Mobiilisovelluksen kehittäminen avoimen lähdekoodin ympäristöjen avulla

Antti Kettunen

12.5.2008

Joensuun yliopisto
Tietojenkäsittelytiede
Pro gradu -tutkielma

Tiivistelmä

Avoimen lähdekoodin periaatteella toimivat ohjelmointiympäristöt ja ohjelmistoprosessin apuvälineenä käytettävät sovellukset mahdollistavat edullisen ohjelmistokehityksen ilman kalliita lisenssimaksuja. Avoimen lähdekoodin kehitystyökaluihin löytyy omia lisäosia mobiilien ohjelmien kehittämistä varten. Aloittelevilla yrityksillä voi olla edullista aloittaa ohjelmistokehitys mobiililaitteisiin käyttämällä vain avoimen lähdekoodin periaatteella tehtyjä sovelluksia ohjelmistojen suunnitteluun, dokumentointiin, toteutukseen, testaukseen ja ylläpitoon. Tällä tavoin koko ohjelmistoprosessin elinkaari tulee katettua.

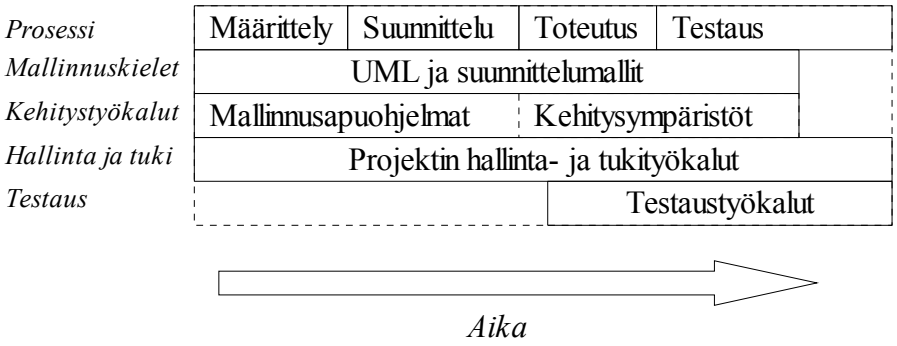
ACM-luokat: (ACM Computing Classification System, 1998 version): C.1.4, D.2.11, D.2.2, K.5.1, K.6.3

Avainsanat: Avoin lähdekoodi, mobiililaite, mobiilialusta, Java, ohjelmointi, ohjelmistotuotanto, ohjelmistoprosessi, kehitystyökalut, UML, lisenssi.

Sisällysluettelo

1 Johdanto

Mobiilisovellusten kehittäminen on pitkä prosessi, joka sisältää useita eri vaiheita ja siinä tarvitaan useita eri työkaluja apuvälineinä. Kuten talonrakennuksessa tarvitaan pieniä ja isoja työkaluja, esimerkiksi vasaraa, sirkkeliä tai vaikkapa suurta kaivinkonetta, niin sovelluksen kehittämisessä tarvitaan kääntäjää, suunnitteluohjelmaa tai vaikka laajempaa kehitysympäristöä. Sovellus täytyy ensin määritellä ja suunnitella hyvin, jonka jälkeen sovellusta on mahdollista aloittaa toteuttamaan ja testaamaan. Tämän ohjelmistoprosessin aikana käytetään erilaisia suunnittelumenetelmiä ja tarvitaan paljon erilaisia helpottavia työkaluja suunnitteluun, toteutukseen ja testaukseen. Mallinnuskieliä, kuten UML:ää käyttämällä, voidaan sovellus suunnitella ja määritellä käyttäen hyväksi mallinnusapuohjelmia. Mallinnusta voidaan tehdä myös kehitysympäristöillä, joilla mallinnuksen jälkeen voidaan aloittaa ohjelman toteutus koodaamalla. Kehitysympäristöihin voidaan liittää erilaisia lisäosia ja työkaluja, joilla esimerkiksi juuri mallinnusta tai testausta on mahdollista suorittaa. Testauksessa voidaan käyttää myös erillisiä testaustyökaluja. Koko tämän prosessin aikana tarvitaan hallinta- ja tukityökaluja dokumentointiin, aikataulujen suunnittelemiseen ja resurssien määrittelemiseen. Kuva 1 kuvastaa, miten nämä eri asiat liittyvät toisiinsa ja mitä työkaluja käytetään missäkin prosessin eri vaiheessa.



Kuva 1: Tutkielman rakenne

Tässä tutkielmassa keskitytään tutkimaan ohjelmistojen kehittämistä mobiililaitteisiin avoimen lähdekoodin ohjelmien avulla kuvan 1 esittämän rakenteen mukaisesti. Avoimeen lähdekoodiin perustuvia kehityksen apuvälineitä on mahdollista saada ilmaiseksi tai ainakin edullisesti ja niiden kehittäminen on jatkuvaa. Avoimen lähdekoodin apuvälineiden käyttämiseen saa tukea yhteisöiltä, jotka noita ohjelmia kehittävät ja heille voi antaa myös palautetta ongelmista tai toiveita tarvittavista ominaisuuksista. Yhteisöihin on mahdollista

myös itse liittyä ja osallistua kehittämiseen sitä kautta tai sitten riippuen ohjelman lisenssistä, ohjelmia voi kehittää edelleen oman tarpeen mukaan. Nämä asiat auttavat monesti yrityksen päätökseen ottaa käyttöön avoimen lähdekoodin ohjelmia.

Tutkielmassa kuvataan mobiililaitteen käsitteet, ohjelmistoprosessin ja suunnittelun kulku, avoimen lähdekoodiin liittyvät asiat sekä joitakin avoimen lähdekoodin työkaluja. Monet tutkielman aiheet sopivat minkä tahansa ohjelmiston kehittämiseen ja kehitysprosessiin, mutta tutkielmassa on tarkasteltu myös asioita, jotka täytyy ottaa huomioon erityisesti kännyköihin tehtävien Java -sovelluksien kehittämisessä ja kehitysprosessissa.

Tutkielman luvussa 2 kerrotaan mobiililaitteiden erityispiirteet ja alustat. Ohjelmistoprosessin kulku ja eri vaiheet kerrotaan tutkielman luvussa 3. Tämän jälkeen tarkastellaan ohjelmiston suunnittelukieltä UML:ää ja suunnittelua luvussa 4 ja suunnittelua helpottavia suunnittelumalleja luvussa 5. Tämän jälkeen luvussa 6 tarkastellaan, mitä avoin lähdekoodi tarkoittaa ja kerrotaan hiukan avoimen lähdekoodin historiaa. Kun tiedetään, mitä mobiililaitteet ovat, miten ohjelmiston kehitysprosessi mobiililaitteisiin etenee, mitä tarkoittaa avoin lähdekoodi ja miten ohjelmia on mahdollista suunnitella, päästäänkin jo keskittymään avoimen lähdekoodin kehitys- ja tukityökaluihin luvussa 7. Luvussa 8 tarkastellaan, mihin nuo avoimen lähdekoodin eri kehitys- ja tukityökalut yltävät, miten niitä voidaan käyttää yhdessä ja mitä ne mahdollistavat ohjelmistojen kehittämisessä mobiileihin laitteisiin. Tutkielman päättää lyhyt yhteenvetoluku.

2 Mobiililaitteet ja alustat

Mobiililaitteet ovat yleistyneet ja yhä useammat ihmiset käyttävät niitä. Mobiililaite käsitteenä on epämääräisesti määritelty, eikä ole olemassa yksikäsitteistä määritelmää, mitä laitteita voidaan kutsua mobiililaitteiksi ja mitkä laitteet ovat vain sitä muistuttavia laitteita. Mikkonen määritteleekin mobiililaitteen mukana kannettavaksi laitteeksi, joka sisältää tietojärjestelmän, kuten esimerkiksi kannettava tietokone, PDA-laite tai rannekelloon tai sykemittariin integroitu tietokone [19]. Tässä tutkimuksessa tarkastelen enemmän kuitenkin sellaisia laajalle levinneitä ja lähes jokaisesta kodista löytyviä mobiililaitteita, joita ohjelmointia harrastanut aloittelija tai ammattilainen voi ohjelmoida internetistä löytyvillä avoimen lähdekoodin periaatteen mukaisilla kehitystyökaluilla. Tällaisia laitteita ovat erityisesti nykyiset matkapuhelimet ja multimedialaitteet.

2.1 Mobiililaitteen erityispiirteitä

Mobiililaitteen ominaispiirteitä ovat pieni koko, vähäinen energiankulutus ja rajoitettu muistin määrä. Perusominaisuuksiin kuuluu mobiililaitteen sisältämä ohjelmisto, joka tekee siitä houkuttelevan ja joka on luonut mobiiliohjelmoinnin yhdeksi ohjelmistotekniikan tärkeäksi osa-alueeksi. Mobiililaitteen tärkein ominaisuus on kuitenkin sen helppo kuljetettavuus ja sen huomaa erityisesti matkapuhelimista, mitkä mahtuvat helposti ja huomaamattomasti taskuun. Suuremmalla koolla on kuitenkin tietty lisäarvo ja esimerkiksi tekstiviestin kirjoittaminen voi olla helpompaa kommunikaattorityyppisellä laitteella, jossa on kirjaimet tarjoava näppäimistö, kuin tavallisella matkapuhelimen näppäimistöllä. Mobiililaitteissa on monesti sellaisia ominaisuuksia, mitä laitteen alkuperäinen käyttötarkoitus ei vaadi. Matkapuhelimissa on esimerkiksi MP3-soittimia, FM-radioita sekä peli- ja ohjelmointiympäristöjä, jotka mahdollistavat laitteen käyttötarkoituksen laajentamisen [19].

Mobiililaitteille on ominaista käyttäjien tarpeiden huomioiminen ja laajennettavuus. Laitteisiin on monesti mahdollista asentaa uusia ohjelmia tai laitteita, kuten handsfree -toiminto. Mobiililaitteiden odotetaan toimivan moitteettomasti ja nopeasti. Jokainen muistivuoto on vaarallinen erityisesti matkapuhelimissa, koska laite sammutetaan harvoin ja useat muistivuodot voivat syödä käytettävissä olevan muistin loppuun [19].

2.2 Mobiilialustat

Mobiililaitteiden ohjelmistot toimivat *mobiilialustojen* päällä. Erilaisia mobiilialustoja on useita ja alustat poikkeavat toisistaan joissakin määrin [19]. Tässä kappaleessa kuvataan pääsääntöisesti matkapuhelimissa käytettyjä mobiilialustoja.

2.2.1 Java

Java -ohjelmointikieli kehittyi alunperin sulautettujen järjestelmien (embedded systems) verkottamista varten luodusta Sun Microsystemsin projektista, jossa alunperin käytettiin OAK -ohjelmointikieltä. Kieltä kehitettiin edelleen ja projektin sivutuotteena syntyi täysin uusi ohjelmointikieli, jonka suunnitteluun otettiin piirteitä myös Smalltalk -ohjelmointikielestä. Java on oliopohjainen, järjestelmästä riippumaton ohjelmointikieli, mikä tarkoittaa sitä, että ohjelmia voi kehittää ja testata eri ympäristöissä. Javalla voi ohjelmoida muun muassa www-palvelimia, digitaalitelevisioita, kämmentietokoneita ja matkapuhelimia [21].

Javan avulla on mahdollista kehittää ohjelma, joka toimii useissa eri laitteissa ilman, että ohjelman joutuisi uudelleenohjelmoimaan. Pöytätietokoneissa ja palvelinympäristöissä on mahdollista luoda ohjelma, mikä toimii useilla eri alustoilla ja useissa eri käyttöjärjestelmissä. Lisäksi Javaa on laajennettu toimimaan pienempiin laitteisiin, kuten esimerkiksi matka-puhelimiin [14]. Yksi Java -ympäristö ei käy joka paikkaan, vaan se on jaettu kolmeen osaan [15]:

- J2EE (Java 2 Enterprise Edition)
- J2SE (Java 2 Standard Edition)
- J2ME (Java 2 Micro Edition)

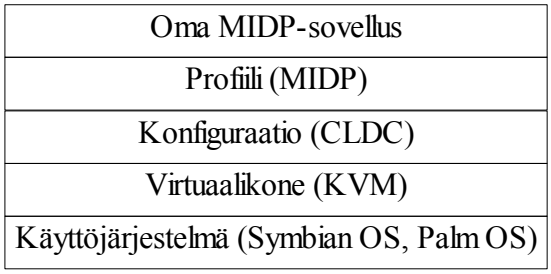
J2SE on perustyyppinen Java, jolla voidaan tehdä pääasiassa työasemassa toimivia sovelluksia [15]. Se sisältää muun muassa Javan peruskirjastot, joita ovat verkko-ohjelmointikirjastot, tietokantarajapinnat sekä AWT- ja Swing -käyttöliittymäkomponentit [21]. J2EE on laajin Javoista ja se on tarkoitettu erityisesti yritysten käyttöön, jotka voivat tehdä laajoja palveluita, kuten on-line kauppoja, asiakastietojärjestelmiä ja taloushallinnon järjestelmiä. Se on tarkoitettu erityisesti hajautettujen järjestelmien palvelinpään ohjelmointiin ja se sisältää käyttöliittymäpuolen tekniikkaa, kuten serveletit ja JSP. J2ME on javan pienin osa, joka on tarkoitettu sulautettujen järjestelmien ja pienten laitteiden ohjelmointiin, kuten

kännyköihin [15]. J2ME on ollut tuettuna useissa matkapuhelimissa vuodesta 2001 lähtien ja sitä tukevat myös muun muassa digitaalitelevisiolähetysten vastaanoton mahdollistavat digiboksit, PDA-kämmentietokoneet ja informaatiokioskit [21].

