Tallinna Ülikool

Informaatika Instituut

Informaatika (multimeedium ja õpisüsteemid)

Sven-Olav Paavel

Java sertifikaadieksami (SCJP) õpivahendi koostamine RIA-rakendusena

Magistritöö

Juhendaja: Jaagup Kippar, *M.Sc.*

Koostaja:	“ ..... ”	mai 2009
Juhendaja:	“ .... ”	mai 2009
Instituudi direktor:	“ .... ”	mai 2009

Tallinn 2009

Autorideklaratsioon

Deklareerin, et käesolev lõputöö on minu töö tulemus ja seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Töös kasutatud teiste autorite materjalid on varustatud vastavate viidetega.

Sven-Olav Paavel

. . . . . .

Sisukord

Sissejuhatus

Teema põhjendus ja teesid

Java programmeerimiskeele rakendusvaldkond on väga lai: alates väikestest rakenditest (*applets*) kuni ettevõtete suurte infosüsteemideni välja (Java EE). Kuigi Java areng ei ole enam nii tormiline kui algusaastatel, on ta oma 14-aastase vanuse juures saanud populaarseks programmeerimiskeeleks, mille oskusteave on tööturul väga hinnas. Javast on saanud ka omamoodi (*lingua franca*, mida õpetatakse ülikoolides. Java distsiplineerib arendajaid, mis teeb sellest sobiva keele programmeerimise õppimisel. Kuivõrd keele rakendusvaldkond on lai, siis siit on ka ajendatud väitekirja autori teemapüstitus Java õpimetoodikate ja -vahendite kohta: **kuidas omandada ja mõõta Java teadmisi?**

Sun Microsystems on koostanud standardid, mille alusel hinnatakse Java kompetentsi. Seda teed on läinud enamus tarkvaragigante, st loonud oma sertifitseerimise süsteemid, määramaks vastavad kompetentsi tasemed, millele mingi keele oskaja peab vastama. Siit pärineb autori tees, et Java keele õppijal oleks otstarbekas silmas pidada Suni poolt kehtestatud **kompetentside tasemeid**, et olla võimeline vastav sertifikaadieksam ära sooritama. See annab õppurile kindlustunde, et minnes koolist tööturule, ei pea omandatud hariduse rakendamiseks hakkama ümber õppima. Sun klassifitseerib Java oskusteavet ja vaatab regulaarselt üle vastavalt turu situatsioonile. Siinkirjutaja väide on, et see on õppija seisukohalt kõige turvalisem viis antud tehnoloogiat omandada: ta ei pea kartma, et ta omandab mitterelevantset ning omanda-

tava oskusteabe tase on samas turul kõrgelt hinnatud. Olukord on mõneti sarnane võõrkeele õppimisele, mille oskuste hindamisel ja üle kandmisel teise süsteemi, kasutatakse üldaktsepteeritud standardeid nagu TOEFL inglise keele puhul või DELF/DALF prantsuse keele puhul.

Nõudlus Java oskusega spetsialistide järgi on Eesti tööturul eriti teravalt tuntav, sest meie IT sektor on noor, meil praktiliselt puuduvad pärandiks saadud vanad süsteemid, mistõttu kipuvad ka meie arendustehnoloogiad (sh. keelte valik, rakendusserverid jne) olema novaatorlikud. Ent kuidas saada Java kompetentsiga spetsialiste? Põhilised viisid Java programmeerijate hankimiseks on:

(1) teistest firmadest üleostmine

(2) odava välistööjõu või allhankimise kasutamine

(3) senise personali ümberõpetamine

(4) värskelt kooli lõpetanute koolitamine

Esimene meetod on end ammendanud ja pikemas perspektiivis ei ole jätkusuutlik. Teine on vaid ajutine leevendus, sest põhineb kolmanda riigi odava tööjõu kasutamisel. Kolmas punkt on rakendatav, aga väga väikeses mahus. Ainuke viljakas lähenemine on neljas. Seda praktiseeritakse Eestis organisatsioonide sees kas lihtsa kompetentsikeskuste, mentor-programmide või noorteprojektide (nn „akadeemiate" või „ülikoolide") loomise kaudu [1]. Eeskujulik Microsofti tehnoloogia-keskne veebistuudiumi lahendus on realiseeritud Eesti .NET Assotsiatsiooni poolt, kus koolitusprogrammiga liitunud koolide õpetajatele ja õpilastele antakse võimalus sooritada Microsofti sertifitseerimise eksameid tasuta [2].

[1] Nt Playtechi *Academy* (`http://www.playtech.ee/?id=2&nr=29`) või Webmedia *University* (`http://www.webmedia.ee/?event=Show_default&page_id=133`). Vt ka „Tõõjõupõua hirmus suurfirmad koolitavad noori endale töötajateks" — *Postimees*, 13.04.2007; „Playtech Academy valiti aasta personaliprojektiks" — *Äripäev*, 16.04.2007.

[2] `http://www.eneta.ee/KOMMUUN/Projektid/Veebistuudium/Sertifitseerimine.aspx`

Eesmärgid ja nende täpsustumine stuudiumi jooksul

Käesoleva magistritöö kitsama eesmärgina näeb autor luua SCJP (*Sun Certified Java Programmer*) eksami jaoks õpivahendi prototüüp, kasutades uue põlvkonna veebiprogrammeerimise — nn „rikka Interneti-rakenduse" (*rich Internet application*) võimalusi. Selle eesmärgi saavutamiseks analüüsitakse SCJP nõudmiseid, õpivahendeid ja -metoodikaid. Kõigepealt avatakse tarkvarainseneride sertifitseerimise probleemi laiem kontekst. Laiema ühiskondliku konteksti avamise taga on autori soov näidata, et tarkvaraarendajate sertifitseerimine ei kujuta endast pelgalt ühe tööstusharu marginaalset sisemist detaili, vaid sellel on tarkvarainseneri elukuse hea nime, vastastikuse usaldamise ja ühiskonna turvalisuse näol avaram tähendus.

Rikaste Interneti-rakendustega (Adobe Flexiga) on siinkirjutaja tööalaselt kokku puutunud viimased kaks aastat. Adobe Flex sai valitud väitekirja lahenduse tehniliseks platvormiks, sest rakenduse kavandamise ajal oli see kõige toodanguküpsem RIA-raamistik. Valiku tegemisel oli ka teisi põhjuseid (suur arenduskogukond, arendusvahendi tasuta litsents jne), mille detaile käsitletakse 3. peatükis pikemalt.

Siinkirjutaja näeb ka oma väitekirja ühe laiema eesmärgina välja selgitada, kui sobiv on Adobe Flex e-õppe õpiobjektide programmeerimiseks. Töö käigus koostatud SCJP eksami õpivahend omab piisava taseme ja keerukusega funktsionaalsust, et olla sobivaks prototüübiks. RIA-rakendust võib integreerida ükskõik missuguse olemasoleva õpikeskkonnaga. Käesolev magistritöö võib pakkuda väärtuslikku analüüsimaterjali neile, kes soovivad koostada iseseisvaid e-kursuseid või õpivahendeid, kasutades selleks rikaste Interneti-rakenduste võimalusi.

Väitekirja autor alustas oma magistriõpinguid 2005. aastal. Võrreldes töö algse plaaniga on selle koostamise käigus tehtud mitmeid olulisi muudatusi. Algselt oli plaanis keskenduda rohkem ühele viimase aja edukaimale Java õpetamise metoodikale — *Head First* raamatusarjale ja selle metakognitiivse-

le alusteooriale. Eriseminarilt saadud tagasiside osutas sellele, et *Head First*-i metoodika rakendmine väljaspool programmeerimise valdkonda pakuks suurt huvi, seetõttu ei pidanud autor *Head First*-i metoodikat analüüsides silmas vaid programmeerimise valdkonda. Heaks näiteks, kuidas seda metoodikat on juba rakendatud võib tuua *Head First*-i õpiku projektijuhtimise *The PMBOK Guide*-i ja selle sertifikaadieksami kohta [3]; hiljuti on veel lisandunud *Head First Statistics*, *Head First Physics*, *Head First Data Analysis*, mainides vaid mõningaid väheseid [4].

Teine oluline muudatus toimus rakendusloome käigus. Autor peab Flexiteemalist ajaveebi [5], millest saadud tagasisidet viis mõttele, et väitekiri võiks pakkuda märksa laiemat huvi, kui ta ühelt poolt esitaks tehnilist informatsiooni vähekäsitletud rikaste Interneti-rakenduste koostamise võimaluste kohta e-õppe kontekstis, ja teiselt poolt käsitleks tarkvarainseneride sertifitseerimise probleemi avaramalt (mitte ainult Java arendajatega seoses). Nii on väitekirja lõpp-versioonis Java SCJP sertifikaadieksamit käsitlev materjal omandanud rohkem **näitlik-abstraktse** iseloomu, kaotamata siiski selle teema kohta süvitsiminekut.

Magistritöö ülesehitus ja uurimismetoodika

Käesolev töö on tüübilt **rakendustloov uurimus**. Metoodiliselt jaguneb töö kaheks osaks: teoreetiliseks ja rakenduslikuks. Teoreetilise osa käsitluslaad on kvalitatiivne (seletav ja kirjeldav). Eesmärgist tulenevalt jaguneb töö kolmeks peamiseks **ülesandeks**, igaühele on pühendatud üks peatükk. Väitekirja teoreetilise osa moodustavad esimene ja teine, rakendusliku osa kolmas peatükk. Iga peatükk keskendub ühele kesksele probleemile, mis on järgmised:

[3] Stellman, A., Greene, J. (2007), *Head First PMP*, O'Reilly. 2009. aasta juulis ilmub sellest raamatust teine trükk.

[4] Vt `http://oreilly.com/store/series/headfirst.csp`

[5] `http://www.mindtheflex.com`

(a) Määratleda, milline on arvutialase tehnilise sertifitseerimise roll tarkvarainseneri elukutses (pt 1).

(b) Analüüsida ühe konkreetse sertifikaadieksami (SCJP) nõudmiseid, õpivahendeid ja -metoodikaid (pt 2).

(c) Luua SCJP eksami jaoks õpivahendi prototüüp kasutades uue põlvkonna veebiprogrammeerimise (RIA) võimalusi (pt 3).

Esimene peatükk esitab laiema konteksti kirjelduse, küsides, mida kujutab endast tarkvarainseneride sertifitseerimine kui selline; mille poolest see erineb litsentsimisest; milline koht on sertifitseerimisel tänasel tarkvaraarenduse maastikul. Autori soov on tõestada, et tarkvarainseneride sertifitseerimine kujutab endast ühte olulist etappi tarkvarainseneri enesetäiendamise ja elukestva õppe protsessis. Samuti näidatakse, et sertifitseerimine on tarkvarainseneri elukutse küpsuse üks fundamentaalsetest alustaladest, haakudes otseselt tarkvarainseneri eetikakoodeksiga. Töö autor jagab Steve McConnelli seisukohata, et vabatahtlikku sertifitserimist võib vaadelda kui omamoodi „raudset sõrmust" (McConnell 1999, 112), mis traditsioonilistes insenerivaldkondades sümboliseerib ühelt poolt inseneriameti küpsust, sest see on teadlikult võtnud vastutuse ühiskonna ees ja teiselt poolt sümboliseerib see konkreetse inseneri küpsust, kes on saavutanud piisava professionaalse taseme.

Peatükk püüab ka näidata, et tarkvarainseneride litsentsimine ei ole saavutanud sellist populaarsust kui sertifitseerimine, sest põhjused, miks tarkvarainsenerid sertifitseerimisprotsessi ette võtavad, lähtuvad eelkõige tema enda sisemisest veendumusest ja kohustusest. Pelgalt välise kohustusena omaks sertifitseerimine sama marginaalset tähendust tarkvaratööstuses kui tarkvarainseneride litsentsimine täna.

Teine peatükk on rohkem analüütilise suunitlusega. Selle eesmärk on analüüsida magistritöö käigus loodava õpivahendi-prototüübi vajadusi. Vaatluse alla tulevad SCJP sooritustingimused aga antakse ka ülevaade Sun Microsystems-i Java sertifikaadieksamite õpiteekonna (*learning path*) kon-

tekstist laiemalt, mis võib huvipakkuv olla eelkõige neile, kes Java sertifitseerimisega ei ole varem kokku puutunud. Peatükis analüüsitakse sertifikaadiõppes edukamaid õpivahendeid ja -metoodikaid. Siinkirjutaja üritab neid kombineerida agiilse tarkvaraarenduse ühe levinuima metoodika — ekstreemprogrammeerimise (XP) — kaudu tuntud paarisprogrammeerimisega. See on teemadering, millega autor on oma stuudiumi vältel nii teoreetiliselt õppetöös kokku puutunud (Paavel 2006) kui ka tööalaselt praktiseerinud.

Kolmandas peatükis kirjeldatakse SCJP õpivahendi-prototüübi enda loomist kasutades Adobe Flexi arendusraamistikku.

*

Magistritöös on kasutusel tekstisisene APA-stiilis viitamissüsteem [6]. Joonealustes märkustes esinevad viited on reeglina kas seadused (nt Riigi Teataja) või Interneti-aadressid. Magistritöö vormistamisel on kasutatud LaTeX-i.

Magistritöö autor on tegelenud programmeerimisega 12 aastat, neist 8 aastat programmeerimiskeelega Java. Autor omab järgmiseid Sun Microsystemsi Java sertifikaate:

(1) *Sun Certified Programmer for the Java Platform 1.4* (2003)

(2) *Sun Certified Web Component Developer 1.4* (2003)

(3) *Sun Certified Business Component Developer* (2005)

(4) *Sun Certified Programmer for the Java Platform 5, Update* (2007)

[6] `http://www.apastyle.org`

1. Tarkvaraarendajate sertifitseerimine

Käesoleva peatüki eesmärk on vastata kolmele küsimusele:

(a) Mida kujutab endast arvutialane tehniline sertifitseerimine ja milline on selle koht tänasel tarkvaraarenduse maastikul.

(b) Millist lisaväärtust pakub tarkvaraarendajate sertifitseerimine tarkvaraarenduse teenuste kasutajatele ja laiemalt ühiskonnale tervikuna.

(c) Mis on sertifitseerimise väärtus ja milline on motivatsioon tarkvaraarendaja enda seisukohalt. Miks ta peaks saama sertifitseeritud?

1.1. Sertifikaadi mõiste

Kõige laiemalt võiks sertifikaati (ing kl *certificate*) defineerida kui formaalset tunnistust, et sertifikaadi omanikul on piisavad teadmised mingis valdkonnas. Sertifikaadi annab üldjuhul välja kutseliit või organisatsioon, kes määratleb ka selle valdkonna oskusteabe standardid. Meid huvitab eelkõige arvutialane tehniline sertifitseerimine, ent lisaks on olemas ka lõppkasutjate sertifikaadid (AO, MOUS, ECDL), tugiisikute sertifikaadid (A+, N+, I+) ja koolitajate sertifikaadid (Sarv 1999a, 44).

Arvutialane tehniline sertifikaat tõendab, et selle omanikul on piisav oskusteabe tase ning teda võib lugeda selle valdkonna professionaaliks. Steve McConnell rõhutab, et sertifikaadi saamise eelduseks on tavaliselt nii vastav haridus kui ka kogemus (McConnell 1999, 102). Teine edukas kombinatsioon oleks täiendkoolitus ja kogemus (Sarv 1999b, 45). Kogemuse nõue võib aga ka mitte olla eksplitsiitselt välja toodud, küll aga on sertifikaadi eksami sooritamine ilma erialase töökogemuseta väga raske, kui mitte võimatu.

Sertifitseerimise puhul on oluline ka, et teadmiste kvaliteeti ei hinnata mitte üksikindiviidi isikuomadustest lähtuvalt. Pigem vastuoksa: oskusteabe sertifitseerimine on oma olemuselt intersubjektiivne, sest teadmise hulka mingi valdkonna, toote, teenuse vms kohta hinnatakse tööstusharu eelnevalt väljatöötatud standardite alusel.

Sertifikaati on sobiv võrrelda TOEFL keeletestiga, mis ei mõõda inimese keelelist a n d e k u s t, vaid kaardistab miinimumnõuded. Sertifikaat annab garantii teadmiste kvantiteedi kohta, mite ei filtreeri välja talente. Mõttetu oleks sertifitseerida kunstnikke, küll aga insenere. See ei tähenda, et (tarkvara)inseneri töös ei oleks looming oluline, vaid pigem seda, et sertifikaat ei ole kreatiivsuse mõõtmise jaoks õige vahend.

