Koodi*

Antti-Juhani Kaijanaho†

11. marraskuuta 2003

Tiivistelmä

Tämän luennon tarkoitus on käsitellä ohjelmistoprojektin toteutusvaiheeseen ja projektin tuloksena syntyvään ohjelmakoodiin liittyviä kysymyksiä. Luennolla tarkastellaan pääasiassa ohjelmointikielten valintaa ja vapaaohjelmia.

> *How many times do I have to tell you?*
> *The right tool for the right job!*
> *— Scotty elokuvassa Star Trek V*

Sisältö

---

* Luento IT130 Johdatus ohjelmistotekniikkaan -kurssilla Jyväskylän yliopiston tietotekniikan laitoksella 11.11.2003

† Jyväskylän yliopisto, tietotekniikan laitos, ohjelmistotekniikka

1 Ohjelmointikielistä

Tämä osa luennosta perustuu pitkälti syksyn 2002 Ohjelmointikielten periaatteet -kurssin (TIE 356) materiaaliin. Jonkin verran olen käyttänyt myös tämän syksyn Funktio-ohjelmoinnin kurssin (TIE 328) materiaalia hyväkseni.

Ohjelmointi on ongelmien ratkaisemista. Asiaa voi katsoa kahdelta kannalta: voi keskittyä ymmärtämään

- *ongelmaa* tai
- *ratkaisumalleja*.

Voidaan ajatella, että maailma jakautuu kahteen osaan,

- *ongelma-avaruuteen* ja
- *ratkaisuavaruuteen*.

Ongelma-avaruus koostuu kaikista mahdollisista ongelmista, ja ratkaisuavaruus koostuu kaikista mahdollisista ratkaisuista. Ohjelmoijan tehtävänä on siirtää ongelma (joka elää ongelma-avaruudessa) ratkaisuavaruuteen, jolloin siitä tulee ratkaisu.

Ohjelmointikieli on ohjelmoijan pääasiallisin työkalu, ja kielen toteutus (kääntäjä tai tulkki) auttaa ohjelmoijaa ratkaisun luomisessa: kielen ominaisuudet ratkaisevat, kuinka syvälle ratkaisuavaruuteen ohjelmoijan tulee ongelmaansa viedä suunnitteluvaiheessa ja mistä lähtien kielen toteutus tekee sen ohjelmoijan puolesta.

1.1 Kielten jaotteluja

Ohjelmointikieliä on mielettömän paljon. Jonkinlaista järjestystä tähän suureen joukkoon saadaan erilaisilla jaotteluilla.

1.1.1 Sukupolvet

Eräs tunnettu mutta nykyisin vähämerkityksinen jaottelu jakaa kielet eri sukupolviin:

Ensimmäinen sukupolvi koostuu konekielistä. Konekielellä tarkoitan tässä nyt ohjelman sitä muotoa, jonka kone pystyy sellaisemaan lukemaan. Seuraavassa on Hello World -ohjelma erään tietokoneen konekiellä (esitettynä 16-lukujärjestelmässä tavujonona):

```
980901013d6e18579802010098060036f686a656c6d612e6d6d7300
98070002f4ff000000000701000000009804000398070007486 56c6c
980700076f2c2057980700076f726c6498070007210a0000980a00ff
000000000000100980b0000203a4050104040204d20612069026e01
00812053207410100272010c82000000980c0008
```

Käytännössä konekieliset ohjelmat ensin suunniteltiin kirjoittamalla ohjelmat ensin paperilla symbolisesti, käyttämällä konekäskyistä muistikasnimiä. Myös vuokaavioita saatettiin harrastaa monimutkaisten ohjelmien kirjoittamisessa. Konetta varten ohjelma käännettiin käsin konekielelle.

Ensimmäinen sukupolvi oli vallalla 1940-luvun lopulta 1950-luvun alkupuolelle.

Toinen sukupolvi syntyi, kun keksittiin, että voidaan kirjoittaa ohjelma suorittamaan yllämainittu käännöstyö muistikkaista konekielelle. Toinen sukupolvi koostuu siis symbolisista konekielistä. Edellä esitetty Hello World -ohjelma käännettiin seuraavasta symbolisella konekielellä kirjoitetusta ohjelmasta:

```
argv   IS    $1
       LOC   #100
Main   LDOU  $255,argv,0
```

```
        TRAP  0,Fputs,StdOut
        GETA  $255,String
        TRAP  0,Fputs,StdOut
        TRAP  0,Halt,0
String BYTE  ", world",#a,0
```

Toinen sukupolvi oli vallalla 1950-luvun puoleenväliin asti.

Kolmas sukupolvi syntyi vuonna 1955, kun FORTRAN kehitettiin. Lähes kaikki nykyisin ohjelmointikieliksi kutsumamme kielet ovat kolmannen sukupolven kieliä.

