeisiä kysymyksiä ovat miten järjestelmät toimivat ja mitä mahdollisuuksia niihin sisältyy, mitä ongelmia niiden käyttöönotossa voi tulla esiin ja mitä puutteita järjestelmissä on. Tarkoituksena oli testata kummassakin järjestelmässä 3 - 4 erilaista ohjelmointitehtävää, jotka sisältävät muun muassa toisto- ja if-lauseita. Yhdestä tehtävästä palautin useita erityyppisiä ratkaisuja ja tarkastelin, minkä tyyppisissä ohjelmoinnin harjoituksissa järjestelmät toimivat. CourseMaster-järjestelmän testaus toteutuikin suunnitelmien mukaisesti, mutta Boss-järjestelmässä en testannut kuin kahta tehtävää, koska ohjelman antamat tulokset olivat erilaisia erityyppisissä vastauksissa eivätkä näin ollen meneet läpi Bossin automaattisista testeistä.

Joensuun yliopiston virtuaaliappro-opinnoissa käytetään WebCT-järjestelmää, joka painottuu kurssin hallintaan, sekä Jeliot-ohjelmaa, jota käytetään visualisoimaan ohjelmointitehtäviä. Opiskelijoiden palauttamat tehtävät on tarkastettu käsin eikä tarvetta kovin tarkkoihin ja rajoitettuihin tehtävänantoihin ole ollut. Tämän takia opiskelijoiden vastaukset saattavat poiketa toisistaan huomattavasti.

5.1 CourseMaster-järjestelmä

CourseMasterin käyttöliittymä on selkeä ja helposti ymmärrettävä. Opiskelijan liittymässä ohjelmointitehtävän etsiminen käy nopeasti ja tehtävän tekemisen aloittamiseen riittää parin painikkeen painaminen. Koko tehtävän tekeminen ja palauttaminen tapahtuu järjestelmän sisällä. CourseMaster-järjestelmän mukana tulee kurssi nimeltään Java.course, joka on johdantokurssi Java-ohjelmointiin. Siitä on paljon hyötyä järjestelmää testattaessa tai käyttöön otettaessa. Ensinnäkin se toimii esimerkkinä siitä, millaisia tehtäviä järjestelmässä on mahdollista automaattisesti arvioida ja helpottaa siksi huomattavasti uusien tehtävien suunnittelua. Toiseksi se nopeuttaa ja helpottaa huomattavasti järjestelmään tutustumista.

Uusien tehtävien tekeminen vie melko paljon aikaa, koska useita eri tiedostoja joudutaan muokkaamaan tai kirjoittamaan kokonaan alusta. Suurin osa tiedostoihin tehtävistä muutoksista on kuitenkin melko yksinkertaisia, varsinkin jos kopioitu tehtävä ja uusi tehtävä ovat saman tyyppisiä. Vaativampien tehtävien tekemisessä saattaa tosin tulla vaikeuksia, erityisesti jos dynaamiset testit ovat monimutkaisia, niin kuin esimerkiksi taulukkojen määrittelemisessä. Monissa tiedostoissa riittää pelkkä tehtävän nimen vaihtaminen. Eniten aikaa menee varmaankin tehtävän määrittelyn ja malliratkaisun suunnitteluun. Itse en joutunut keksimään uusia tehtäviä, vaan sain käyttööni muutaman Joensuun yliopiston virtuaaliopinnoissa käytetyistä Java-tehtävistä ja niihin annettuja erilaisia vastauksia.

Kokeilin neljän eri tehtävän tekemistä CourseMaster-järjestelmään. Ensimmäinen niistä oli yksinkertainen muunnos esimerkkikurssin mukana tulleesta printAddress.java-tehtävästä. Muut tehtävät olivat virtuaaliappro-opinnoissa käytettyjä tehtäviä ja niiden nimet olivat Min.java, Kauppa.java ja Rooli.java (malliratkaisut ovat liitteissä 3, 4 ja 5). Näiden ohjelmien toimintaa ja testausta on käsitelty myöhemmin tässä luvussa.

Tehtävänanto on kirjoitettava tarkasti ja esimerkki odotetuista syötteistä ja tulosteista on annettava. Tämä tekee dynaamisten testien suunnittelemisen järjestelmään helpommaksi. Muussa tapauksessa opiskelija saattaa esimerkiksi tehdä ohjelman, joka pyytää käyttäjää tekemään jonkin valinnan syöttämällä kirjaimen sijasta numeron, tai toisin päin. Annettava testidata ei tällöin käy ohjelmalle, eikä testi mene läpi.

CourseMaster on joustava järjestelmä siinä mielessä, että se antaa arvosanan myös osittain väärästä vastauksesta. Tehtävästä annettavaan yleisarvosanaan vaikuttavat käännöksen virheetön suoritus, typografia, dynaamiset testit ja ohjelman ominaisuudet. Järjestelmä antaa oman arvosanan jokaisesta edellä mainitusta osa-alueesta ja muodostaa yleisarvosanan niiden perusteella. Eri osa-alueiden suhteellista painoarvoa yleisarvosanaan voidaan muokata.

Itse tarkastuprosessia voi muokata haluamakseen. Järjestelmän antamaa palautetta voi käsitellä tarkastamista määriteltäessä ja opiskelijalle annettavan automaattisen palautteen voi kirjoittaa suomeksi. Se, kuinka monesti opiskelija voi lähettää vastauksen samaan tehtävään, on myös

rajoitettavissa. Tarkastusprosessissa ohjelman eri osille, kuten piirteille, typografialle ja dynaamisille testeille voi antaa haluamansa painoarvon. Opiskelija saa tarkastuksesta täydet pisteet, vaikka tehtävän vastaus olisikin erilainen, jos tehtävänantoa vain on noudatettu tarkasti ja ohjelma on toimiva.

CourseMaster-järjestelmää ei saatu toimimaan täysin oikein yliopistolla, mikä tuotti joitakin ongelmia testaukseen. Minun täytyi poistua vastauksen tallentamisen jälkeen ohjelmasta ja kääntää tekemäni java-tiedosto käsin. Tämän jälkeen avasin taas CourseMasterin ja lähetin tekemäni ohjelman. Lähetyksen jälkeen painoin *Mark*-painiketta ja järjestelmä näytti minulle arvosanan sisältävän palautepuun. Oikein toimiessaan CourseMaster kääntää tiedostot itse, eikä ohjelmasta tarvitse välillä poistua. En voinut myöskään testata vastauksia, jotka eivät menneet kääntäjästä läpi, koska tarvittava class-tiedosto puuttui.

5.1.1 Uuden kurssin ja tehtävien luominen

Uudet kurssit luodaan melko matalalla tasolla tekemällä tiedostojen kokoelmia palvelimen tiedostojärjestelmään (Rawles 2001). Yksinkertaisin tapa tehdä uusi kurssi on kopioida esimerkkinä tullut kurssihakemisto uudella nimellä hakemistoon \cms\courseArea\system, joka on kaikkien kurssien juurihakemisto. Kurssihakemistoilla on hierarkkinen rakenne, jossa kurssi-hakemiston alla ovat moduulien ja tehtävien hakemistot, yksi kutakin tehtävää kohden, esimerkiksi Java.coursen alla on unit-päätteisiä moduuleja (unit), kuten u6.unit, joiden alla on exercise-päätteisiä tehtävähakemistoja, kuten bsort.exercise. (CourseMaster 2001)

Uusien tehtävien tekeminen vaatii useita vaiheita. Helpoin tapa lisätä uusia tehtäviä on kopioida aluksi jonkin vanhan tehtävän tiedostot ja tehdä niihin tarvittavat muutokset. Osaa vanhoista tiedostoista ei välttämättä tarvitse editoida, osaa tiedostoista on editoitava ja osa niistä on kirjoitettava kokonaan alusta asti (CourseMaster 2001). Järjestelmän mukana tulee vaiheittaiset ohjeet tehtävien tekemiseksi, jotka ovat suomennettuna liitteessä 1.

Taulukossa 3 olevien tiedostojen tehtävänä on määritellä yksityiskohtaisesti tiettyyn ohjelmointitehtävään liittyvät asiat, kuten tehtävän ominaisuudet, tehtävänanto, tehtävän

malliratkaisu ja -runko, testidatat ja niiden tulokset sekä tehtävästä annettava palaute. Tiedostot on jaoteltu Bsort-tehtävästä suoraan kopioitaviin ja editoitaviin tiedostoihin, sekä uutta tehtävää määritellessä uudelleen kirjoitettaviin tai kääntämällä luotavin tiedostoihin. Suoraan kopioitavia tiedostoja ovat *mark.ft.compile*, joka sisältää palautepuun virheilmoitukset, *mark.tv*, jossa on määritelty palautepuun typografiset viheilmoitukset ja *mark.scale*, jossa on palautepuun arvosanat pisteiden mukaan sekä *mrun.bat*, joka ajaa mark.java-tiedoston. Suoraan kopioitavat tiedostot ovat liitteessä 2, eikä niitä käsitellä sen tarkemmin tässä tutkielmassa.

Taulukko 3: Bsort-tehtävän tiedostot

Suoraan kopioitavat tiedostot:	**Editoitavat tiedostot:**	**Uudelleen tehtävät tiedostot**:	**Kääntämällä luotavat tiedostot:**
UserInput.java mark.tv mark.ft.compile mark.scale mrun.bat	Mark.java Properties.txt Setup.txt Save.txt Mark.makefile	mark.testdata bsort.java mark.dv mark.oracledata question.txt bsort.java.SOL mark.ft	bsort.class mark.class UserInput.class

Seuraavaksi käsittelen tehtävän automaattisen tarkastamisen määrittelyä CourseMaster-järjestelmässä ja käyn läpi Rooli-tehtävään liittyvät tiedostot ryhmiteltyinä editoitaviin ja täysin uudelleen kirjoitettaviin tiedostoihin. Sen jälkeen esittelen testauksesta saamiani tuloksia tarkemmin.

5.1.2 Tiedostoihin tehtävät muutokset rooli-tehtävässä

Rooli-tehtävässä opiskelijat harjoittelevat arvojen tallentamista taulukkoon ja niiden tulostamista. Ohjelma laskee taulukkoon hahmojen maantieto, mahti, magia ja selviytymisarvot. Ohjelma tulostaa taulukon arvot ja sanallisen arvion hahmon selviytymistaidoista. Käyttäjältä ei kysytä mitään arvoja suorituksen aikana. Rooli-tehtävän malliratkaisu on liitteessä 3.