Ka kogenud programmeerija võib eksamil läbi kukkuda (Green 2004). Kuidas on see võimalik? Levinum kriitika sertifitseerimise vajalikkuse kohta ongi seotud tavaliselt näidetega, et leidub piisava tööstaažiga või originaalsete töövõtetega programmeerijad, kes on eksamil läbi kukkunud, kuigi nad vaieldamatult on suurepärased spetsialistid. Silmas tuleb aga pidada seda, et sertifikaat on tõend selle omaniku laiapidiste teadmiste miinimumhulga, mitte aga programmeerija geniaalsuse kohta. Suni Java sertifikaadid ei pane sertifitseeritavale „hinnet" — kas eksamil lahendati ülesanded 100% tulemusega või saadi miinimumnõuded, ei kajastu mujal kui eksami raportis ja need jäävad ainult eksamineeritava enda teada.

1.1.1. Sertifikaat, litsents või kutsetunnistus?

Tarkvara valdkonnas on sertifikaadi mõiste puhul oluline ka selle omandamise v a b a t a h t l i k k u s. Tegevuslitsents on aga seevastu seadusega ette nähtud k o h u s t u s, mille eesmärk on kaitsta ühiskonda võimaliku kahju eest, mida amatöörid võiksid tekitada. Näiteks raamatupidajad ja audiitorid on arstide ja juristide kõrval ühe enimreguleeritud tegevusala esindajad. Vannutamise ja sertifitseerimise vajaduse tingib asjaolu, et oskamatus või

ebaeetilisus neis valdkondades põhjustab teenuste tarbijatele suuremat kahju kui muudes sfäärides. Vabatahtlikuse ja kohustuslikkuse eristamine on nende mõistete puhul olemuslik (Ford and Gibbs 1996, 13). Paraku kasutatakse mitmetes eluvaldkondades sertifikaadi ja tegevuslitsentsi mõisted segamini, eriti igapäevases kõnepruugis.

Taunitav on, kui see terminoloogiline hägusus hakkab jõudma seadustesse. Eestis on näiteks võetud kasutusele ilus omakeelne termin „kutsetunnistus". Paraku tähistatakse aga sellega nii sertifikatsiooni kui ka litsentsimist. Näiteks kui võtta isegi kaks lähedalolevat professiooni nagu raamatupidajad ja audiitorid, siis mõlemad sooritavad k u t s e e k s a m i ja saavad k u t s e t u n n i s t u s e, ent raamatupidajate kutseeksam on vabatahtlik, audiitoritel oma aga kohustuslik[1]. Esimene on sisuliselt sertifikaat ja teine tegevuslitsents. Eesti kutsekvalifikatsiooni süsteemis määratletakse kutsekvalifikatsiooni nõudeid viiel tasemel. I tase on madalaim ja V tase kõrgeim. Iga konkreetse kutse kvalifikatsioonitasemed, sealhulgas vajaduse korral ka haridusnõuded, määrab kindlaks konkreetne kutsenõukogu.

1.1.2. Inseneride litsentsimine

Ameerika Ühendriigid on esimene maa, kus hakati reguleerima inseneride valdkonda ja hakati insenere litsentsima. USA-s välja pakutud ühine definitsioon terminile „insener". Selle pakkus välja 1986. aastal USA Riiklik Inseneride Eksamineerimise Liit (*U. S. National Council of Engineering Examiners*):

> „Insener" on isik, kes on kvalifitseeritud spetsiaalsete teadmiste põhjal tegutsema insenerina ja kasutama matemaatika-, füüsikaning inseneriteadusi; juhtimise analüüsi ja planeerimise meetodeid ning printsiipe, mis on saavutatud inseneri hariduse ja kogemuste põhjal. (Gordon 2008, 45)

[1] Audiitortegevuse seadus, RT I 1999, 24, 360, `http://www.riigiteataja.ee/ert/act.jsp?id=264381`.

Tarkvarainseneride litsentsimispoliitika väljatöötamisega on tegeletud Ameerika Ühendriikides 90-date aastate keskpaigast alates. Insneride erialaühing *National Society of Professional Engineers*[2] on saavutanud mõiste i n s e n e r erialase kasutamise seadusega kontrollimise 48 osariigis. Florida ja Texase osariigis käib see piirang ka mõistete „arvutiinsener" (*computer engineer*) ja „tarkvarainsener" (*software engineer*) kohta. Tarkvaraarendajate litsentsimise protsess on juurutatud USA-s Texase osariigs (alates 1998. aastast) ja Kanadas (Briti Kolumbia). McConnell, kes ise on litsentsimise pooldaja, kirjutas 1999. aastal optimistlikult: „Kõigi silmad on Texase peal [...] Texas ees, teised osariigid järel" (McConnell 1999, 105).

Paraku peab tunnistama, et litsentsimise süsteem on väga aeglaselt toimima hakanud. Liialdamata võib ütelda, et litsentsimine ei ole tarkvaraarenduses seni laialt levinud. Näiteks võiks tuua sama Texase osariigi andmed 2008. aasta kohta: välja oli antud üle 50 000 PE litsentsi, millest vaid 58 olid SWE litsentsid, neist omakorda viiendik mitteaktiivsed [3]. 2006. aastal oli aktiivseid litsentsiaate veel 63 [4].

Kindlasti on olnud üheks oluliseks põhjuseks liialt suured nõudmised tarkvaraarendajatele. Texase osariigi tarkvarainseneri PE litsentsi saamiseks peab taotleja vastama vähemalt ühele järgmistest nõuetest: 1) 16 aastat töökogemust; 2) 12 aastat töökogemust ja bakalaureusekraad riiklikult akrediteeritud programmiga ülikoolist; 3) 6 aastat töökogemust ja PhD kraad. Lisaks peab litsentsi taotleja hankima üheksa professionaalse inseneri soovituse (viis nendest võivad olla ka mitte tarkvarainseneride poolt).

Tarkvarainseneride litsentsimine ei ole levinud ka seetõttu, et sertifitseerimine toimib lihtsama alternatiivina. Kolmanda põhjusena võiks välja tuua selle, et teadmiste tuum, mille kohta litsentsi anda, ei ole stabiliseerunud,

[2]NSPE (`http://www.nspe.org/`) asutati 1934. aastal New Yorgis. NSPE-l on 2009. aasta seisuga üle 45 000 liikme 53 osariigis.

[3]`http://www.tbpe.state.tx.us`

[4]Danielle Boykin (2007). Is It Time to License Software Engineers? — *PE Magazine*, December, 2007, `http://www.nspe.org/PEmagazine/pe_1207_Software_License.html`.

mistõttu seatakse küsimuse alla litsentsimise kui sellise mõttekus tervikuna (McBreen 2002, 40) (me tuleme selle küsimuse juurde allpool tagasi). Neljas probleem on kindlasti ka selles, et erialaorganisatsioonid ei ole saavutanud konsensust. Nii astus ACM (*Association for Computing Machinery*) juba 1999. aastal litsentsimise vastu välja [5]. 2002. aastal küsitakse retooriliselt kas tarkvarainseneride litsentsimine on olnud Texase osariigis edukas või on läbi kukkunud ja ACM-i artikli autorite vastus on järgmine: *In our view, the news is good: licensing had practically no effect* (Kennedy and Vardi 2002).

PE litsents omab tarkvara vallas kaalukat tähendust Ameerika Ühendriikides vaid militaar- ja meditsiinitarkvara sfääris. Muudes eluvaldkondades on aktsepteeritud alternatiiviks kommertsfirmade seritfikaadid, näiteks MCSE (*Microsoft Certified Systems Engineer*) kvalifikatsioon.

Litsentsimine on seotud ühe juristiktsiooni alla kuuluva territoriaalse üksusega, mille üle litsentsiaaril on voli. Tarkvarafirmade (Apple, Oracle, Microsoft, Sun Microsystemsi) poolt väljaantavad sertifikaadid omavad aga globaalset mõõdet. See on üks tarkvaratootjate sertifikaatide iseloomulik omadus, mis eristab neid nii piiritletud territoriaalse juristdiktsiooni all välja antud tegevuslitsentsidest kui ka sertifikaatidest, mis omavad lokaalset mõõdet.

1.2. Tarkvarainseneri roll ühiskonnas

Termini „tarkvaratehnika" (*software engineering*) [6] võttis kasutusele Inglise arvutiteadlane Brian Randell ning sellele anti laiem kõlapind NATO tarkvara konverentsil 1968. aastal (Naur and Randell 1969) (selle termini kasutusele võtmise idee autor oli tegelikult Fritz L. Bauer). Termin tundus tol ajal koraldajatele „provokatiivne", suunamaks tarkvaratootjate tähelepanu vajadu-

[5] ACM Position on Software Engineering as a Licensed Engineering Profession, Association for Computing Machinery, 2000, `http://www.cs.wm.edu/~coppit/csci690-spring2004/papers/selep_main.pdf`

[6] Eesti keelde soovitab Keelenõuanne *engineering* tõlkida ka i n s e n e r l u s (kuigi ÕS-s sellist vormi ei esine)

sele põhineda rohkem teoreetilistel alustel ja praktilistel printsiipidel, mis oli senistes väljakujunenud inseneri-valdkondades juba traditsiooniliseks saanud (Naur and Randell 1969, 13). Kongressi korraldamise üheks peamiseks ajendiks oli teadvustada laiemat üldsust, et tarkvarasüsteemide tähtsuse plahvatusliku kasvuga ühiskonnas (Naur and Randell 1969, 15) kaasneb sellega ka rida ohtusid (sest süsteemid muutuvad keerukaks ning vigade arv kasvab eksponentsiaalselt). F. L. Bauer rääkis isegi „tarkvara kriisist" (*software crisis*) (Naur and Randell 1969, 16, 120). Mis on selle kriisi põhjused ja kuidas need väljenduvad? Järgnev nimekiri ei pretendeeri täielikkusele, aga sisaldab enamlevinud probleeme, mis tarkvaraprojekte vaevavad:

(1) Projektid lähevad üle eelarve
(2) Projektid lähevad ajaliselt lõhki
(3) Tarkvara kvaliteet on madal
(4) Tarkvara on ebaefektiivne
(5) Tarkvara ülalhoidmine on raske
(6) Tarkvara ei vasta ootustele (või nõuetele)
(7) Tarkvara ei saa valmis

Tarkvara arendusprotsessi on kammitsenud alati nõudmiste ja vajaduste konflikt: ühelt poolt soovivad tellijad pidevalt uusi omadusi, aga samas tahetakse vähendada arendustööle kulutatavat aega ja raha. Tarkvara k r i i s i k s muutub see olukord siis, kui nõudmised ja arengutempo kasvavad hüppeliselt. 1972. aastal ütles Edsger Dijkstra oma kuulsas loengus *The Humble Programmer*, et tarkvara kriisi peamine põhjus on see, et masinate võimsus on kasvanud kordades: „Kui arvuteid ei olnud, ei olnud programmeerimine üldse probleem, kui meil olid mõned viletsad arvutid, sai sellest väike probleem, ja nüüd, mil meil on gigantsed arvutid, on programmeerimisest saanud samuti gigantne probleem" (Dijkstra 1972, 861). Dijkstra tabab probleemi tuuma öeldes, et proportsionaalselt masinate võimsusega kasvab ka ühiskonna soov

seda võimsust kasutada. Programmeerija jääb mitme tule vahele (tööandja, tellija, ühiskond, erialane eetikakoodeks) ja talle langeb väga suur vastutus.

Tarkvarainseneri elukutse on piisavalt keerukas, et ühiskond hakkab seda müstifitseerima, juurdlemata tarkvaraga seonduvaid turva- ja finantsriskide üle (McConnell 1999, 63, 102). Tarkvararendaja professioon on võrreldav kõrgkeskaegse kivimeistri (*magister lapidum*) elukutsega. Seda eriti gooti katedraalide suurejoonelisuse kui ka nende sagedate kokkuvarisemiste ja pideva parandamise tõttu[7].

Joonis 1.1. Beauvais' koor: *code-and-fix* näide arhitektuuris.

Tarkvarasüsteemide arendamises on levinud nn *code-and-fix* meetod (McConnell 1999, 11). Ühiskonna kõrgendatud huvi ja majandsuliku surve tingimustes on see kõige kergem töömeetod, mille järgi tarkvaraarenduses haaratakse. Apple-i laadsed garaažis sündinud (ja vahel väga edukad) innovatsiooni-projektid on *code-and-fix* meetodi ilmekamad näited. Siin tuleb aga silmas pidada ühte iseloomulikku joont: *start-up* projekti risk langeb täielikult innovaatori (või riskiinvestori) õlgadele.

[7]... keskaja toodangu üldine kvaliteet on vastupidi levinud arvamusele ikkagi üldiselt halb. [...] Hoonete — ja eriti kirikute — kokkuvarisemine oli sagedane. Beauvais' koori sissevarisemine 1284. aastal [...] märgib paljude keskaegsete ehituste ühist saatust. [...] parandamise ekspertiisid muutuvad XIII sajandi lõpul isegi arhitektide peamiseks sissetulekuallikaks ja enamik keskaegse arhidektuuri meistriteoseid püsib veel ainult tänu parandamistele ja taastamistele (Le Goff 2001, 305-306)

Kui vennad Wrightid leiutasid lennuki ja seda katsetasid, siis riskisid nad ainult iseenda eluga. Kui kaasaegse lennuki pardakompuutrit programmeeriv tarkvarainsener teeb vea, riskib suur osa ühiskonnast. Vendade Wrightide meetodi levimisest tarkvaraarenduses hoiatati 1968. aasta NATO konverentsil: „me ehitame süsteeme nagu vennad Wrightid ehitasid lennukit: teeme kogu asja valmis, viskame selle kaljult all, laseme puruks kukkuda, ja alustame otsast peale" (Naur and Randell 1969, 17).

Joonis 1.2. Orville Wright sooritab testlendu (1902)

Suur majandsulik surve tarkvara valdkonnale, mida konverentsil korduvalt välja toodi (Naur and Randell 1969, 17-18,113,123), ei ole neljakümne aasta jooksul kadunud. Pigem vastupidi: ühiskonna huvi tarkvarainseneri professiooni vastu on kasvanud veelgi.

Selle ilmekaimaks näiteks on tõik, et tarkvarainseneri elukutse valiti MONEY Magazine ja Salary.com poolt 2006. aastal Ameerika Ühendriikides kõige paremaks elukutseks. Töökohtade võrdlemisel peeti silmas nende kasvupotentsiaali, tasutamist, stressi taset, loovust, raskust tööturule siseneda ja ametialast tuge leida ning tööja ja töökekkonna paindlikkust. Toimetus leidis, et „tarkvarainseneri vajab praktiliselt iga majandusharu, mis teeb sellest ühe kiireimini kasvava ametikoha Ameerika Ühendriikides" (Kalwarski et al. 2006). 2007. aastal tunnistati sama ajakirja poolt Ameerika kõige

nõutavamaks (*in-demand*) ametikohaks programmeerija ning uute ametikohtade esikolmikus on robotite programmeerija (*Robot programmer*) ja infoinsener (*Information engineer*) [8].

Ajalooliselt on inseneri kvalifikatsiooni formeerumisele kaasa aidanud ühiskonda raputanud õnnetused. Inseneri elukuse kujunemisele on nendest rohkem mõju avaldanud Quebeci silla kokkuvarisemine 1907. aastal ning Texase algkooli plahvatus 1937. aastal. Mõlemad nõudsid palju ohvreid ja andsid tõuke inseneriameti riikliku litsentsimise väljakujundamisele. McConnell ütleb, et on vaid aja küsimus, mil tarkvarast põhjustatud õnnetused ja nende tagajärjed saavutavad mõõtmed, mis kutsuvad ühiskonnas esile samasuguse kaitserefleksi nagu inseneriameti kujunemisel 100 aastat tagasi ning hakatakse kujundama ühiskonnapoolseid nõudmisi tarkvaratöötuse kõrgemate standardite järele (McConnell 1999, 57).

1.2.1. Tarkvarainseneri elukutse kui professioon

Nn tarkvarakriisist väljumiseks peaks tarkvarainseneri elukutse ja tarkvara tehnoloogia saavutama küpsuse (*maturity*). Paradoks on aga selles, et tööturg ja ühiskond laiemalt oma kõrgendatud huviga ei tekita juurde mitte tarkvarainsenere, vaid pigem nn MacGyver-programmeerijaid. Õigem oleks tegelikult ütelda, et MacGyveritel ei lasta kasvada inseneridеks. MacGyveri töövahendeid iseloomustab sõna t e i p. Tema lahendused töötavad niikaua, kui ta ise on kohal. Need on *ad hoc* lahendused, mida reeglina ei saa korrata. Ta on omapäraste töövõtetega kangelastüüp (*self-styled heroe*) [9], mitte meeskonnamängija. Tema hüpoteese on raske falsifitseerida. Edu kriteeriumiks ei ole lahenduskäigu universaalsus ja teoreetiline tõestamine, vaid pigem lihtne empiiriline konstanteering, et asi töötab (siin ja praegu). *IT works!* on ainuke tõekriteerium. Edasine uurimine ei ole vajalik.