Neljäs ja viides sukupolvi esiintyvät puheissa toisinaan, mutta niiden sisältö riippuu täysin puhujasta. Tavallisimmin neljännen tai viidennen sukupolven kielellä tarkoitetaan joko sovelluskehittimiä (Delphi ym.) tai ohjelmoitavia sovellusohjelmia (Excel ym.).

1.1.2 Paradigmat

Edward Sapir ja Benjamin Whorf esittivät viime vuosisadan alkupuolella kielitieteellisen hypoteesin, joka tunnetaan nykyisin nimellä Sapirin–Whorfin hypoteesi. Sen heikompi muoto, johon nykyisetkin kielitieteilijät enemmän tai vähemmän uskovat, kuuluu: *Kieli vaikuttaa ajatteluun.* Myös ohjelmoinnin yhteydessä tämä heikompi muoto on totta: *se, mitä ohjelmointikieltä osaa, vaikuttaa siihen, miten ohjelmointiongelmia ryhtyy ratkaisemaan*.

Kaikki kielet eivät ole tässä suhteessa samanarvoisia: esimerkiksi Pascalin ja C:n tukema ajattelutapa on niin samanlainen, että kummastakin kielestä voi varsin vähällä vaivalla siirtyä toiseen. Vastaavanlainen samankaltaisuus pätee, tosin vähemmässä määrin, myös C++:aan ja Javaan. Jotkin kielet puolestaan suosivat niin erilaisia ajattelutapoja, että ne tuntuvat aluksi aivan eri maailmoilta, eikä toisen osaamisesta ole kovin suurta apua toisen osaamisessa. Ajatustavoiltaan läheisten kielten katsotaan yleensä kuuluvan samaan *kieliparadigmaan*, mutta niiden väliset rajapyykit eivät ole useinkaan kovin selvät.

Kieliparadigmoille sukua ovat *ohjelmointiparadigmat*, tosin aika usein näiden välistä eroa ei nähdä (myös tämän kirjoittaja on ne monesti sekoittanut!). Kukin ohjelmointiparadigma on oikeastaan idealisoitu versio niistä ajattelutavoista, joita vastaava kieliparadigma suosii. Alan Perlis [5] kirjoitti kaksi vuosikymmentä sitten: *A language that doesn't affect the way you think about programming, is not worth knowing.* Minä kääntäisin tämän väitteen toisin päin: *A language that affects the way you think about programming is worth knowing.* Itse asiassa väittäisin, että jokaisen ohjelmointiin vakavissaan suhtautuvan tulisi tuntea ainakin pari-kolme eri kieliparadigmoihin kuuluvaa kieltä – eli oikeastaan pari-kolme eri ohjelmointiparadigmaa.

Ohjelmointiparadigmoiksi lasketaan tavallisimmin ainakin seuraavat (huomaa, että nämä paradigmat eivät ole toisistaan erillisiä, vaan ne menevät osittain päällekkäin):

Imperatiivisen ohjelmoinnin (käskyohjelmoinnin) juuret ovat syvällä ratkaisuavaruudessa, von Neumannin arkkitehtuurin perusrakenteissa. Imperatiivinen ohjelmoija näkee maailman joukkona muistipaikkoja, joita muutellaan yksitellen "von Neumannin pullonkaulaa" pitkin. Imperatiiviselle ohjelmoinnille tyypillistä on tiedon muuttaminen *in place* (sitä kopioimatta), silmukoiden runsas käyttö, ohjelmakoodin korostuminen sekä käskymetafora.

Deklaratiivisen ohjelmoinnin (kuvausohjelmoinnin) juuret ovat ongelma-avaruudessa, ongelmien täsmällisessä kuvaamisessa yleensä matematiikan keinoin. Sen taustalla on yleensä vahva matemaattinen teoria, joka on sovitettu ohjelmoinnin tarpeisiin. Deklaratiiviselle ohjelmoinnille tyypillistä on tiedon muuttumattomuus (muutos tehdään tekemällä uusi, erilainen kopio), rekursion runsas käyttö, tietorakenteiden korostuminen sekä laskentametafora.

Proseduraalisen ohjelmoinnin (toiminto-ohjelmoinnin) perusabstraktio on aliohjelma. Proseduraalinen ohjelmoija keskittyy etsimään koodistaan tai suunnitelmastaan toimintoja, jotka toistuvat, ja tekee niistä aliohjelmia. Proseduraalinen ohjelmointi esiintyy yleensä imperatiivisen ohjelmoinnin kanssa yhdessä.

Olio-ohjelmointi näkee kaiken olioina ja niiden välisenä kommunikaationa. Olio-ohjelmoija keskittyy hahmottamaan olioita ja niiden välisiä suhteita. Myös olio-ohjelmointi on yleensä imperatiivista ohjelmointia, vaikka deklaratiivinen olio-ohjelmointi ei olekaan käsitteenä mahdoton ajatus.