Tarkastusta tehdessä editoitavia tiedostoja ovat *mark.java, properties.txt, setup.txt, save.txt ja mark.makefile*. Mark.java-tiedosto sisältää tehtävän tarkastuskaavion. Mrun.bat-tiedosto kääntää mark.java-tiedostosta mark.class-nimisen binääritiedoston. (CourseMaster, 2001) Käytännössä tiedostosta joutuu muuttamaan tiedoston alkuun paketin nimen (kurssi.moduuli.tehtävä), tehtävän nimen tarvittaviin kohtiin ja halutessa sanallisen arvostelun voi kääntää englannista suomeksi tai muuttaa tarkastuksen pisterajoja. Kuvassa 8 on osa mark.java-tiedostosta. Tiedoston editoidut kohdat on selvyyden vuoksi kirjoitettu lihavoidulla tekstillä ja kolme pistettä ilmaisee välistä puuttuvaa koodia.

```
package Java.oma.Rooli;
…
// the feedback range. We will use this for the Typography tests
// You don't have to specify a feedback range. There is a default one (equal to the next one)
String[] strRange = {"Huono", "Heikko", "Keskitaso", "Ok", "Hyvä", "Oikein hyvä", "Täydellinen"};
int[]    intRange = {   40   ,   50 ,    60    ,   70 ,   80 ,    90    ,    100    };

// Another feedback range. We will use this for the Dynamic tests.
String[] strRangeDyn = {"Huono", "Heikko", "Hyvä", "Täydellinen"};
int[]    intRangeDyn = {  40    ,   50 ,   80 ,    100    };

// Another feedback range. We will use this for the Feature tests.
String[] strRangeFea = {"Huono", "Saisi olla parempi", "Hyvä", "Täydellinen"};
int[]    intRangeFea = {  70    ,   80 ,   90 ,    100    };

// COMPILE
…
TmarkingResult tmTypog = execute(new TypographicCMD("mark.tv","Rooli.java"));
TmTypog.setWeight(20);
TmTypog.setFeedbackRange(strRange, intRange);

TmarkingResult tmDynamic = execute(new DynamicAllCMD("Rooli.java"));
…
// execute one dynamic command with the 1rst test data set
TmarkingResult tm2 = execute(new DynamicCMD("Rooli.java",1)); tm2.setWeight(20);

TmarkingResult tmFeat = execute(new FeaturesCMD("mark.ft","Rooli.java"));
tmFeat.setWeight(30);
tmFeat.setFeedbackRange(strRangeFea, intRangeFea);

MarkingCompositeResult mcr = new MarkingCompositeResult("Yleisarvosana");
mcr.addChild(tmCompile); …
```

Kuva 8: Mark.java-tiedosto.

Kuvassa 9 on properties.txt-tiedosto, jossa määritellään tehtävän ominaisuuksia. Kurssin tyyppi on ohjelmointikursseilla arvo PROGRAMMING ja projektin tyyppi on esimerkiksi Java-kurssilla PFE-editoria käytettäessä PFEJAVA. Muita ominaisuuksia ovat ohjelman aikakatkaisu (PROG_TIMEOUT), tehtävien lähetysten maksimimäärä (MAX_SUB), painotus (WEIGHT) ja automaattinen tarkastus (AUTO_MARKING), sekä tehtävän tila (STATE), joka voi olla avoin (OPEN), suljettu (CLOSED) ja myöhässä (LATE). Tiedoston viimeinen rivi tarkoittaa tehtävän rungon nimeä. (CourseMaster 2001)

COURSE_TYPE	PROGRAMMING
PROJECT_TYPE	PFEJAVA
PROG_TIMEOUT	15
MAX_SUB	10
PROG_MAXOUTLINES	500
WEIGHT	5
AUTO_MARKING	TRUE
STATE	OPEN
SKELETON_NAME	**Rooli.java**

Kuva 9: Properties.txt-tiedosto.

Setup.txt-tiedostossa on annettu niiden tiedostojen nimi ja tarkenne, jotka kopioidaan palvelimelta käyttäjän työalueelle asiakassovellukseen. (CourseMaster 2001) Tässä tapauksessa asiakassovellukseen kopioidaan Rooli.class-tiedosto ja ohjelmassa käytetyn Lue-apuluokan class-tiedosto. Setup.txt-tiedosto on kuvassa 10.

```
:Rooli.java:UserPreprocessor:
:Rooli.class:
:Lue.class:
:save.txt:
@end@
```

Kuva 10: Setup.txt-tiedosto.

Save.txt on ASCII-muotoinen tiedosto, joka sisältää ne tehtävän tiedostot, jotka haetaan ja tallennetaan CourseMaster-järjestelmän sisällä asiakassovelluksesta palvelimelle. Nämä ovat tiedostoja, jotka CourseMaster tarkastaa (CourseMaster 2001). Save.txt-tiedosto on kuvassa 11.

Rooli.java
Rooli.class

Kuva 11: Save.txt-tiedosto.

Kuvassa 12 on Rooli-tehtävän mark.makefile-tiedosto. Tiedostossa annetaan Rooli.java-tiedoston kääntämiskäsky. Tämä tapahtuu ennen kuin Rooli.java-tiedostosta on muokattu opiskelijalle annettava tehtävän runko.

Rooli.class :
Javac **Rooli.java**

Kuva 12: Mark.makefile-tiedosto.

Seuraavaksi käsitellyt tiedostot on kirjoitettava kokonaan itse uutta tehtävää tehdessä. Näistä tiedostoista ensimmäinen on Rooli.java-tiedosto, jossa on opiskelijalle valmiiksi annettu tehtävän runko. Testauksen suunnitelija voi itse määritellä millaisen rungon opiskelijoille antaa. Rooli.java-tiedosto sisältää aluksi malliratkaisun, josta on käännetty Rooli.class-tiedosto ja joka on kopioitu Rooli.java.SOL-tiedostoksi. Tämän jälkeen runko tehdään editoimalla Rooli.java-tiedostoa. Rooli.java-tiedosto on kuvassa 13.

```
// Rooli-tehtävän runko
public class Rooli
{
  public static void main(String [] args)
  {
       //Alusta arvot tässä
    {
     }
    //Tulosta taulukon arvot tässä
  }
}
```

Kuva 13: Rooli.java-tiedosto.

Rooli-tehtävän Mark.dv-tiedosto on kuvassa 14. Mark.dv -tiedostoa käytetään dynaamisten testien määrittelyyn. Tiedoston jokaista riviä kohden määritellään dynaamisten testien sarja (CourseMaster 2001). Kuvassa kummankin dynaamisen testin arvo vaikuttaa arvosanaan yhtä paljon, koska molemmilla riveillä on luku 50. Ensimmänen testi etsii ohjelmasta tiettyjä sanoja, jotka määritellään mark.oracledata-tiedostossa. Toinen testi etsii taulukon antamia tuloksia.

```
50: Dynaaminen testi (testaa tekstiä):
50: Dynaaminen testi (testaa taulukon tuloksia):
```

Kuva 14: Mark.dv-tiedosto.

Question.txt-tiedosto sisältää tehtävän kysymyksen, nimeää ohjelmassa käytettävät metodit, sanoo kuinka monesti opiskelija voi lähettää vastauksensa ja näyttää arvioinnin painottumisen eri arviointityökalujen välillä. Lisäksi tiedostossa on esimerkki ohjelmalle annettavasta tyypillisestä syötteestä ja tulosteesta. Rooli-tehtävän question.txt-tiedosto on kuvassa 15.

Coursemaster kurssi Java yksikkö oma tehtävä Rooli: Roolipeli (Painotus: 5)

==========

====

Kysymys

========

Jatketaan edelleen Roolipeli-ohjelmaa. Lisää nyt ohjelmaan hahmojen arvojen tulostus taulukkoon. Lisäksi tulosta hahmon selviytymisarvo sanallisesti seuraavan kaavan mukaan: Selviytymisarvo: Kuvaus selviytymistaidosta: 1-2 huono, 3-6 hyvä, 7-10 erinomainen. Vihje: käytä if-else if-rakennetta tulostukseen.

Metodi

======

-

Huomautus

=====

Voit lähettää ratkaisusi enintään 10 kertaa.

Arviointi

==========

20% : Dynaaminen oikeellisuus

30% : Typografia

50% : Ohjelman ominaisuudet

Tyypillinen syöte

==========

-

Tyypillinen tuloste

==========

Hahmojen kyvyt taulukossa:

Nimi	Juuso	Eetu
Hahmo	velho	samooja
Maantieto	5	6
Mahti	3	3
Magia	6	3
Selviytymisarvo	hyvä	hyvä

Kuva 15: Question.txt-tiedosto.

Rooli-tehtävän Mark.oracledata -tiedostossa (kuvassa 16) määritellään kaksi eri testiä, jotka on erotettu toisistaan viidellä @-merkillä. Ensimmäisessä testissä etsitään ohjelman tulostuksesta sanoja "hahmojen", "nimi", "hahmo", "maantieto", "mahti", "magia" ja "selvitymisarvo". Luvut riveillä tarkoittavat rivin painoarvoa testin arvosanaan. Hakasulut ensimmäisien kirjaimien ympärillä tarkoittavat, että kirjain voi olla joko suuri tai pieni kirjain. Viimeinen testi etsii ohjelman tulostamasta taulukosta numeroita 5, 1, 3, 2 ja 6. Merkit @end@ ilmaisevat tiedoston loppumista.

```
@@@@
10: [Hh]ahmojen: Etsi tuloksista "Hahmojen": "Hahmojen" löytyi: "Hahmojen" EI löytynyt:
10: [Nn]imi: Etsi tuloksista "Nimi": "Nimi" löytyi: "Nimi" EI löytynyt:
10: [Hh]ahmo: Etsi tuloksista "Hahmo": "Hahmo" löytyi: "Hahmo" EI löytynyt:
10: [Mm]aantieto: Etsi tuloksista "Maantieto": "Maantieto" löytyi: "Maantieto" EI löytynyt:
10: [Mm]ahti: Etsi tuloksista "Mahti": "Mahti" löytyi: "Mahti" EI löytynyt:
10: [Mm]agia: Etsi tuloksista "Magia": "Magia" löytyi: "Magia" EI löytynyt:
10: [Ss]elviytymisarvo: Etsi tuloksista "Selviytymisarvo": "Selviytymisarvo" löytyi: "Selviytymisarvo" EI löytynyt:
@@@@@
50: 0[^0-9]+5[^0-9]+1[^0-9]+3[^0-9]+2[^0-9]+6: Etsii tuloksista taulukon numeroita: Tulos on oikein: Tulos ei ole oikein:
@@@@@
@end@
```

Kuva 16: Mark.oracledata-tiedosto.