Millised peaks olema tarkvarainseneri tegevuse uurimisstrategiad ja nen-

[8] `http://money.cnn.com/magazines/business2/jobs/2007/`
[9] McConnell 1999, 34

de hindamiskriteeriumid, et tarkvara valdkonda võiks nimetada küpseks ja võrdseks teiste insenerivaldkondadega? Mary Shaw ütleb, et kõigepealt peaks saavutama selle, et tarkvara valkonna enda sees kirjutataks eksplitsiitselt ja detailselt lahti juhised, uurimisstrateegiad ja valdkonnas väärtustatud hindamiskriteeriumid.

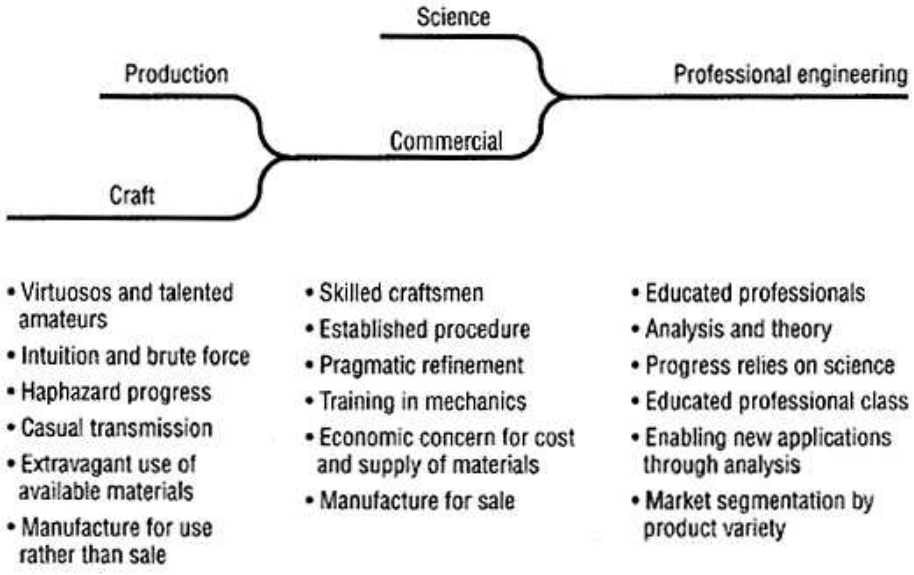
Teine oluline küpsust iseloomustav aspekt on Shaw meelest see, et nn outsaideritele (olgu selleks avalikkus, valdkonnaga kokku puutuvad õpilased) on kättesaadav lihtsustatud ja populariseeritud juhiste kogum, mis aitaks üldsusel ära tunda selles valdkonnas väärtustatud paradigmasid (Shaw 2002, 1). See on sama probleem, millele eelmises peatükis sai viidatud, et ühiskond tegelikult ei hooma tarkvara valdkonna spetsiifikat, hakates seda müstifitseerima ja väärarusaamu kujundama. Siit võivad kergesti tekkida põhjendamatult kõrged ootused ja nendega tavaliselt haakuvad pettumused.

Shaw kriitika juures on siinkirjutaja meelest hästi oluline see, et tarkvara inseneritegevuse printsiipide eksplitseerimine tähendab temal tegelikult endale kolme lihtsa küsimuse püstitamist. Need iseloomustavad Shaw meelest teaduslikku ja insneritegevuse uurimusi ning on järgmised:

(a) Mis tüüpi küsimused on antud valdkonnas „huvitavad"?
(b) Mis tüüpi tulemused aitavad nendele küsimustele vastata ja milliste uurimismeetoditega neid tulemusi võib saavutada?
(c) Mis tüüpi tõendid näitavad tulemuste kehtivust (*validity*) ja kuidas on head ja halvad tulemused eristatavad? (Shaw 2002, 2)

Shaw on tegelikult juba 1990. aastal üritanud modelleerida tarkvaratehnika evolutsiooni. See algab tema järgi amatöörlikust käsitööst ja kulmineerub inseneri professionaalse elukutsega. Skeem koos põhikarakteristikutega on kujutatud joonisel 1.3 [10].

[10]Shaw skeem põhineb tegelikult Ameerika inseneri James Kip Finchi (1883-1967) monograafial *Engineering and Western Civilization*, mille too avaldas 1951. aastal.



Joonis 1.3. Inseneri-distsipliini evolutsioon Shaw järgi (Shaw 1990, 17).

Esimest staadiumi, mida Shaw skeemil nimetatakse „käsitöö etapiks" oleme juba piisavalt kirjeldanud. MacGyver on selle etapi peakangelane: tal on kõik Shaw poolt loetletud omadused: erakordne talent, intuitsioon segamini brutaalse jõuga, spontaanne edu, kommunikatsiooni juhuslik iseloom, ekstravagantne materjalide kasutamine ja *ad hoc* lahendused (mitte seeriatootmine). Tarkvara vallas oli see staadium valdav eelmise sajandi 50-date lõpus ja 60-date aastate alguses. Amatöörlikule iseloomule vaatamata, ei tähenda, et need süsteemid oleksid olnud halvad või primitiivsed — vastupidi, mõned nendest olid piisavalt keerukad ja toimisid hästi (Shaw 1990, 20). Paljud tarkvaraprojektid opereerivad suurema või väiksema eduga endiselt sellele tasemel (McConnell 1999, 60).

Teine etapp on nn kommerts-etapp. Seda iseloomustab tugev majanduslik orientatsioon, pidev meetodite ja protseduuride viimistlemine. Nõudmine ületab pakkumise ja seeriatootmiseks on vajalik lisakapital. Finantsjuhtimine tõuseb ausse. Shaw toonitab, et kommertsfaasis tekib tugev seos tärkava teadusega ja see mõju on kahepidine: ühelt poolt praktilised probleemid stimuleerivad teaduse arengut, aga teisalt ka võetakse teaduslikud avastused

sageli kohe kasutusele (Shaw 1990, 17-18). Siin sünnivad Leonardo Da Vinci leiutised. Me võrdlesime eelnevalt tarkvaraarendajaid keskaegsete kivimeistritega, need on just selle etapi kangelased. Ehitusmeistrid ei ole enam lihtsad käsitöölised-müürsepad, vaid lasevad ennast nimetada „arhitektideks" ja „meistriteks" [11]. Igat liiki „arhitektide" ilmumine tarkvarasfääri ei ole juhuslik, vaid on sellele etapile väga iseloomulik [12].

Kolmandas etapis muutub Shaw järgi tarkvarainseneri tegevus professionaalseks. Probleeme ei lahendada enam katse-eksituse meetodil, vaid kaastakse ka teaduslik analüüs. Vaatleme seda etappi lähemalt.

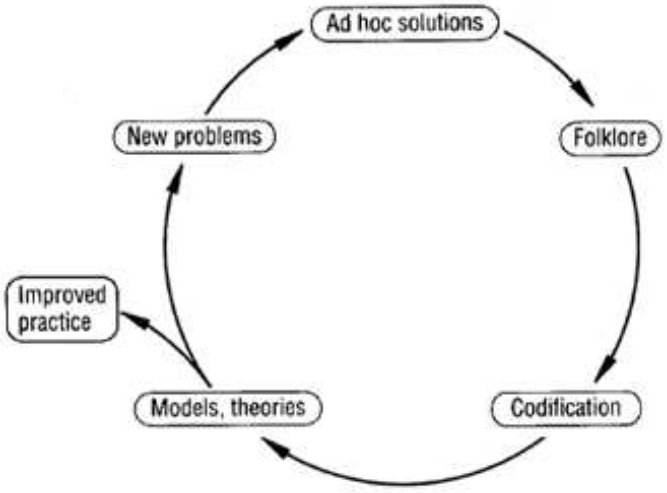
1.2.2. Professionaalne tarkvarainsenerlus ja teadus

Kui meenutada eelnevat Shaw skeemi, siis torkas silma, et juba teises etapis oli tarkvaratehnika ja teaduse vaheline seos kahepoolne. Viimasel etapil seos teadusega süveneb: ühelt poolt saab tarkvaratehnika arvutiteaduselt sisendina usaldusväärsed meetodid, teadusliku analüüsi ja väljatöötatud mudelid. Teiselt poolt aga niisutab ta teaduse kuiva kõnnumaad värskete ideedega millega too ülepea võiks tegeleda ning mida modelleerida.

Ideede areng toimub Shaw järgi ringikujuliselt. Esialgu insenerid implementeerivad lahendusi, kuidas nad parasjagu oskavad ja paremaks peavad. Mõne aja pärast märgatakse, et osad nendest *ad hoc* lahendustest töötavad päris hästi, mõned aga mitte. Need, mis enamasti töötavad, sisenevad „folkloori" (*folklore*): st toimub inimestevaheline mitteformaalne diskussioon nende lahenduste üle. Selle faasi läbimisel muutub asjadest arusaamine üha süstemaatilisemaks, lubades juba esmast kodifitseerimist heuristilise meetodi alusel ja kirjapanemist protseduurireeglitena. Lõpuks muutub kodifitsee-

[11] „Nende kahe ehitajatüübi kõrvutus ja mõnikord ka konflikt kestab teadupärast keskaja lõpuni ning just Milano katedraali ehitusplatsil XVI ja XV sajandi vahetusel toimub tähendusrikas vaidlus, kus vastanduvad prantslasest arhitekt, kelle jaoks „pole tehnikat ilma teaduseta" (*ars sine scientia nihil est*), ja lombardia müürsepad, kelle jaoks „pole teadust ilma tehnikata" (*scientia sine arte nihil est*)" (Le Goff 2001, 305).

[12] Sisuliselt on tegu terminoloogilise lõtvusega nagu ka disainerite nimetamisel „kujundusinseneridеks" ja psühholoogide nimetamisel „inimhingede inseneriks".



Joonis 1.4. Tarkvaratehnika ja teaduse interaktsioon (Shaw 1990, 21).

rimine piisavalt täpseks, et toeteda mudeleid ja teooriaid. Need omakorda parandavad praktikat ja selle praktika kogemus viimistleb teooriat. Minnes üha edasi, praktika prandamine lubab meil mõtelda juba raskematele probleemidele, mis alguses olid märkamatud ja mõeldamatud (Shaw 1990, 21-22).

Nüüd on igati õigustatud kriitiline küsimus, et kahtlemata võib kohata igal sammul esimese etapi käsitöölisi-programmeerijaid. Ka teine (nn kommertslik) etapp tundub tuttav. Aga kas kolmanda etapi esindajaid — nn professionaalseid tarkvarainsenere — võib leida ka tavaelus või on need rohkem insenerid „Start Treki" seriaalist? „Evolutsiooni" mõiste kasutamine Shaw poolt võib jätta mulje, justkui peaks üks etapp tarkvaratehnika arengus asenduma täiel määral teisega. Ometi see nii ei ole. McConnell märgib õigesti, et paljud tarkvaraprojektid opereerivad endiselt esimesel tasemel (McConnell 1999, 60).

Teine, nn kommertsfaas on tänapäeval ilmselt kõige laiemalt levinum. Samas toob Mary Shaw näiteid juba tarkvaratehnika varaste professionalse ilmingute kohta. Üheks selliseks on algoritmide ja andmestruktuuride abstraheerumine 60-date lõpus, mida veel kümnendi alul lahendati *ad hoc* iga programmmi raames eraldi (Shaw 1990, 21). Siia juurde võiksime omalt poolt

lisada veel Gang of Fouri [13] mustrite (*design patterns*) kataloogi, mis formaliseerivad ja abstraheerivad rakenduste ülesehitust.

Järelikult Mary Shaw skeemi kasutades on oluline silmas pidada, et eri etapid ei arene üksteise järgi nagu Marxil üheteisele järgnevad ühiskondlikud formatsioonid, vaid pigem eri etappide karakteristikud võivad igapäevases inseneripraktikas ilmneda läbisegi.

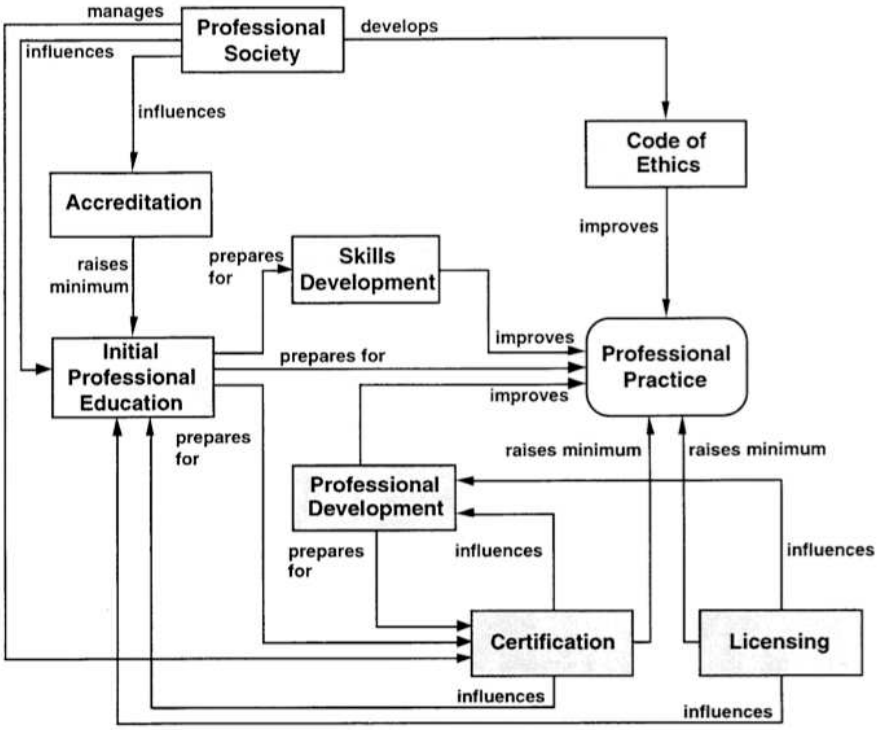
See on pannud mitmeid autoreid kahtlema selliste „evolutsiooniteooriate" paikapidavuses. Pete McBreen, *Software Craftmanship* autor (McBreen 2002) on seisukohal, et tarkvaratehnika on eksitav metafoor, sest ta läheneb asjadele mehhaaniliselt ja ei taba tarkvaraarenduse tõelist — so sotsiaalset ja intellektuaalset — loomust. McBreeni meelest peaks keskenduma väikestele hästitoimivatele tiimidele. Kui tarkvaraarenduses mingit edasiminekut on, siis on see saavutatav meistri–õpilase suhte läbi. McBreen võtab sisuliselt selle etapi, mida Shaw skeemil nimetatakse „käsitöö tasemeks" (*craft level*) ja kõrvaldab sealt lihtsalt olulised puudused. McBreeni ei vastanda enda teooriat tarkvarainsenerlusele ega arvutiteadusele, aga ta lisab olulise nüansi juurde, mis on siinkohal oluline, ja see on s õ l t u m a t u s t e a d u s e s t:

> „Tarkavarameisterlikkus ei vastandu tarkvarainsenerlusele ega arvutiteadusele. Pigem on tarkvarameisterlikkus erinev traditsioon, mis eksisteerib õnnelikult koos ja saab kasu teadusest ja inseneritegevusest. Nii nagu moodne sepp kasutab paremaid tööriistu, materjale ja teadmist, nii ka tarkvarameisterlikkus naudib paremaid arvuteid, taaskasutatavaid komponente ja programmeerimiskeeli. Just nagu sepp on sõltumatu teadusest ja inseneritegevusest oma oskuste ja kunsti osas, nii on ka tarkvarameisterlikkus sõltumatu arvutiteadusest ja tarkvarainsenerlusest, et luua suurepäraseid programme, rakendusi ja süsteeme" (McBreen 2002, XVI).

[13] Eric Gamma, Richard Helm, Ralph Johnson, John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994

Tarkvaraarendaja McBreenil niisiis pigem „sepp", kes töötab gildis meistri käe all, kui industriaalse vabriku „keevitaja". On vist asjatu lisada, et McBreeni meelest on tarkvarainseneride litsentsimine puhas „illusioon" (McBreen 2002, 40). Siinkirjutaja ei näe aga seda, et McBreeni mudel kummutaks sertifitseerimise vajadust, pigem ta isegi suurendab seda, sest tema mudel toob endaga kaasa paratamatult teatud suletuse: ühe meistri käe all saavutatud töövõtted ja renomee ei ole ülekantavad ja universaalsed.

Väga detailne skeem tarkvarainsenerluse kui professiooni arengus läbitavate elementide kohta on esitanud Garry Fordi ja Normann E. Gibbsi poolt (Ford and Gibbs 1996, 7).



Joonis 1.5. Eriala küpsuse mudel (Ford and Gibbs 1996, 7)

Fordi ja Gibbsi erialaküpsuse mudel koosneb praktiku ja infrastruktuuri tasemest ning kaheksast elemendist. McConnell on seda skeemi oma monograafias märkimisväärselt täiendanud, lisades juurde veel erialaühingute taseme ja organisatsioonide sertifitseerimise elemendi (McConnell 1999, 93-95).