Funktio-ohjelmointi (funktionaalinen ohjelmointi) on eräässä mielessä proseduraalisen ohjelmoinnin yleistys. Funktio-ohjelmoija näkee ohjelman (matemaattisena) funktiona. Funktio-ohjelmalle tyypillisiä ilmaisukeinoja ovat funktionaalit (funktiot, jotka ottavat toisia funktioita parametrikseen). Funktio-ohjelmointi lasketaan perinteisesti deklaratiiviseksi ohjelmoinniksi, vaikka oikeastaan deklaratiivinen funktio-ohjelmointi onkin harvinaisempaa kuin imperatiivinen funktio-ohjelmointi, jossa funktiot ovat oikeastaan vain yleistettyjä aliohjelmia.

Logiikkaohjelmointi perustuu ajatukseen, että looginen päättely voidaan mekanisoida (joka pitää paikkansa vain rajoitetusti). Logiikkaohjelmoija näkee ohjelman loogisena väitteenä, joka pitää todistaa. Logiikkaohjelma koostuu kyseisen väitteen muotoilusta sekä niiden oletusten muotoilusta, joita päättelyssä voidaan käyttää. Puhtaimmillaan logiikkaohjelmointi on täysin deklaratiivista, mutta käytännössä logiikkaohjelmointia harrastetaan myös imperatiivisesti, jossa päättelysäännöt vastaavat aliohjelmia.

Olio-ohjelmoinnista ja funktio-ohjelmoinnista on kummastakin tiedekunnassamme laudaturkurssi (TJTL33 ja TIE328).

Ohjelmointikielet jaotellaan usein myös yllämainitttuihin paradigmalokeroihin. Tämä on kuitenkin varsin ongelmallista. Jokaisella ohjelmointikielellä on ”omin” paradigmansa, jota se tukee parhaiten, mutta hyvin monet kielet ovat aidosti moniparadigmakieliä siinä mielessä, että niitä voidaan käyttää mainiosti useammassakin ohjelmointiparadigmassa. Seuraavassa olen luetellut tyypillisiä ohjelmointikieliparadigmojen tunnusmerkkejä ja tyyppiesimerkkejä; kannattaa kuitenkin muistaa, että minkään paradigman tyyppiesimerkit eivät rajoitu pelkästään sen paradigman sisälle.

Imperatiivisen kielen (käskykielen) merkittävimmät tunnusmerkit ovat *sijoituslause* (tai -lauseke) ja *peräkkäistäminen*. Useimmissa kielissä peräkkäistäminen on implisiittistä: peräkkäin kirjoitetut lauseet suoritetaan peräkkäin. Niissäkin kielissä yleensä puolipiste on peräkkäistyksen merkkinä.

Lähes kaikki kielet ovat imperatiivisia. Tyyppiesimerkkejä ovat Algol-suvun kielet (Pascal, C, C++, Java).

Deklaratiivisen kielen (kuvailukielen) tunnusmerkki on se, että siitä puuttuu sijoituslause ja peräkkäistäminen. Deklaratiivisen kielen pääetu on, että siinä pätee viittausläpinäkyvyys (*referential transparency*), mikä helpottaa huomattavasti ohjelmien formaalia pyörittelyä käsin tai koneellisesti.

Aidosti deklaratiiviset kielet ovat harvinaisia, ja niihin pätee Alan Perlisin [5] lausahdus: *Purely applicative languages are poorly applicable.* (Perlis käytti deklaratiivisista kielistä nimeä applikatiivinen kieli.) Esimerkkejä lienevät λ-laskenta ja John Backusin FP-kieli [1].

Viime vuosikymmenellä kehitettiin muutamia tekniikoita, joilla kielestä saatettiin tehdä eräässä mielessä sekä deklaratiivinen että imperatiivinen. Näitä tekniikoita käyttäviä kieliä ovat Clean ja Haskell.

Proseduraaliset kielet (toimintokielen) tukevat aliohjelmarakennetta. Tyyppiesimerkkejä ovat C ja Pascal. Myös C++ ja Java kuuluvat tähän luokkaan, vaikka ne ovatkin ensisijai-

sesti oliokieliä.

Oliokielet tukevat olioita, joilla on identiteetti ja toiminta (metodit) sekä toimintojen perimisen oliolta toiselle (tai luokkapohjaisissa kielissä luokalta toiselle). Tyyppiesimerkkejä ovat Smalltalk, Python, Java ja C++.