Kuvassa 17 on rooli-tehtävän mark.ft-tiedosto. Tiedosto määrittelee rooliohjelman koodin ominaisuuksia. Käytännössä sen avulla voi etsiä ohjelmasta tiettyjä sanoja ja tarkastaa, että vastauksessa on käytetty tehtävänannossa määritellyn tyyppisiä lauseita. Rooli-tehtävän vastauksista on etsitty sanojen “static”, “if”, “else” ja “println” käyttöä.

```
10: static: Etsi "static": "static" Löytyi: "static" Ei löytynyt:
10: if: Etsi "if": "if" Löytyi: "if" Ei löytynyt:
10: else: Etsi "else": "else" Löytyi: "else" Ei löytynyt:
10: println: Etsi "println": "println" Löytyi: "println" Ei löytynyt:
```

Kuva 17: Mark.ft-tiedosto.

5.1.3 Rooli.java-tehtävän testaaminen

Taulukossa 4 on verrattu CourseMaster-järjestelmän automaattisesti antamia arvosanoja virtuaaliappro-opiskelijoiden tekemistä erilaisista rooli-tehtävän ratkaisuista. Kaikki testatut ratkaisut menivät läpi kääntäjästä. Malliratkaisu tarkoittaa kurssin opettajan kyseiseen tehtävään tekemää esimerkkiratkaisua, kuvassa 18 on tästä malliratkaisusta saatu palautepuu. Tehtävän malliratkaisusta CourseMaster antoi täydet pisteet kaikista testauksen osa-alueista.

Rooli-tehtävästä CourseMaster tarkasti käännösvirheet ja varoitukset, typografian, dynaamiset testit ja ohjelman ominaisuudet. Dynaamisissa testeissä arvioinnin kohteena olivat tulostuksessa sanat “hahmojen”, “nimi”, “hahmo”, “maantieto”, “mahti”, “magia” ja “selvitymisarvo”, sekä taulukon tulostamat arvot 5, 1, 3, 2 ja 6. Rooli-ohjelman ominaisuuksista CourseMaster tarkisti, onko koodissa käytetty sanoja ”static”, “if”, “else” ja ”println”.

CourseMaster-järjestelmä antoi myös Rooli_A.java-vastaukselle yleisarvosanan A. Sulkumerkkien sisennyksessä oli puutteita, mutta se ei kuitenkaan yksinään laskenut typografista arvosanaa. Dynaamiset testit eivät ole menneet täysin läpi, koska taulukon antamat tulokset eivät ole olleet täysin oikein ja sanat “hahmojen”, “nimi” ja “selviytymisarvo” eivät löytyneet ohjelman tuloksista. Opiskelija on korvannut sanan selviytymisarvo sanalla selviytyminen, eikä saanut siitä pisteitä. Ohjelman ominaisuuksista vastaus on saanut täydet pisteet, eli etsityt sanat löytyivät koodista.

Rooli_B-vastausta ajettaessa tuli virheilmoitus. Typografisissa testeissä sulkumerkkejä ei oltu sisennetty oikein ja ohjelmassa oli kommentoimattomia sulkumerkkejä. Dynaamiset testit eivät

menneet läpi, koska mitään etsityistä sanoista ei löytynyt tuloksista. Ohjelman ominaisuudet olivat kuitenkin kohdallaan. Vastauksen kokonaisarvosana oli B.

Kolmas, eli Rooli_C.java-vastaus toimi oikein, mutta sen tulostusmuoto oli vääränlainen. Typografian testauksessa merkkejä rivillä oli liikaa keskiarvon mukaan, sulkumerkkien oikeassa sisennyksessä on virheitä, eikä sulkumerkkejä ole kommentoitu. Dynaamisista testeistä tulos on F, koska mitään sanoista ei löydetty ja taulukosta ei löytynyt oikeita tuloksia. Ohjelma antoi eri numeroarvot taulukkoon ja tulostuksessa ei oltu käytetty tarkastuksessa etsittyjä sanoja. Toisin sanoen, oikein tehty ja toimiva ohjelma ei välttämättä saa parasta arvosanaa, jos se ei käytä samoja arvoja kuin tehtävänannossa on annettu, eikä ole tulostusasultaan odotetun kaltainen.

Taulukko 4: Rooli-tehtävän arvosanat.

Rooli.java	**Yleisarvosana** (General Grade)	**Kääntäminen** (Compilation Tool)	**Typografinen arvosana** (Typographic Tool)	**Dynaamiset testit** (All Dynamic Tests)	**Ohjelman ominaisuudet** (Feature Tool)
Malliratkaisu	A	A	A	A	A
Rooli_A-java	A	A	A	F	A
Rooli_B.java	B	A	B	F	A
Rooli_C.java	B	A	B	F	A

Kuva 18: Rooli-tehtävän malliratkaisun palautepuu.

5.1.4 Tiedostoihin tehtävät muutokset min-tehtävässä

Min.java-ohjelmassa opiskelijat harjoittelevat muuttujien ja vakioiden käyttöä. Ohjelma pyytää käyttäjää antamaan ajan sekunteina ja lukee käyttäjän antaman luvun lue.java-apuluokkaa apuna käyttäen. Sen jälkeen ohjelma muuntaa käyttäjän syöttämän sekuntimäärän minuuteiksi ja sekunneiksi. Ohjelma tulostaa vastauksen println-lauseella. Min-tehtävän malliratkaisu on liitteessä 4.

Min-tehtävä eroaa edellä käsitellystä rooli-tehtävästä siten, että siinä käyttäjä syöttää ohjelmalle arvoja, jotka vaikuttavat ohjelman antamiin tuloksiin. Ohjelmalle annettavat syötteet kirjoitetaan mark.testdata-tiedostoon ja syötteitä vastaavat odotetut tulokset mark.oracledata-tiedostoon. Kuvassa 19 on tehtävän mark-testdata -tiedosto, jossa nämä kolme testiä on määritelty ja kuvassa 20 on vastaava mark.oracledata-tiedosto.

Min-tehtävässä ohjelmalle on määritelty kolme erillistä testiä. Ensimmäisessä testissa ohjelmalle syötetään luku 250, joka vastaa käyttäjän ohjelmalle antamaa sekuntimäärää. Tämän

sekuntimäärän perusteella Mark.oracledata-tiedostossa etsittään lukuja 4 ja 10, jotka tarkoittavat ohjelman antamaa oikeaa tulosta 4 minuuttia ja 10 sekuntia. Lisäksi vastauksesta etsitään tekstejä “minuuttia” ja “sekuntia”. Toisessa testissä tarkastetaan, että ohjelma laskee oikein myös alle minuutin olevat vastaukset. Kolmas testi testaa, toimiiko ohjelma oikein, jos käyttäjä antaa syötteeksi negatiivisen luvun.

```
@@@@@
250
@@@@@
10
@@@@@
-30
@@@@@
@end@
```

Kuva 19: Mark.testdata-tiedosto.

```
@@@@@
50: 4: Etsi tuloksista "4": "4" löytyi: "4" EI löytynyt:
50: 10: Etsi tuloksista "10": "10" löytyi: "10" EI löytynyt:
@@@@@
40: 0: Etsi tuloksista  "0": "0" löytyi: "0" EI löytynyt:
40: 10: Etsi tuloksista "10": "10" löytyi: "10" EI löytynyt:
30: [Ss]ekuntia: Etsi tuloksista "sekuntia": "sekuntia" löytyi: "sekuntia" EI löytynyt:
30: [Mm]inuuttia: Etsi tuloksista "minuuttia": "minuuttia" löytyi: "minuuttia" EI löytynyt:
@@@@@
40: -30: Etsi tuloksista  "-30": "-30" löytyi: "-30" EI löytynyt:
40: -135: Etsi tuloksista  "-135": "-135" löytyi: "-135" EI löytynyt:
30: [Ss]ekuntia: Etsi tuloksista  "sekuntia": "sekuntia" löytyi: "sekuntia" EI löytynyt:
30: [Mm]inuuttia: Etsi tuloksista  "minuuttia": "minuuttia" löytyi: "minuuttia" EI löytynyt:
@@@@@
@end@
```

Kuva 20: Mark.oracledata-tiedosto.

5.1.5 Min.java-tehtävän testaaminen

Min-tehtävässä CourseMaster tarkisti ohjelmasta käännösvirheet, typografian ja dynaamiset testit sekä ohjelman piirteet. Dynaamisissa testeissä tarkastettiin, antoiko ohjelma oikeat vastaukset edellä esiteltyihin testeihin. Ohjelman ominaisuuksia testattaessa tarkistettiin, onko ohjelmassa käytetty sanoja ”static” ”final” ja ”println”. Taulukossa 5 on verrattu CourseMaster-järjestelmän automaattisesti antamia arvosanoja Min.java-tehtävään saaduista erityyppisistä ratkaisuista. Malliratkaisusta saatu arvosana oli A. Kuvassa 21 on malliratkaisusta saatu palautepuu.