1. **Esialgne erialane haridus** (*Initial professional educaton*). Hariduse omandamine enne erialasele tööle asumist. Siia alla kuulub ülikooliharidus ja erialakoolid.

2. **Akrediteerimine** (*Accreditation*). Ülikoolide programmide akrediteerimissüteem.

3. **Erialaste oskuste täiendamine** (*Skills development*). Paljudel puhkudel on enne kutsetunnistuse saamist lisaks erialasele haridusele nõutav ka vastav tööstaaž (Eesti nt audiitoritel).

4. **Sertifitseerimine** (*Certification*). Peale hariduse ja tööstaaži etapi läbimist on võimalik omandada sertifikaat.

5. **Litsentsimine** (*Licensing*). Sama mis sertifitseerimine, aga kohustuslik ja reeglina riigi juristdiktsiooni all.

6. **Professionaalne enesearendamine** (*Professional development*). Erialane täiendkoolitus kui elukestva õppe üks element. Kuulub eelkõige nende erialade juurde, kui toimuvad pidevad muutused, millega on vaja kaaas käia (nt meditsiinis, raamatupidamises, tarkvaravaldonnas).

7. **Erialaliidud** (*Professional societies*). Kuulumine erialaliitudesse professionaalse abi saamiseks. Erialaliidud aitavad hoida ka kutsestandardit (sertifitseerimisnõuded, eetikakoodeks).

8. **Eetikakoodeks** (*Code of ethics*). Kirjapandud või kirjutamata reeglid.

9. **Organisatsioonde sertifitseerimine** (*Organizational certification*). Lisaks indiviidide sertifitseerimisele võib taotleda ka organisatsioonide juhtimisprotsesside sertifitseerimist (ISO-sertifikaadid), seda eii juhul kui üksikisikute sertifitseerimine ei ole piisav. Selle elemendi on lisanud Fordi ja Gibbsi mudelile Steve McConnell (McConnell 1999, 95).

1.2.3. Teadmiste tuum

Elektri ja Elektroonika Inseneride Instituut (*Institute of Electrical and Electronics Engineers*), üks suuremaid USA inseneride seltse, on välja töötanud kõlava nimega korpuse — tarkvaratehnika teadmiste korpuse (*Software Engineering Body of Knowledge*) (SWEBOK 2004). Kuigi nime poolest sarnane projektijuhtimises levinud standardkorpusele PMBOK (*A Guide to the Project Management Body of Knowledge*), ei ole tal sellist autoriteeti. SWEBOKi kriitika edastamine ei ole käesoleva töö raamides võimalik, ometi peab märkima kindlasti selle dokumendi ühte fundamentaalset nõrka kohta, nimelt konsensuse puudumist — kuigi see oli üks selle dokumendi koostamise peamiseid eesmärke: st pakkuda üldaktsepteeritud teadmist (*consensually-validated characterization; generally accepted knowledge*) (SWEBOK 2004, IX). SWEBOK-i koostamisega oli alguses seotud ka ACM (*Association for Computing Machinery*), aga eitava seisukohtade tõttu tarkvarainseneride litsentsimise küsimuses on ta jäänud kõrvale [14]. SWEBOK on 2006. aastast ka ISO-standard (ISO/IEC TR 19759:2004) [15].

Selge on see, et pelgalt soovitava standardina, on selle dokumendi kasutegur väike ja kuna ta ei esinda tööstusharu konsensust, jääb tõenäoliselt riiulile tolmuma. Küll saab seda teadmiste korpust kasutada alusdokumendina tarkvarainseneride riiklikul litsentsimisel. Nii ongi kujunenud SWEBOK-ist tööriist litsentsimise-vaidluses. Selle dokumendi suhtes on tavaliselt positiivselt meelestatud need, kes pooldavad tarkvarainseneride litsentsimist. Võiks isegi ütelda, et litsentsimine omaette eesmärgina on saanud takistuseks tarkvarainsenerluse teadmise korpuse väljakujunemisel.

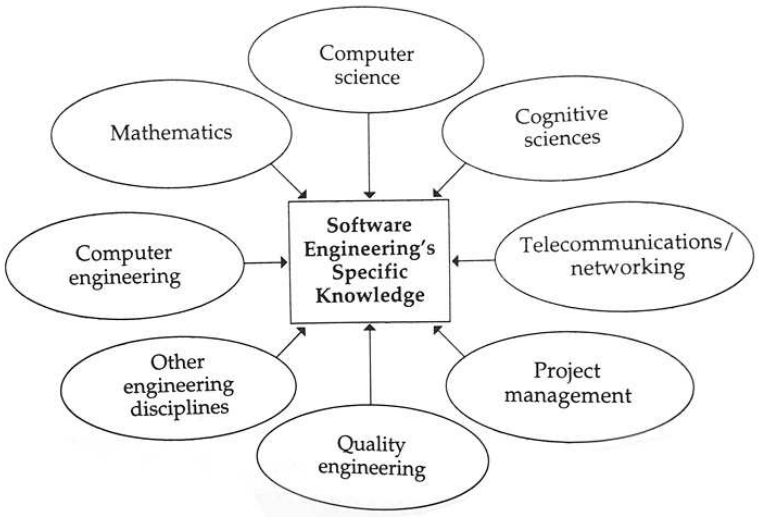
Väga iseloomulik on siinkohal Pete McBreeni seisukoht. Nagu eelnevalt sai juba öeldud, on ta litsentsimise suhtes kindlalt eitaval positsioonil. Nii on

[14] ACM Position on Software Engineering as a Licensed Engineering Profession, Association for Computing Machinery, 2000, `http://www.cs.wm.edu/~coppit/csci690-spring2004/papers/selep_main.pdf`

[15] `http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=33897`.

ta kujundanud ka eitava seisukoha tarkvara teadmiste korpuse suhtes. Millised on tema argumendid? McBreen ütleb, tarkvara on võrratult keerukam (*complex*) ja mahukam võrreldes traditsiooniliste inseneri-valdkondadega. Nt BMW auto pannakse kokku vähem kui 15 000 tükist, aga McBreeni monograafia ise sisaldab 50 000 sõna. Kosmosesüstiku tarkvara koosneb aga juba vähemalt 420 000 koodireast (McBreen 2002, 40). Tema argumendid on ilmselgelt retoorilised, aga iseloomulik on see, et tegelikult seisab ta vastu litsentsimisele.

Siinkirjutaja arvab, et McConnell annab olukorrast adekvaatsema pildi, et mingi stabiilne teadmiste korpus on saavutatav. McConnell ütleb, et kui jälgida tarkvarainsenerluse arengut 1968. aasta NATO konverentsist kuni aastatuhande vahetuseni, siis teadmiste stabiilne tuum (*stable core*) on kasvanud. St et inimese tänane investeering haridusse, kui ta on oma tarkvarakarjääri alguses, jääb oluliselt relevantseks terve karjääri jooksul (McConnell 1999, 83).



Joonis 1.6. Tarkvarainseneri teadmiste kaardistus (McConnell 1999, 86).

Antud skeemi vaadates tekib kindlasti küsimus, kuidas on võimalik, et tarkvarainsener omaks teadmisi nii mitmes valdkonnas: alates tarkvara konstrueerimisest kuni kognitiivsete teaduste (sotsioloogia, psühholoogia) ja tu-

runduseni välja? Vastus on see, et tarkvarainseneri teadmiste korpus on l a i a p õ h i n e. Tal peavad olema teadmised mitte ainult programmeerimisest. Huvitav on see, et kui me kõrvutaksime eelnevalt osutatud (ja litsentsimise tõttu eri seisukohta omavat) McBreeni ja McConnellit, siis siin langeb nende arvamus kokku. McBreen ütleb, et spetsialiseerumine ei ole efektiivne (McBreen 2002, 20-21). Sama väidab McConnell: tarkvarainseneril peab olema lai ettekujutus kõikidest faktoritest, mis puudutavad tema poolt loodavat toodet. See ristab tarkvarainseneri teadlasest, kes võib spetsialiseeruda (McConnell 1999, 38-39, 88).

1.2.4. Tarkvarainseneri eetikakoodeks

Kirjapandud eetikakoodeksid ei teki tühjale kohale. Enne seda peab olema tekkinud konsensuslik arusaam, mis on väärtuslik. 1925. aastal kasutati esimest korda Rudyard Kiplingi koostatud inseneride vabatahtlikku vannet „Insenerlikud kohustused". Vande andnud on äratuntavad raudse sõrmuse järgi, mis antakse üle „Inseneride vannutamise tseremoonial" (Gordon 2008, 49). Traditsiooni järgi on see sõrmus tehtud kokkukukkunud sillast, mis sümboliseerib inseneri vastutust ühiskonna ees.

Tarkvaratehnika valdkond elas pikka aega üldse ilma üldaktsepteeritud eetikakoodeksita. Nagu peatükis 1.2.2 sai näidatud, on eetikakoodeksi formeerumine üks erialaküpsuse tunnuseid. Tarkvarainseneri eetikakoodeksit hakati koostama ACM-i ja IEEE organisatsioonide poolt 1990-date aastale lõpus. Eetikakoodeks koosneb preambulast ja kaheksast põhiprintsiibist, ning sellest on kaks versiooni: lühike ja pikk (SECE 1999). Märkimisväärne on, et juba 1999. aastal juhtis McConnell tähelepanu asjaolule, et Eetikakoodeksi preambula suunab tarkvarainsenere oma ametikutsest alles t e g e m a (*make*), mitte aga hoidma (*maintain*) „kasulikku ja lugupeetud professiooni", mis justkui tähendaks, et ta v e e l seda ei ole (McConnell 1999, 136). 10 aastat hiljem on koodeksi preambula samasugune.

Lühidalt on eetikakoodeksi mõte järgmine: kuna arvutitel on keskne ja kasvav roll äris, tööstuses, valitsussfääris, meditsiinis, hariduses, meelelahutuses ning ühiskonnas laiemalt, siis on tarkvarainsenerid need, kes otseselt või kaudselt osalevad tarkvarasüsteemide analüüsis, spetsifitseerimisel, disainimisel, arendamises, testimises ja ülalhoius. Eetikakoodeks rõhutab, et selle rolli tõttu on tarkvarainseneridel märkimisväärsed võimalused teha head või põhjustada halba (või mõjutada teisi seda tegema). Kindlustamaks seda, et tarkavarainseneride püüdlused kannaksid makimaalselt head vilja, peavad tarkvarainsenerid oma ametist kujundama kasuliku ja respekteeritud professiooni. Selleks peab tarkvarainsener järgima kaheksat põhiprintsiipi, mis on järgmised:

1. **Avalikkus** (*Public*). Tarkvarainsener toimib avalikkuse huve silmas pidades.

2. **Klient ja tööandja** (*Client and employer*). Tarkvarainsener toimib nii, kuidas on tema kliendile ja tööandjale parim, kooskõlas avalikkuse huvidega.

3. **Toode** (*Product*). Tarkvarainsenerid peavad tagama, et nende tooted ning toodete modifikatsioonid vastavad kõrgeimatele võimalikele kutsestandarditele.

4. **Otsustus** (*Judgment*). Tarkvarainsener on oma professionaalses otsustuses aus ja iseseisev.

5. **Juhtimine** (*Management*). Tarkvarainseneridest juhid ja liidrid toetavad ja soodustavad eetilisi lähenemis tarkvara ülalhoiu ja arenduses juhtimises.

6. **Professioon** (*Professio*). Tarkvara insenerid edendavad eriala terviklikkust ja head mainet kooskõlas avalikkuse huvidega.

7. **Töökaaslased** (*Colleagues*). Tarkvarainsener on aus ja abivalmis oma töökaaslaste suhtes.

8. **Isiklik** (*Self*). Tarkvarainsener osaleb elukestvas õppes oma kutsealal ning edendab eetlisi lähenemisi kutsetegevuses.

1.3. Sertifitseerimise motivaatorid

Milleks saada sertifitseeritud? Kui õigusaktid ei nõua kutseeksami või sertifikaadi omandamist ja otseseid hüvesid sellega ei garanteerita, siis on see küsimus igati õigustatud.

Tarkvaraarendust võiks mitmete tunnuste alusel võrrelda raamatupidamisega (täpsus, vigadest tuleneva kahju suurus, erialase kogemuse tähtsus). Eestis loodab Raamatupidajate Kogu, et „piisava arvu sertifitseeritud raamatupidajate korral tekib siin nii isereguleeruvaid mehhanisme kui ka võimalusi otseseks reguleerimiseks" [16]. Paljudes riikides ei ole mõeldav raamatupidamise alane karjäär ilma kutsetunnistuseta [17].

Nagu eelnevalt sai väidetud, on sertifikaat tõend selle omaniku teadmiste hulga kohta, sertifikaati kasutamine on aga seotud peamiselt teadmiste k v a l i t e e d i argumendina konkurentsitingimustes. Kui kandidaat A väidab tööintervjuul endal olevat teadmised x, y ja z valdkonna kohta ja sama väidab kandidaat B, aga viimasel on oma teadmiste kohta pädevustunnistus, siis sertifikaadi mõte on aidata B-l oma teadmiste mahu kvaliteeti paremini tõestada. Näiteks, et tema teadmised ei piirdu vaid üksikute valdkondadega, millega ta senistes projektides on kokku puutunud, vaid tal on ka tööstusharust laiapinnalisem ülevaade. Samas sertifitseeritute arvu kasvades, võib sertifikaadi omamine konkurentsieelisena oma tähtsuse minetada — oma teadmiste kvaliteeti peab suutma tõestada ka muul moel.

[16] Tints, M. (2007). Raamatupidaja kutsestandarditest — *Raamatupidaja*, `http://raamatupidaja.ee/195482art/`.

[17] *ibid.*

Lihtsaim viis on vaadelda sertifikaati eeliste kaudu, mida selle omamine kaasa toob. Kõik organisatsioonid pakuvad teatud sarnaseid hüvesid:

(1) tunnustus ja tuntus oma erialal

(2) juurdepääs täiendavale tehnilisele infole

(3) tasuta tarkvara, ajakirjad jms

(4) logo kasutamise õigus (Sarv 1999a, 45)

Järgnevalt üritame kategoriseerida sertifitseerimise rakendamise motivaatoreid laiemalt. Motivaatorid sõltuvad kindlasti inimese vanusest. Huvitav on aga see, et värsked uuringud noorte (so alla 30 aasta vanuste) inseneride motivatsioonifaktorite kohta näitavad, et ühelt poolt joonistub välja selgelt vanusest sõltuv eelistus, teisalt aga otsib noorem inseneride generatsioon samasugust tunnustust ja respekti, mida eelnev inseneride põlvkond (Skibiski 2008, 1, 16).

(a) Sertifikaat kui k o h u s t u s, mis tuleneb vastava tööstusharu kutsestandardist. Kohustus on siin positiivne mõiste, kuna lähtub sertifitseeritavast endast. Näiteks sertifikaadita ei ole võimalik astuda kutseliitu [18]. Sertifikaadi omandamine võib olla kujunenud ka elukestva õppe üheks oluliseks komponendiks. Elukestev õpe on aga eetikakoodeksi osa (nagu tarkvarainseneride puhul).

(b) Sertifikaat kui k i n d l u s t u s. Sertifitseeritav näeb vajadust sertifikaadi järele, lootes selle abil oma konkurentsivõimet säilitada või tõsta. Reaalne hetkeline vajadus sertifikaadi järele võib isegi puududa.

(c) Sertifikaat kui i n v e s t e e r i n g. Sertifikaadi omandamine nõuab rahalisi, aga kindalasti ajalise ressursi olemasolu. Sertifitseeritav käsitleb sertifikaati kui vabade ressursside investeeringut lootes sellest tulu teenima hakata (kasutada sertifikaati müügiargumendina klientide juures, samuti töövõtjana karjääri planeerimisel või palgaläbirääkimistel).

[18] Eestis on olemas nt Sertifitseeritud Arvutispetsialistide Liit, mis loodi juba 1999. aasta juunis (Sarv 1999a, 44). See eksisteerib tänini, aga ei tegutse aktiivselt.

(d) Sertifikaadi taseme ü l e t a m i n e. Sertifitseeritaval on visioon kuidas tööstusharu standardeid muuta ja konstruktiivselt reformida. Selleks peab ta omandama täielikult senise oskusteabe, saavutama tööstuaharus piisava autoriteedi ja renomee.

(e) Sertifikaat kui m a n i p u l e e r i m i s v a h e n d. Kasutatakse enamasti tööandja poolt (nt arenguvestlustel, palgaläbirääkimistel, koondamistel) argumendina töövõtja vastu. Tarkvaratehnika vallas vastuolus eetikakoodeksiga, aga kahjuks (sh Eestis) levinud kasutusviis.