Funktiokielten (funktionaalisten kielten) tärkein tunnusmerkki on tuki täysivaltaisille funktioille (funktioille, joita voi palauttaa funktiosta, antaa funktiolle parametriksi ja tallettaa tietorakenteisiin). Muita tyypillisiä mutta ei universaaleja ominaisuuksia ovat tuki halvalle tiedon kopioinnille, algebralliset tietotyypit ja niiden hahmonsovitus ja tehokas rekursio. Tyyppiesimerkkejä ovat Lisp, SML, Haskell ja Clean. Myös useimmat oliokielet voidaan sijoittaa tähän kategoriaan, sillä olioilla ja täysivaltaisilla funktioilla on paljon yhteistä ainakin pellin alla.

1.1.3 Ousterhoutin jaottelu

John Ousterhout [4] määrittelee järjestelmäohjelmointikielen ja juontokielen eron seuraavasti:

järjestelmäohjelmointikieli Nämä kielet on tarkoitettu tietorakenteiden ja yksittäisten komponenttien toteuttamiseen siirrettävästi ja korkeammalla tasolla kuin symbolisella konekielellä kirjoitettaessa. Ne käyttävät vahvaa staattista tyypitystä. Järjestelmäohjelmointikieliä ovat esimerkiksi PL/I, Pascal, C, C++ ja Java.

juontokieli Juontokielet olettavat, että on jo olemassa joukko hyödyllisiä komponentteja, jotka on kirjoitettu järjestelmäohjelmointikielillä. Juontokieliä käytetään komponenttien yhdistelemiseen, eräänlaisina liimoina (usein puhutaankin englanniksi *glue languageistä*). Juontokielet käyttävät tyypillisesti dynaamista tyypitystä - tyypityksen vahvuus vaihtelee. (Ousterhout puhui tyypittömyydestä mutta hän tarkoitti dynaamista tyypitystä.) Juontokieliä ovat esimerkiksi Perl, Python, Rexx, Tcl, Visual Basic ja Unixin kuoret.

Ousterhoutin jaottelua on kovasti kritisoitu.

1.2 Kielen valinta

Tyypilliset tavat valita toteutuskieli ovat

- *me ollaan aina käytetty tätä kieltä*
- *tuo kieli on kaikkien huulilla; käyttäkäämme sitä*

Kumpaakin tapaa voidaan kenties perustella liiketoiminnallisilla syillä, mutta teknisesti molemmat tavat ovat huonoja. Varsinkin ison projektin suunnittelijoiden kannattaa pohtia kielen valintaa tosissaan, sillä sillä voi olla iso vaikutus projektin tuottavuuteen.

Kannattaa huomata, kielen valintaa ei voi tarkastella irrallaan kielen tukityökalujen (kääntäjä, tulkki, debuggeri yms.) tarkastelusta, mutta niitä ei silti pidä sekoittaa toisiinsa. Useimmissa ”kielisodissa”, mitä esimerkiksi netissä käydään, kielet ja niiden tukityökalut sekoitetaan toisiinsa niin autuaasti, että vertailun mielekkyys katoaa kokonaan.

Lawlis [3] on esittänyt prosessin, jolla toteutuskielen voi valita (näennäisen?) täsmällisesti vaatimusten mukaan. Burgess [2] on esittänyt kriteereitä, jota hänen mukaansa on syytä tarkastella valittaessa toteutuskieltä korkealaatuisen tuotteen tekemistä varten. Raymond [7] on käsitellyt kielen valintaa Unix-ympäristöön ohjelmaa rakentavan vapaaohjelmaprojektin näkökulmasta, mutta hänen havaintonsa ovat yleisemminkin sovellettavissa.

Seuraavassa olen koonnut Lawlisin, Burgessin ja Raymondin pohjalta muutaman kysymyksen, joita voidaan käyttää kielten arvioinnissa (ei missään erityisessä järjestyksessä):

1. Tukeeko kieli hyvää ohjelmointityyliä ja tekeekö se huonon tyylin hankalaksi?
2. Onko kielestä olemassa standardi? Miten tarkasti ja selkeästi se määrittelee kielen? Tukevatko kielen toteutukset standardia ja tekevätkö ne epästandardin koodin kirjoittamisen hankalaksi?
3. Onko kielestä olemassa laadukkaita toteutuksia?
4. Onko kieli riippumaton yksittäisistä käyttöjärjestelmistä ja laitteistoista? Toisin sanoen, onko se siirrettävä myös käytännössä?
5. Tukeeko kieli sovellusalueita, joilla projekti tai organisaatio toimii?
6. Auttaako vai haittaako se turvallisuuden (safety) ja luotettavuuden tavoittelua?
7. Ovatko sen toteutukset ajanmukaisia?
8. Onko sille olemassa tarvittavat prosessia tukevat työkalut?

2 Vapaaohjelmat

Vapaaohjelmat (engl. *free software*, *open source software*) ovat tämän hetken trendi, mutta ne eivät ole mikään uusi keksintö. Vapaaohjelmien käytöllä ja kehityksellä on joitakin etuja, jotka myös perinteisen ohjelmistokehityksen kannattaa huomioida.