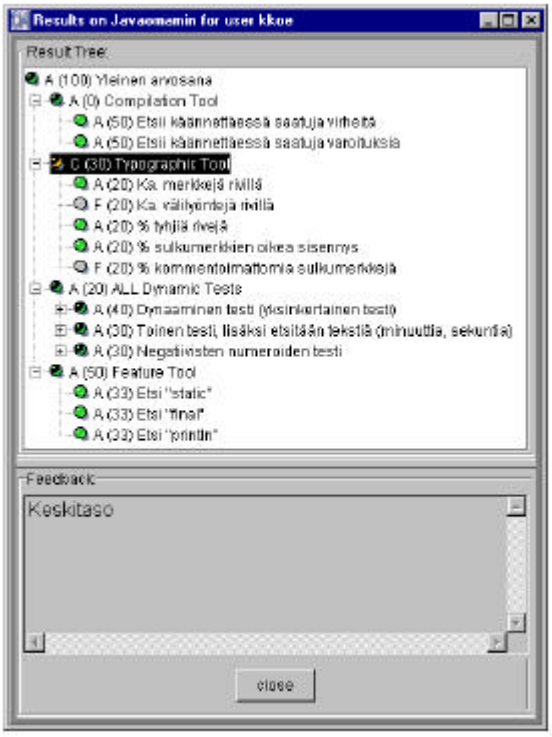
Min_A.java-vastaus poikkesi malliratkaisusta siten, että siinä oli käytetty switch case -rakennetta. Ohjelma toimi oikein ainoastaan tietyillä ohjelmakoodissa määritellyillä arvoilla. Koska ohjelma ei antanut kaikilla testisyötteillä oikeita tuloksia, dynaamiset testit eivät menneet läpi. Myös typografiassa oli puutteita, vastaus sai typografiasta arvosanaksi D:n, koska CourseMaster-järjestelmän mukaan ohjelmassa oli liian vähän välilyöntejä, loppusulkuja ei oltu kommentoitu ja sisennyksiä oli käytetty liian vähän. Ohjelman ominaisuuksia testattaessa arvosana oli A, if, static ja println -sanat löytyivät, mutta else-lause puuttui. Min_A.java-vastauksen kokonaisarvosanaksi muodostui B.

Vastauksesta Min_B.java CourseMaster antoi yleisarvosanaksi A:n. Vastauksen ainoat puutteet olivat typografiassa, välilyöntejä riviä kohden oli liian vähän ja kaarisulkujen sisennyksiä oli liikaa. Lisäksi loppusuluista puuttui kommentointi.

Min_C.java-vastaus laskee ajan väärin ja tulostaa pelkkiä sekuntteja. Tämän vuoksi mikään dynaamisista testeistä ei mennyt läpi tarkastuksesta. Ohjelmassa ei ollut käännösvirheitä, mutta typografian testauksessa välilyöntejä oli liian vähän riviä kohden, sulkuja oli sisennetty väärin ja ohjelmassa oli kommentoimattomia sulkumerkkejä. Ohjelman ominaisuuksia testattaessa static, final- ja prinln -sanat löytyivät ohjelmasta. Vastauksen yleisarvosana oli B, mikä on aika hyvä arvosana vääriä vastauksia antavalle ohjelmalle.

Taulukko 5: Min.java-tehtävän arvosanat.

Min.java	**Yleisarvosana** (General Grade)	**Kääntäminen** (Compilation Tool)	**Typografinen arvosana** (Typographic Tool)	**Dynaamiset testit** (All Dynamic Tests)	**Ohjelman ominaisuudet** (Feature Tool)
Malliratkaisu	A	A	A	A	A
Min_A.java	B	A	D	F	A
Min_B.java	A	A	C	A	A
Min_C.java	B	A	D	F	A



Kuva 21: Min-tehtävän malliratkaisusta saatu palautepuu.

Tehtävien tarkastus oli suunniteltu malliratkaisujen perusteella. CourseMaster antoi automaattisesta tarkastuksesta täydet pisteet kaikilla malliratkaisuilla, tosin lisäsin itse kommentoinnit sulkeviin lainausmerkkeihin. Olen jättänyt kauppa-tehtävän osuuden CourseMasterin testauksesta pois, koska siitä saamani tulokset olivat hyvin samantyyppisiä kuin edellisistä kahdesta tehtävästä saadut tulokset. Tehtävä oli lisäksi niin yksinkertainen, että

kaikkien vastauksien yleisarvosana oli A. Ainoat virheet kauppa-tehtävässä olivat typografiassa ja dynaamisissa testeissä. Oikein toimivista ohjelmista sai yleensä hyvän arvosanan. Uskon kuitenkin, että CourseMaster-järjestelmään tekemäni automaatinen tarkastus ei oikealla kurssilla toimisi täydellisesti, koska automaattiset testit eivät ota riittävän tarkasti huomioon kaikkia erityistapauksia, joita opiskelijoiden ohjelmat saattavat sisältää.

5.2 Boss-järjestelmä

Mike Joy asensi Joensuun yliopiston koneelle tällä hetkellä vielä kehityksen alla olevan version Boss-järjestelmästä. Palvelinohjelma on asennettu yliopiston palvelimelle ja ohjelman asiakassovellus RedHat 6.2 Linux -käyttöjärjestelmään testauksessa käytettävän koneen kiintolevylle. Asiakassovellus on asennettu erikseen kiintolevylle, koska Bossin painikkeet eivät suostuneet muuten näkymään oikein, virhe johtui kuulemma erilaisista fonteista. Koneen kiintolevyltä järjestelmä lisäksi toimi nopeammin.

Järjestelmään on lisättävä käyttäjiä, eli opiskelijoita ja henkilökuntaa, ennen kuin sitä voidaan alkaa käyttää. Uusien käyttäjien tekeminen on hieman hankalaa, koska tiedot pitää syöttää Boss-tietokannan *member*-tauluun sql-lauseen avulla. Lauseiden kirjoittaminen manuaalisesti tai vanhan tietokannan muuttaminen järjestelmän ymmärtämään muotoon vie aikaa.

Boss-järjestelmän testaaminen poikkesi CourseMasterin testaamisesta: virtuaaliappro-opinnoissa käytetyt tehtävät eivät sovellu Boss2-järjestelmän uudemman paradigman automaattisille testeille, koska niissä ei ole tarvittavaa I/O-koodia. Boss 1 -järjestelmän mukaiset automaattiset testit vain kääntävät tarkastettavan ohjelman ja tarkastavat ohjelman antamat tulokset. Ne eivät ota kantaa ohjelman sisäiseen toimintaan. Tarkastajan tehtävänä on tarkastaa ohjelman muut ominaisuudet. Koska kaikkien testattavien vastausten antamat tulokset olivat erimuotoisia, automaattiset testit eivät olisi menneet läpi. Jos taas kaikkien vastausten antamat tulokset olisi muutettu samanlaisiksi, kaikki kääntäjästä läpi menneet ohjelmat olisivat läpäisseet testit. Mike Joy teki automaattisen testin Kauppa.java-tehtävälle, jonka tekemisen käyn läpi seuraavaksi.

5.2.1 Kauppa-tehtävän tekeminen

Kauppa-tehtävässä ohjelma kysyy käyttäjältä, minkä tuotteen käyttäjä haluaa ostaa ja ilmoittaa hänelle, paljonko tuote maksaa, sekä kiittää käynnistä ja toivottaa tervetulleeksi uudelleen. Tässä vaiheessa asiakas voi ostaa vain yhden tuotteen kerrallaan. Käyttäjä voi ostaa 1. maitoa, 2. juustoa tai 3. makkaraa. Ostaminen tapahtuu antamalla tuotteen numero 1, 2, tai 3. Kauppa-tehtävän malliratkaisu on liitteessä 5.

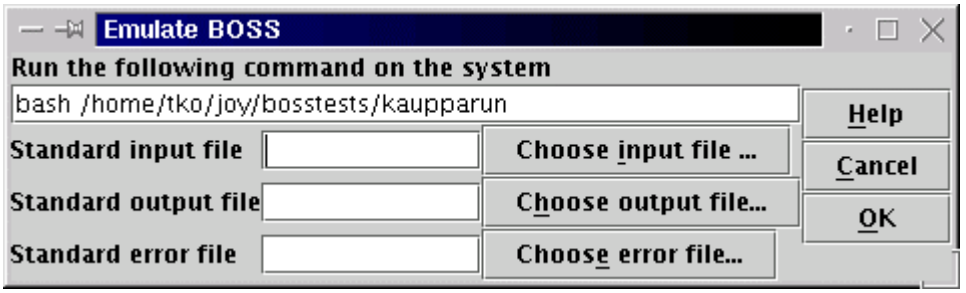
Automaattiset testit tehdään Boss-järjestelmässä lisäämällä tehtäville arviointikategorioita moduulin hoitajan oikeuksilla. Aluksi lisätään uusi tehtävä moduulin hoitajan asetukset sisältävässä *Course manager options* -ikkunassa. *Add Assessment Wizard -niminen* tehtävän lisäysvelho käynnistyy ja pyytää täyttämään ikkunoihin tehtävän ominaisuudet, kuten esimerkiksi nimen, tehtävän palautusajat, rakenteen, suhteellisen painotuksen ja java-tiedostojen kääntämisen. Näitä asetuksia voi muuttaa myöhemmin harjoitustehtävän editointi (Edit Assessment) -ikkunassa, jonka *Problems*-välilehdellä voi lisätä *arviointikategorioita* (Marking Category). Valitsemalla *automatic tests* ja painamalla *Emulate Boss 1*-painiketta, voidaan lisätä Boss1-järjestelmän automaattisia testejä.

Tämän jälkeen tehdään *kaupparun, kauppain* ja *kauppaout* -nimiset tiedostot. Kaupparun-tiedosto on kuvassa 22 ja siinä määritellään ohjelman kääntäminen sekä ajaminen. *Kauppain*-tiedoston sisältö on käyttäjän ohjelmalle antama syöte, joka tässä tapauksessa on numero 1. *Kauppaout*-tiedoston sisältö on tyhjä.

```
$WORKING_DIR
CLASSPATH=.:$CLASSPATH
Export CLASSPATH
/usr/j2se/bin/javac Kauppa.java
/usr/j2se/bin/java Kauppa
```

Kuva 22: Kaupparun-tiedosto.