J2ME toimii erityyppisissä laitteissa, joille on mahdotonta määritellä yhtä ohjelmointirajapintaa. Laitteissa on erilaiset prosessorit, virtalähteet, muistimäärä, tallennusmahdollisuudet ja käyttöliittymät. J2ME onkin tavallaan kompromissi ja laite-erojen vuoksi J2ME on jaettu arkkitehtuuriltaan eri osiin, joista jokainen hoitaa tietyt tehtävät. J2ME tarjoaa ainakin seuraavat työkalut [21]:

- Java-virtuaalikone (esim. KVM)
- sovellusohjelmoinnin API-rajapinnat
- sovellusten jakelutyökalut
- laitteiden konfigurointityökalut

J2ME sisältää kolme tasoa: virtuaalikoneen, konfiguraation ja profiilin (ks. kuva 2). Tasot riippuvat toisistaan, eli joku konfiguraatio saattaa vaatia tietyn virtuaalikoneen. Siksi J2ME-ohjelmat eivät ole täysin siirrettävissä järjestelmästä toiseen. J2ME-alusta on suunnattu kahteen tuoteryhmään: henkilökohtaisiin verkkoon kytkettyihin mobiililaitteisiin, kuten matkapuhelimiin, hakulaitteisiin ja PDA-laitteisiin sekä useammalle käyttäjälle jaettuihin ja kiinteällä yhteydellä verkkoon kytkettyihin laitteisiin, kuten kosketusnäytöllä käytettäviin informaatiokioskeihin tai ajoneuvon paikannusjärjestelmiin. Näille eri tuoteryhmille on kehitetty omat konfiguraatiot, jotka ovat CDC (Connected Device Configuration) esimerkiksi navigointijärjestelmille ja CLDC (Connected Limited Device Configuration) esimerkiksi matkapuhelimille [21].



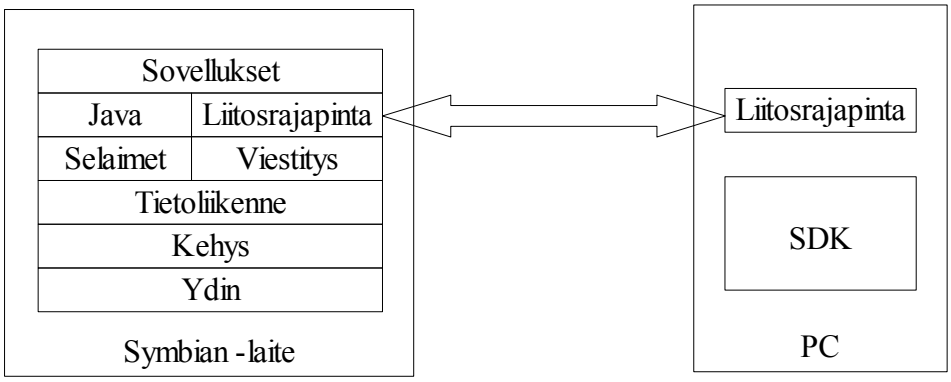
Kuva 2: J2ME-arkkitehtuuri (esimerkkinä MIDP-sovellus) [21]

Konfiguraatio ei pelkästään riitä sovellusten kehittämiseen tietylle alustalle, koska siitä on karsittu kaikki ylimääräinen pois, kuten kaikki käyttöliittymään liittyvä. Erityyppiset laitteet

vaativat lisätoimintoja ja nämä lisätoiminnot toimivat profiilien avulla. Matkapuhelinten käyttöliittymä on esimerkiksi yksi profiili. CLDC -konfiguraatiossa ei tueta AWT- tai Swing-rajapintoja, jotka eivät sellaisenaan sovi pienille näytöille tehtäviin käyttöliittymiin. Siksi matkapuhelinten LDC-näyttöjä varten suunniteltiin uusi käyttöliittymärajapinta LCDUI, joka sisältyy matkapuhelimessa käytettyyn *MIDP* (Mobile Information Device Profile) -profiiliin. Profiilit siis laajentavat konfiguraatiota ja tietylle profiilille tehdyt ohjelmat toimivat ilman erityisvaatimuksia muissa laitteissa, jotka tukevat samaa profiilia. Siksi matkapuhelimiin tehdyt MIDP-profiilia tukevat ohjelmat toimivat kaikissa muissa samaa profiilia tukevissa laitteissa. Kaikki suurimmat matkapuhelinvalmistajat tukevat nykyisin MIDP-profiilia, kuten Nokia, Siemens, Motorola, Samsung ja Sony-Ericsson. J2ME-konfiguraatioita ja -profiileja on olemassa useita eri käyttötarkoituksiin. Lisäksi on olemassa valinnaisia kirjastoja, kuten SMS-tekstiviestien lähettämistä tukeva WMA-kirjasto ja MMAPI (Multimedia API) [21].

2.2.2 Symbian OS

Symbian on mobiililaitevalmistajien, kuten Nokian, Sony-Ericssonin, Psionin ja Matsushitan (eli Panasonic) perustama yritys, jonka kehittämää Symbian OS -ohjelmistoalustaa ovat lisensioineet myös monet muut suuret yritykset [19]. Symbian OS -ohjelmistoalusta, josta käytetään myös nimeä Symbian Platform, tarjoaa sovelluskehitysalustat Java ja C++ -ohjelmointikielille [15].



Kuva 3: Symbian Generic Technology -arkkitehtuurikuvaus [15]

Symbian -käyttöjärjestelmän tarkoitus on soveltua monille eri laitteille, joten se on jaettu geneeriseen teknologiaan (ks. kuva 3) ja graafiseen käyttöliittymään. Geneerinen teknologia toimii käyttöjärjestelmän sydämenä ja graafinen käyttöliittymä mahdollistaa räätälöinnin erilaisille kommunikaattoreiden ja älypuhelinten näytöille. Tällä pyritään siihen, että järjestelmän ydin on kaikilla Symbian -laitteilla sama ja käyttöliittymä on mahdollista toteuttaa

laitekohtaisesti. Geneeriseen teknologiaan kuuluvat moniajoa tukeva ydin, tiedonhallinta, kommunikointi, grafiikka, multimedia, tietoturva, sovellus-, viestitys- ja selainohjelmat, Java Runtime Environment (JRE) sekä tuki tiedon synkronointiin. Graafisia käyttöliittymiä ovat muun muuassa Crystal, Quartz sekä Nokian oma Series 60 -käyttöliittymä [15].

2.2.3 Windows Mobile

Windows Mobile on käyttöjärjestelmä mobiililaitteille ja se perustuu Microsoft Win32 API:iin. Windows Mobile toimii useilla eri laitteilla, joita ovat esimerkiksi Pocket PC:t, älypuhelimet sekä Portable Media Center -laitteet. Käyttöjärjestelmä on suunniteltu samantyyppiseksi pöytätietokoneen Windows-käyttöjärjestelmän kanssa. Windows Mobile perustuu alunperin Pocket PC 2000 -käyttöjärjestelmään, josta sitä on kehitetty edelleen ja laajennettu. Siitä on kehitetty muun muassa seuraavat versiot: Pocket PC 2000, Pocket PC 2002, Windows Mobile 2003, Windows Mobile 2003 SE, Windows Mobile 5.0, Windows Mobile 6 sekä Windows Mobile 6.1 [49].

Edellisin versio käyttöjärjestelmästä, joka on kehitetty erityisesti älypuhelimille, käyttää nimeä Windows Mobile 6 Standard. Vastaava versio Pocket PC -laitteille, joissa on puhelintoiminto, on Windows Mobile 6 Professional. Pocket PC -laite ilman puhelintoimintoa käyttää Windows 6 Mobile Classic -versiota. Windows Mobile 6 -käyttöjärjestelmä sisältää muun muassa Office Mobile -tuen älypuhelimille, käyttöjärjestelmän live-päivitys-ominaisuuden sekä VoIP-tuen. Uusin versio käyttöjärjestelmästä on Windows Mobile 6.1, joka päivittää joitakin edeltävän version ominaisuuksia. Seuraava versio alustasta tulee olemaan Windows Mobile 7 ja sen julkaisu on suunniteltu vuodelle 2009 [49].

2.2.4 Embedded Linux

Embedded Linux on sulautettuun ympäristöön muokattu versio Linux -käyttöjärjestelmästä. Linuxin edut mobiililaitteissa on samat kuin työasemapuolella, eli toteutus on vapaasti muunneltavissa ja edelleen kehitettävissä [19]. Etuina ovat myös se, että ei tarvitse maksaa lisenssimaksuja, tukea on laajasti saatavilla sekä alusta on kypsä ja vakaa [48]. Embedded Linux on jaettu kolmeen osaan, jolla mahdollistetaan erilaisten kokoonpanojen yksinker-taisempi valinta [19]:

- minijärjestelmä (minimal system environment)
- keskitason järjestelmä (intermediate system environment)

- täysi järjestelmä (full system environment)

Minijärjestelmä käsittää laitteet, joissa käytetään vain yhtä prosessia ja joita kutsutaan syvästi sulautetuiksi laitteiksi. Keskitason järjestelmä mahdollistaa useamman prosessorin ja lisää minijärjestelmään massamuistin ja siihen liittyvät toiminnot, kuten I/O:n, dynaamisen linkkaamisen ja tiedostojärjestelmärajapinnat. Täysi järjestelmä on lähes täysi Linux -ympäristö, josta on jätetty pois monet systeemitason apujärjestelmät [19].

Embedded Linux -käyttöjärjestelmää käytetään matkapuhelimissa, PDA-laitteissa, mediasoittimissa ja muissa kulutuselektroniikan tuotteissa. Käyttöjärjestelmä sopii myös muihin sulautettuihin järjestelmiin, kuten teollisuuden automaatiolaitteisiin, navigointi-laitteisiin sekä lääkintälaitteisiin. Matkapuhelinten Linux -standardointia varten perustettiin vuonna 2004 Linux Phone Standards -foorumi ja vuonna 2006 perustettiin LiMo Foundation -järjestö, jonka tehtävänä on helpottaa kolmansien osapuolten ohjelmistokehitystä Embedded Linux -alustalle [48].

Matkapuhelimissa käytettyjä Linux-muunnelmia on useita ja esimerkiksi MontaVista Software -yritys on kehittänyt MontaVista Linux -käyttöjärjestelmän sulautettuihin järjes-telmiin, kuten matkapuhelimiin [48]. Käyttöjärjestelmä on käytössä Motorolan useissa malleissa, NEC:n muutamissa malleissa sekä Panasonic P901i -matkapuhelimessa [47]. Muita matkapuhelimiin kehitettyjä tai kehitteillä olevia Linux -muunnelmia ovat muun muassa Qtopia sekä OpenMoko [48]. Maemo nimistä Linux -muunnelmaa on käytetty muun muassa Nokia 770 Internet Tablet -laitteessa ja sen seuraajissa Nokia N800 ja Nokia N810 -laitteissa [46].