2.1 Määritelmä

Vapaaohjelma on ohjelma,

1. jota kuka tahansa saa käyttää mihin tahansa tarkoitukseen,
2. jota kuka tahansa saa tutkia ja muuttaa,
3. jota kuka tahansa saa levittää eteenpäin maksua vastaan tai maksutta (levittäjän, ei kehittäjän valinta!) ja
4. jota kuka tahansa saa parannella (ja julkistaa näin tehdyt muutoksensa).

Ohjelma, joka ei ole vapaaohjelma, on omisteinen ohjelma (*proprietary software*).

2.2 Mitä hyötyä…

2.2.1 …yksityiselle kuluttajalle?

Kuluttaja saattaa saada vapaaohjelman ilmaiseksi netistä (laillisesti!) tai ostamalla sen kaupasta. Saamistavasta riippumatta kuluttajalla on oikeus kopioida vapaaohjelmansa kavereilleen ja naapureilleen. Jos kuluttaja sattuu olemaan ohjelmoija, hän voi myös parannella ohjelmaa ja korjailla sitä, ja joka tapauksessa hän voi antaa parantelut ja korjaukset jonkun muun tehtäväksi. Kuluttajan ei tarvitse odottaa, että ohjelman kehittänyt firma sattuisi armollisesti saamaan aikaan korjauksen bugiin tai puutteeseen.

2.2.2 …asiakasorganisaatiolle?

Asiakasorganisaation kannalta vapaaohjelma on samankaltainen kuin ohjelma, jonka tekijänoikeudet se saa itselleen — tärkeimpinä eroina se, että toimittajaorganisaatio voi silti tarjota samaa ohjelmaa muillekin, ja se, että asiakasorganisaatio ei välttämättä voi tehdä ohjelmasta omaa omisteista versiotaan (tämä tosin riippuu lisenssistä). Asiakasorganisaatiolle merkittävä hyöty on siinä, että se ei ole sidottu alkuperäiseen toimittajaan vaan voi kilpailuttaa ohjelman edelleenkehityksen tai vaikkapa tehdä sen sisäisenä työnä. Alkuperäisen toimittajan luotettavuus (ylläpitovaiheessa) ja vakavaraisuus ei ole tällöin yhtä merkittävä valintakriteeri kuin tavallisesti.

2.2.3 ... toimittajaorganisaatiolle?

Toimittajaorganisaatiolle vapaaohjelma on parempi diili kuin ohjelma, jonka tekijänoikeudet se luovuttaa asiakkaalle. Toisaalta se on huonompi diili kuin omisteinen ohjelma. Toimittajaorganisaatio voi tarjota samaa ohjelmaa myös muille asiakkaille ja näin saada enemmän fyrkkaa samasta ohjelmasta. Toisaalta joku asiakkaista tai joku kolmas osapuoli, joka on saanut ohjelman joltakulta asiakkaalta tai muin keinoin, saattaa ruveta kilpailemaan toimittajan kanssa. Toisaalta se, että asiakasorganisaatio ei ole sidottu toimittajaan, antaa mahdollisuuden saada töitä myös sellaisille toimittajille, joiden vakavaraisuus on kyseenalainen. Vapaaohjelman tekijälla on myös mahdollisuus käyttää hyväkseen kaikkia jo olemassa olevia vapaaohjelmia, joita on melko runsaasti, ilman, että jokaisesta pitäisi erikseen neuvotella sen toimittajan kanssa.

2.2.4 ... yhteiskunnalle?

Tekijänoikeus on luonteeltaan valtion erityisistä syistä myöntämä rajoitettu monopoli tekijänoikeuden kohteeseen. Omisteisen ohjelman toimittajalla on aina monopoli tuotteeseensa. Yhteiskunnan kannalta on aina parempi, jos toimittaja kykenee pärjäämään ilman monopolia.

Vapaaohjelma edistää myös kilpailua, kuten edellä kävi ilmi.

Julkistetut vapaaohjelmat ovat kaikkien tutkittavissa. Tämä on hyödyllistä, sillä hyväksi ohjelmoijaksi tullaan pitkälti lukemalla olemassaolevia ohjelmia ja kirjoittamalla uusia. Julkiset vapaaohjelmat auttavat näin yhteiskuntaa kouluttamaan ohjelmoijia.

2.3 Kuinka?

Kansainvälisten sopimusten mukaisesti (lähes) jokainen ohjelma on automaattisesti tekijänoikeuden suojaama. Tekijänoikeuslaki kieltää kopioinnin ja muuttelun pääsääntöisesti ilman tekijänoikeuden haltijan lupaa. Jotta ohjelmasta olisi vapaaohjelmaksi, tulee sen tekijänoikeuden haltijan (tai haltijoiden yhdessä, jos heitä on monta) antaa sopivanlainen yleinen lupa eli lisenssi.