Ameerika Ühendriikide riikliku inseneride liidu NSPE (*National Society of Professional Engineers*) alamühingu *National Society of Professional Engineers* poolt viidi aastatel 2006–2007 läbi uuring, mis keskendus just noorte inseneride motivatsioonifaktoritele (Skibiski 2008, 5). Seitse kõige olulisemat motivatsiooni faktorit olid järgmised:

(1) Karjäär ja ametiredelil edasijõudmine
(2) Enesearendamine
(3) Palga kasv
(4) Soov tõestada enda väärtust
(5) Klientide rahulolu
(6) Töö atraktiivsus
(7) Professionaalse taseme saavutamise kohustus

Kui vaadata neid motivaatoreid, siis vaid töö atraktiivsus (*Interest Level in Job*), mille all mõeldi seda, et töö ei muutu rutiinseks ja ebahuvitavaks, ei ole otseselt seotud sertifitseerimise motivaatoritega. Ülejäänute puhul on sertifikaadi omamine vähemal või rohkemal määral asjakohane ning aitab eesmärke saavutada.

1.4. Kokkuvõte

Käesoleva peatüki eesmärk oli vastata kolmele peamisele küsimusele, mille töö käigus sai pakutud järgmised vastused:

(a) **Mida kujutab endast arvutialane tehniline sertifitseerimine ja milline on selle koht tänasel tarkvaraarenduse maastikul?** Erinevalt litsentsimisest on sertifitseerimine vabatahtlik ja universaalne. Kuna levinud sertifitseerijatena figureerivad tootjafirmad ise, siis on tarkvarasertifikaadid ka tehnoloogilises mõttes ajakohased. Sertifikaadi omandamist võib vaadelda tarkvarainseneri eetikakoodeksi kaheksanda punkti (elukestva õppe) valguses kui vältimatut etappi tarkvarainseneri enesetäiendamise protsessis. Käesoleva töö autor jagab Steve McConnelli seisukohata, et sertifitseerimist võib vaadelda samuti kui omamoodi „raudset sõrmust" (McConnell 1999, 112), mis traditsioonilistes insenerivaldkondades sümboliseerib ühelt poolt inseneriameti küpsust, sest see on teadlikult võtnud vastutuse ühiskonna ees, ja teiselt poolt sümboliseerib see konkreetse inseneri küpsust, kes on saavutanud professionaalse taseme.

(b) **Millist lisaväärtust pakub tarkvaraendajate sertifitseerimine tarkvaraarenduse teenuste kasutajatele ja laiemalt ühiskonnale tervikuna?** Kõige olulisem moment on siin avalikkuse turvatunne. Kui tegemist ei ole just väga spetsiifilise valdkonnaga, mis vajab eraldi litsentsimist, pakub tarkvarainseneride sertifitseerimine tarkvaraarenduse teenuse kasutajatele ja ühiskonnale tervikuna samu garantiisid, mille edukat toimimist võib jälgida raamatupidajate sertifitseerimise juures.

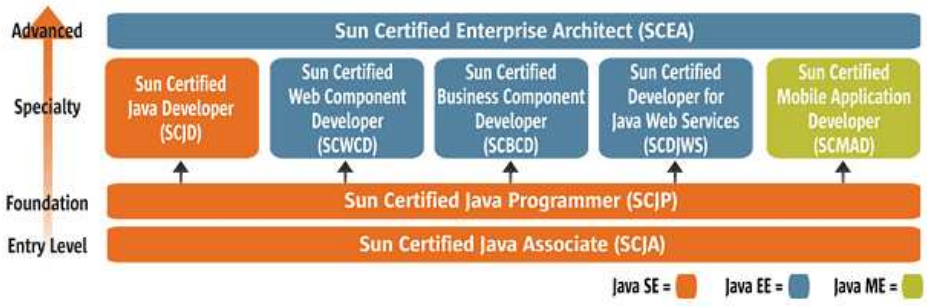
(c) **Miks peaks tarkvaraarendaja saama sertifitseeritud?** Me nägime, et sertifikaatide kasutusvaldkonnad on olulisel määral kokku langevad noorte inseneride motivatsioonifaktoritega. See tähendab, et noor tarkvarainsener võib loota, et sertifikaat aitab saavutada enamlevinud erialaseid eesmärke.

2. Java arendajate sertifitseerimine

Käesoleva peatüki eesmärk on analüüsida magistritöö käigus loodava õpivahendi-prototüübi vajadusi. Kõigepealt antakse ülevaade Java sertifikaadieksamite õpiteekonnast (*learning path*), seejärel SCJP eksami sooritustingimustest. Järgmisena võtab autor vaatluse alla kaks olemasolevat ja omas klassis parimat õpivahendit. Õpimetoodikatest peatutakse *Head First* raamatusarjas kasutatul ja selle metakognitiivsel alusteoorial. Viimaks kirjeldatakse, kuidas ekstreemprogrammeerimise (XP) kaudu tuntud paarisprogrammeerimine võib sertifikaadiõppes rakendamist leida.

2.1. Java sertifikaadieksamite süsteem

Sun Microsystems-i Java sertifikaadieksamite süsteemis võib eristada kolme põhilist taset: 1) baastaset, 2) spetsialisti taset ja 3) kõrgtaset. Õigupoolest lisati 2006. aastal juurde ka nn sisenemistase (*entry level*), aga see ei ole järgmiste eksamitega seotud ning ei ole suunatud programmeerijatele, vaid projektijuhtidele. Java sertifitseerimise õpiteekond näeb välja järgmine:



Joonis 2.1. Java sertifitseerimise õpiteekond (SUN 2009).

Java arendajate esimene põhieksam on *Sun Certified Java Programmer* (SCJP). See on eelduseksamiks kõikidele spetsialisti-tasemetele. Kõige kõrgema taseme (Java arhitekti eksami) puhul SCJP sooritamise kohustust ei ole. SCJP eksami nõuetel peatume allpool pikemalt. Märgima siinkohal vaid, et soorituse tingimusi (õigete vastuste protsendi nõuet, aega ja eksamil küsitava materjali sisu) on korduvalt muudetud. Praegu on võimalik teha eksamit kahe Java versiooni kohta (5 ja 6).

Kõige vanem spetsialisti-taseme eksam on *Sun Certified Java Developer* (SCJD). See eksam omandas suurema tähenduse peale 2003. aastat, mil programmeerija eksamist tuli välja Java 1.4 kohta käiv versioon, kust oli eemaldatud ekraanigraafika vahendite osa (`AWT, SWING`) ning samuti `java.io` paketi kohta käivad küsimused. SCJD eksam koosneb kahest osast: esimene on projekt, kuis tuleb kirjutada paari nädala joosul (otsest ajalist piiri ei ole) ca 3500 koodirea mahuga projekt. Eksami teine osa on essee nelja küsimuse põhjal eksamikeskuses. Eksami vautšerite hind on 375+300 USA dollarit.

Kui SCJD on töölaua-rakenduste spetsiifiline, siis *Sun Certified Web Component Developer* (SCWCD) on vastavalt veebirakenduste eksam. Eksam on Java Servleti API kohta (Servlet 2.4, JSP 2.0).

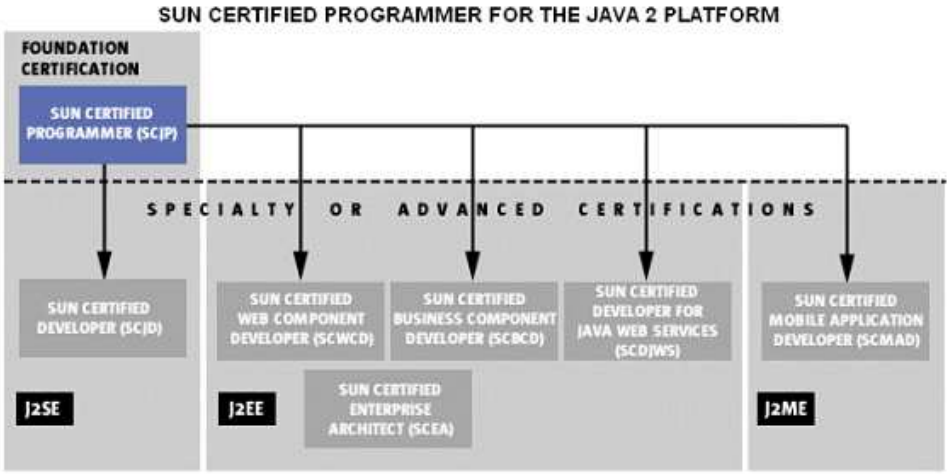
Kõlava nimetusega *Sun Certified Business Component Developer* (SCBCD) eksam käib tegelikult ainult Enterprise Java Beanide kohta. See (nagu ka veebirakenduste eksam SCWCD) sisaldab vähem koodinäiteid ja rohkem tekstilisi (aga väga detailseid) küsimusi konkreetse API versiooni spetsifikatsiooni kohta.

Sun Certified Developer For Java Web Services (SCDJWS) ja *Sun Certified Mobile Application Developer* (SCMAD) on noorimad spetsialisti eksamid (lisatud aastal 2004). Esimene hõlmab järgmiseid valdkondi: Java Web Services Developer Pack, JAX-WS ja JAXB, XML, JSON. Mobiilirakenduste eksam käib Java 2 Micro Edition (J2ME) kohta.

Kõikide eelmainitud (va SCJD) sertifikaatide saamiseks on vaja sooritada

üks eksam, mis maksab hetkel 300 dollarit. Hinda on aastate jooksul alates 150 dollarilt järk-järgult tõstetud.

Java sertifikaatide kõige kõrgem tase on *Sun Certified Enterprise Architect* (SCEA). Kui täpne olla, siis see kujutab endast kõrgtaset ainult JavaEE osale, mis on hästi näha ühel varasemal sertifitseerimise õpiteekkonna joonisel (vt joonis 2.2). SCEA erineb teistest eksamitest selle poolest, et tal ei ole eelduseksameid. Teiseks koosneb ta ise kolmest eksamist, millest esimene sarnaneb eelpoolmainitud spetsialistieksamitega (valikvastustega küsimused). Teine eksam on projekti koostamine (varem ajalist limiiti ei olnud, praegu on üks aasta). Projekti hinnatakse komisjoni poolt kvalitatiivselt, aga tulemus (mille saab teada 6–8 nädala jooksul) väljendub ikkagi protsendis. Mingeid kommentaare ja põhjendusi ei edastata. Kolmas etapp on projekti kaitsmine. See leiab aset jälle eksamikeskuses. Arhitekti eksami vautšerid maksavad vastavalt 200+375+300 dollarit.



Joonis 2.2. Varasem versioon Java õpiteekonnast (Steve Rowe, 2005).

2005. aastal muutis Sun Microsystems oma Java eksamid mitte-aeguvaks. Varem kehtisid need 2 aastat. Eksami vautšereid on võimalik osta Interneti kaudu, mis saadetakse postiga soovitud aadressile. Reeglina kehtivad need alates väljaandmisest üks aasta. Aegunud vautšereid välja ei vahetata ega kompenseerita. Eksami sooritaja peab arvestama veel eksami administreeri-

mise lisakuludega (Eestis ca 300 krooni). Eestis saab Java eksameid teha IT Koolituses ja BSC Koolituses.

2.2. SCJP eksami sooritustingimused

SCJP eksamist on alates 2000. aastast koostatud neli versiooni: Java 1.2, 1.4, 5 ja 6 kohta. Eksamid tulevad välja tavaliselt peale seda, kui JavaSE vastav versioon on piisavalt levinud ja omaks võetud. Kui eelneva versiooni JavaSE kohta on sertifikaadieksam sooritatud, võib teha ka lihtsustatud uuendus-eksami, mis on lühem ja keskendub suuresti vaid muudatustele. Allolevas tabelis on näha SCJP eksamid. Töö kirjutamise ajal (märts, 2009) saab neist teha ajaliselt kahte viimast:

Nr	Eksam	Aasta
CX-310-065	Sun Certified Programmer for the Java Platform, Standard Edition 6	2007
CX-310-055	Sun Certified Programmer for the Java Platform, Standard Edition 5.0	2005
CX-310-035	Sun Certified Programmer for the Java Platform, Standard Edition 1.4	2003
CX-310-025	Sun Certified Programmer for Java 2 Platform 1.2	2000

Joonis 2.3. SCJP eksamite versioonid (autori tabel)

Seitsme aasta jooksul on SJCP eksami sooritustingimustes tehtud olulisi korrektiive: muutunud on eksamiküsimuste arv, läbisaamiseks vajalike õigete vastuste protsent ning eksami pikkus. Alates versioonist 1.4 on eksamit võimalik sooritada ka enamlevinud võõrkeeletes. Erinevalt Microsofti

eksamitest inglise keelt mitte-emakeelena rääkivale eksamineeritavale Suni eksamitel lisaaega ei anta. Läbi aegade on samaks jäänud see, et *ca* 80% küsimustest on 6–30 realised koodinäited. Enamus on valikvastustega küsimused, kusjuures õigete vastuste arv on ette üteldud. Järgmine tabel annab ülevaate SCJP eksami sooritustingimuste muutustest:

Nr	Java	Küsimusi	Min õigeid	Min protsent	Sooritusaeg
CX-310-065	6	72	47	65%	210 min
CX-310-055	5	72	43	59%	175 min
CX-310-035	1.4	61	32	52%	120 min
CX-310-025	1.2	59	36	61%	120 min

Joonis 2.4. SCJP eksamite sooritustingimuste võrdlus (autori tabel)

Suuremad muutused on toimunud kolme esimese eksamiversiooni sisu osas: Java 1.4 eksamiga lisati juurde kinnitamised (*assertations*) ning võieti välja `java.io` ja ekraanigraafika vahendite (`AWT, SWING`) osa. Sun pidas ise 1.2 eksamit raskemaks kui 1.4. Kahtlemata ta seda ka oli, sest sooritusprotsenti alandati 52-le, samas kui eksami kestvus jäi samaks (2 tundi), aga maht oli oluliselt väiksem. Java 1.2 sertifikaadieksami ühes parimas õpikus (Mughal and Rasmussen 1999) moodustab ekraanigraafika ja `java.io` osa üle kolmandiku kogu õpimaterjali mahust.

Järgmises versioonis (Java 5) toodi `java.io` teek sisse tagasi, teegist `java.util` lisati juurde mitmeid uusi klasse (`Locale, Formatter, Scanner, regex.Pattern, regex.Matcher`). Java 5 tõi üldse arendaja jaoks kaasa palju muutusi, enamus neist kajastuvad ka SCJP eksamis: geneerilised tüübid (*generics*), staatilised impordid (*static import*), `enum`, *autoboxing/unboxing*. Eksamist jäeti välja igapäevases praktikas marginaalset tähendust omav nihutamine (*shifting*).

Java 6 eksam on praktiliselt sama, mis talle eelnev versioon. Eksami eesmärgid (*objectives*) on samad, muutused on detailides: lisatud on uued klassid `System.Console` (*API Contents*), `NavigableSet, NavigableMap` (*Collections/Generics*) ning välja on võetud meetodite ja konstruktorite ülekate (*OO Concepts*, 5-2). Eksam on järgmiste teemade kohta:

(1) Deklareerimine, isendiloome, skoop

(2) Voo reguleerimine

(3) Rakendusteegid

(4) Lõimed

(5) Objektorienteeritud programmeerimine

(6) Kollektsioonid / geneerilised tüübid

(7) Keele põhialused

Peale eksami sooritamist, saab eksamineeritav kohe teada, kui mitu protsenti küsimustest ta igas valdkonnas edukalt vastas ja kas eksam on sooritatud. Detailset informatsiooni küsimuste kohta raport ei esita. Kui eksamineeritav kukub läbi, võib ta korduseksamile tulla alles kahe nädala pärast. Ajaline piirang ei ole mitte „karistuseks", vaid seepärast, et vähendada võimalust, et esimese eksami küsimused meeles oleksid, sest need võivad uuesti korduda. Eksamiküsimused valitakse programmi poolt automaatselt.

Eksami edukal sooritamisel saadetakse eksamineeritavale US Letter formaadis tunnistus, magnetkaardi suurune kaart ja rinnamärk. Osade riikide puhul (sh Eesti) nõutakse eraldi lisalepingu sõlmimist, et saada õigust kasutada Java logoga kujunduselementi enda korporatiivse identiteedi kandjatel (nt visiitkaardil) [1].

Joonis 2.5. SCJP kaart ja märk.

[1] Vt `http://www.sun.com/training/certification/faq/#logos`

Ülevaadet sooritatud eksamitest, planeerida järgmiseid eksameid, laadida üles eksamitöid ning saada tuge on võimalik elektroonilises keskkonnas *CertManager* [2].