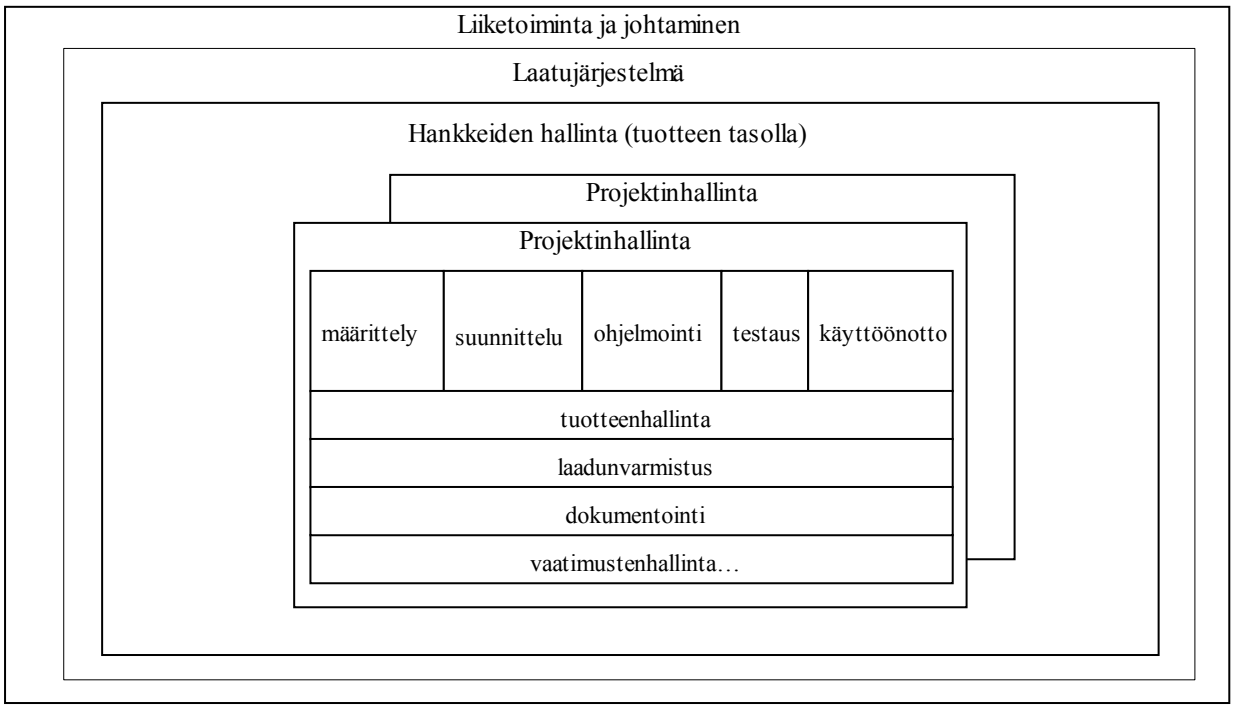
Yksi uusimmista Linuxiin perustuvista käyttöjärjestelmistä on vasta kehitteillä oleva Android. Se on Linuxiin perustuva avoimen lähdekoodin matkapuhelinalusta, jota kehittää Open Handset Alliance -ryhmittymä. Ryhmittymän tarkoituksena on kehittää avoimia standardeja mobiileihin laitteisiin ja siihen kuuluu useita yrityksiä, kuten Google, HTC, Intel, Motorola, Qualcomm, T-Mobile ja NVIDIA. Ryhmittymä julkaisi marraskuussa 2007 kehittäjiä varten SDK:n, mihin kuuluvat toteutus- ja debuggaustyökalut, joukko kirjastoja, emulaattori, dokumentaatio, esimerkkiprojekteja, ohjekirja ja paljon muuta. Kehitysympäristön myötä Androidin ominaisuuksia on julkistettu ja niitä ovat muun muassa useat yhteystekniikat (Bluetooth, EDGE, 3G ja WiFi), tekstiviestit (SMS ja MMS), WebKit -internetselain, Java -virtuaalikone, medioitten tuki (MPEG-4, H.264, MP3 ja AAC) sekä useat eri laitteistotuet

(kamera, kosketusnäyttö, GPS, kompassi, kiihtyvyysanturi ja 3D-grafiikka). Adroid -alusta on tarkoitus päästää markkinoille vuonna 2008 [45].

Android -käyttöjärjestelmä ei eroa puhelimen ydinohjelmien tekijöille ja kolmansien osapuolten ohjelmien tekijöille, vaan kaikki ohjelmat ovat samanarvoisia. Tämä tarkoittaa sitä, että kaikki ohjelmat pääsevät käsiksi puhelimen kaikkiin ominaisuuksiin, mikä laajentaa tulevien ohjelmistojen tarjontaa ja ominaisuuksia. Ohjelmien on esimerkiksi mahdollista tehdä puheluja, lähettää tekstiviestejä tai käyttää puhelimen kameraa. Käyttäjät voivat räätälöidä puhelimensa omien tarpeittensa mukaan ja esimerkiksi puhelimen päänäytön voi vaihtaa tai käyttäjä voi valita haluamansa ohjelman valokuvien katselemiseksi [34].

3 Ohjelmistoprosessi

Ohjelmistotuotanto voidaan jakaa eri osa-alueisiin kuten kuvassa 4 on kuvattu. Yrityksellä voi olla useita eri ohjelmistoprojekteja samanaikaisesti, mutta kaikki projektit pyritään suunnittelemaan ja toteuttamaan saman kaavan mukaisesti. Joskus ohjelmistoprojektit voivat olla osaprojekteina laajemmissa tuotekehityshankkeissa. Jokaisen projektin kehitysprosessiin kuuluvat määrittely, suunnittelu, ohjelmointi, testaus, käyttöönotto ja ylläpito[1]. Lisäksi ohjelmistoprojektiin liittyy koko projektin elinkaaren kestäviä tukitoimintoja, joista tärkeimpiä ovat laadunvarmistus, tuotteenhallinta ja dokumentointi [11].



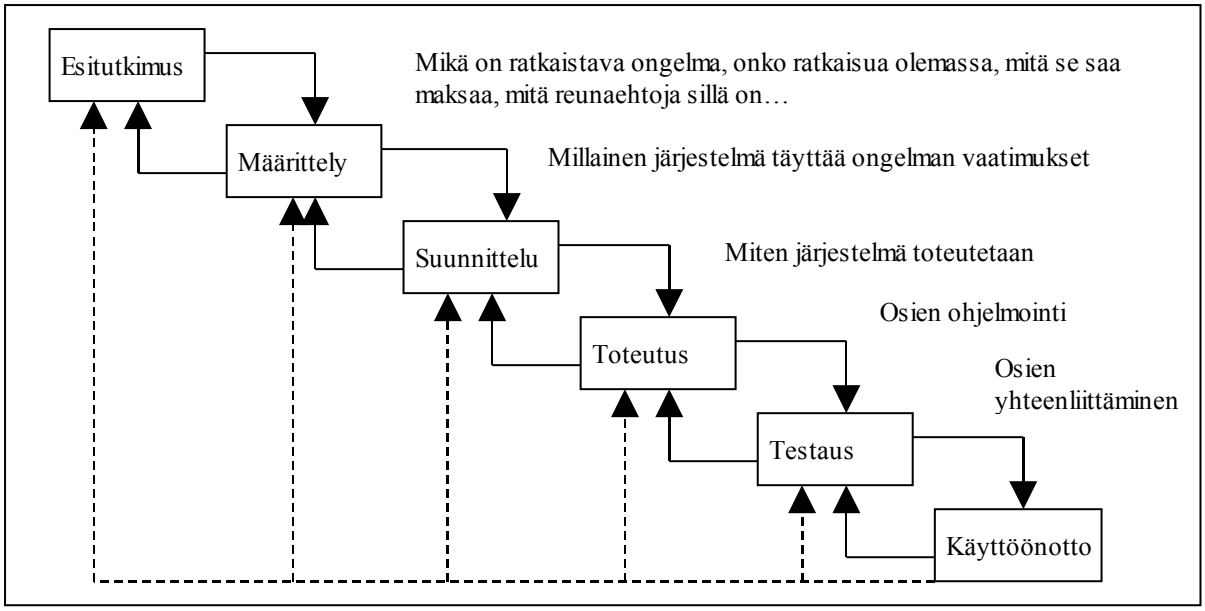
Kuva 4: Ohjelmistotuotannon osa-alueet

Ohjelmiston elinkaari kuvaa hyvin ohjelmiston kehitysvaiheita sen kehittämisen aloittamisesta aina sen käytöstä poistamiseen saakka. Vaihejakomallilla voidaan ohjelmiston kehitystyö ja koko elinkaari jakaa eri vaiheisiin. Vaihejakomalleista yleisin on vesiputousmalli, jota on hahmoteltu kuvassa 5. Vesiputousmalleissa on yleensä eroteltu ainakin määrittely, suunnit-

1 B'Far on kehittänyt prosessia edelleen erityisesti mobiiliohjelmien kehittämistä varten. Prosessi pohjautuu UML:ään ja laajentaa yleistä prosessia soveltumaan paremmin mobiiliohjelmien kehittämiseen. Prosessissa tarkastellaan tarkemmin erityisesti käyttöliittymän suunnittelua ja käytetään käyttötapauskaavioita alemmalla tasolla ja käyttöliittymäkeskeisemmin. Prosessissa esitellään menetelmä mobiilien käyttötapausten laatimiseksi [1].

telu ja toteutusvaiheet. Vaiheisiin voi liittyä laadunvarmistustoimenpiteitä, joita ovat tarkistukset ja testaukset. Näiden avulla pyritään poistamaan virheitä ohjelmasta mahdollisimman varhaisessa vaiheessa [11].

Esitutkimuksen tarkoituksena on kartoittaa järjestelmän yleiset vaatimukset ja sen perusteella voidaan päätellä, miksi ohjelmisto tulisi tehdä tai miksi sitä ei kannattaisi tehdä. Monesti esitutkimus on osana määrittelyvaihetta ja joissakin jatkuvan tuotekehityksen ympäristöissä esitutkimus voi kuulua vaatimustenhallinnan tukitoiminnon piiriin [11]. Muita ohjelmistoprojektin elinkaaren vaiheita on kuvattu seuraavissa kohdissa.



Kuva 5: Vesiputousmalli

3.1 Määrittely

Ohjelmistokehitysprosessissa ensimmäisenä vaiheena esitutkimuksen jälkeen on tavallisesti *määrittelyvaihe*, joka sisältää vaatimusten keräämisen. *Vaatimukset* voidaan jakaa toiminnallisiin ja ei-toiminnallisiin vaatimuksiin. Toiminnalliset vaatimukset määrittelevät mitä toimintoja järjestelmältä halutaan saada aikaan. Ei-toiminnalliset vaatimukset määrittelevät mitä ehtoja näiden toimintojen toteutuksen tulee täyttää [16].

Määrittelyvaiheessa asiakkaalta saatavia vaatimuksia analysoidaan ja niistä johdetaan vaatimukset, jotka määrittelevät toteutettavan järjestelmän. Tämän prosessin tuloksena saadaan dokumentti, jota kutsutaan toiminnalliseksi määrittelyksi tai yleisemmin vaatimusmäärittelyksi. Dokumentissa kuvataan sekä toiminnalliset että ei-toiminnalliset vaatimukset

sekä rajoitukset. Ei-toiminnalliset vaatimukset liittyvät usein suoritustehoon tai käytettävyyteen. Rajoituksia voivat olla laitteessa olevan muistin määrä tai toteutuksessa käytettävä ohjelmointikieli [11].

Määrittelyvaiheen tarkoitus on muuttaa asiakkaalta saatavat vaatimukset tarkoiksi ohjelmistovaatimuksiksi. Asiakasvaatimuksella tarkoitetaan asiakkaan ongelmaa tai tarvetta, minkä pohjalta asiakas haluaa jonkun ohjelmiston tai uuden ominaisuuden jo olemassa olevaan järjestelmään. Ominaisuus voidaan toteuttaa useiden järjestelmän toimintojen avulla ja joku tietty toiminto voi liittyä useisiin eri ominaisuuksiin. Ohjelmistovaatimukset pitävät sisällään sekä ominaisuudet että toiminnot [11].

Vaatimuksia voidaan kerätä asiakkaalta useilla eri keinoilla. Vaatimusten keruukeinoja voivat olla loppukäyttäjien haastatteleminen, nykyisen järjestelmän dokumentaation tutkiminen tai vaikka uuden ja vanhan järjestelmän välisten erojen analysointi. Vaatimukset tulee määritellä riittävän tarkasti järjestelmän testattavuuden varmistamiseksi. Vaatimusten laatimisen jälkeen ne käydään läpi asiakkaan kanssa ja varmistetaan, että molemmat osapuolet ovat ymmärtäneet asian samalla tavalla ja että vaatimukset on dokumentoitu kunnolla [20].

3.2 Suunnittelu

Suunnitteluvaiheessa on tarkoitus suunnitella määrittelyvaiheessa saatujen toimintojen toteutus. Suunnitteluvaiheessa ei toteuteta vielä mitään käytännössä, vaan tarkoituksena on määrittää ne keinot, joilla vaatimusten mukainen järjestelmä toteutetaan [20]. Suunnitteluvaihe voidaan jakaa kahteen tasoon; arkkitehtuurisuunnitteluun sekä moduulisuunnitteluun. Moduuli voidaan tulkita ohjelmasta erotettavissa olevaksi loogiseksi kokonaisuudeksi, joka sisältää tietomäärittelyitä ja joukon tietoja käsitteleviä funktioita [11]

Arkkitehtuurisuunnittelu on korkeamman tason näkemys järjestelmästä ja siinä järjestelmä jaetaan moduuleihin, joiden rajapinnat määritellään samalla. Arkkitehtuurisuunnittelu on pääasiassa moduulien työnjaon määrittelyä. Tavoitteena on saavuttaa mahdollisimman uudelleenkäytettäviä ja mahdollisimman vähän toisistaan riippuvia moduuleita siten, että yksittäisen moduulin toteutuksen muuttaminen ei näy moduulin ulkopuolelle. Tämä helpottaa muutosten tekemistä sekä projektin osien tekemisen toisistaan riippumattomasti [11].