Seuraavassa luettelen ja kuvailen joitakin suosittuja vapaaohjelmalisenssejä. (Kehoitan jokaista lukemaan kaikkien käyttämiensä ohjelmien lisenssit ja ne lisenssit, joita aikoo omissa ohjelmissaan käyttää.)

Ensin kuitenkin vähän lisenssiteoriaa. Vapaaohjelmien lisenssit voidaan jakaa kahteen kategoriaan:

Transitiiviset lisenssit (Englanniksi yleisimmin copyleft licenses.) Lisenssi on transitiivinen, jos se sallii ohjelmaan tehtävät muutokset tai ohjelman käyttämisen osana toista ohjelmaa vain sillä ehdolla, että kyseistä lisenssiä sovelletaan myös näiden tekojen tuloksena syntyviin ohjelmiin.

Intransitiiviset lisenssit (Englanniksi yleisimmin non-copyleft licenses.) Lisenssi on intransitiivinen, jos se sallii ohjelmaan tehtävät muutokset ja ohjelman käyttämisen osana toista ohjelmaa rajoittamatta näin syntyvien ohjelmien lisensointia.

2.3.1 GNU General Public License (GPL) version 2

Eräs yleisimmistä vapaaohjelmalisensseistä on GNU-projektin General Public License (on myös epävirallinen suomennos). GPL on transitiivinen lisenssi, joten kaikki ohjelmat, jotka

on kerran asetettu GPL:n alaisuuteen, säilyvät myös aina GPL:n alaisina (tietyin harvinaisin poikkeuksin).

GPL:llä on seuraavat tärkeät ominaisuudet:

- GPL:n alaista ohjelmaa saa käyttää täysin vapaasti mihin tahansa tarkoitukseen. Käytöstä ei tarvitse ilmoittaa kenellekään.
- Jos olet saanut käsiisi binääriversion GPL:n alaisesta ohjelmasta, on sinulla oikeus saada sitä vastaava lähdekoodi sellaisessa kunnossa, että vastaavan binäärin voi niistä kääntää. Jos et ole saanut sitä binäärin mukana (eikä sinulla ollut mahdollisuutta hakea niitä samasta verkkopaikasta, josta hait binäärin), sinun olisi pitänyt saada binäärin sinulle luovuttaneelta vähintään kolme vuotta voimassa oleva kirjallinen lupaus toimittaa lähdekoodi kenelle tahansa toimituskuluja vastaan. Jos kuitenkin sait binäärin epäkaupallisesti, on sen sinulle luovuttaneella ollut oikeus antaa sinulle eteenpäin se lupaus, jonka tämä oli itse saanut.
- GPL:n alaista ohjelmaa saa muuttaa täysin vapaasti käyttäjän omaan käyttöön. Mitään velvollisuutta ei ole ilmoittaa muutoksista kenellekään, saati lähettää niitä kenellekään. Muutoksista tulee kuitenkin mainita kussakin muutetussa lähdekooditiedostossa. Jos muutettua versiota jakelee muille, tulee tämän tapahtua GPL:n ehtojen mukaisesti.
- GPL:n alaista ohjelmaa saa levittää (muutettuna tai muuttamattomana) lähdekoodin kanssa vapaasti (lähdekoodin tarjoaminen samassa verkkopaikassa binäärien kanssa riittää). Mitään velvollisuutta levittää ohjelmaa ei ole.
- GPL:n alaista ohjelmaa saa levittää (muutettuna tai muuttamattomana) binäärimuodossa ilman lähdekoodia vapaasti sillä ehdolla, että binäärien mukaan laittaa kirjallisen, vähintään kolme vuotta voimassa olevan lupauksen toimittaa kyseisiä binäärejä kenelle tahansa korkeintaan veloittaen fyysisen median valmistus- ja toimituskulujen verran. Jos ohjelmaa levittää binäärimuodossa epäkaupallisesti ja on sen itse saanut edellämainitun lupauksen kanssa, saa kyseisen luvan antaa binäärien mukana eteenpäin.
- GPL:n alaisen ohjelman (tai sen osan) ottaminen toisen ohjelman osaksi (myös linkittämällä) lasketaan ohjelman muuttamiseksi. Tällaisen kokonaisuuden levittäminen on sallittua vain, jos kokonaisuus on GPL:n alainen. Pelkiltä ohjelmistokokoelmilta (esimerkiksi useita ohjelmia sisältävät CD:t) ei GPL:isyyttä kuitenkaan vaadita.
- Jonkin muun ohjelman (tai sellaisen osan) ottaminen GPL:n alaisen ohjelman osaksi (myös linkittämällä) lasketaan GPL:n alaisen ohjelman muuttamiseksi. Tällaisen kokonaisuuden on sallittua vain, jos osaksi otettu ohjelma on GPL:n alainen (tai jonkin sellaisen lisenssin alainen, jonka ehdot ovat yhteensopivat GPL:n kanssa).
- Jos harkitset ohjelman laittamista GPL:n alle ja ohjelmaa koskee yksi tai useampi patentti, lue lisenssi tarkasti läpi (erityisesti kohta 7).