2.3. Ülevaade tüüpilistest õpivahenditest

Mis õpivahendeid kasutada SCJP eksamiks ettevalmistumisel? SCJP eksami eesmärk on eelkõige reaalse programmeerimiskogemuse testimine (Mughal and Rasmussen 1999, 607). Siinkirjutaja väidab, et järgmised neli asja on sertifikaadieksamiks valmistumise juures hädavajalikud:

1. **JDK ja käepärane arendusvahend**. Õppimise käigus tuleb kirjutada sadu triviaalseid pisikesi programme. Pole vahet, kas arendusvahendiks on Emacs või viimane Eclipse'i versioon — peaasi, et arendusvahend oleks käepärane, ja et ta oleks kogu aeg k ä e p ä r a s t. Marcus Green ütles väga tabavalt, et „valmistu saama i n i m k o m p i l a a t o r i k s" (*human Java compiler*) (Green 2004). Tüüpiline küsimus eksamil on *ca* 10-realine koodinäide, mille kohta peab otsustama, kas see 1) kompileerub ja käivitub edukalt; 2) kompileerub, aga saab käivitades erindi; 3) ei kompileerugi; 4) kompileerub, kui muuta rida x.

2. **API Dokumentatsioon**. Lisaks API dokumentatsioonile oleks hea käepärast hoida *The Java language specification* (Gosling et al. 2005).

3. **Üks SCJP eksami raamat**. Piisab ühest väga heast eksamile keskenduvast raamatust. Neid on turul mitmeid, aga kaks tükki on teistest peajagu üle: võiks valida kas Mughali raamatu (Mughal et al. 2008) kolmanda trüki (mis on ajakohastatud CX-310-066 eksami jaoks) või Kathy Sierra ja Bert Bates'i õpiku (Sierra and Bates 2008).

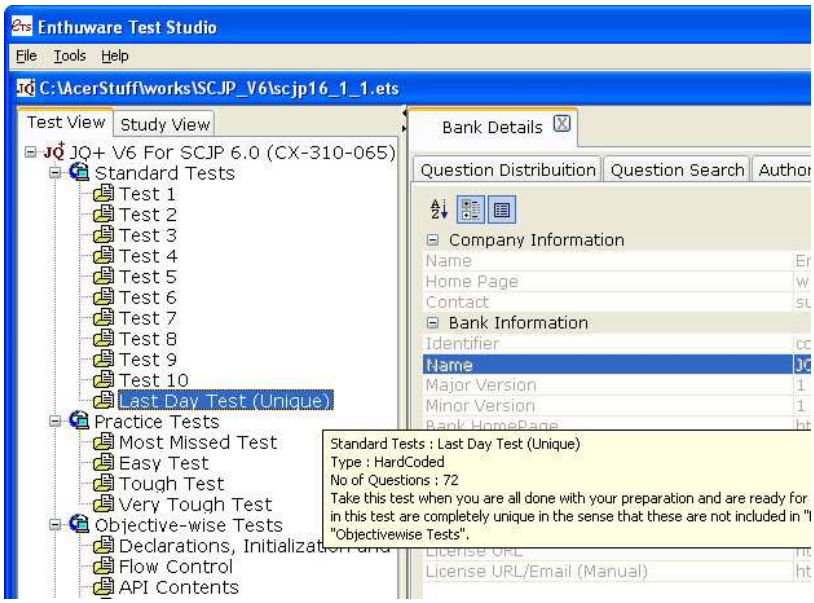
[2] Vt `http://www.certmanager.net/sun/`

4. **Eksami õpivahend**. Hea õpivahend on: 1) sama raskusega, mis Suni eksam; 2) vigadeta; 3) piisavalt suure küsimuste hulgaga; 4) abifoorumiga. Eksami õpivahendid jagunevad laias laastus kaheks: ühed on lihtsad ekasmisimulaatorid, teised on juba keerukamad õpikeskkonnad, mis lisaks testide lahendamisele aitavad ka analüüsida tehtud vigu.

Järgnevalt võtame vaatluse alla kaks tehnoloogilises mõttes erinevalt õpikeskkonda, ent oma klassis parimat lahendust: üks on veebi-, teine iseseisev kliendirakendus.

2.3.1. Enthuware Test Studio: JQPlus

EnthuWare JQPlus [3] on turul olnud juba kuus aastat. Alguses oli see lihtne simulaator, mis jooksis kõikidel platvormidel (kirjutatud Javas) ja saavutas suure populaarsuse Suni SCJP eksamile sarnaste küsimuste tõttu.

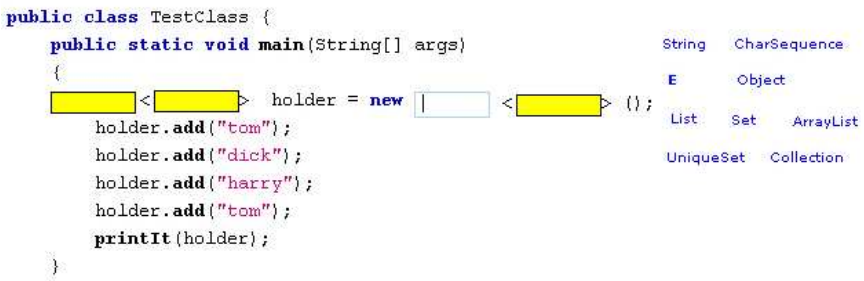


Joonis 2.6. Enthuware Test Studio kasutajaliides.

Aastate jooksul on JQPlus arenenud täismahuliseks õpikeskkonnaks (*Enthuware Test Studio*). Praegu sisaldab selle andmebaas 792 küsimust, mis

[3] Vt `http://www.enthuware.com/jqplus/`

tähendab, et kasutajal on võimalik koostada 11 eksamit, mille küsimused ei kordu. Suni Java eksamitel on järjest suurenenud nn hiirega lohistamise (*drag and drop*) tüüpi küsimuste osakaal, mida iseseisvas kliendirakenduses on kahtlemata kergem implementeerida kui veebirakenduses. Joonisel 2.7 on üks lihtsamat sorti lohistamisküsimuse näide. Selles peab kasutaja lohistama paremal pool olevatest võtmesõnadest õiged valikud vasakul pool olevatele kohtadele ning lisaks ühe välja täitma:



Joonis 2.7. *Drag and drop* küsimus JQPlus programmis.

Kuigi JQPlusi näol on tegu iseseiva allalaaditava rakendusega, kus ka küsimuste andmebaasi hoiakse lokaalselt, on lahendatud selle pidev ajakohastamine. Samuti on Enthuwarel häistitoimiv ja populaarne tugifoorum *JDiscuss.com* [4].

2.3.2. JavaCertificate.com

JavaCertificate.com [5] on veebipõhine õpikeskkond. See on hea näide, kuidas suhteliselt lihtsate veebiprogrammeerimise vahenditega on realiseeritud eksamisimulaatori põhifunktsionaalsus. Kasutajal on võimalik lahendada valikvastustega küsimusi ja õpikeskkond kontrollib need ära, esitades graafilise raporti ning kasutajal on võimalik vastatud küsimusi üle vaadata ja lugeda küsimuste juurde lisainformatsiooni. Nad on küll jäänud pidama Java 1.4 eksami versiooniga, aga see ei vähenda selle õpikeskkonna kontseptsioo-

[4] `http://www.jdiscuss.com`
[5] `http://www.javacertificate.com`

ni väärtust. Sarnaselt päris eksamile saab *JavaCertificate.com*-i keskkonnas küsimusi hilisemaks üle vaatamiseks ära märkida; realiseeritud on ka primitiivne ajaloendur. *JavaCertificate.com*-i üks huvitav kontseptsioon on see, et ta hoiab kasutajal silmade ees kogu aeg raportit tema edasijõudmise kohta eksami eri eesmärkide kaupa. Eraldi on välja toodud vigaselt vastatud küsimused, mida saab uuesti sooritada:

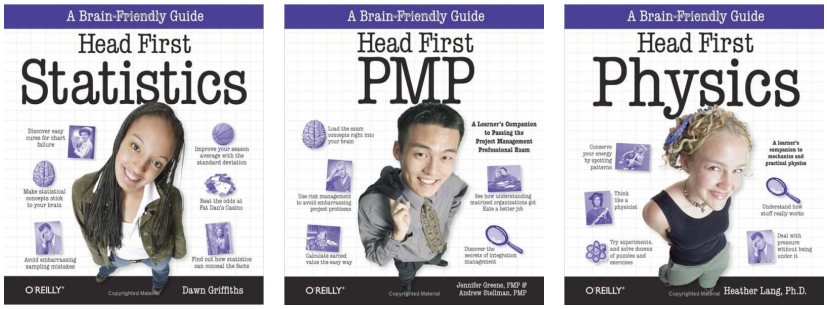
Joonis 2.8. Valdkondade grupid JavaCertificate.com keskkonnas.

2.4. Kathy Sierra *Head First* raamatusari

Head First on väga kiiresti populaarsust võitnud õpimetoodika, mille on välja töötanud Kathy Sierra ja Berth Bates'i oma *Head First* õpikuteseerias, mida annab välja O'Reilly kirjastus. Esimesed raamatud ilmusid Java kohta 2003. aastal. Nii Sierra kui Bates omavad pikaajalist Java õpetamise kogemust.

Sierra on ka Java kogukonna JavaRanch.com üks asutajatest asutaja. JavaRanchi idee oli luua Java õppurile sõpralik keskkond (selle moto on: *friendly place for Java greenhorns!*). Sierra on tunnistanud, et JavaRanchi loomise ajal 1997. aastal oli vähe Java kohta kirjutatud raamatuid ja Internetis figureeris Java kogukonnana vaid uudisegrupp `comp.lang.java`, kus algaja võis oma „rumalate" küsimuste esitamise peale lihtsalt sõimata saada (Schreckmann 2004). „Minu eesmärk oli luua naljakas ja sõbralik koht, kus inimene saaks õppida tundmata ennast rumalana" (*ibid.*). Selle sama kontseptsiooni põhjal on sündinud ka *Head First*-i raamatusari. Sierra ütleb, et ta eesmärk oli luua selline raamat, mida ta ise tahaks omada, kui ta esmakordselt mingi teemaga kokku puutuks (*ibid.*).

Õpikud põhinevad metodoloogiliselt metakognitiivsel lähenemisel. Autorite viis Javat õpetada on osutunud lühikese ajaga viljakaks, leides kasutamist ka väga abstraktse ja raskesti edasiantava programmerimisteemade õpetamisel, nt disainimustrite (*design patterns*) puhul [6] ja objekt-orienteeritud analüüsi ja disaini [7] käsitlemisel. Samas ei ole *Head First* populaarne mitte ainult programmeerimise või IKT valdkonna teemade õpetamisel. Sissejuhatuses sai juba viidatud, et üks selle metoodika implementatsioone on projektijuhtimise vallast, täpsemalt projektijuhtimise *The PMBOK Guide*-i sertifikaadieksami õpik [8]. *Head First* raamatusarjas on ilmunud seitsme aasta jooksul juba üle kahekümne raamatu, mitmetest kordustrükid [9].



(a) HF Statistics (b) HF PMP (c) HF Physics

Joonis 2.9. Mõned värsked *Head First* õpikute näited.

Esmapilgul tekitavad need õpikud tavaliselt kummastust, kuna meenutavad vormiliselt isegi koomiksit ja on justkui vastuolus käsitletavate teemade raskuse ja tõsidusega. Ent autorite püüd ei ole teps mitte lugeja lõbustamine, vaid panna teda kiiremini õpitavat omandama. See saavutatakse metakognitiivse lähenemise kaudu: lugejat üritatakse panna mõtlema sellest, k u i d a s

[6] E. Freeman, B. Bates, K. Sierra (2003), *Head First Design Patterns*, O'Reilly.
[7] B. McLaughlin, G. Pollice, D. West (2006), *Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D*, O'Reilly.
[8] A. Stellman, J. Greene (2007), *Head First PMP*, O'Reilly.
[9] Vt `http://oreilly.com/store/series/headfirst.csp`

ta mõtleb ja teda teadvustama k u i d a s ta õpib. *Head First*-i metakognitiivse lähenemise juured on hariduspsühholoogias. Mõistet „metatunnetus" (*metacognition*) seostatakse tavaliselt Ameerika psühholoogi John H. Flavelli nimega (Flavell 1979). Flavell defineerib metatunnetust kui „tunnetust tunnetuse kohta" (*cognition about cognition*) (Flavell 1985, 104). Seda võib mõista kui teadmist oma kognitiivsest tegevusest, mis on tekkinud läbi kogemuse (*accumulated through experience*) ja see on salvestatud meie pikaajalisse mällu (Flavell 1985, 105). Flavelli järgi tekib selline metatunnetuse ja reflekteerimise võime alles täiskasvanuks saades, mängides olulist rolli õppimise juures (Flavell 1985, 116).

Rohkem kui metekognitiivse lähenemise klassikud (nagu Flavell) on Sierra *Head First*-i kontseptsiooni mõjutanud e-õppe teoreetikud, eelkõige Roger Schank ja tema nn „lugude-teooria" (*storytelling process*). Schanki lugude-teooria on lühidalt hästi lihtne: ta vaatab kõigele läbi lugude rääkimise prisma: me anname edasi oma kogemusi ja isegi abstraktseid mõtteid rääkides lugusid, sama moodi me võtame vastu teiste mõtteid lugude kaudu. Inimmälu, mõtlemine ja teadmine on Schank väitel loo-põhine (Schank 1995, xliv, 12). Niiviisi refereerituna tunduvad Schanki ideed ehk isegi triviaalsed, ent selle taga on tal detailne kirjelduse mälu funktsioneerimisest märgistamise ja indekseerimise süsteemi kaudu (Schank 1995, 84). *Head First*-i materjali ülesehitus põhineb suuresti Schanki lugude-rääkimise skeemil. Iga *Head First*-i raamatu lehekülg räägib mingi loo, kasutades atraktiivsemaks tegemisel kõikvõimalikku visuaalset abimaterjali. *Head First*-i ülesehituse põhiprintsiipidena võib välja tuua järgmised punktid (Sierra and Bates 2003, xxii-xxiii):

1. **Aeglane õppimine.** Raamatut ei tule lihtsalt lugeda, vaid selle üle tuleb järele mõtelda. „Rumalaid küsimusi ei ole"-printsiip tähendab, et kõikidele küsimustele tuleb püüda vastata. Esitatud küsimustele ei ole üks-üheselt võimalik vastata.

2. **Harjutuste tegemine.** Oluline on, et harjutuste tegemise juures oleks õppija füüsiliselt aktiivne (nt kasutaks pliiatsit vms).

3. **Kõva häälega õpitavast rääkimine.** Parem veel, kui oleks võimalik kellegi teisega vestelda. Rääkimise kaasamine aktiveerib mõlemad ajupoolkerad.

4. **Millegi tundmine.** Igasugune emotsioon (huumor, üllatus, huvi) tugevdab materjali kinnistumist. Seda eesmärki teenivad mitteformaalses stiilis jutustatud lood ja visuaalne (ja ootamatul viisil kasutatud) materjal. Õppimise ajal tuleks ka vahetada kohta (ruumi jne).

5. **Enda vasuvõtuvõime monitoorimine.** Üleväsinuna pole mõtet õppida. Joo palju vett. Meenuta õpitut vahetult enne magamaheitmist.

6. **Korduste kasutamine.** Üht ja sama asja kirjeldatakse korduvalt ja eri meeli afekteerides.

7. **80/20 printsiip.** *Head First* katab reeglina vaid osa, aga sagedamini kasutatava alamhulga õpitava teema kohta.

2.5. Paarisprogrammeerimine ja sertifikaadiõpe

Nn paarisprogrammeerimine [10] on ühe levinuma agiilse tarkvaraarenduse — ekstreemprogrammeerimise (XP) poolt kasutatud praktika, mille käigus ühte programmeerimisülesannet lahendavad kaks arendajat koos. See toimub järgmiselt: üks programmeerija kirjutab tellija loole testi, teine selle testi implementatsiooni. Testid kirjutatakse enne koodi kirjutamist (nn *test-driven development*). Praktiliselt näeb see nii välja, et töötatakse ühe paarilise töölaua taga. Kodeerija ja kõrvalistuja rolle vahetatakse päevas mitmeid kordi. Mõlemad on aktiivsed, üks ei ole pelgalt passiivne kuulja. Pidevalt

[10] Vt `http://www.pairprogramming.com`

toimub refaktoorimine ja integreerimine. Kui paarisprogrammeerimine on kasutusel projektis, võib paare moodustada jooksvalt igahommikusel projektikoosolekul. Joonisel 2.9 on kaks paarisprogrammeerijat. Pildil võib näha, et vaskpoolne programmeerija, on isegi aktiivsem, kuigi ta ise hetkel koodi ei kirjuta.

Joonis 2.10. AS Webmedia paarisprogrammeerijad (autori foto).

Kogemus on näidanud, et sellist metoodikat ei jaksa praktiseerida korraga üle poole tööpäeva. Ent tulemuseks on kood, mis on tunduvalt kvaliteetsem. Vigade arv programmis on reeglina väiksem. Programmeerijate kompetents on suurem, sest teadmine, kuidas midagi teha, ei ole üksikul asendamatul inimesel, vaid on hajutatud projektitiimi laiali. Väheneb oht ühte arendajat üle koormata ning et ühe arendaja puudumisel jäävad tema asjad tegemata. Kehtivad ühtsed kodeerimise standardid. Sellise arendamise juures on aga ületunnid välistatud (reegel on see, et töönaädal ei tohi venida üle 40-tunni ja hiljemalt kl 5 koju).