Moduulisuunnittelussa suunnitellaan jokaisen moduulin sisäinen rakenne. Tavoitteena on saada aikaan kokonaisuuksia ja osia, joita voidaan jakaa yksittäisten suunnittelijoiden

tehtäväksi. Joskus moduulisuunnittelua ei pidetä omana vaiheena, vaan moduulin suunnittelu, toteutus ja testaus muodostavat yhdessä yhden vaiheen ja kuuluu tällöin toteutusvaiheen alapuolelle. Suunnitteluvaiheen tarkoitus on vastata kysymykseen, miten järjestelmä toimii, kun määrittelyvaiheessa saadaan vastaus kysymykseen, mitä järjestelmä tekee [11]

Ohjelmia suunnitellessa mobiileihin laitteisiin on tärkeää ottaa huomioon yhtenäinen käytettävyys, joka saavutetaan monilla erilaisilla suunnitteluratkaisuilla. Ongelmana on myös se, että mobiililaitteen käyttäjä haluaa monesti suorittaa nopeita ja keskitettyjä toimintoja, jotka vaativat mahdollisimman vähän näppäinten painamista. Seuraavat asiat onkin hyvä ottaa huomioon suunnitellessa ohjelmia mobiililaitteisiin, jos halutaan erityisesti kiinnittää huomiota käytettävyyteen ja käyttäjän toimintaan [18]:

- rajaus
- suorituskyvyn huomioon ottaminen
- käyttöliittymän suunnittelu
- tietomallin ja muistinkäytön merkitys
- tiedonsiirto ja I/O

Rajaamalla pyritään hahmottamaan, mitä ohjelma voi ja ei voi tehdä ottamalla huomioon toimintojen tärkeys ja laitteen fyysiset ominaisuudet. Rajausta voidaan helpottaa tekemällä ohjelmasta kuvia, malleja ja luomalla prototyyppejä. Harvoin käytetyt toiminnot voidaan jättää vähemmälle huomiolle, kun taas usein käytettyjen toimintojen tulee olla hyvin toimivia. Suorituskyky täytyy ottaa huomioon jo suunnitteluvaiheessa, sillä mobiililaitteissa voi olla joitain rajoituksia sen suhteen. Käyttöliittymän suunnittelussa on hyvä ottaa huomioon, että useimmiten käytetyt toiminnot toimivat hyvin. Käyttöliittymän suunnitteluun vaikuttaa myös se, millainen mobiililaite on kyseessä. Käyttöliittymän suunnittelu isonäyttöiseen mobiili-laitteeseen ja pienäyttöiseen mobiililaitteeseen on erilaista [18].

Mobiililaitteissa on monesti rajatut laiteominaisuudet kuten laitteen koko, virrankäyttö ja muistimäärät ja niiden taso on riippuvainen laitteen hinnasta. Laitteen käsittelemän tiedon esitysmuoto vaikuttaa siihen, miten se sijoitetaan laitteen muistiin, miten laite käyttäytyy vaativissa tehtävissä ja miten ohjelma järjestelee tiedon. Sen takia tietorakenteet ja muistin käyttäminen yleensä täytyy ottaa huolellisesti huomioon jo suunnitteluvaiheessa. Ohjelmien

suunnittelussa mobiililaitteisiin täytyy ottaa huomioon ohjelman tietoliikenteen ja I/O:n määrittely ja miten ohjelma kommunikoi muiden laitteiden kanssa. Laitteen omiin tietoihin pääsy on usein nopeaa, kun ulkopuolisten laitteiden kanssa se on hitaampaa. Tallennettavien tietojen abstraktiotason suunnittelu etukäteen vaikuttaa ohjelman suorituskykyyn. Suuret tietomäärät on hyvä tallentaa yksinkertaisiin ja nopeasti luettaviin rakenteisiin, kun taas pienemmät tietomäärät on mahdollista tallentaa enemmän ihmisten ymmärtämään ja monikäyttöisempään muotoon, jota laite joutuu kuitenkin enemmän prosessoimaan [18].

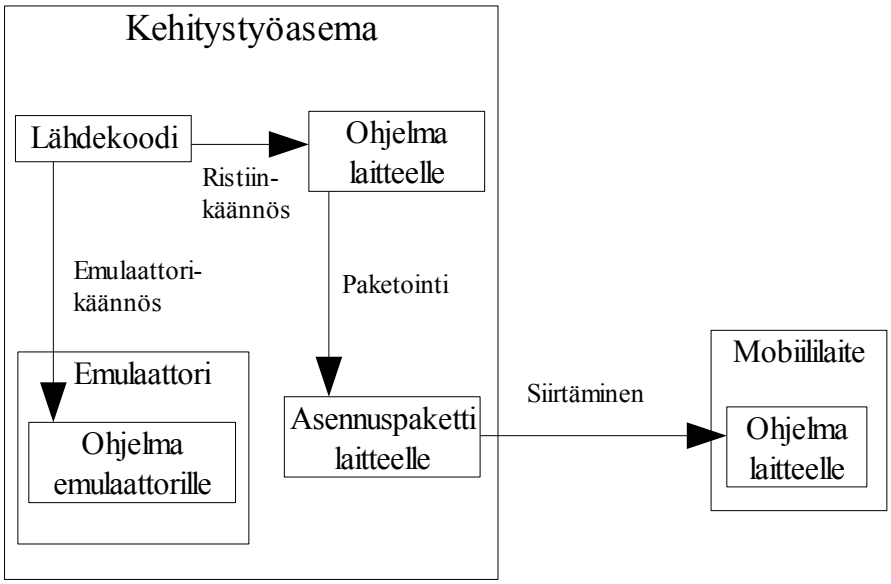
3.3 Toteutus

Toteutusvaiheessa suunnitteluvaiheen moduulit muutetaan oliopohjaisen ohjelmointikielen lähdekoodiksi. Ohjelmointikielestä riippuen se voi olla joko helppoa tai vaikeaa [6]. Järjestelmä toteutetaan sovitussa ympäristössä ja sovittuja työkaluja käyttäen. Tarkoituksena on toteuttaa moduulit ja etsiä niistä virheitä testaamalla ne yksikkötestauksella. Moduulit on tarkoitus liittää toisiinsa ja testata moduulien väliset yhteydet, verrata saatuja tuloksia odotettuihin tuloksiin ja korjata löytyneet virheet. Toteutusvaiheen tuotoksia ovat lähdekoodi, suoritettava koodi, testikoodi, testaustietokanta sekä testien tulokset [20].

Koskimiehen [16] mukaan toteutusvaihe ei vaadi enää merkittävien suunnitteluratkaisujen tekemistä, jolloin se on periaatteessa pitkälle automatisoitavissa. Tällöin apuna voi käyttää ohjelmageneraattoreita, jotka tuottavat valmista ohjelmakoodia suunnittelukaavioiden pohjalta. Pelkästä luokkakaaviosta koodin tuottaminen on taas kiistanalaisempaa, koska ohjelmoija joutuu panostamaan paljon luokan yksityiskohtien määrittelyyn työkalulle ja sen lisäksi vielä täydentämään ja muuttamaan tuotettua koodia. Tällöin työkalun konkreettinen hyöty voi olla kyseenalaista [16].

Ohjelman toteuttaminen mobiililaitteeseen voi erota työasemaohjelman toteuttamisesta. Tärkeimpänä syynä siihen on, että ohjelma mobiililaitteeseen testataan ensin kehitys-ympäristössä esimerkiksi PC:llä emulaattoria apuna käyttäen. Vasta kun ohjelma on todettu toimivaksi emulaattoriympäristössä, se siirretään oikealla mobiililaitteelle testausta varten (kuva 6). Emulaattorissa testattu ohjelma ja oikealle mobiililaitteelle siirretty ohjelma eivät välttämättä ole identtisiä, jos kohteiden perusrakenne ei ole samanlainen. Tällöin tarvitaan erilaisia käännösversioita ohjelmasta. Jos kuitenkin perusrakenne on samanlainen, kuten Java-ohjelman tapauksessa on, ei erilaisia käännöksiä tarvita, mikä helpottaa ohjelman testausta.

Prosessia, missä työasemaa käytetään ohjelman kääntämiseen toiselle laitteelle, kutsutaan ristiinkäännökseksi [18].

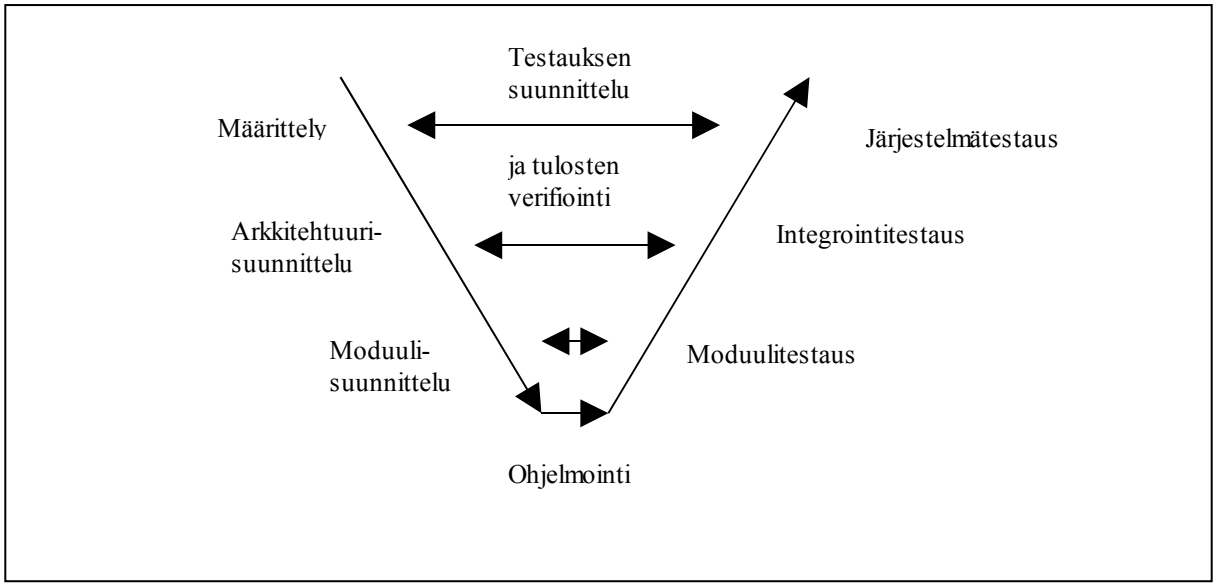


Kuva 6: Mobiiliohjelman toteutusprosessi

3.4 Testaus

Testauksen tarkoituksena on virheiden löytäminen ohjelmistosta. Testaus jaetaan yleensä yksikkö-, integrointi-, järjestelmä- ja hyväksymistesteihin. Yksikkötestaus suoritetaan toteutusvaiheessa ja siinä tarkistetaan yksittäisten luokkien tai luokkien kokonaisuuksien toiminta toteuttajan itsensä toimesta. Integrointitestissä testataan luokkien ja komponenttien yhteistoiminta. Järjestelmätestissä järjestelmä testataan mustana laatikkona, eli sen sisäiseen toimintaan ei oteta kantaa, vaan varmistetaan, että järjestelmä pääsee aina oikeaan lopputulokseen käyttäjän näkökulmasta katsottuna. Hyväksymistestaus kuuluu asiakkaan vastuulle, ja siinä asiakkaan tulee varmistaa, että järjestelmä täyttää sille asetetut vaatimukset [6].

Yleensä testaus tapahtuu monella eri tasolla V-mallin mukaisesti, kuten kuvassa 7 on ilmennetty. V-mallissa järjestelmätestaus tehdään vertaamalla valmista järjestelmää sen määrittelydokumentaatioon. Samalla tavalla integrointitestaus tehdään arkkitehtuurisuunnitelmaa vasten ja moduulitestaus moduulisuunnitelmaa vasten. Kukin näistä testeistä suunnitellaan samaan aikaan, kun kyseisen järjestelmän suunnitelmat tehdään. Tuotekehityksessä testaus, virheiden jäljittäminen ja korjaaminen vie merkittävän osan järjestelmän kokonaiskustannuksista [11].



Kuva 7: Testauksen V-malli

Testauksessa varmistetaan, että järjestelmä sisältää kaikki tarvittavat toiminnot ja että järjestelmä suorittaa toiminnot oikein. Testauksessa varmistetaan myös, että järjestelmä toimii hyvin kaikkien siihen liittyvien järjestelmien kanssa. Lisäksi varmistetaan, että kaikki laatuvaatimukset tulee täytettyä. Testausvaiheessa tehdään testaussuunnitelma, jossa kuvataan testauksen tyyppi, aikataulu, resurssit sekä komentojonot. Testaussuunnitelmassa kuvataan myös testiolosuhteet ja oletetut tulokset. Testausvaiheessa syntyy tuotoksina testitulokset, joista käy ilmi mitkä testit menivät läpi ja missä oli puutteita [20].