Ohjelman voi asettaa GPL:n alle vain tekijänoikeuden omistaja (lähtökohtaisesti ohjelman tekijä tai joissakin tapauksissa hänen työnantajansa). Tämä tapahtuu laittamalla ohjelman jokaiseen lähdekooditiedostoon seuraava ilmoitus (muutettavat muutettuina):

> Copyright © *vuosi tekijänoikeuden omistajan nimi*
> This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

> You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Jos ohjelma on interaktiivinen, on hyvä muuttaa se tulostamaan seuraavan kaltainen huomautus käynnistyessään:

```
Gnomovision version 69, Copyright (C) <vuosi>  <tekijänoikeuden omistajan nim
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’.
This is free software, and you are welcome to redistribute it
under certain conditions; type ‘show c’ for details.
```

Myöskin on syytä liittää ohjelman mukaan kappale GPL:n tekstistä.

Huomaa, että GPL:n nimi ei ole GNU Public License.

2.3.2 GNU Lesser General Public License (LGPL) version 2.1

Toinen GNU:n vapaaohjelmalisenssi on GNU Lesser General Public License. LGPL on rajoitetusti transitiivinen lisenssi: se käyttäytyy GPL:n kanssa samalla tavalla lukuunottamatta linkittämistä.

LGPL:n alaisen ohjelman (esimerkiksi kirjaston) saa linkittää osaksi toista ohjelmaa riippumatta kyseisen toisen ohjelman lisenssistä. Tällaista binääriä levitettäessä ilman toisen ohjelman lähdekoodia on binäärin mukaan laitettava sellainen materiaali, jonka avulla käyttäjä voi linkittää ohjelmaan eri version LGPL:n alaisesta ohjelmasta (jos kyseinen eri versio säilyttää ABI:n).

Muilta osin LGPL on GPL:n kaltainen.

Ohjelman voi asettaa LGPL:n alle vain tekijänoikeuden omistaja (lähtökohtaisesti ohjelman tekijä tai joissain tapauksissa hänen työnantajansa). Tämä tapahtuu laittamalla ohjelman jokaiseen lähdekooditiedostoon seuraava ilmoitus (muutettavat muutettuina):

> Copyright ©*vuosi tekijänoikeuden omistajan nimi*
> This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.
> This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.
> You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Myöskin on syytä liittää ohjelman mukaan kappale LGPL:n tekstistä.

2.3.3 MIT-lisenssi

Seuraava lisenssi on yleistetty muoto MIT:n X11-lisenssistä. Se on intransitiivinen lisenssi, joka sallii melkein minkä vain. Ainoat rajoitukset liittyvät siihen, että lisenssiteksti tulee säilyttää myös muutoksia tehtäessä tai ohjelman osia kopioitaessa toisiin ohjelmiin. Se ei estä ohjelman omisteistamista.

Copyright © *vuosi tekijänoikeuden omistajan nimi*
Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:
The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL *TEKIJÄNOIKEUDEN OMISTAJAN NIMI* BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
Except as contained in this notice, the name of *tekijänoikeuden omistajan nimi* shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization from *tekijänoikeuden omistajan nimi*.

Ohjelman voi asettaa tämän lisenssin alle vain tekijänoikeuden omistaja (lähtökohtaisesti ohjelman tekijä tai joissain tapauksissa hänen työnantajansa). Tämä tapahtuu laittamalla ohjelman jokaiseen lähdekooditiedostoon ylläoleva ilmoitus (muutettavat muutettuina).

2.4 Usein esitettyjä kysymyksiä vapaaohjelmista

2.4.1 Onko Open Sourcella mitään tekemistä tämän kanssa?

Open Source Software ja Free Software ovat englannin kielessä nimiä samalle asialle. Näistä Free Software on vanhempi ja perinteikkäämpi nimitys; termi Open Source Software luotiin, koska pelättiin, että sana *free* tulkitaan liian usein ilmaisuudeksi kuin vapaudeksi. Suomen kielessä tätä monimerkityksellisyysongelmaa ei ole, joten suosittelenkin termin *vapaaohjelma* käyttämistä suomennoksena kummallekin termille.