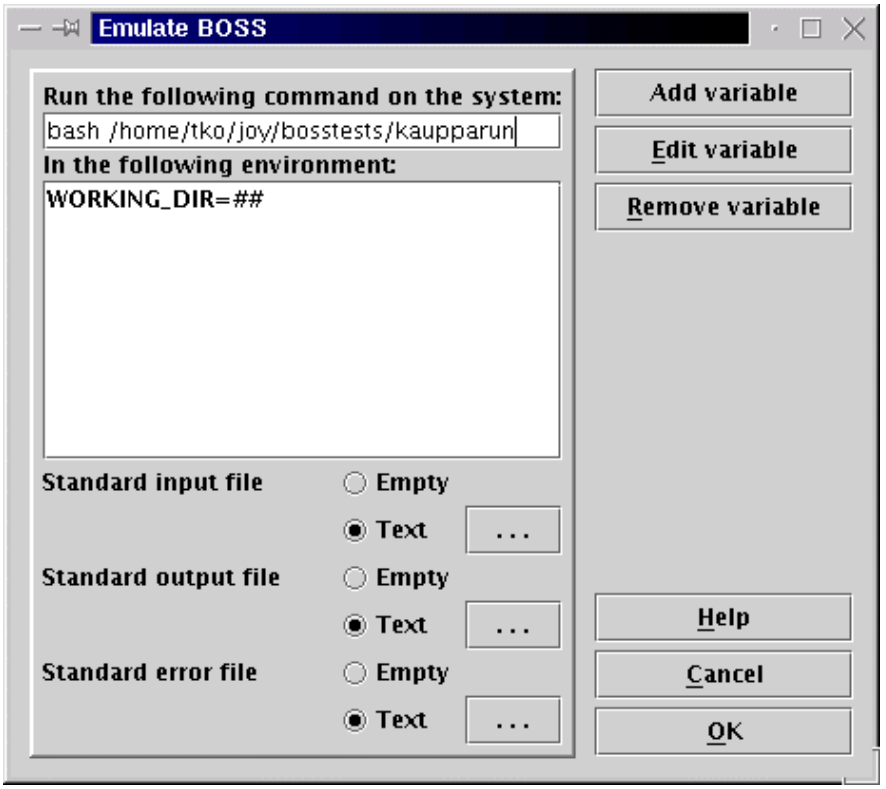
Emulate Boss -ikkunassa *Run the following command on the system*: -kohtaan kirjoitetaan komento: *bash /home/tko/joy/bosstests/kaupparun*, jossa /home/tko/joy/joy/bosstests/ on run-tiedoston hakemisto. Run-tiedosto voi olla missä tahansa hakemistossa palvelimella, mutta testauksen hoitavalle *Slave*-käyttäjälle on annettava tiedoston suoritusoikeus, joka pitää olla myös kaikilla run-tiedoston yläpuolella olevilla hakemistoilla. Valitaan *Standard input file*-kohtaan tiedosto kauppain ja *Standard output file* -kohtaan kauppaout-tiedosto. Viimeiseen, eli *Standard error file* - kohtaan, määritellään standardi virhe (Standard Error). Tässä tapauksessa tiedosto puuttu ja kohtaan voidaan kirjoittaa dev/null. Emulate Boss -ikkuna on kuvassa 23.



Kuva 23: Standardi-tiedostojen lisääminen.

Kuvassa 24 näkyvässä *Emulate Boss*-ikkunassa painike *Add Variable* avaa ikkunan, johon kirjoitetaan WORKING_DIR-niminen ympäristömuuttuja ja sen arvoksi laitetaan kaksi ##-merkkiä, mikä tarkoittaa satunnaisesti luotua väliaikaista tiedostoa. *Standard input file*- kohdasta valitaan radiopainike *Text* ja painetaan sen viereistä painiketta. Tekstilaatikkoon lisätään ohjelmaa testattaessa syötettävä tulos, joka on tässä tapauksessa numero 1. Tehdään samoin *Standard output file*- kohdassa ja tekstilaatikkoon lisätään ohjelmalta odotettu tulos, joka näkyy kuvassa 25, testiraportin kohdassa *Expected output*.

Tehtävien määrittely pitää tehdä hyvin huolellisesti ja yllättäviä ongelmia voi tulla UNIX:n rivin loppumerkkien kanssa, esimerkiksi ohjelmalta odotetun tuloksen lopussa viimeisellä rivillä pitää olla rivin lopetusmerkki. Boss-järjestelmässä pitäisi voida käyttää myös Java-appletteja, mutta niitä ei testattu Warwickin yliopistossa. (Joy 2002)



Kuva 24: Boss 1 -järjestelmän emulointi.

5.2.2 Kauppa-tehtävän lähettäminen

Boss-järjestelmän opiskelijakäyttöliittymä käynnistetään koneen kiintolevyltä, eikä yliopiston palvelimelta, sillä fontit eivät muuten näy ollenkaan ja yhteys on todella hidas käyttää. *Select a problem*-ikkunassa valitaan tehtävä, johon vastaus lähetetään. *Select files to submit* -ikkunassa valitaan lähetettävät tiedostot, eli Kauppa.java ja ohjelmassa käytetty Lue.java apuluokka. Jos apuluokan lähetys unohtuu, testausraportti antaa virheilmoituksen: error: cannot read: Kauppa.java 1 error. Painetaan *Test*-painiketta, jotta järjestelmä suorittaa testauksen ja avaa testausraportin. Kauppatehtävälle tehty automaattinen testi meni läpi malliratkaisulla. Boss-järjestelmän antama testausraportti on kuvassa 25.

Kuva 25: Kauppa-tehtävän testausraportti.

Kauppa-tehtävän automaattinen testi epäonnistui lähetettäessä Kauppa_A.java- ja Kauppa_B.java -vastaukset tehtävään. Testaus epäonnistuisi myös muilla käytettävissä olevilla testivastauksilla, koska yksikään niistä ei antanut kirjaimelleen täysin samanlaista tulostetta, kuten jo aikaisemmin todettiin. Kuvassa 26 on Kauppa_A.java-vastauksen saama testausraportin sisältö. Kuvasta käy ilmi testin epäonnistuminen ja se, miten ohjelmalta odotettu tulos (Expected output) ja sen antama tulos (Actual output) eroavat toisistaan.

One test
Expected output (stdout)
Tervetuloa Tanelin SuperMarkettiin!
****
1. Maitoa 3 mk (0,50 E)
2. Juustoa 16 mk (2,70 E)
3. Makkaraa 10 mk (1,70 E)
Tilataksesi tuotteen paina sen numeroa.
Maitoa 3 mk (0,5 E)
Kiitoksia kaynnista ja tervetuloa uudelleen!

Actual output (stdout)
Kaupasta voit ostaa:
1. Maitoa
2. Juustoa
3. Makkaraa
Mit? haluat ostaa?
Haluat ostaa maitoa ja se maksaa 3 mk!

Incorrect amount of output from submission: expected 254 bytes, but received 114 bytes.
Expected output (stderr)

Actual output (stderr)

Failed. Ending test.
The test failed.

Kuva 26: Testausraportin sisältö Kauppa_A-vastauksesta.

5.3 Järjestelmien soveltuvuus virtuaaliopintoihin

Harjoitustehtäviä ei oltu suunniteltu kumpaakaan testattavaa järjestelmää silmällä pitäen, mikä vaikeutti testausta ja vaikutti tehtävistä saatuihin tuloksiin. Jos näitä järjestelmiä olisi käytetty tehtävien arvioinnissa, tehtävänantojen olisi pitänyt olla tarkempia ja kummallekin järjestelmälle

erikseen määriteltyjä, niistä olisi esimerkiksi pitänyt käydä tarkasti ilmi, mitä ja missä muodossa ohjelmien pitäisi antaa tuloksiksi. Myös palautettavien tehtävien tyyliseikkoja, kuten oikeanlaista kommentoinnin ja sisennyksien käyttöä olisi pitänyt erikseen tehtäviä annettaessa opettaa. Toisin sanoen opiskelijoiden olisi pitänyt tietää, että myös kommentointi ja sisennykset otetaan arvioinnissa huomioon ja ne vaikuttavat osaltaan arvosanaan.

Käytännön testaamisen ja järjestelmän mukana tulevan esimerkkikurssin perusteella molemmat järjestelmät soveltuisivat virtuaaliappro-opintoihin. Tosin CourseMaster sopisi käytettäväksi vain, jos järjestelmä saadaan kääntämään opiskelijan tekemät ohjelmat automaattisesti, eikä sitä tarvitse tehdä itse ohjelman ulkopuolella. Järjestelmän mukana tulevan esimerkki-kurssin Java-tehtävät kävisivät sellaisinaan virtuaaliopintojen tehtäviksi.

CourseMasterissa menee Bossia enemmän aikaa tehtävien suunnitteluun, mutta tarkastaminen tapahtuu täysin automaattisesti ja siihen normaalisti käytetty aika säästetään. CourseMaster antaa myös välittömästi palautetta ohjelmasta ja opiskelija voi korjata ohjelmaansa. Tämä on hyödyllistä opiskelijalle, koska hänen mielenkiintonsa tehtävään säilyy paremmin. Toisaalta järjestelmän antama palaute on hieman "teknistä"; se ei auta opiskelijaa ymmärtämään ohjelman sisältöä, eikä sillä voi kokonaan korvata opettajan antamaa palautetta. Lisäksi Java-appletit eivät toimi CourseMaster-järjestelmässä, joten niitä ei enää voitaisi käyttää virtuaaliopinnoissa.

Boss-järjestelmässä hyvänä puolena olisi ihmisen antama palaute. Opiskelijalle voidaan myös tarjota testidata, jolla hän saa testata ohjelmansa toimintaa. Näin opiskelijalle ei jää epätietoisuutta ohjelman toimimisesta. Lisäksi tieto tulee kätevästi suoraan sähköpostiin, eikä sitä tarvitse erikseen hakea. Huono puoli palautteessa on, että se annetaan vasta myöhemmin tarkastuksen yhteydessä. Ohjelmasta saatava palaute tulee liian myöhään opiskelijan kannalta siinä mielessä, että hän ei enää voi korjata ohjelmaansa, vaan saa vain tietoa ohjelman puutteista ja virheistä. Boss-järjestelmä on lisäksi ilmainen, eikä sen käyttöötosta tulisi yliopistolle ylimääräisiä kuluja.

Mike Joyn mukaan (Joy 2002) Joensuun yliopiston virtuaaliappro-opinnoissa käyttämät tehtävät eivät suoraan sovellu testattavaksi Boss-järjestelmän uudemmalla testausparadigmalla. Boss 1 -

järjestelmää emuloimalla tehtäviä voidaan kuitenkin testata. Huonoa tässä on, että vanhemmasta paradigmasta on pyritty eroon sen aiheuttamien ongelmien takia. Jos uudempaa paradigmaa halutaan käyttää virtuaaliappron opinnoissa, on tehtäviä muutettava sisältämään input- ja output-koodit, jotta niitä voidaan verrata.