Paarisprogrammeerimist on võimalik edukalt rakendada ka sertifikaadieksamiks ettevalmistumisel. Eriti just tehtud vigade analüüsimisel. Mitmed printsiibid, mis me eelnevalt *Head First* õpimetoodikat kirjeldades välja tõime, on kattuvad: ületöötamise vältimine, kõva häälega rääkimine, küsimuste esitamine (Sierra and Bates 2003, xxiii). Mõningaid metakognitiivse

õpimetoodika põhimõtteid on paarisprogrameerides oluliselt loomulikum rakendada: nt kellelegi teisele seletamine või probleemide „läbi elamine" (läbi nalja, üllatuse või huvi) (Sierra and Bates 2003, xxii). Teadmise kinnitamiseks naljade või lugude rääkimine on partneriga võrratult loomulikum, kui üksinda arvuti taga ülesandeid lahendades.

2.6. Kokkuvõte

Käesoleva peatüki eesmärk oli analüüsida magistritöö käigus loodava õpivahendi-prototüübi vajadusi. Esmalt määratlesime SCJP sooritustingimused ja näitasime, kuhu see eksam kuulub Sun Microsystems-i Java sertifikaadieksamite õpiteekonna (*learning path*) kontekstis laiemalt.

Järgmisena võtsime vaatluse alla kaks olemasolevat ja omas klassis parimat õpivahendit (Enthuware JQPlus ja JavaCertificate.com). Need rakendused on oluliseks eeskujuks järgmises peatükis kirjeldatava õpivahendi-prototüübi koostamisel. Käesolev peatükk peatus ka *Head First* raamatusarjas kasutatud matakognitiivsel metoodikal. Seejärel näidati, et *Head First*-i õppimise printsiibid on edukalt kombineeritavad agiilses tarkvaraarenduses levinud paarisprogrammeerimisega ja koos sertifikaadiõppes rakendatavad.

3. Õpivahendi loomine RIA-rakendusena

Käesoleva peatüki eesmärk on kirjeldada SCJP õpivahendi-prototüübi loomist kasutades Adobe Flexi arendusraamistikku. Prototüübi loomise käigus püütakse samuti välja selgitada, kas Adobe Flex on sobilik arendusplatvorm loomaks korduvkasutatavaid õpiobjekte, mida saab integreerida suurematesse õpikeskkondadesse.

Adobe Flex sai valitud autori poolt prototüübi arendusplatvormiks järgmistel põhjustel:

(1) FlexBuilder Pro haridus-litsents on tasuta [1].

(2) Arendusvahend FlexBuilder on olemas ka Mac OS X-ile.

(3) Flexi arendajate kasutajakogukond on suur ja aktiivne.

(4) Flex SDK on avatud lähtekoodiga.

(5) Flexi rakendust on kerge integreerimida teiste Flashi objektidega.

(6) Flexi rakendust on kerge integreerimida serveripoolse Java kihiga.

(7) Programmeerimiskeeleks on Flashi kaudu tuntud ActionScript.

(8) Flexi toetavad tänu Flashi levikule juba enamus arvuteid [2].

(9) Flex on kõige toodanguküpsem RIA raamistik (hetkel versioon 3.3).

Mis on RIA? RIA ehk rikas Interneti-rakendus (*rich Internet application*) on tüüpiliselt brauseri kaudu avatav veebirakendus, mis on oma võimaluste poolest võrreldav traditsiooniliste töölaua-rakendustega. Tuntuimad RIA-raamistikud on Adobe Flex/AIR, Microsoft Silverlight ja JavaFX. RIA-rakendus vajab, et brauser toetaks seda vastava plugina abil. RIA-rakendusi

[1] `https://freeriatools.adobe.com/`

[2] Adobe väitel 98-99 % Internetti-ühendatud arvutied `http://www.adobe.com/products/player_census/flashplayer/version_penetration.html`

võib käivitada ka väljaspool brauseri piiratud ligipääsuga liivakasti (*sandbox*). Adobel on selleks puhuks tehtud veel eraldi käituskeskkond *Adobe Integrated Runtime* (AIR), mis võimaldab jooksutada Flashi, Flexi ja HTMLi/AJAXi rakendusi. RIA-rakendus ei ole veebilehekülje-keskne, see mäletab kliendi olekut (on *stateful*).

3.1. Adobe Flexi programmeerimismudel

Flex on koondnimetus mitmele eri tehnoloogiale (ActionScript, Java, XML), mille abil on loodud arendusplatvorm koos standartsete valmiskomponentidega. Rakenduste kõige lihtsam arendamine käib deklaratiivsel meetodil (MXML). Flex SDK on platvormist sõltumatu. Üks väärarusaam Flexi kohta on see, et see on midagi muud kui Fash. Flex ongi Flash: see jookseb Flash Playeris; Flex on ehitatud Flahi API peale, saades kasutada kõiki Flashi rikkusi. Flexi mõte oli teha Flashi platvorm kättesaadavaks traditsioonilistele tarkvaraarendajatele (Sanchez and McIntosh 2009, xix).

Flexi programmeerimismudel on väga sarnane JavaServer Pages-i programmeerimismudelile: Flex konverteerib MXMLi deklaratiivse koodi automaatselt ActionScripti klassideks (samuti nagu JSP-d konverteeritakse Servlettideks) ja seejärel kompileerib selle baitkoodiks (SWF), mida käivitatakse Flashi virtuaalmasinas (ActionScript Virtual Machine). Alates 9. versioonist paranes jõudlus sellega, Flashi Playeri JIT kompilaator konverteerib SWF baitkoodi veel eelnevalt masinkoodiks.

Sellise programmeerimismudeli eesmärk on eraldada kasutajaliidese disain (mis on defineeritud MXMLis) ja rakenduse kood (ActionScripti klassid). Suurtes projektides on nende arendajad reeglina erinevad inimesed. MXMLi sisse võib kirjutada ka otse ActionScripti koodi, aga see muudab kogu rakenduse väga raskesti hallatavaks ja automaatselt testitavaks. Flexi koodi on võimalik testida kasutades avatud lähtekoodiga FlexUnit raamistikku [3].

[3] `http://code.google.com/p/as3flexunitlib/`

Suures Flexi projektis on tavaliselt eri taustaga programmeerijad ja igaüks kipub oma seniseid häid arendustavasid (*best practices*) ja keele paradigmasid Flexi üle tooma. Flexi ja Java koolitaja Yakov Fain tõi näite kuidas COBOLi, Java ja Smalltalki taustaga programmeerijad lahendaksid sama ülesannet Flexis — tulemused erineksid kardinaalselt, aga ükski nendest ei ole Flexi seisukohalt ainuõige (Fain et al. 2007, 101-104). See tähendab seda, et tänasel päeval on tuleb arvestada asjaoluga, et Flexi on hakanud arendama väga erinevate programmeerimiskultuuridega arendajad, mistõttu disainimustrite ja raamistike (*frameworks*) kasutamine on veel kaunis stiihiline ja efektiivsemate väljaselgitamine alles kujunemisjärgus (Paavel 2008, 66-67). Adobe on seni panustanud pigem Flexi kasutajaliidese komponentide täiustamisse, jättes rakenduse arhitektuurilised lahendused arendajate endi valida. Flexi kogukonnas on enim levinud Cairngorm [4] raamistik, mis põhineb *Model-View-Controller* (MVC) disainimustril.

3.2. Mitte-funktsionaalsed nõuded

Siinkirjutaja klassifitseeris õpivahendi prototüübi nõuded kaheks: ühed on mitte-funktsionaalsed ja teised rakenduse enda kasutusloogikat puudutavad funktsionaalsed nõuded. Mitte-funktsionaalsed nõuded puudutavad rakenduse ülesehitust, käideldavust ja talitlusvõimet. Need on järgmised:

(1) Rakendus peab olema platvormist sõltumatu.

(2) Rakendus peab olema brauserist sõltumatu.

(3) Prototüüp peab töötama ka *standalone* rakendusena (nt CD-lt).

(4) Rakendus peab töötama ka Interneti-ühenduseta.

(5) Küsimuste andmebaas peab olema XML-i kujul.

(6) Rakendust peab saama integreerida suvalise veebikeskkonnaga.

(7) Rakendust peab saama parameetriseerida.

[4] `http://opensource.adobe.com/wiki/display/cairngorm/Cairngorm`

Siinkirjutaja annab endale aru, et selline nõuete komplekt on suhteliselt harvaesinev. Aga karmid nõuded ei tee kindlasti prototüübi testile halba. Esimesed kolm punkti on saavutatud sellega, et loodav rakendus kasutab Flashi Playerit.

Neljandat ja viiendat punkti võib vaadelda kompleksselt. Prototüübi rakendus on kompileeritud käsuga `-use-network=false`, mis paigutab ta *local-with-filesystem* turvalisuse liivakasti. Teine alternatiiv oleks seadistada rakendus kasutama *local-with-networking* liivakasti ning laadida küsimused Interneti aadressilt ja salvestada tulemused samuti serverisse. Küsimuste andmebaas on hetkel samas kataloogis kus rakenduse SWF-fail. Küsimuse kirjeldamine andmebaasi XML-faili struktuuris näeb välja järgmine:

Koodinäide 1 Küsimuse kirjeldamine andmebaasi XML-faili struktuuris.

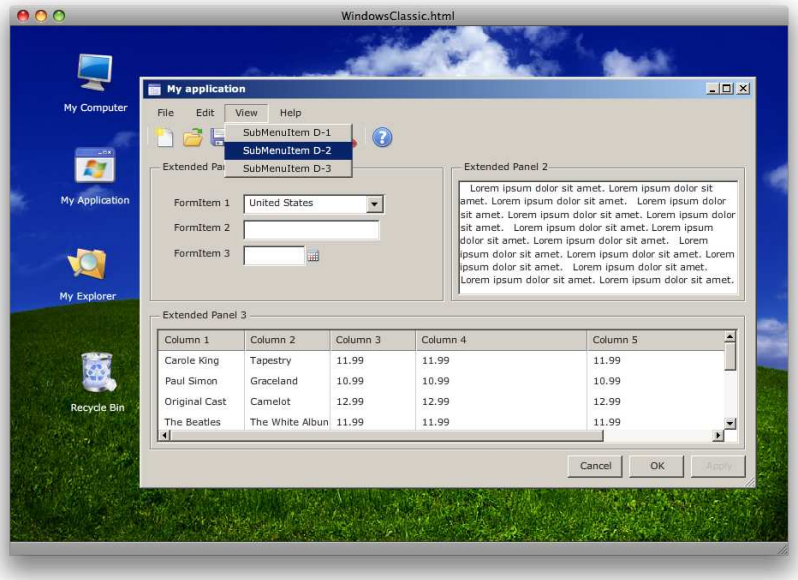
```
<question id="2007122" category="2" weight="2">
   <title>Consider the following code snippet. What can be inserted
      at // 1, which will destroy the object referred to by obj?
   </title>
   <code>
       void myMethod() {
          Object obj = new Object();
          // 1 insert code here
       }
   </code>
   <question_body>
      <element type="checkBox" isTrue="false">
         <qtext type="code">obj.destroy();</qtext>
      </element>
      <element type="checkBox" isTrue="false">
         <qtext type="code">System.getRuntim().gc();</qtext>
         <qcom>It will not necessarily run the garbage collector.</qcom>
      </element>
      ...
      <element type="checkBox" isTrue="true">
         <qtext>None of these.</qtext>
      </element>
   </question_body>
   <comment>
      Nothing can ensure that an object will definitely be destroyed
      by the garbage collector. You can at most make an object eligible
      for GC by making sure that there are no references to it.
   </comment>
</question>
```

Küsimused loetakse XML-failist ActionsScripti klassidena defineeritud objektide hierarhiasse (küsimustik, küsimus, vormi element) Flexi rakenduse initsialiseerimise ajal ja hoitakse mälus. Nt raadionupu vormi element initsialiseeritakse XML-failist järgmise ActionScripti klassi abil:

Koodinäide 2 Raadionupu klass.

```
public class RadioButtonVO extends FormElementVO {
   public var checked:Boolean = false;
   public function RadioButtonVO(o:XML) {
      super(o.@type,o.qtext,o.@id,o.qtext.@type,o.@predefValue,o.qcom);
   }
}
```

Mitte-funktsionaalsete nõuete kuues ja seitsmes punkt on saavutatav sellega, et Flexi rakenduse SWF-objekti saab parameetriseerida JavaScripti kaudu (`flashVars`) ja integreerida suvalise veebilehega. Järgnev näide illustreerib Flexis koostatud Windowsi traditsioonilise töölaua-laadset keskkonda. Rakendus on avatud Safari brauseriga. Õpivahend või terve õpikeskkond võib omandada kasutajale harjumuspärase väljanägemise [5].



Joonis 3.1. Flexis tehtud Windowsi-stiilis töölaud (autori koostatud).

[5] Vt `www.scalenine.com/themes/windowsclassic/WindowsClassic.html`

3.3. Prototüübi funktsionaalsed nõuded

SJCP õpivahendi funktsionaalsed nõuded tulenevad põhiliselt tarkvarast, mida kasutatakse eksamikeskustes. Teine osa funktsionaalsust on lisatud õppimist abistava eesmärgiga. Viimaste hulka kuuluvad näiteks testide ajutine peatamine, testide tegemine vastavalt raskusastmele ja kategooriale, samuti testide ülevaatamine ja selgituste-põhjenduste lugemine. Eeldefineeritud testid andmebaasis garanteerivad unikaalsuse (et sama küsimus ei satuks kahte järjestikusse testi korduvalt). JQPlusi programmis on lisatud veel küsimuste kohane märkmete tegemise võimalus ja küsimuste integreerimine Enthuware veebifoorimiga *JDiscuss.com*.

Üks levinud võte testide raskemaks tegemisel on see, et õigete küsimuste arvu ei ole ette antud (nagu päris ekamil). Seda viimast on praktiseerinud mitmetel Java sertifikaadieksamitel Kathy Sierra (Sierra and Bates 2003, 637). Viited spetsifikatsioonile on siinses prototüübis välja toodud iga üksiku küsimusega seoses. Õpivahendis on kaasa pandud veel eksami eesmärgid (*Exam Objectives*). Prototüübi funktsionaalsed nõuded on järgmised:

(1) Teste peab saama teha vastavalt raskusastmele.

(2) Teste peab saama teha vastavalt kategooriale.

(3) Andmebaasis peab saama moodustada eeldefineeritud teste.

(4) Pikemad koodinäited peavad olema küsimustest eraldatud.

(5) Küsimusi peab saama märkida.

(6) Testi tegemist peab saama panna seisma.

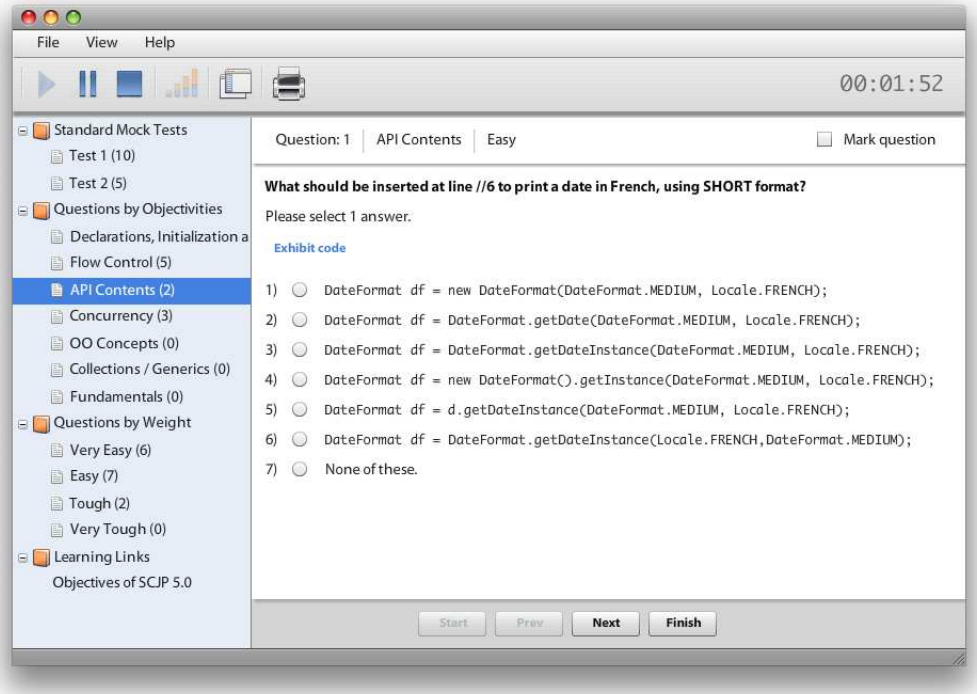
(7) Testi tulemusi peab saama printida.