Englannin kielessä termeillä Open Source Software ja Free Software on yksi vivahde-ero: Open Source Software -termiä käytetään yleensä korostamaan vapaaohjelmien käytännöllisiä ominaisuuksia, kun taas Free Software -termi korostaa käsitteen takana olevaa filosofis-ideologista teoriaa.

Haluan vielä muistuttaa, että on tärkeää puhua vapaudesta. Viime aikoina on tavallisten ihmisten vapautta haluttu rajoittaa mitä ihmeellisimmillä tekosyillä. On vain yksi tapa varmistaa, että vapaus pysyy: meidän kaikkien täytyy puolustaa sitä jatkuvasti, esimerkiksi puhumalla siitä. Ihminen, joka puhuu open sourcesta, vaikenee vapaudesta, mutta hän, joka puhuu vapaaohjelmista, tekee oman pienen osansa maailman parantamiseksi.

2.4.2 Eikö vapaaohjelma ole pakko julkaista?

Lyhyt vastaus: Ei.

Pitkä vastaus: Se, että ohjelmasta tehdään vapaaohjelma, ei tuota kenellekään velvollisuutta antaa ohjelmaa kenellekään tai julkistaa sitä. Jotkin vapaaohjelmalisenssit saattavat kyllä

vaatia, että *jos* ohjelma annetaan jollekin, niin silloin on tiettyjä velvollisuuksia (kuten lähdekoodin antaminen samalla). Nämä vaatimukset eivät ole mitenkään universaaleja (esimerkiksi MIT-lisenssi ei tällaisia vaatimuksia aseta).

2.4.3 Saako vapaaohjelmaa käyttää omisteisen ohjelman teossa?

Lyhyt vastaus: Ehkä.

Hieman pidempi vastaus: Riippuu lisenssistä.

Pitkä vastaus on liian pitkä tälle luennolle. Joskus saa, joskus ei. Lue lisenssit, tai jos et jaksa, palkkaa lakimies.

2.4.4 Olen kuullut, että Open Source on tehokas tapa tehdä ohjelmia

Ensiksi lue vastaus Open Source -termistä yllä.

Monet vapaaohjelmien kannattajat, varsinkin Open Source -termiä suosivat, puhuvat, että vapaaohjelmakehitys olisi jotenkin ylivertainen ohjelmistonkehitysmenetelmä. En aio tässä arvioida, onko se ylivertainen, mutta eräänlainen menetelmä kyllä on olemassa. Kutsun sitä *basaarimenetelmäksi*. Menetelmän on ensimmäisenä kirjoittanut eksplisiittisesti näkyviin Raymond [6], mutta se ei ole hänen keksintönsä

Basaarimenetelmän idea on tämä: jollakulla on jokin (yleensä henkilökohtainen) tarve ratkaista jokin ongelma ohjelmalla. Hän kirjoittaa nopeasti ohjelman, joka hänelle riittää. Hän julkistaa sen vapaaohjelmana ja ilmoittaa siitä relevanteilla listoilla. Jos hyvin käy, niin ohjelma kerää huomiota muilta kehittäjä-käyttäjiltä. Nämä käyttävät ohjelmaa, huomaavat siinä puutteita ja alkavat lähettää alkuperäiselle kehittäjälle muutoksia koodiin, joka liittää ne mukaan omaan versioonsa ja julkaisee näin muutetun version. Sitten taas joku huomaa puutteita ja... Ennen pitkää elinvoimaisen ohjelman ympärille muodostuu yhteisö, joka kehittää ja ylläpitää ohjelmaa.

Viitteet

[1] John Backus. "Can programming be liberated from the von Neumann style? A functional style and its algebra of programs". *Communications of the ACM*, 21 (8), Aug. 1978.
URL http://portal.acm.org/citation.cfm?doid=359576.359579

[2] C. J. Burgess. "Software quality issues when choosing a programming language". In "Software Quality Management III Vol. 2", (pp. 25–31). Computational Mechanics Publications, 1995. ISBN 1- 85312-417-6.
URL http://citeseer.nj.nec.com/49884.html

[3] Patricia K. Lawlis. "Guidelines for choosing a computer language: Support for the visionary organization", Aug. 1997.
URL http://archive.adaic.com/docs/reports/lawlis/

[4] John K. Ousterhout. "Scripting: Higher-level programming for the 21st century". *Computer*, 31 (3), Mar. 1998.

[5] Alan J. Perlis. "Epigrams on programming". *ACM SIGPLAN Notices*, 17 (9), Sep. 1982.
URL http://www.bio.cam.ac.uk/~mw263/Perlis_Epigrams.html

[6] Eric S. Raymond. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an accidental revolutionary*. Beijing: O'Reilly, 2001, revised ed.
URL http://catb.org/~esr/writings/cathedral-bazaar/

[7] —. *The Art of Unix Programming*. Boston: Addison-Wesley, 2003.
URL http://catb.org/~esr/writings/taoup/