Toinen ongelma Boss-järjestelmän soveltumisessa virtuaaliapproon on, että se toteutetaan etäopetuksena, jolloin palomuuriin on tehtävä tarvittavat reiät. Asiakas-sovellus ja vähintään javan versio 1.3 on asennettava erikseen jokaiselle käytettävälle koneelle. Asiakas-sovellus piti asentaa erikseen erilaisten fonttien takia, osa painikkeista ja teksteistä ikkunoiden sisällä ei näkynyt kunnolla ilman erillistä asennusta. Asiakas-sovellus toimii onneksi myös Windowsissa. (Joy 2002)

Molemmissa järjestelmissä niiden ylläpito ja kurssien suunnittelu vaatisi osaavaa henkilökuntaa. Kummankin järjestelmien asiakassovellus, eli opiskelijan tarvitsema sovellus, pitäisi asentaa jokaiselle opiskeluun käytettävälle koneelle erikseen, mikä voi tuottaa ongelmia. Lisäksi yliopiston palomuuriin pitäisi tehdä reikiä, jotta opiskelijat pääsisivät kirjautumaan sisälle ohjelmiin.

6. YHTEENVETO

Mitä olisin voinut tehdä eri tavalla? Näin jälkeenpäin mietittynä tekisin tutkielmani hieman toisin. Ensinnäkin tekisin tutkielmasta hieman tiiviimmän paketin, aihe osoittautui niin laajaksi, että kaikkea ei voi saada mahtumaan yhteen pro gradu -tutkielmaan. Ensimmäisessä luvussa kirjoittaisin arvioinnin eri muodoista lyhyemmin ja erilaisista tehtävien tyypeistä enemmän. Ceilidhistä en varmaankaan kirjoittaisi niin tarkasti, vaan olisin keskittynyt CourseMasteriin. Olisin myös tutustunut tarkemmin Boss-järjestelmän uudempaan testausparadigmaan.

Järjestelmien toimimaan saamisessa oli yllättävän paljon ongelmia ja se vei paljon aikaa, vaikka apua haettiin sähköpostilla jopa järjestelmien kehittäjiltä asti. Aikaisempi testaaminen olisi helpottanut järjestelmistä kertovan luvun kirjoittamista ja antanut enemmän aikaa empiiriselle arvioinnille ja kokeilulle. Onneksi Professori Mike Joy Warwickin yliopistosta tuli Joensuun yliopistoon luennoimaan syyskuun alkupuolella. Hän asensi yliopiston palvelimelle uudemman version Boss-järjestelmästä. Sain häneltä myös paljon apua järjestelmän testaamisessa ja vastauksia kysymyksiini järjestelmän soveltuvuudesta virtuaaliappro-opintoihin.

Yksiselitteistä vastausta pro gradu -tutkielmani pääongelmaan, eli soveltuvatko Boss- ja CourseMaster -järjestelmät Joensuun yliopiston virtuaaliappro-opintoihin, on vaikea antaa. Toisaalta ne kyllä sopivat etäopiskeluun, mutta kummankin käyttöönotossa olisi omat ongelmansa. Käytännössä asia vaatii lisää tutkimista.

Tämän jälkeen olisi hyvä testata paremmin kyseessä olevia järjestelmiä, CourseMaster pitäisi saada toimimaan virheettömästi ja Boss-järjestelmän uusin versio pitäisi asentaa sen valmistuttua. Järjestelmiä, tai jompaa kumpaa niistä voisi myös kokeilla jollakin tietojenkäsittelytieteen ohjelmointikurssilla, joilla niiden käyttäminen voisi olla vapaaehtoista. Ennen tällaisen kurssin järjestämistä olisi hyvä tutustuttaa opiskelijat käytettävään järjestelmään ja tarjota heille apua sähköpostilla itse kurssin aikana. Tutoreiden käyttäminen kurssilla jättäisi luennoitsijalle aikaa järjestelmän käytön opetteluun, tehtävien ja niiden automaattisen tarkastamisen suunnitteluun ja kurssin hallinnointiin.

Vaikka Boss- ja CourseMaster -järjestelmät ovatkin kaksi tunnetuinta koodintarkastus-järjestelmää ja ne soveltuvat käytettäväksi virtuaaliappro-opintoihin Joensuussa, automaattisen koodintarkastusjärjestelmän valintaa ei tulisi rajoittaa pelkästään näihin kahteen järjestelmään. Kumpikin järjestelmä soveltuisi teknisesti paremmin pelkästään Joensuun yliopiston sisällä tapahtuvaan opetukseen kuin etäopiskeluun. Testattavaksi voitaisiin lisäksi ottaa jokin web-pohjainen, etäopiskelua varten suunniteltu järjestelmä, kuten esimerkiksi *VIOPE-oppimisympäristö* (Virtuaali Opetus), joka soveltuu ainakin Java- ja C-ohjelmoinnin opetukseen (Ageenko, Vihtonen 2002). Jos yliopistolla on aikomus suunnitella oma vastaavanlainen järjestelmänsä, kannattaisi pohtia eri järjestelmien etuja ja heikkouksia ja sitä, kuinka eri järjestelmien edut voitaisiin yhdistää toteutettavaan järjestelmään. Automaattisen koodin tarkastusjärjestelmän kehittäminen ei tapahdu nopeasti ja huolellisella etukäteen suunnittelulla voidaan välttää monta sudenkuoppaa.

CourseMaster-järjestelmässä annetaan opiskelijoille runko valmiiksi pohjaksi ja kannattaisi tutkia, miten opiskelijat suhtautuvat siihen. Valmis runko on annettu myös joissakin tehtävissä Joensuun yliopiston virtuaaliappro-opintojen ohjelmointikurssilla eikä siinä ole mitään uutta. Mitä merkitystä rungon antamisella on esimerkiksi oppimistuloksiin? Helpottaako rungon antaminen liikaa vai haittaako se luovuutta? Opiskelijoiden suhtautumista automaattisiin koodin tarkastamisjärjestelmiin pitäisi myös miettiä huolimatta siitä, että suhtautumista on tutkittu järjestelmien kokeilemisen yhteydessä.

Koodin automaattista arviointia kannattaa käyttää, jos kurssilla on paljon opiskelijoita tai heillä teetetään paljon ohjelmointiharjoituksia, tai heille halutaan antaa yksilöllisiä tehtäviä. Näin opettajalta jää enemmän aikaa opiskelijoiden varsinaiseen opettamiseen ja neuvomiseen. Etuna opiskelijoiden kannalta on että he voivat määrätä ajan ja paikan, jolloin he tekevät tehtäviä. Lisäksi tehtävien palautteen ja arvosanojen saaminen suoraan verkon kautta olisi vaivatonta.

Koodin automaattinen arviointi edellyttää opettajalta enemmän aikaa ohjelmointitehtävän laatimisessa ja arvioinnin määrittelyssä. CourseMaster- ja Boss -järjestelmissä malliratkaisun tekeminen ja testaaminen järjestelmässä vievät aikaa, CourseMasterissa aikaa kuluu erityisesti automaattisen tarkastuksen määrittelyyn ja Boss-järjestelmässä automaattisten testien tekemiseen

ja tehtävien manuaaliseen arviointiin. Itse järjestelmän käytön opiskeluun ja järjestelmälle soveltuvien ratkaisujen tekemään opettelu ei tapahdu yhdessä yössä. Lisäksi kummankin järjestelmän ylläpito voi olla hankalaa. Tämän vuoksi kannattaa pohtia tarkkaan, mikä määrä opiskelijoita kurssilla on oltava, että tarkastamisen automatisoiminen tuo enemmän hyötyä kuin haittaa ja kuinka paljon se helpottaa saman kurssin järjestämistä seuraavina vuosina.

LÄHDELUETTELO

Ageenko, E., Vihtonen, E.: VIOPE - Computer Supported Environment for Learning Programming Languages. *Proceedings of Technology of Information and Communication in education for engineering and industry*, November 13 - 15, Lyon, France, 2002.

Alexander, S.: Computer Assisted Assessment Tools, Resources and Articles. Internet WWW-sivu, URL: http://www.ulst.ac.uk/cticomp/CAA.html (27.9.2002).

Benford, S. D., Burke, E. K., Foxley E., Higgins, C. A.: Ceilidh: A Courseware System for the Assessment and Administration of Computer Programming. In Levonen, J.J, Tukiainen, M.T, *Proceedings of the Interdisciplinary workshop on Complex Learning in Computer Environments: Technology in Schools, University, Work and Life-long Education*. Technical Reports TOTY, University of Joensuu, 1994.

Boss-järjestelmä.:BOSS Online Submission System. Internet WWW-sivu, URL: http://www.dcs.warwick.ac.uk./boss (3.7.2002).

Bridgeman, S., Goodrich, M., Kobourov, S., Tamassia, R.: PILOT: An Interactive Tool for Learning and Grading. *ACM SIGCSE Bulletin*, **32**(1), 133 - 136, 2000.

Brunsmann, J., Homrighausen, A., Six, H.-W., Voss, J.: Assignments in a Virtual University - The WebAssign System. *Proceedings of the 19th World Conference on Open Learning and Distance Education*, June 20 - 24, Vienna, Austria, 1999.

Ceilidh: Ceilidh on the World Wide Web, 1999. Internet WWW-sivu, URL: http://www.cs.nott.ac.uk/~ceilidh (10.10.02).

Choate, J., Evans, S.: Authentic assessment of special learners: Problems or promise? *Preventing School Failure*, **37**(1), 1992.

CourseMaster: Setting Exercises under CourseMaster at Ngee Ann Polytechnic, käyttöohje, 1 - 27, 2001.

Culwin, F., Lancaster, T.: Plagiarism, Prevention, Deterrence & Detection. The institute for learning and teaching in higher education. South Bank University, UK, 2001b. Internet WWW-sivu, URL: http://www.ilt.ac.uk/resources/Culwin-Lancaster.htm (27.09.2002).

Culwin F., Lancaster, T.: Towards an Error Free Plagiarism Detection Process. *ACM SIGCSE Bulletin*, **33**(3), 57 - 60, 2001a.

Culwin, F., MacLeod, A., Lancaster, T.: Source Code Plagiarism in UK HE Computing Schools, Issues, Attitudes and Tools. JISC Source Code Plagiarism Report. Joint Information Systems Committee, South Bank University, London, 2001. Internet WWW-sivu, URL: www.jisc.ac.uk/pub01/southbank.pdf (27.09.02).