3.5 Ylläpito

Ylläpitovaihe tulee sen jälkeen, kun alkuperäinen ohjelmisto on jo toimitettu asiakkaalle. Ylläpidossa ratkotaan asiakkaan ongelmia, korjataan virheitä, muutetaan ohjelmaa uusien vaatimusten mukaisiksi sekä lisätään uusia ominaisuuksia. Ohjelmistotuotteiden kohdalla ylläpitovaihetta ei virallisesti ole ollenkaan, vaan edellä mainitut asiat toteutetaan kokonaan uudessa projektissa, jonka lopputuloksena on tuotteen seuraava versio. Asiakasprojekteissa ylläpito on taas käytössä ja erityisen tärkeä ja keskeinen vaihe [11].

3.6 Tukitoiminnot

Projektiin liittyy koko projektin elinkaaren kestäviä *tukitoimintoja*, joista tärkeimpiä ovat laadunvarmistus, tuotteenhallinta ja dokumentointi. Isoissa projekteissa tukitoimintoja voi olla enemmänkin ja vaatimustenhallinta ja riskienhallinta voi kuulua tukitoimintojen piiriin, kun pienemmissä projekteissa ne kuuluvat johonkin muuhun vaiheeseen [11].

Ohjelmiston laatu on yleensä subjektiivinen käsite ja riippuu käyttäjästä ja käyttöympäristöstä ja siitä, miten käyttäjän toiveet ja odotukset täyttyvät. Laadunvarmistuksen tarkoitus on estää virheiden pääseminen tuotteeseen ja auttaa löytämään virheet mahdollisimman aikaisessa vaiheessa, jolloin saavutetaan myös kustannussäästöä. Laadunvarmistusta voidaan tehdä testaamalla tai katselmoimalla suunnitteludokumentteja tai ohjelmakoodia. Katselmoinnin tarkoituksena on poistaa mahdollisimman paljon virheitä heti niiden synnyttyä niin, että seuraava vaihe voi alkaa puhtaalta pöydältä. Katselmointi on todettu parhaaksi tavaksi vähentää lopputuotteeseen jääviä virheitä ja puutteita [11].

Ohjelmistoprojektin aikana kirjataan paljon asioita erityyppisiin dokumentteihin, joita voi laajimmillaan olla kymmeniä erilaisia. Yleisimpiä dokumentteja ovat projektisuunnitelma, määrittelydokumentti, suunnitteludokumentit sekä testaussuunnitelma. Dokumentointi tulee tehdä mahdollisimman ylläpidettävään muotoon, ja aina kun ohjelmistoa muutetaan tulee myös dokumentointi päivittää ajan tasalle. Jos muutoksia ei kirjata dokumentteihin, muuttuvat dokumentit vähitellen hyödyttömiksi [11].

4 UML

UML tulee sanoista Unified Modeling Language eli tarkasti suomennettuna se tarkoittaa yhdistettyä mallinnuskieltä. Käytännössä se tarkoittaa visuaalista mallinnuskieltä, jolla voidaan suunnitella oliopohjaisten ohjelmien rakennetta korkealta tasolta aina alemmille tasoille saakka. Tässä tutkielmassa on käytetty UML:n versiota 1.4.2, mikä on saanut ISO standardin [32].

Minkä tahansa alan suunnittelussa on aina ollut tärkeää mallin käyttäminen tai mallinnus. Rakennettaessa tehdään aina piirustuksia, jotka hahmottavat sitä millainen kohteen tulisi olla tai miltä valmiin tuotteen tulisi näyttää. Suunnitelmien sisältämien tietojen perusteella voidaan suunnitella myös resurssien käyttäminen, kuten ajan, rahan sekä työvoiman käyttäminen [6].

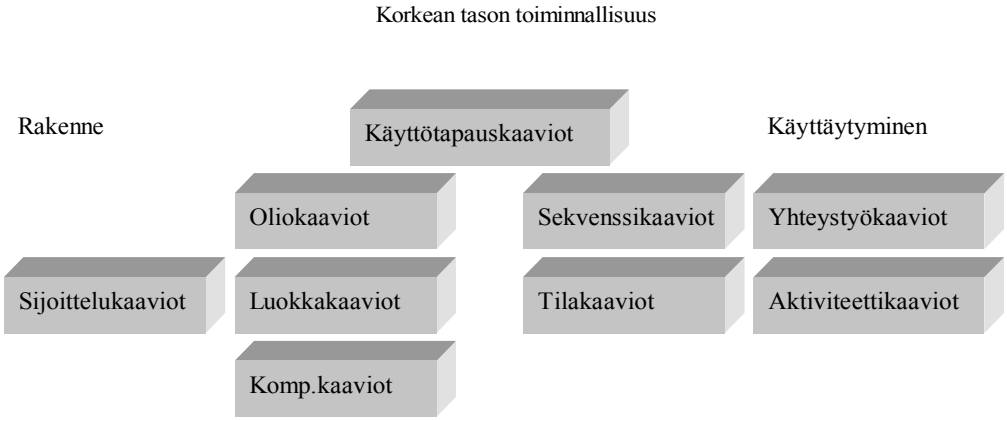
Mallit ovat yleensä visuaalisia kuvauksia, jolloin sisältö esitetään graafisilla symboleilla ja yhteyksillä. Tässä tapauksessa sanonta, että kuva kertoo enemmän kuin tuhat sanaa pitää hyvin paikkaansa. Kaikkeen ei kuitenkaan visuaalinen kuvantaminen ole paras vaihtoehto, koska jossain tapauksissa teksti on parempi vaihtoehto. Käyttökelpoinen malli onkin Eriksonin ja Magnusin [6] mukaan:

- tarkka, jolloin se kuvailee järjestelmää oikein,
- yhtenäinen, jolloin eri näkymät eivät kuvaa keskenään ristiriitaisia ominaisuuksia,
- helposti muille selitettävä,
- helposti muunneltavissa ja
- ymmärrettävä, jolloin se on mahdollisimman yksinkertainen olematta liian rajoittava.

UML -mallinnuskielenä on yritys ratkaista noita yllä kuvattuja ongelmia. UML on saavuttanut sellaisen suosion, että se on virallisesti ja käytännössä muodostunut mallinnuskielten standardiksi [6]. UML[2] on yleisesti käytetty mallinnusväline myös sovellusten kehittämisessä mobiililaitteisiin[3].

2 UML-mallinnuskielestä on olemassa laajennus, UML profiili, jota kutsutaan Mobiiliksi UML:ksi (Mobile UML). Mobiili UML tarjoaa uusia mallinnuselementtejä mobiilien systeemien määrittelemiseen, visualisointin ja dokumentointiin [2].

3 Mikkonen [19] käyttää mobiiliohjelmoinnin mallinnuksessa UML-mallinnusta. Samoin B'Far [1] käyttää UML-mallinnusta kuvatessaan ohjelmistoprosessia mobiilikehitykseen.

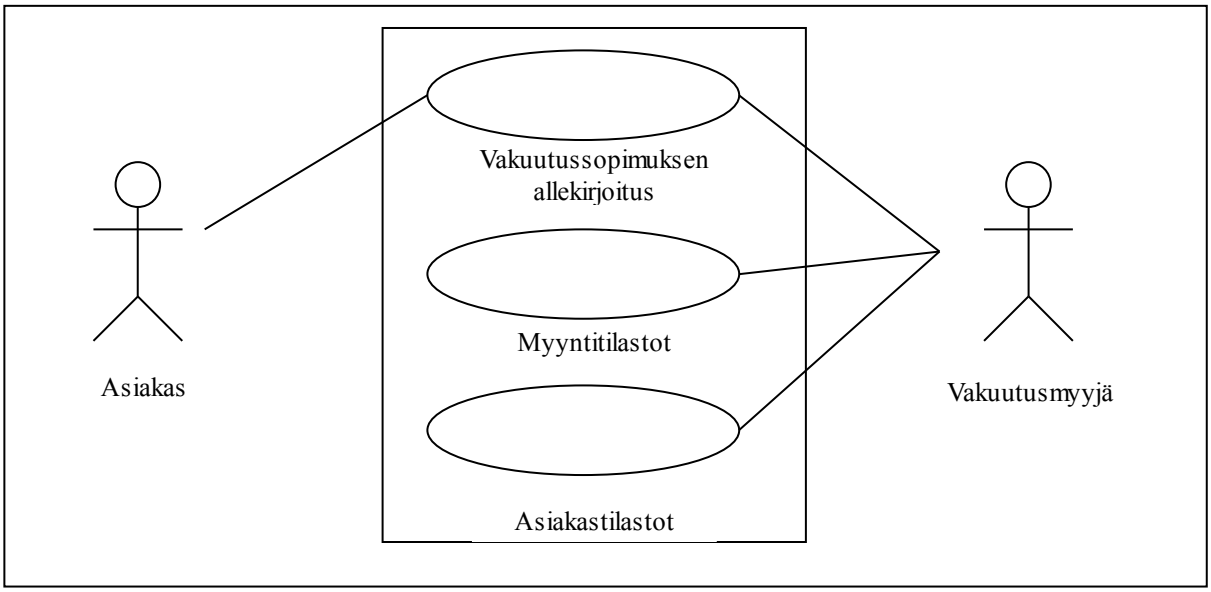


Kuva 8: UML:n kaaviotyypit [16]

Erilaisia *kaavioesityksiä* UML:ssä on yhdeksän. Näillä kaavioesityksillä voidaan kuvata järjestelmää korkealta tasolta (käyttötapauskaaviot), järjestelmän eri rakenteita (oliokaaviot, luokkakaaviot, sijoittelukaaviot, komponenttikaaviot) sekä järjestelmän dynaamista käyttäytymistä (sekvenssikaaviot, yhteistyökaaviot, tilakaaviot, aktiviteettikaaviot). Osa kaavioesityksistä voi olla päällekkäisiä, koska niissä voi esiintyä mm. samoja symboleita. Eri kaaviotyyppejä voidaan soveltaa eri ohjelmistotuotannon vaiheissa. Kaavioesityksiä on hahmoteltu kuvassa 8.

4.1 Korkean tason kaaviot

Korkean tason kaavioilla, kuten *käyttötapauskaaviolla*, pyritään hahmottamaan järjestelmä ulkopuolisen toimijan näkökulmasta. Ulkopuolinen toimija, joka voi olla ihminen tai toinen järjestelmä, kommunikoi järjestelmän kanssa jollakin tavalla [6]. Käyttötapauskaavio koostuu käyttötapauksista ja kuvaa järjestelmän käyttötapausten väliset suhteet, sekä suhteet ulkopuolisiin toimijoihin [16]. Käyttötapaus itsessään on kuvaus siitä, miten järjestelmää voidaan käyttää ulkopuolisen toimijan näkökulmasta. Käyttötapauksen tulisi tarjota kullekin toimijalle jotakin hyödyllistä. Jokainen käyttötapaus on kuvaus jostain järjestelmän toiminnosta. Käyttötapaukset on tarkoitus kuvata vain toimijan näkökulmasta, eikä niiden toteutukseen oteta vielä tässä vaiheessa kantaa. Käyttötapauksista muodostuvat järjestelmän toimintovaatimukset [6]. Käyttötapauksia voidaankin siis käyttää apuna vaatimusmäärittelyn tekemisessä.



Kuva 9: Vakuutuslaitoksen käyttötapauskaavio [6]

Kuvassa 9 on ilmennetty yksinkertaista vakuutuslaitoksen käyttötapauskaaviota. Toimijoina ovat Asiakas ja Vakuutusmyyjä, jotka molemmat ovat tekemisessä vakuutuslaitoksen toimintojen kanssa. Vakuutusmyyjä on yhteydessä kaikkiin järjestelmän käyttötapauksiin ja Asiakas vain yhteen eli Vakuutussopimuksen allekirjoitus –käyttötapaukseen. Asiakas ja Vakuutusmyyjä siis yhdessä allekirjoittavat sopimuksen. Vakuutusmyyjällä on pääsy tilastoihin asiakkaista sekä myyntitilastoihin.

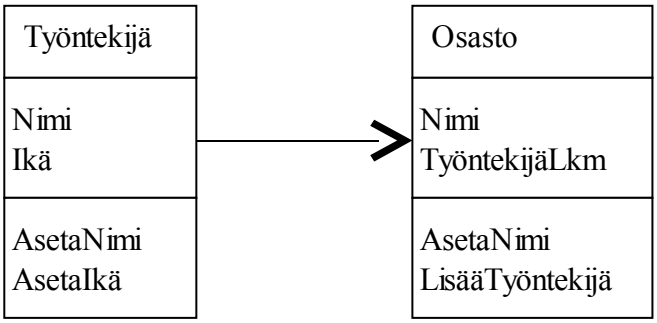
4.2 Rakenteelliset kaaviot

Rakenteellisia kaavioita käytetään järjestelmän staattisen rakenteen kuvaukseen. Staattisia kaavioita ovat luokkakaaviot, oliokaaviot, sijoittelukaaviot sekä komponenttikaaviot. Staattisuus tarkoittaa sitä, että kaavio on pätevä minä tahansa hetkenä. Olio-ohjelmien suunnittelussa luokat, oliot sekä niiden väliset suhteet ovat tärkeimmät mallinnuselementit. Nämä elementit on helppo muuttaa suoraan lähdekoodiksi [6].