(8) Küsimuste andmebaas peab olema kasutaja poolt muudetav.

3.3.1. Kasutajaliidese funktsionaalne kirjeldus

Prototüübi kasutajaliides koosneb neljast põhilisest paneelist: Vasakpoolne menüü annab ülevaate küsimuste andmebaasist eeldefineeritud testide, kü-

simuste kategooriate ja küsimuste raskusastmete kaupa. Ülemine tööriba sisaldab nuppe testide alustamise, seiskamise ja lõpetamisese kohta. Samuti tulemuste vaatamist ja printimist. Ülemine menüüriba näitab ka testile kulunud aega. Kui ajalimiit on ületatud, muutuvad numbrid punaseks, aga testi on endiselt võimlik sooritada. Alumine menüüriba kordab testide alustamise ja lõpetamise nuppe, aga lisaks nendele on allumisel menüüribal küsimuste vahel navigeerimise nupud.

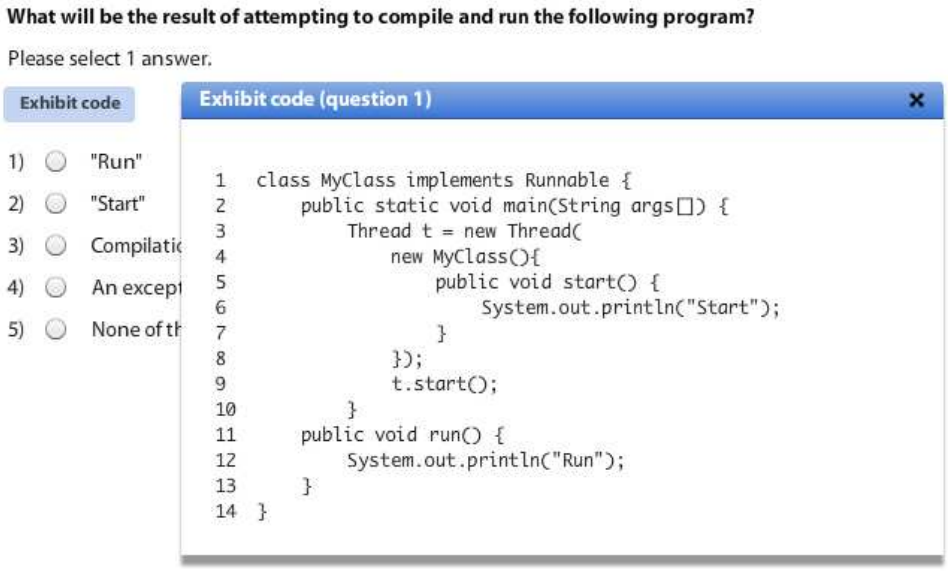


Joonis 3.2. SCJP õpivahendi prototüübi kasutajaliides.

Parempoolses paneelis toimub testi enda tegemine. Navegeerimisnuppe kasutades on võimalik küsimuste vahel liikuda. Küsimusi on võimalik hilisemaks ülevaatamiseks ära märkida, samuti näidatakse, mis kategooriasse antud küsimus kuulub ja millise raskusastmega see on. Küsimuste vahelist liikumist võimaldab samuti päris eksam. Viimane näitab ka koondnimekirja küsimustest, nii oon võimalik liikuda nt otse hilisemaks ülevaatamiseks märgitud küsimusele. Parempoolset paneeli on võimalik jooksvalt suurendada,

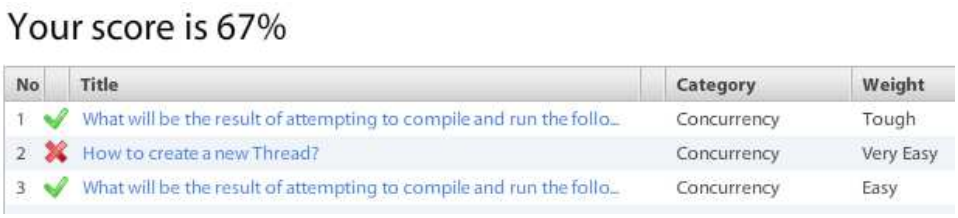
samuti saab (nt testi tegemise ajaks) vasakpoolse menüü täiesti ära peita.

Sarnaselt päris eksamile on lühikesed koodinäited esitatud kohe põhiaknas, ent pikemad programmikoodid tuleb eraldi avada vajutades Link-Buttonile *Exhibit code* nagu on näitatud joonisel 3.3.



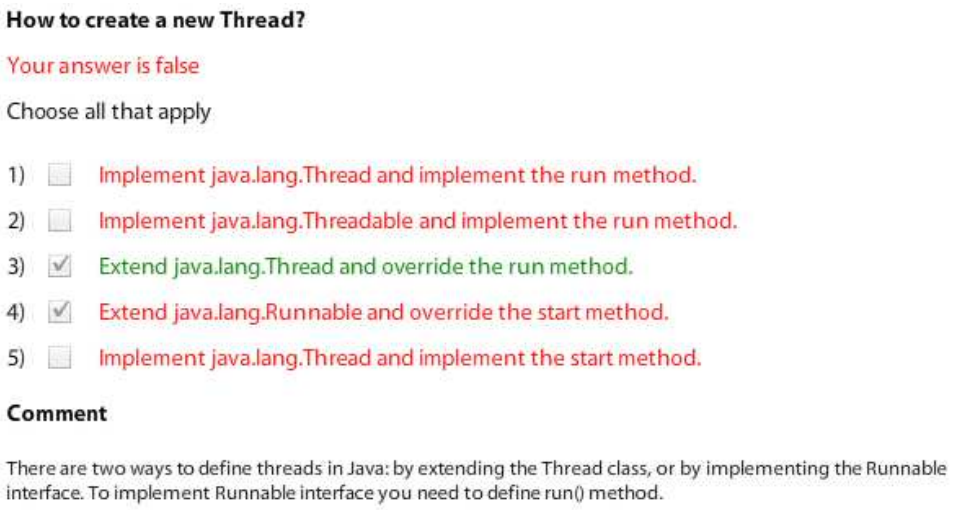
Joonis 3.3. Pikema programmikodiga küsimus avab lisaakna.

Kui test on ära lõpetatud, kontrollitakse küsimuste vastused süsteemi poolt ära ja näidatakse kasutajale kokkuvõtet. Samuti arvutatakse õigete vastuste protsent.



Joonis 3.4. Testi tulemuste paneel.

Küsimuse peale klikates on võimalik üksikasjaliselt näha, mis valitud küsimuse puhul õigesti/valesti vastati. Samuti näidatakse küsimuse üldiseid kommentaare. Kui on, siis näidatakse kommentaare ka üksiku küsimuse elemendi kohta. Joonisel 3.5 on kujutatud kontrollitud küsimus koos kommentaariga.



Joonis 3.5. Kontrollitud küsimus koos kommentariga.

Prototüüp on kättesaadav aadressil `http://sven.mindtheflex.com/scjp/`.

3.4. Kokkuvõte

Käesoleva peatüki eesmärk oli kirjeldada SCJP õpivahendi-prototüübi loomist kasutades Adobe Flexi arendusraamistikku. Lisaks nõuete spetsifitseerimisele ja prototüübi loomisele näidati, et Adobe Flex on sobilik arendusplatvorm programmeerimaks korduvkasutatavaid rakendusi-objekte, mida saab integreerida suurematesse õpikeskkondadesse.

Kokkuvõte

Käesoleva magistritöö käsitles Java programmeerimiskeele oskuste omandamist ja mõõtmist vastavalt Sun Microsystemsi koostatud Java kompetentsitasemetele. Magistritöö kitsam eesmärk oli Java programmeerijate esimese taseme sertifikaadieksami (SCJP) õpivahendi prototüübi koostamine, kasutades selleks uue põlvkonna veebiprogrammeerimise — nn „rikka Interneti-rakenduse" (RIA) võimalusi. Väitekirja avaramateks eesmärkideks oli uurida, kui sobilik on ühe konkreetse RIA raamistiku (Adobe Flexi) kasutamine õpiobjektide programmeerimisel, samuti avada tarkvarainseneride sertifitseerimise laiem ühiskondlik kontekst.

Tulenevalt magistritöö eesmärkidest püstitati järgmised ülesanded, millele töö käigus anti järgmised vastused:

(a) **Määratleda, milline on arvutialase tehnilise sertifitseerimise roll tarkvarainseneri elukutses.** Siin selgus, et sertifitseerimine on seotud tarkvarainseneri eetikakoodeksiga ning et sertifitseerimist võib vaadelda kui omamoodi „raudset sõrmust", mis traditsioonilistes insenerivaldkondades sümboliseerib ühelt poolt inseneriameti küpsust, sest see on teadlikult võtnud vastutuse ühiskonna ees, ja teiselt poolt sümboliseerib see konkreetse inseneri küpsust, kes on saavutanud professionaalse taseme. Me nägime, et sertifikaatide kasutusvaldkonnad on olulisel määral kokku langevad noorte inseneride motivatsioonifaktoritega. See tähendab, et noor tarkvarainsener võib loota, et sertifikaat aitab saavutada enamlevinud erialaseid eesmärke.

(b) **Analüüsida ühe konkreetse sertifikaadieksami (SCJP) nõudmiseid, õpivahendeid ja -metoodikaid.** Siin määratleti SCJP sooritustingimused ja näidati, kuhu see eksam kuulub Sun Microsystems-i Java sertifikaadieksamite õpiteekonna (*learning path*) kontekstis laiemalt. Analüüsiti omas klassis parimad SCJP õpivahendid, mis said eeskujuks käesoleva töö jooksul loodava õpivahendi-prototüübi nõudmiste kirjeldamisel. Java õpimetoodikatest käsitleti *Head First* raamatusarjas kasutatud matakognitiivset metoodikat. Seejärel näidati, et *Head First*-i õppimise printsiibid on edukalt kombineeritavad agiilses tarkvaraarenduses levinud paarisprogrammeerimisega ja koos sertifikaadiõppes rakendatavad.

(c) **Luua SCJP eksami jaoks õpivahendi prototüüp kasutades uue põlvkonna veebiprogrammeerimise (RIA) võimalusi.** Siin kirjeldati SCJP õpivahendi-prototüübi loomist kasutades Adobe Flexi arendusraamistikku. Prototüübi abil näidati ka, et Adobe Flex on sobilik arendusplatvorm loomaks korduvkasutatavaid õpiobjekte, mida saab integreerida suurematesse õpikeskkondadesse.

Väitekirja koostamise käigus tekkisid ka mitmed uurimisteemad, mis — arvestades käesoleva töö skoopi — pidid jääma käsitluse alt välja, näiteks RIA-rakenduste arhitektuuri ja disainimustrite küsimused, *Head First*-i metoodika metakognitiivse alusteooria edasine uurimine, samuti ei jõutud loodud õpivahendi-prototüübi tootestamiseni.

Abstract

Building an e-Learning RIA Environment for Java Programmer Certification (SCJP) Exam.

The more the society depends on information systems, the more increases the responsibility of software developers. In order to guarantee the quality of services provided by them, different kind of mechanisms have to be set up that would create a framework for testing their skills and at the same time providing the path for developing their skills further. Current thesis concentrates on the framework created by Sun Microsystems that was built for measuring and developing the software developers Java programming skills. The author demonstrates how the emerging RIA (rich Internet application) technology can be used in creating the tools for developers that would help them to succeed in this framework.

Purpose of this thesis is to build a Flash-based rich Internet application e-Learning environment for Java Programmer Certification (SCJP) Exam using Adobe Flex framework. The thesis has following goals:

(a) To find out how important role does the certification process play in a software developer's career,

(b) To analyze the requirements, learning tools and methods for the Sun Certified Java Programmer (SCJP) certificate exam,

(c) To create a prototype of a learning tool for SCJP by using the new generation RIA technology Flex.

Kirjandus

Dijkstra, E. (1972). The Humble Programmer. — *Communications of the ACM*, 15(10):859–866. http://www.jdl.ac.cn/turing/pdf/p859-dijkstra.pdf [01.05.09].

Fain, Y., Rasputnis, V., and Tartakovsky, A. (2007). *Rich Internet Applications with Adobe Flex and Java*. SYS-CON Media, Woodcliff Lake, NJ, first edition.

Flavell, J. H. (1979). Metacognition and cognitive monitoring: A new area of cognitive-developmental inquiry. — *American Psychologist*, 34(10):906–911.

Flavell, J. H. (1985). *Cognitive development*. Prentice-Hall, Englewood Cliffs, N.J., 2nd edition.

Ford, G. and Gibbs, E. N. (1996). *A Mature Profession of Software Engineering (Technical Report CMU/SEI-96-TR-004)*. Carnegie Mellon University, Software Engineering Institute, Pittsburgh, Pennsylvania.

Gordon, B. (2008). *Eetika ja kultuuridevahelised seosed: lühendatud loengukonspekt*. Tallinna Tehnikaülikool, Automaatikainstituut, Tallinn.

Gosling, J., Joy, B., and Steele, G. L. (2005). *The Java language specification*. Addison-Wesley, third edition. http://java.sun.com/docs/books/jls/ [01.05.09].

Green, M. (2004). *Deep into the Basics: Tackling Sun's SCJP 1.4 Exam — CertCities.com*. http://certcities.com/editorial/exams/story.asp?EditorialsID=86 [01.05.09].

Kalwarski, T., Mosher, D., Paskin, J., and Rosato, D. (2006). *Best Jobs in America. — MONEY Magazine. CNN*. http://money.cnn.com/magazines/moneymag/bestjobs/2006/ [01.05.09].

Kennedy, K. and Vardi, M. Y. (2002). A Rice University perspective on software engineering licensing. — *Communications of the ACM*, 45(11).

Le Goff, J. (2001). *Keskaja Euroopa kultuur*. Kupar, Tallinn.

McBreen, P. (2002). *Software Craftsmanship: The New Imperative*. Addison-Wesley, Boston.

McConnell, S. (1999). *After the gold rush: creating a true profession of software engineering*. Microsoft Press, Redmond, Wash.

Mughal, K. A. and Rasmussen, R. (1999). *A programmer's guide to Java certification: a comprehensive primer*. Addison-Wesley.

Mughal, K. A., Rasmussen, R., and Mughal, K. A. (2008). *A programmer's guide to Java SCJP certification: a comprehensive primer*. Addison-Wesley, Upper Saddle River, NJ, 3rd edition.

Naur, P. and Randell, B., editors (1969). *Software engineering: Report on a conference sponsored by the NATO Science Commitee (Garmisch, Germany, 7th to 11th October 1968)*. Scientific Affairs Division, NATO, Brussels. `http://www.europrog.ru/book/nato1968e.pdf` [01.05.09].

Paavel, S.-O. (2006). *Agiilse metoodika kasutuselevõtt tarkvaraprojektis: Arvestustöö aines „Projektijuhtimine" (MII7007)*. Tallinna Ülikool, Tallinn.

Paavel, S.-O. (2008). Uue põlvkonna veebirakenduste arendusvahend. — *Arvutimaailm*, (5(150)):66–67.

Sanchez, J. and McIntosh, A. (2009). *Creating visual experiences with Flex 3.0*. Addison-Wesley.

Sarv, H. (1999a). Arvutialasest tehnilisest sertifitseerimisest Eestis. — *Arvutimaailm*, (7):44–47.

Sarv, H. (1999b). Arvutialasest tehnilisest sertifitseerimisest Eestis 2. — *Arvutimaailm*, (8):43–45.

Schank, R. C. (1995). *Tell me a story: narrative and intelligence*. Northwestern University Press, Evanston, Illinois.

Schreckmann, D. (2004). *Meet the Authors: Kathy Sierra and Bert Bates — JavaRanch Journal*. `http://www.javaranch.com/journal/200406/MeetTheAuthors-KathySierraAndBertBates.html` [01.05.09].

SECE (1999). *Software Engineering Code of Ethics and Professional Practice (Version 5.2)*. Association for Computing Machinery. `http://www.acm.org/about/se-code/` [01.05.09].

Shaw, M. (1990). Prospects for an engineering discipline of software. — *IEEE Software*, pages 15–24.

Shaw, M. (2002). What makes good research in software engineering? — *International Journal of Software Tools for Technology Transfer*, 4(1):1–7.

Sierra, K. and Bates, B. (2003). *Head first EJB*. O'Reilly.

Sierra, K. and Bates, B. (2008). *SCJP Sun certified programmer for Java 6 study guide: exam (310-065)*. McGraw-Hill, New York.

Skibiski, K. C. (2008). *Motivating Factors of Young Engineers*. National Society of Professional Engineers, Alexandria, Virginia.

SUN (2009). *Java Certification*. Sun Microsystems. `http://www.sun.com/training/certification/java/index.xml` [01.05.09].

SWEBOK (2004). *IEEE's Guide to the Software Engineering Body of Knowledge — 2004 Version*. Institute of Electrical and Electronics Engineers. `http://www.swebok.org` [01.05.09].

Joonised