Daly, C.: RoboProf and an Introductory Programming Course. *ACM SIGCSE Bulletin*, **31**(3), 155 - 158, 1999.

Dalziel, J., Gazzard, S.: Next generation computer assisted assessment software: The design and implementation of WebMCQ. *Proceedings of 3 rd annual CAA Conference*, June 16 - 17, Loughborough, UK, 1999.

Dowsing, R.D., Long, S.: Evidence, Assessment Criteria and the Difficulty of Automated IT Skills Assessment. *Proceedings of 3 rd annual CAA Conference*, June 16 - 17, Loughborough, UK, 1999.

Foxley, E.: CourseMaster WWW Implementation, 2000. Internet WWW-sivu, URL: http://www.cs.nott.ac.uk/~cmp/CourseMaster (18.2.2002).

Foxley, E.: Policy on Plagiarism and Late Handing in of Work. LTR Technical Report, University of Nottingham, 1997. Internet WWW-sivu, URL: http://www.cs.nott.ac.uk/~ceilidh/papers/Plag.html (1.7.2002).

Foxley, E., Higgins, C. Gibbon, C.: Ceilidh-CourseMaster System An Introduction. *Monitor* **7**, December, 1996.

Foxley, E., Higgins, C., Hegazy, T., Symeonidis, P., Tsintsifas, A.: CourseMaster CBA System: Improvements over Ceilidh. *Proceedings of Fifth International Computer Assisted Assessment Conference*, July 2 - 3, Laughborough, UK, 2001.

Foxley, E. Higgins, C. Tsintsifas, A.: Security Issues in Ceilidh Software, Technical report. Learning Technology Research, Department of Computer Science, University of Nottingham.

Foxley, E., Higgins C., Tsintsifas, A., Symoniedes, P.: The Ceilidh-CourseMaster System An Introduction 1999, *The 4th Java in the Computing Curriculum Conference (JICC 4)*, January 24, 2000, SouthBank University, 1999.

Foxley, E., Zin, A.: The ”oracle” program, LTR Report, Department of Computer Science, University of Nottingham, 1996. Internet WWW-sivu, URL: http://www.cs.nott.ac.uk/~ceilidh/papers/Oracle.html (3.7.2002).

Foxley, E., Zin, A. M.: Analyse - An Automatic Program Assessment System, *Malaysian Journal of Computer Science*, **7**, 1994.

Gibbon, C.: Writing Typographically Sound Programs with Ceilidh. Internet WWW-sivu, URL: http://www.cs.nott.ac.uk/CourseMaster/more_info/html/Typog.htm (18.6.2002).

Haataja, A. Suhonen, J. Sutinen, E. Torvinen, S.: High School Students Learning Computer Science over the Web. *Interactive Multimedia Electronic Journal of Computer Enhanced Learning*, **3**(2), 2001.

Hagan, D., Lowder, J.: Web-based Student Feedback to Improve Learning. *ACM SIGCSE Bulletin*, **31**(3), 151 - 154, 1999.

Higgins, C., Symeonidis, P., Tsintsifas, A.: The Marking System for CourseMaster. *Proceedings of the 7th Annual Conference on Innovation and Technology in Computer Science Education*, June 24 - 26, Aarhus, Denmark, 46 - 50, 2002.

Itoh, Y. Konishi, T., Sakai, T., Suzuki, H.: Automated Evaluation of Learners' Programs by using Algorithm Representations Independent of Programming Languages. *Proceedings of the 9th International Conference on Computers in Education ICCE/Schoolnet 2001*, November 12 - 15, Seoul, Korea, 883 - 891, 2001.

Jackson, D., Usher, M.: Grading Student Programs using ASSYST. *ACM SIGCSE Bulletin*, **29**(1), 335 - 339, 1997.

Jones, E.: Grading Student Programs - A Software Testing Approach, *The Journal of Computing in Small Colleges*, **16**(2), 185 - 192, 2001.

Joy, M.: Keskustelu, 2002.

Joy, M., Luck, M.: A Secure On-line Submission System. *Software -- Practice and Experience*, **29**(8), 721 - 740, 1999a.

Joy, M., Luck, M.: Effective Electronic Marking for On-line Assessment. *Proceedings of the 3rd Annual Conference on Integrating Technology into Computer Science Education*, 134 - 138, 1998a.

Joy, M. Luck M.: On-line Submission and Testing of Programming Assignments. *Innovations in Computing Teaching*, 97 - 103, 1995.

Joy, M., Luck M.: Plagiarism in Programming Assignments. *IEEE Transactions on education,* **42**(2), 129 - 133, 1999b.

Joy, M., Luck, M.: The BOSS System for On-line Submission and Assessment. *Monitor On-line,* **10**, 27 - 29, 1998b.

Korhonen, A. Malmi, L. Saikkonen, R: Experiences in Automatic Assessment on Mass Courses and Issues for Designing Virtual Courses. *Proceedings of the 7th Annual Conference on Innovation and Technology in Computer Science Education,* June 24 - 26, Aarhus, Denmark, 55 - 59, 2002.

Korhonen, A. Malmi, L. Saikkonen, R: Fully Automatic Assessment of Programming Exercises. *ACM SIGCSE Bulletin,* **33**(3), 133 - 136, 2001.

Malmi, L.: Automaattinen tarkastaminen opetuksen apuvälineenä. *Tietojenkäsittelytiede-lehti,* **17** Toukokuu, 24 - 35, 2002.

McAlpine, M.: Principles of assessment, CAA Centre, 2002. Internet WWW-sivu: http://caacentre.lboro.ac.uk/dldocs/Bluepaper1.pdf (29.09.2002)

Nicholson, D.: Computer-Aided Assessment as a Holistic Learning Tool in Geoscience. *Proceedings of Fifth International Computer Assisted Assessment Conference*, July 2 – 3, Loughborough, UK, 2001.

Odekirk-Hash, E., Zachary, J.L.: Automated Feedback on Programs Means Students Need Less Help From Teachers. *ACM SIGCSE Bulletin,* **33**(1), 55 - 59, 2001.

O'Leary, R.: An Introduction to Computer Assisted Assessment. Learning Technology Support Service. Internet WWW-sivu, URL: http://www.ltss.bris.ac.uk/pdfs/CAA.pdf (29.07.2002).

Petre M., Price, B.: Teaching Programming through Paperless Assignments: an empirical evaluation of instructor feedback. *Proceedings of ACM SIGCSE/SIGCUE Conference on Integrating Technology into Computer Science Education*, Uppsala, Sweden, 94 - 98, 1997.

Preston, J., Shackelford, R.: Improving On-line Assessment: an Investigation of Existing Marking Methodologies. *ACM SIGCSE Bulletin*, **31**(3), 29 - 32, 1999.

Rawles, S.: CAA Review Exercise CourseMaster, Centre for Information and Computer Sciences. *Computer Assisted Assessment Workshop*, April 5 - 6, University of Warwick, 2001.

Reek, K.: A Software Infrastructure to Support Introductory Computer Science Courses. *ACM SIGCSE Bulletin* **28**(1), 125 - 129, 1996.

Rowe, G.: Assessment of Programs by Matching Flowgraphs, *Proceedings of LTSN-ICS 1st Annual Conference* August 23 - 25, Heriot-Watt University, Edinburgh, 2000.

WebAssign: WebAssign. Internet WWW-sivu, URL: http://webassign.net/index.html (20.10.2002).

Wise, M.: Detection of Similarities in Student Programs: YAP'ing may be preferable to Plague'ing. *ACM SIGCSE Bulletin,* **24**(1), 268 - 271, 1992.

Young, J.: The Cat-And-Mouse Game of Plagiarism Detection, *Chronicle of Higher Education*, **47**, July 6, 2001.

Liite 1: Uuden tehtävän luominen CourseMasterissa.

1) Tee uusi hakemisto Java-kurssi-hakemiston alle nimellä u1.unit, jos sitä ei ole ennestään olemassa ja luo tarvittavat moduulin tiedostot, joita ovat title.txt, summary.txt, notes.txt, properties.txt.
2) Tee tiedosto, joka sisältää tehtävän otsikon ja anna sille nimeksi *title.txt*. Opiskelijat näkevät tämän tiedoston sisällön CourseMasterin käyttöliittymässä.
3) Tee tiedosto, joka sisältää tehtävänannon ja anna sille nimeksi *question.txt*. Tähän kysymykseen opiskelijat vastaavat.
4) Tee tiedosto, joka sisältää tehtävän ratkaisun ja tallenna se nimellä *tehtävä1.java.* Toimivan tehtävän kääntäminen luo ajettavan *tehtävä1.class*-tiedoston.
5) Kopioi ratkaisu tehtävä1.java tiedosto *tehtävä1.java.SOL*-tiedostoksi.
6) Editoi *tehtävä1.java*-tiedostoa, kunnes se on sopivassa muodossa opiskelijoille annettavaksi rungoksi.
7) Kopioi muut tarvittavat apuluokat, kuten esimerkiksi Lue-luokka.
8) Kopioi ja editoi *setup.txt*-tiedostoa. Tiedosto määrittelee, mitä toimintoja CourseMaster suorittaa, kun opiskelija tekee setup-toiminnon.
9) Kopioi ja editoi *save.txt*-tiedostoa. Tiedosto määrittelee, mitä tiedostoja CourseMaster tallentaa tarkastusprosessin aikana. Käytännössä tiedosto sisältää java-tehtävän nimen ja mahdollisten apuluokkien nimet.
10) Kopioi ja editoi *properties.txt*-tiedostoa. Tiedostoa käytetään osoittamaan muun muassa tehtävän statusta (avoin, suljettu), painotus ja maksimilähetysten määrä.
11) Kopioi ja editoi *mark.java*-tiedostoa. Tiedostoa käytetään kontrolloimaan tarkastusprosessia. Yleensä riittää, että tehtävän nimi muutetaan, mutta monimutkaisempi tarkastus voidaan saavuttaa kun tarkastusprosessi ymmärretään yksityiskohtaisesti. Piirteiden suhteelliset painotukset ja dynaamiset- ja typografiset testit voidaan asettaa tässä tiedostossa. Tiedoston ensimmäinen rivi sisältää kurssin, yksikön ja tehtävän nimet ilman tiedostotunnisteita, esimerkiksi package Testi_Java.u1.tehtävä1;
12) Kopioi *mrun.bat*. Tiedostoa käytetään kääntämään mark.java, joka luo *mark.class*-tiedoston.