Luokkakaaviolla kuvataan järjestelmän luokkaranne sekä niiden suhteet [16]. Luokat ovat järjestelmän käsittelemiä asioita ja ne voivat olla yhteydessä toisiinsa monilla eri tavoilla. Yhteyksiä on seuraavanlaisia [6]:

- Assosiaatio: luokkien välillä on jonkinlainen yhteys.
- Riippuvuus: toinen luokka tarvitsee tai käyttää toista luokkaa toimiakseen.

- Erikoistuminen: toinen luokka on toisen luokan erikoisversio.
- Pakkaus: luokat on ryhmitelty samaan kokonaisuuteen kuuluviksi.



Kuva 10: UML-Luokkakaavio

Yhteydet näytetään luokkakaaviossa luokkien attribuuttien ja operaatioiden kanssa, kuten kuvassa 10 on esitetty. Yksi luokkakaavion tehtävistä on määrittää perusta järjestelmän muita piirteitä kuvaaville kaavioille, kuten dynaamisille kaavioille, jotka kuvaavat olioiden tiloja ja olioiden välistä yhteistyötä. Ennen luokkakaavion luomista järjestelmän luokat on tunnistettava ja kuvattava. Luokkien etsimisessä voi käyttää apuna esimerkiksi vaatimusmäärittelyä tai liiketoiminta-analyysiä [6].

Oliokaaviot on tarkoitettu kuvaamaan olioita sekä niiden välisiä suhteita. Tavallaan oliokaavio on luokkakaavion mahdollinen ajonaikainen ilmentymä [16]. Oliokaavion ulkoasu on lähes kuin luokkakaaviolla, erotuksena että olioiden nimet alleviivataan ja kaikki yksittäiseen yhteyteen liittyvät ilmentymät näytetään. Oliokaavio ei ole yhtä tärkeä ja käytetty kuin luokkakaavio [6].

Sijoittelukaavioita käytetään kuvaamaan prosessointia suorittavia laiteyksiköitä, kuten tietokoneita sekä niiden yhteyksiä kuten tietoliikenneyhteyksiä. Lisäksi kuvataan ohjelmiston osien sijoittumista niihin. Sijoittelukaaviota tarvitaan vain siinä tapauksessa, jos järjestelmään liittyy useita laiteyksiköitä. Sijoittelukaavio on arkkitehtuuritason kuvaus, jossa järjestelmää on tarkasteltu korkealta abstraktiotasolta [16].

Komponenttikaavioita käytetään kuvaamaan ohjelmistokomponentteja sekä niiden välisiä suhteita. Komponentilla on UML:ssä monta merkitystä riippuen tapauksesta. Komponentti voi olla lähdekielinen tai binäärimuotoinen rajapinnallinen ohjelmayksikkö, datatiedosto, exe-

tai dll-tiedosto, kirjasto jne. Yleisesti mikä tahansa selkeästi erotettavissa oleva järjestelmän fyysinen osa voidaan mallintaa komponentiksi. Komponenttien välille voi liittyä mm. assosiaatioita, yleistyssuhteita, attribuutteja samalla tavalla kuten luokkiinkin [16]. Komponentti sisältää tiedot luokasta tai luokista, mitkä kyseiseen komponenttiin kuuluu. Komponentit voidaan ryhmitellä paketeiksi. Komponentin symboli UML:ssä on laatikko, jonka vasemmassa laidassa on kaksi pientä suorakulmiota. Komponentit voivat määritellä rajapintoja muiden komponenttien käytettäväksi. Komponentin rajapinta merkitään pienellä ympyrällä, joka on liitetty lyhyellä viivalla komponentin symboliin [6].

4.3 Käyttäytymiskaaviot

Käyttäytymiskaaviot kuvaavat käyttäytymistä eri olioiden välillä viestejä välittämällä. Käyttäytymiskaaviot kertovat dynaamiset lähtökohdat yhteistyöstä. Yhteistyö perustuu viestien välittämiseen jonkun tavoitteen aikaansaamiseksi. Seuraavissa alakohdissa on kuvattu eri käyttäytymiskaavioita, joita ovat sekvenssikaaviot, yhteistyökaaviot, tilakaaviot sekä aktiviteettikaaviot. Sekvenssikaavioita ja yhteistyökaavioita kutsutaan monesti yhteisesti vuorovaikutuskaavioiksi [16].

Sekvenssikaavio kuuluu käyttäytymiskaavioihin. Se koostuu johonkin vuorovaikutukseen liittyvistä olioista ja näiden olioiden välisistä viesteistä. Sekvenssikaavio voi kuvata olioiden vuorovaikutuksia liittyen johonkin tiettyyn käyttötapaukseen [16]. Sekvenssikaavioilla on pystyakselina aika-akseli ja sen vaaka-akseli ilmentää eri olioita. Sekvenssikaavio näyttää tietyn vuorovaikutuksen olioiden välillä, joka tapahtuu käytettäessä jotain järjestelmän tiettyä toimintoa. Sekvenssikaavion oliot kuvataan oliolaatikolla, jonka alapuolelle tulee niin sanottu elämänviiva, joka kuvaa olion elinkaarta ja suoritusta kyseisessä tilanteessa. Vaakasuorat viivat olioiden elämänviivojen välillä kuvaat olioiden välistä viestintää. Sekvenssikaavioita luetaan ylhäältä alaspäin, jolloin nähdään nuolia seuraamalla viestien välitys ajan kuluessa [6].

Yhteistyökaavioilla kuvataan myös olioiden välistä vuorovaikutusta. Erotuksena sekvenssikaavioon on se, että yhteistyökaaviossa mikään suunta ei edusta aikaa. Oliot kuvataan oliosymboleina ja olioiden välinen vuorovaikutus kuvataan olioiden välisillä viivoilla. Olioiden sijoittelulla voidaan ilmaista yhteenkuuluvuutta tai symmetrisyyttä. Olioiden välisiin viivoihin voidaan liittää operaation kutsun nimi, viestin suunta tai viestien järjestys numeroimalla ne. Numeroinnin monitasoisuudella voidaan kuvata sisäkkäisiä

operaatiokutsuja [16]. Sekvenssikaavioita ja yhteistyökaavioita voidaan käyttää monesti samassa tilanteessa. Jos ajan kuluminen ja viestisarja on tärkeää tuoda esille, kannattaa käyttää sekvenssikaaviota. Jos taas tärkeämpää on tuoda esille olioiden suhteet toisiinsa, kannattaa käyttää yhteistyökaaviota [6].

Yhteistyökaavio kuvataan piirtämällä oliokaavio, jossa olioiden välille on piirretty viestinuolet, jotka kertovat viestien suunnan. Viestinuoliin voidaan lisätä nimekkeet, jotka kertovat viestien lähetysjärjestyksen. Viestinuolet voivat kertoa myös muita arvoja, kuten palautusarvoja. Yhteistyökaavio alkaa viestillä, joka voi olla vaikka operaatiokutsu ja joka aloittaa koko sitä seuraavan vuorovaikutuksen [6].

Tilakaaviota käytetään monesti täydentämään jonkin luokan kuvausta. Tilakaavion avulla kuvataan kaikki luokan ilmentymän tilat, mihin se voi joutua. Lisäksi tilakaaviosta käy ilmi, mitkä tapahtumat aiheuttavat tilasiirtymiä. Tilakaaviossa on kuvattu eri tilat pyöristettyinä suorakaiteina ja niiden välisiä nuolia käytetään kuvaamaan eri siirtymiä. Siirtymään voi liittyä jokin tapahtuma, mikä määrittää, mitä siirtymän aikana tulisi tehdä [6]. Luokan ilmentymä on aina jossain tilakaavion tilassa ja luokka voi vaihtaa tilaansa vain siirtymänuolten osoittamaan uuteen tilaan. Tilakaaviossa voi olla alkutila, mistä suoritus alkaa sekä monia eri lopputiloja, minne sen suoritus voi päättyä. Tilakaavioita ei ole tarkoituksenmukaista piirtää kaikille luokille, vaan kannattaa käyttää vain sellaisia luokkia, joilla on useita selkeästi erottuvia eri tiloja ja tilasta toiseen siirtymisiä [6].

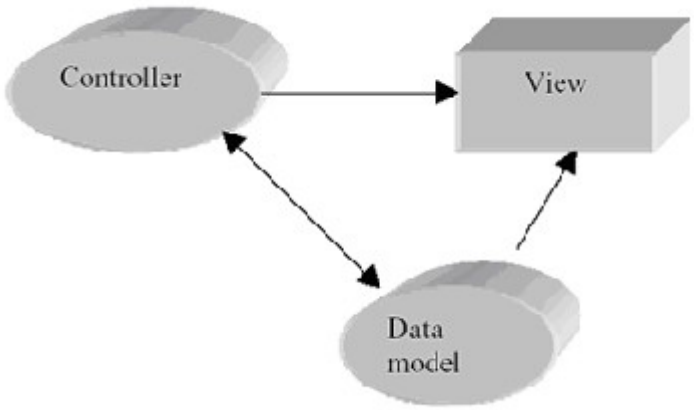
Aktiviteettikaavio on tilakaavion kaltainen. Erotuksena on se, että sitä käytetään kuvaamaan tietyn tehtävän sisäistä logiikkaa eikä olion reagoimista ulkoisiin tapahtumiin [16]. Tapahtumien kulku näytetään aikajärjestyksessä. Aktiviteettikaaviossa on eri tiloja, jotka sisältävät suoritettavan tapahtuman kuvauksen. Kun tapahtuma on suoritettu, siitä poistutaan toisin kuin tilakaaviossa, jossa tilasta poistutaan vasta sopivan viestin vastaanottamisen jälkeen. Lisäksi aktiviteettikaavion toiminnot voidaan sijoittaa uimaradoille, mitä tilakaaviossa ei voi tehdä. Uimaradoilla voidaan toimintoja ryhmitellä niiden sijainnin tai vastuuolion perusteella [6]. Aktiviteettikaaviosta käytetään joissain lähteissä myös nimitystä toimintokaavio.

5 Suunnittelumallit

Suunnittelumallit ovat tiedon esitysmenetelmiä, joiden avulla kokeneiden ohjelmisto-suunnittelijoiden tietämys on saatu helposti ymmärrettävään ja luettavaan muotoon. Suunnittelumallit tarjoavat tehokkaan keinon suunnitteluongelman tärkeimpien ongelmien ratkaisemiseen sekä kertovat kyseisen mallin käytöstä saavutettavan hyödyn [17]. Suunnittelumalli tulee sanoista Design Pattern. Jos tämä termi jaetaan kahteen osaan, voidaan malli (Pattern) määritellä seuraavasti [8]: ”Malli on idea, joka on ollut hyödyllinen kerran ja todennäköisesti tulee olemaan hyödyllinen myös useamman kerran”. Suunnittelumalleilla siis poistetaan useasti esiintyviä ongelmia tarjoamalla niille ratkaisumallit.

5.1 Yleistä

Ohjelmistoarkkitehtuuria voidaan havainnollistaa kuvan 11 avulla. Yleensä suunnittelu-mallien yhteydessä käyttöliittymän suunnitteluongelmat on jaettu kolmeen osaan: sisältö (data model), näkymä (view) ja kontrolleri (controller). Tämä jako edesauttaa ohjelman muunnel-tavuutta ja uudelleenkäytettävyys paranee [9]. Jokainen osista on olio, joilla on omat tapansa tiedon hallintaan ja käsittelyyn. Näkymän ja sisällön välillä tapahtuu tiedonvaihtoa ja kontrolleri toimii näkymän ja käyttäjän välillä [4].



Kuva 11: Mallin kuvaus [4]

Suunnittelumalli kuvaa jollekin tunnetulle ohjelmistosuunnittelussa vastaan tulleelle ongel-malle yleispätevän ratkaisun. Arkkitehti Christopher Alexander on sanonut: ”Jokainen

ratkaisumalli kuvaa ongelman, joka toistuu jatkuvasti ympäristössämme ja määrittelee ongelmalle ratkaisuperiaatteen, jota voidaan soveltaa miljoonia kertoja aina uudella tavalla" [9]. Vaikka tämä ajatus oli sanottu rakennusten suunnittelun yhteydessä, voidaan sitä soveltaa myös olio-ohjelmoinnin suunnittelumalleihin.

Suunnittelumalleista kuvataan yleensä neljä keskeistä osaa, joita ovat mallin nimi, ongelma, ratkaisu sekä seuraukset [9]. Mallin nimestä käy ilmi suunnittelun kohteena oleva ongelma, sen ratkaisu ja ratkaisun seuraamukset. Ongelma-osiosta saadaan selville mallin sovellus-kohteita, sekä ympäristö, missä ongelma yleensä esiintyy. Joskus ongelma voi sisältää listan ehtoja, joiden perusteella voidaan päätellä, onko mallin käyttö kyseisessä tapauksessa järkevää. Ratkaisu-osiossa kuvataan ongelman ratkaisuperiaate jollakin kuvausmenetelmällä. Siitä käy ilmi elementit, joista ratkaisu koostuu sekä niiden väliset suhteet ja keskinäinen vuorovaikutus. Seuraukset-osiossa analysoidaan ratkaisun hyviä ja huonoja puolia sekä vaihtoehtoisia ratkaisuja.

5.2 Suunnittelumallien käyttäytyminen

Olennaista suunnittelumallien käyttämisessä on huomata, että suunnittelumallit tarjoavat ongelmien ratkaisuksi sellaisia asioita, joita ei tule ensimmäiseksi mieleen. Perimistä käytettäessä tulisi ottaa huomioon rajapinnan suunnittelun tärkeys. Tämä tarkoittaa sitä, että tulisi käyttää enemmän abstrakteja luokkia, joilla on hyvin ja selkeästi määritelty rajapinta ja muodostaa tämän luokan avulla olioperheitä, joilla on identtiset rajapinnat. Kaikki abstraktista luokasta perityt luokat ainoastaan lisäisivät ominaisuuksia tai korjaisivat operaatioita, mutta eivät kätkisi mitään yliluokan operaatioita. Tällöin kaikki aliluokat voisivat ottaa vastaan abstraktille luokalle tulevat pyynnöt [9] .

Toinen tärkeä asia on käyttää mahdollisimman paljon luokkien perimisen sijasta olioiden välistä kommunikointia, jota kutsutaan koostamiseksi (object composition). Koostamisella tarkoitetaan uuden toiminnallisuuden saavuttamista olioita yhdistämällä, jolloin ei tarvitse tietää olioiden sisäisiä yksityiskohtia. Koostaminen auttaa pitämään luokat kapseloituna ja yhteen tehtävään keskittyneinä. Tämä taas vaikuttaa siihen, että luokkahierarkiat pysyvät pienempinä ja helposti hallittavina [9].

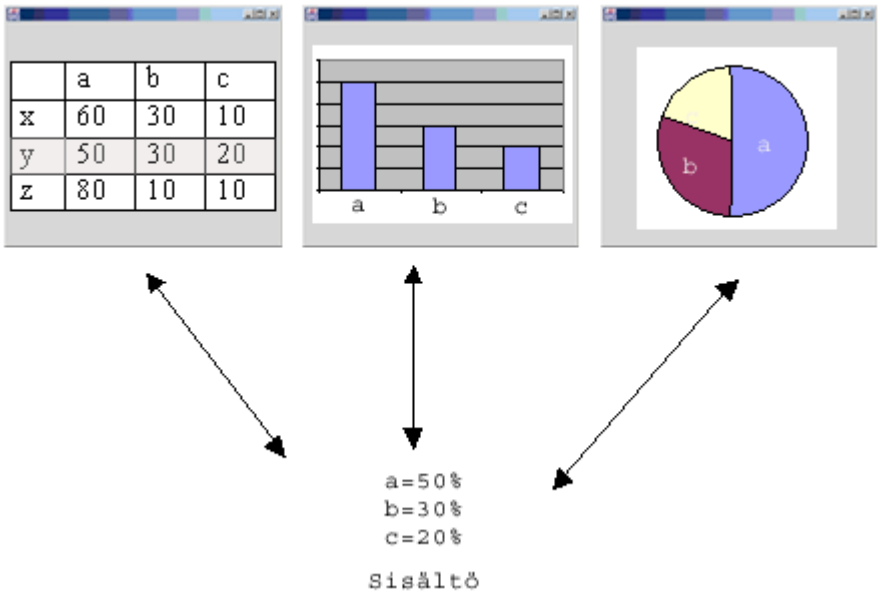
Molemmilla menetelmillä on hyvä ja huonot puolensa, mutta koostamisella saavutetaan monia etuja, joita ovat mm. ajonaikainen dynaamisuus. Tämä voi tarkoittaa sitä, että mikä tahansa olio voidaan korvata ajon aikana toisella oliolla, kunhan sillä on vain samanlainen

rajapinta [9]. Suunnittelumallit jaetaan kolmeen eri ryhmään niiden käyttötarkoituksen perusteella: luontimallit, rakennemallit ja käyttäytymismallit.

Luontimallit käsittelevät olioiden luontiprosesseja, rakennemallit liittyvät luokkien ja olioiden koostamiseen ja käyttäytymismallit käsittelevät tapoja, joilla luokat ja oliot ovat vuorovaikutuksessa ja joilla ne jakavat vastuita. Suunnittelumallien käyttöönotossa tuleekin tarkastella, mihin ryhmään suunnitteluongelman voisi rajata, ja sen perusteella lähteä etsimään sopivaa suunnittelumallia [9].

5.3 Esimerkkiongelma ja ratkaisu

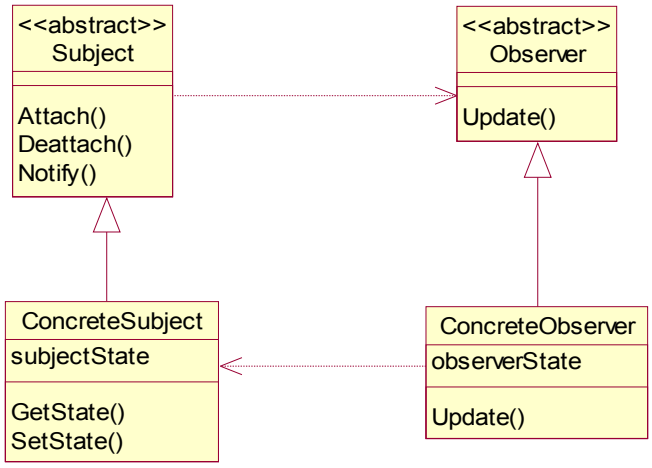
Suunnittelumalleilla voidaan ratkaista monenlaisia suunnitteluongelmia. Yksi esimerkkiongelma voi olla taulukkolaskentaohjelmassa esiintyvä tilanne, jossa käyttäjällä on taulukossa tietoa, ja samalla hänellä on auki graafisia esityksiä tämän tiedon pohjalta. Kun käyttäjä muuttaa jotain taulukossa olevaa arvoa tai muuta tietoa, päivittyy tieto sisältöön ja graafisten esitysten tulee tämän jälkeen pysyä ajan tasalla ja päivittyä vastaamaan uusinta tilannetta. Tilanne vastaa kuvan 11 tilannetta siltä osin, että graafiset esitykset ovat näkymiä (views) ja taulukosta tallennettava tieto on sisältöä (data model). Tilannetta voidaan hahmottaa tarkemmin kuvan 12 avulla [9].



Kuva 12: Taulukkolaskentaohjelman näkymät ja sisältö

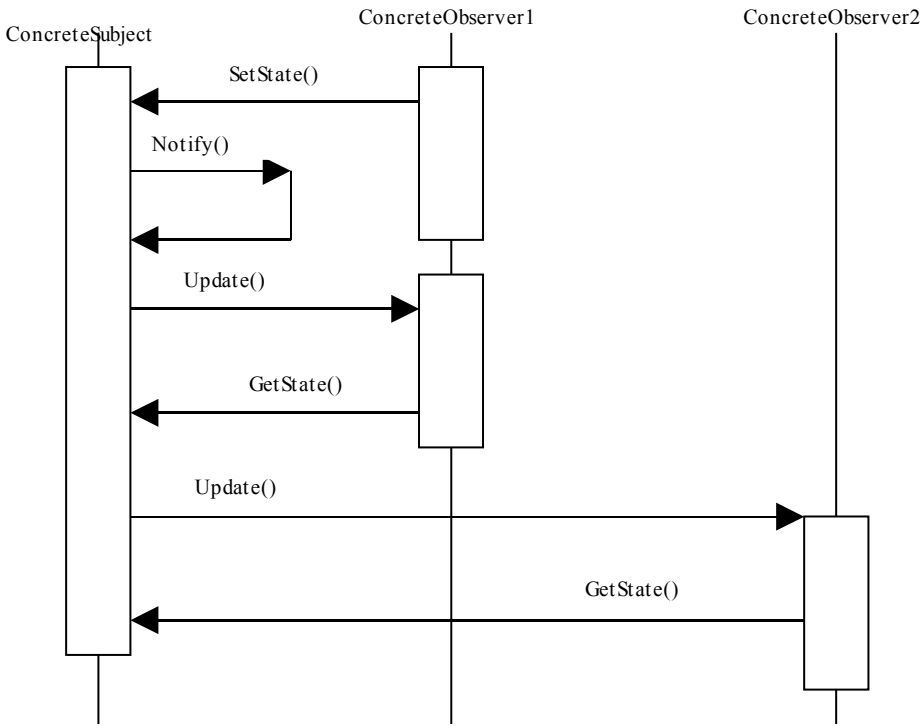
Ongelma liittyy olioiden käyttäytymiseen ja sen perusteella voidaan rajata sopivien suunnittelumallien kokoelmaa. Yksi ratkaisu tähän ongelmaan löytyy Tarkkailija-mallista, joka on yksi Gamman et al. [9] kuvaamista suunnittelumalleista. Tarkkailija-mallin tarkoitus on määritellä olioiden välille sellaiset riippuvuudet, että yhden olion tilan muuttuessa siitä riippuvat oliot saavat ilmoituksen tilanmuutoksesta ja päivittyvät automaattisesti.

Tarkkailija-mallin tärkeimmät oliot ovat subjekti (subject) ja tarkkailija (observer). Subjekti kuvastaa oliota, jonka tilaa tarkkaillaan ja siihen voi liittyä monia tarkkailija-oliota. Subjektin tilan muuttuessa se ilmoittaa kaikille tarkkailijoille tilan muutoksesta ja tällöin tarkkailijat käyvät hakemassa subjektilta ajan tasalla olevat tiedot ja päivättävät oman tilansa vastaavaksi. Subjektin ei tarvitse tietää, ketkä ovat sen tarkkailijoita. Kuvasta 13 käy ilmi Tarkkailija-mallin rakenne.



Kuva 13: Tarkkailija-mallin rakenne luokkakaaviona [9].

Subjekti tuntee tarkkailijansa, joita voi olla rajaton määrä. Lisäksi subjekti määrittelee rajapinnan, jolla tarkkailijoita kiinnitetään ja irrotetaan subjektiin. Tarkkailija määrittelee päivitysrajapinnan olioille, joiden tulee saada ilmoitus subjektin tilassa tapahtuneista muutoksista. ConcreteSubject-olio tallettaa tilatiedot, jotka kiinnostavat konkreettisia ConcreteObserver-olioita. ConcreteObserver-olio pitää yllä viitettä ConcreteSubject-olioon ja tallettaa tilatiedot. Lisäksi se toteuttaa Observer-päivitysrajapinnan ja käyttää sitä tilatietojensa päivittämisessä [9].



Kuva 14: Sekvenssikaavio [9].

Olioiden välinen yhteistyö on selkeästi määritelty Tarkkailija-mallissa. Tilanteen muuttuessa subjekti-olio ilmoittaa kaikille tarkkailijoille tilanmuutoksesta. Tällöin kukin tarkkailija käy ilmoituksen saamisen jälkeen hakemassa subjektilta sen tilatiedot ja päivittää omat tiedot vastaamaan näitä tietoja. Kuvasta 14 käy ilmi subjektin ja kahden tarkkailijan välinen vuorovaikutus ja yhteistyö. Subjektin tilan muutoksen aiheuttaa yksi tarkkailijoista, joka esimerkkiongelman mukaan voisi olla vaikka kuvan 12 taulukkonäkymä. Tarkkailija kutsuu subjektin SetState-operaatiota ja muuttaa näin subjektin tilaa. Kun tilanmuutos on tehty, päivittää subjekti tarkkailijoille tiedon tilanmuutoksesta Notify-operaatiolla, joka taas kutsuu kaikkien tarkkailijoiden Update-operaatiota. Tarkkailijoita esimerkkiongelmassa ovat pylväs-diagrammi ja ympyrädiagrammi. Jokainen tarkkailija käy sen jälkeen hakemassa uuden tilanteen GetState-operaatiolla [9].

6 Avoin lähdekoodi

Tässä luvussa kuvataan, mitä on *avoin lähdekoodi* ja tutkitaan hiukan avoimen lähdekoodin historiaa. Luvussa käydään läpi myös avoimen lähdekoodin lisensseihin liittyviä perustietoja.