13) Aja *mrun.bat*.
14) Kopioi *mark.scale*. Tiedosto sisältää tehtävän arvosanojen porrastuksen, joka kartoittaa karkean % arvosanan ja on yleensä muuttumaton tietyllä kurssilla. Arvosanat annetaan kirjaimina A:sta F:n.
15) Kopioi *mark.tv*. Tässä tiedostossa on typografisen työkalun komennot. Tiedostoa voidaan tarvittaessa muuttaa.
16) Tee tiedosto nimeltä *mark.ft*, joka on ohjelman piirteiden testaamisen kontrollitiedosto. Tiedostossa on yksi rivi, jokaista testattavaa piirrettä kohden. Rivi noudattaa muotoa: suhteellinen painotus:varattu sana:kommentti opiskelijalle:kommentti, jos löydetty:kommentti, jos ei löydetty.
17) Kopioi tiedosto *mark.ft.compile*. Tätä tiedostoa ei tarvitse todennäköisesti muuttaa.
18) Kopioi *mark.makefile* ja editoi tiedostoa. Ohjelman nimen muuttaminen riittää, jos uusi tehtävä on kirjoitettu samalla ohjelmointikielellä kuin vanha.
19) Tee dynaaminen testausosio, eli tiedostot *mark.dv*, *mark.oracledata* ja *mark.testdata.*
20) *Mark.dv* on tiedosto, joka määrittelee painotuksen jokaiselle dynaamiselle testille suhteessa toisiin dynaamisiin testeihin ja palautekommentteja. Jokaista dynaamista testiä kohden on yksi rivi, joka on muotoa: suhteellinen painotus:varattu sana:kommentti opiskelijalle:kommentti jos alle seuraavan arvon:% arvo:kommentti, jos alle seuraavan arvon:% arvo ja niin edelleen.
21) *Mark.testdata* on jokaisen ohjelman dynaamisen testin lähdetiedosto. Jokainen testi vaatii monta tehtäväkohtaista riviä, ja testit on erotettu toisistaan viidellä @-symbolilla ja sen jälkeen rivillä, jossa on @end@.
22) *Mark.oracledata* sisältää varatut sanat ja palautekommentit jokaiselle dynaamiselle testille. Myös se on eritelty viidellä @-symboleilla. Esimerkki, 10:[Uu]nsorted:Search output for ”unsorted:”unsorted” found in output:”unsorted” NOT found in output:
23) Testaa tehtävä. Askeleet 16 - 21 suoritetaan uudelleen, jos uusia piirteitä tai dynaamisia testejä lisätään. (CourseMaster käyttöohje 2001)

Liite2: Rooli-tehtävään kopioituja tiedostoja.

mark.ft.compile (käännetty suomeksi)

50: [Ee]rror==0: Etsii käännettäessä saatuja virheitä: Ei käännösvirheitä: VIRHEITÄ löytyi käännettäessä:
50: [Nn]ote==0: Etsii käännettäessä saatuja varoituksia: Ei varoituksia: VAROITUKSIA löytyi käännettäessä:

mark.scale

40:F:GRAY:
50:D:RED:
60:C:YELLOW:
70:B:BLUE:
100:A:GREEN:

mark.tv (käännetty suomeksi)

10: ACPL: 0: 5: 40: 50: Ka. merkkejä rivillä: Tarpeeksi merkkejä rivillä: Ei tarpeeksi merkkejä rivillä: Liikaa merkkejä rivillä: Lisää merkkien määrää rivillä: Vähennä merkkien määrää rivillä:
10: ASPL: 1: 2: 10: 12: Ka. merkkejä rivillä: Tarpeeksi välilyöntejä rivillä: Liian vähän välilyöntejä rivillä: Liikaa välilyöntejä rivillä: Lisää välilyöntien määrää rivillä: Vähennä välilyöntien määrää rivillä:
10: %BLL: 0: 15: 50: 80: % tyhjiä rivejä: Tarpeeksi tyhjiä rivejä: Liian vähän tyhjiä rivejä: Liikaa tyhjiä rivejä: Lisää tyhjiä rivejä: Vähennä tyhjien rivien määrää:
10: {}ER: 0: 0: 0: 100: % sulkumerkkien oikea sisennys: Ei löytyneitä sulkumerkkien virheitä: Liian vähän sisennyksiä: Liikaa väärin tehtyjä sisennyksiä: Yritä sisentää sulkumerkit paremmin: Vähennä vääriä sulkumerkkien sisennyksiä:

10: UCCC: 0: 0: 0: 100: % kommentoimattomia sulkumerkkejä: Kommentoimattomia sulkumerkkejä ei löytynyt: Liian vähän kommentoimattomia sulkumerkkejä: Liikaa kommentoimattomia sulkumerkkejä: Lisää kommentoimattomien sulkumerkkien määrää: Vähennä kommentoimattomien sulkumerkkien määrää:

mrun.bat

```
javac -classpath .;d:\cmpxs;f:\java\jdk1.3\jre\lib\rt.jar mark.java
```

Liite 3: Malliratkaisu Rooli.java-tehtävään.

```
// rooli.java.SOL-tiedosto
public class Rooli
{
  public static void main(String [] args)
  {
    int kyky_J[];    //Juuson hahmo, velho
        kyky_J=new int[4];
    int kyky_E[];     //Eetun hahmo, samooja
        kyky_E=new int[4];

    for(int i=0; i<=3; i++)   //Arvojen alustaminen
    {
                kyky_J[i]=0;
                kyky_E[i]=0;
    }

    /* Taulukossa 1. alkio sisältää maantiedon, 2. mahdin, 3. magian
       ja 4. selviytymisarvon */

    kyky_J[0] = 5;
    kyky_J[1] = kyky_J[0] -2; //5-2=3
    kyky_J[2] = 2*kyky_J[1]; //2*3=6
    kyky_J[3] = (3*kyky_J[0] + 2*kyky_J[1] + kyky_J[2])/6; //3*5+2*3/6=

    kyky_E[0] = kyky_J[2];
    kyky_E[1] = kyky_E[0]/2;
    kyky_E[2] = kyky_E[1];
    kyky_E[3] = (3*kyky_E[0] + 2*kyky_E[1] + kyky_E[2])/6;
```

```
    System.out.println("Hahmojen kyvyt taulukossa: ");
    System.out.println("-------------------------- ");
    System.out.println("Nimi            Juuso     Eetu ");
    System.out.println("Hahmo           velho     samooja");
    System.out.println("Maantieto         "+kyky_J[0]+"         "+kyky_E[0]);
    System.out.println("Mahti             "+kyky_J[1]+"         "+kyky_E[1]);
    System.out.println("Magia             "+kyky_J[2]+"         "+kyky_E[2]);
    System.out.print("Selviytymisarvo  ");

    if (kyky_J[3] < 3)
                System.out.print("huono     ");
    else if (kyky_J[3] > 6)
                System.out.print("erinomainen ");
    else
                System.out.print("hyvä      ");

    if (kyky_E[3] < 3)
                System.out.println("huono  ");
    else if (kyky_E[3] > 6)
                System.out.println("erinomainen  ");
    else
                System.out.println("hyvä   ");

  }
}
```

Liite 4: Malliratkaisu Min.java-tehtävään.

```
// malliratkaisu sekunnit minuuteiksi muuttavan ohjelmaan

public class Min
{
        public static void main(String[] args)
        {
                final int MINUUTTI = 60;
                int luku, minuutit, sekunnit;

                System.out.println("Anna sekunnit");
                luku = Lue.kluku();

                minuutit = luku / MINUUTTI;
                sekunnit = luku % MINUUTTI;

                System.out.println(luku + " sekuntia on " + minuutit +
                " minuuttia ja " + sekunnit + " sekuntia.");
        }// main-loppuu

}//ohjelma loppuu
```

Liite 5: Malliratkaisu Kauppa.java-tehtävään.

```
// malliratkaisu kauppa-tehtävään
class Kauppa
{
	public static void main(String[] args)
	{
		System.out.println("Tervetuloa Tanelin SuperMarkettiin!");
		System.out.println("**********************************");
		System.out.println("1. Maitoa 3 mk (0,50 E)");
		System.out.println("2. Juustoa 16 mk (2,70 E)");
		System.out.println("3. Makkaraa 10 mk (1,70 E)");
		System.out.println("Tilataksesi tuotteen paina sen numeroa.");
		int valinta = Lue.kluku();
		if (valinta==1)
		{
			System.out.println("Maitoa 3 mk (0,5 E)");
		}
		else if (valinta==2) {
		System.out.println("Juustoa 16 mk (2,70 E)");
		}
		else if (valinta==3) {
			System.out.println("Makkaraa 10 mk (1,70 E)");
		}
		else {
			System.out.println("Valintasi ei ollut pateva!");
		}
		System.out.println("Kiitoksia kaynnista ja tervetuloa uudelleen!");
	}
}
```

Liite 6: Boss-järjestelmän automaattisesti lähettämiä sähköpostiviestejä.

Uusi salasana

Lähettäjä: Your Admin
Otsikko: New Password
Boss Online Submission System - New Password

Your new pass phrase is DOoPeVoGH.
You should change this pass phrase as soon as possible.
Please try this new password by logging on now. If you are a new user then you may wish to browse the help documentation. Simply press the help button in any of the BOSS menus.

Kuittaus opiskelijan lähettämistä tehtävistä

Lähettäjä: Your Admin
Otsikko: Online Assignment Submission - Receipt

You are 111112 Lehtonen

You submitted the following files for
Module 1234: testikurssi2
Assignment A: teht1
ProblemHello hello.java, size 210, checksum 1276177345
Security code: f089add146de548ea71faa2b5343c50337ce3f45

What the codes mean

The codes in this message are digital signatures for each of the files submitted. We store your files as a single archive (in "zip" format), and the security code at the end of the message is the signature for the archive.

Important

Please store this receipt safely, it is your
evidence that the submission was successful

Palaute tehtävästä

Feedback for 1234: testikurssi2 - teht1
You are taina lehtonen (111114)
Your overall percentage for this assessment is **-1%**