6.1 Avoimen lähdekoodin määritelmä

Avoimen lähdekoodin ohjelma on ohjelma, jonka mukana tulee ohjelman lähdekoodi tai lähdekoodi on helposti saatavilla esimerkiksi internetistä lataamalla. Lähdekoodi ei saa olla käännettynä, puutteellisena tai muuten sellaisessa muodossa, että ohjelmoija ei voisi sen avulla muuttaa tai kehittää ohjelmaa edelleen. Ohjelman lisenssi ei rajoita muita koodin levittämisessä tai muokkaamisessa ja muokatun koodin mukaan periytyy sama lisenssi, kuin alkuperäisen ohjelman mukana on ollut [13].

Hyviä esimerkkejä avoimen lähdekoodin ohjelmista ovat esimerkiksi Apachen web-palvelin, jolla on suurin osuus WWW:n palvelimista ja määrä on koko ajan kasvussa. Linux -käyttöjärjestelmää on käytetty miljoonissa palvelimissa, mikä ilmentää sitä, että ei ole olemassa ohjelmaa tai projektia, mikä olisi liian vaikea toteuttaa avoimen lähdekoodin periaatteella. Näitä ohjelmia ovat käyttäneet miljoonat ihmiset useiden vuosien ajan ja ne ovat olleet luotettavia ja pitäneet käyttäjät tyytyväisinä [13].

Avoin lähdekoodi -termiä ei virallisesti omista tai kontrolloi kukaan Yhdysvaltojen lakien mukaan. Yleisesti kuitenkin avoimen lähdekoodin ohjelma on ohjelma, joka on tuotettu avoimen lähdekoodin määritelmän (Open Source Definition, OSD) mukaan [7]. Tätä määritelmää ylläpitää Open Source Initiative (OSI) -organisaatio, joka edistää avoimen lähdekoodin ohjelmistojen käyttöä [44]. Lisää tietoa avoimen lähdekoodin määritelmästä ja OSI -organisaatiosta on luettavissa OSI:n internetsivuilta [35].

On tärkeää huomata, että avoimen lähdekoodin määritelmä ei itsessään ole lisenssi, vaan ennemminkin se on vain määritelmä, minkä avulla ohjelman jakelusääntöjä (Terms of Distribution) voidaan mitata. Jakelusäännöt sisältävät sekä lisenssin että tiedot, miten ohjelmaa saa levittää eteenpäin. Jos ohjelman jakelusäännöt ovat avoimen lähdekoodin määritelmän mukaiset, ohjelman voidaan sanoa olevan avoimen lähdekoodin ohjelma [7].

Avoimen lähdekoodin ohjelmien etuina ovat edullisuus; ohjelmat ovat yleensä ilmaisia tai ovat saatavilla ilmaiseksi esimerkiksi internetistä ladattuna. Tämä on etu yritykselle, mutta

vielä tärkeämpää on, että ohjelmaa voi muuttaa ja tehdä ohjelmaan parannuksia ja jakaa muutettua versiota yrityksen sisällä ilman esteitä. Avoimen lähdekoodin ohjelmat ovat yleensä myös laadukkaita, koska niitä kehitetään koko ajan toteuttajien toimesta ja parannukset ja uudet ominaisuudet tulevat nopeasti saataville. Jotkut avoimen lähdekoodin ohjelmat ovat oman alansa standardeja ja monet työkalut perustuvat niiden pohjalle [12].

Tyypillisen avoimen lähdekoodin periaatteella tehdyn ohjelman projekti käyttää jotakin internetsivua säilytyspaikkanaan, missä koodi, dokumentit, keskustelut, suunnittelu-dokumentit, bugi- ja asiaraportit ja muu projektiin liittyvä materiaali säilytetään. Joku projektiin osallistuva henkilö jakaa käyttöoikeuksia muille projektiin osallistuville henkilöille tai ryhmille, että hekin voivat tehdä muutoksia koodiin versionhallinnan avulla. Yleensä sivuistoilla on eri versioita ohjelmista valmiiksi käännettynä erilaisille alustoille, joita kuka tahansa voi ladata ja kokeilla [10]. Hyvä esimerkki tällaisesta sivustosta on SourceForge -internetsivusto, mikä on internetin suurin tällainen sivusto. Sivusto isännöi yli sataa tuhatta avoimen lähdekoodin projektia ja sille on rekisteröitynyt yli miljoona käyttäjää, jotka ylläpitävät projekteja, keskusteluja ja koodeja [37].

6.2 Avoimen lähdekoodin historiaa

Ennen vuotta 1970 ohjelmat olivat yleensä oletusarvoisesti vapaita tai ilmaisia. Niitä oltiin kehitetty tieteellisissä tutkimusryhmissä, joissa tietoa yleensä jaettiin vapaasti tutkijoiden kesken. Ohjelmat tehtiin jonkun ongelman ratkaisemiseksi ja leviteltiin sitten vapaasti sellaisille tahoille, joilla oli sama ongelma. Ohjelmat toimivat yleensä vain yhden toimittajan järjestelmissä, joten hyvä ohjelma auttoi monesti myymään koko alustan [13].

Ensimmäiset ohjelmistoyritykset aloittelivat 1960-luvun lopuilla ja 1970-luvulla nostettiin kanne IBM:ää vastaan tuotteiden niputtamisesta. Tämän jälkeen IBM avasi ohjelmiensa niputusta käyttöjärjestelmää lukuunottamatta ja sopi oikeusjutun. Näihin aikoihin useiden itsenäisten ohjelmistoyritysten liiketoiminnasta tuli menestyvää, kun he alkoivat tarjoamaan kehitystyökaluja, tietokantoja sekä tuotannon ja kirjanpidon apuohjelmia teollisuuden, pankkien ja vakuutustoiminnan käyttöön [13].

Uusi ajatus ohjelmien sulkeutuneisuudesta alkoi levitä 1970-luvulla. Vuonna 1976 Microsoftin Bill Gates kirjoitti Avoimen Kirjeen Harrastelijoille (Open Letter to Hobbyist), jossa hän toteaa, että ”ohjelma ei ole yhteistä hyvää, vaan yksityistä omaisuutta”. Vuosikymmenen lopussa muun muassa Microsoftin ja Oraclen vaikuttajat keskustelivat

suljetun koodin tuotteista ja ohjelmistojen kehittämisen tärkeydestä, joka on eroteltuna laitteistojen kehittämisestä. Ei ole siis sattumaa, että muun muuassa Microsoft ja Oracle saivat rakennettua vahvan pohjan toiminnalleen ja niistä tuli suurimpia yrityksiä ohjelmistojen tuotannon saralla [13].

Yksi tärkeä etappi avoimen lähdekoodin ohjelmistojen historiassa oli Free Software Foundation (FSF) -säätiön perustaminen vuonna 1985. Säätiön perustaja Richard Stallman käytti työssään Xeroxin printteriä ja pyysi valmistajalta lähdekoodia, että olisi voinut etsiä ja korjata printterissä esiintyviä virheitä. Valmistaja kuitenkin kieltäytyi toimittamasta lähdekoodia ja Stallman päätyi siihen ajatukseen, että ohjelmistojen tulisi olla vapaita ja perusti ajatukseen pohjautuvan säätiön. Hän ennakoi, että suljettujen ohjelmien pohjalta syntyisi helposti monopoleja. Stallman irtisanoutui työstään ja pyhitti aikansa kehitelläkseen sarjan ilmaisia ohjelmistotuotteita, joita hän kutsui GNU -perheeksi. Stallman määritti ”Free Software” -termin, joista myöhemmin muovaantui sekaannusten välttämiseksi Open Source -termi. Free Software Foundation -säätiö oli vastuussa useista tunnetuista ja laajalle levinneistä hyötyohjelmista, mitkä toimivat myöhemmin muiden avoimen lähdekoodin ohjelmistojen ytimenä, kuten GNU C -kääntäjä (GCC) ja Emacs -tekstieditori [7].

Vuonna 1991 Helsinkiläinen 21-vuotias yliopisto-opiskelija Linus Torvalds aloitti Linux -käyttöjärjestelmän kehittämisen. Torvalds perusti järjestelmänsä Unix -käyttöjärjestelmän klooniin Minixiin, joka oli saatavilla ohjelmakoodin muodossa. Muutoksien tekeminen Minixiin vaati järjestelmän kehittäjän Andrew Tanenbaumin hyväksynnän. Torvaldsin tavoitteena oli luoda Unixin tyylinen käyttöjärjestelmä ja hän etsi apua projektilleen keskustelupalstoilta. Projektin sai paljon tukea maailmanlaajuisesti ja arvioilta tuhansia kehittäjiä osallistui yksinään Linuxin kernelin toteutukseen, kun samalla koko käyttöjärjestelmän kehittämisyhteisö oli arvioilta 40 000 kehittäjän suuruinen ja joidenkin lähteiden mukaan jopa satojen tuhansien kehittäjien suuruinen. Torvalds onkin vakuuttanut, että kyseessä on ollut suurin yhteistyöprojekti maailman historiassa [7].

Seuraava etappi avoimen lähdekoodin historiassa oli Apachen Web -palvelimen kehittäminen, mikä alkoi vuonna 1995 vapaaehtoisvoimin. Ryhmä vapaaehtoisia halusi yhdistää kokemuksensa mieluummin kuin työskennellä erillään ja tuplata työmääränsä keksimällä pyörää uudelleen. Apachen ydinryhmässä on lukuisia jäseniä ja kaikki palvelimeen tulevat muutokset päätetään ryhmän äänestyksillä ja työskentelymenetelmät ovat tarkasti määritelty ja virallistettu [7].

Yksi tärkeä avoimen lähdekoodin historian aloitteista oli Mozilla Web -selaimen kehitystyö. Projektin vaikuttavuutena oli median tietoisuus projektista ja yritysten tuki. Projekti sai alkunsa vuonna 1998, kun Netscape julkisti selaimensa lähdekoodin ja nimesi projetin Mozillaksi. Osasyynä julkistamiseen oli markkinosuuksien häviäminen Microsoftin Internet Explorer -selaimelle ja kova kilpailu sen kanssa [7].

6.3 Lisenssit

Ohjelmistojen lisensiointi on osa ohjelmistojen kehitysprosessia. Avoimen lähdekoodin ohjelmistojen lisensiointi ei käytännössä eroa lisensioinnista yleensä. *Lisenssit* avoimen lähdekoodin ohjelmille jaetaan yleensä ilmaisiin (”free”) tai vastavuoroisiin lisensseihin ja avoimiin (”open”) lisensseihin. Esimerkki ilmaisista lisensseistä on GNU General Public License (GPL) ja avoimista lisensseistä Berkeley Software Distribution -lisenssi (BSD) ja Apache -lisenssit [13].

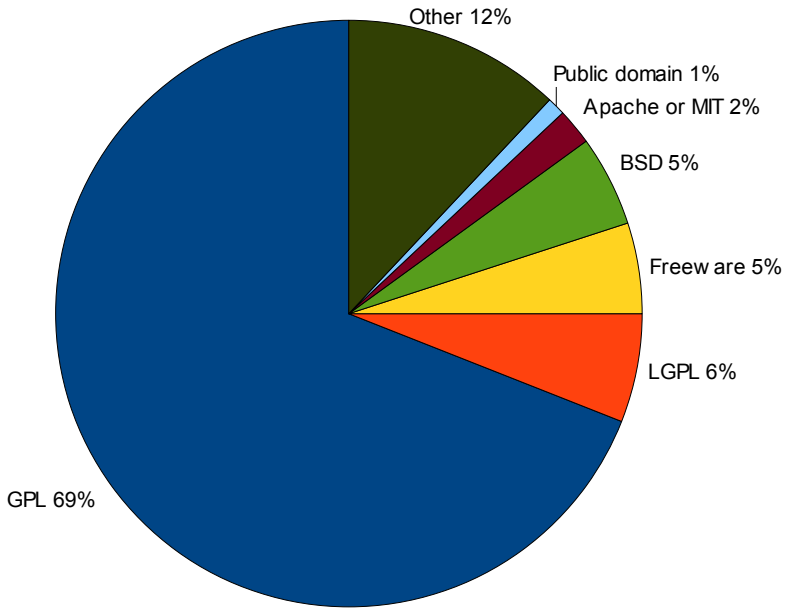
Lisenssin tarkoitus on myöntää ulkopuolisille toteuttajille tietyt oikeudet siitä, mitä he voivat tehdä ohjelmakoodille ja toisaalta taas mitä he eivät voi tehdä ohjelmakoodin kanssa. Suositelluimmat lisenssit antavat luvan käyttää ja muuttaa ohjelmakoodia. Jotkut lisenssit antavat myös oikeuden käyttää patentteja, joita ohjelmakoodi edustaa. Useimmat lisenssit eivät salli kuitenkaan esimerkiksi ottaa tuota patenttia käyttöön ja käyttää sitä jossain toisessa sovelluksessa [10].

Lisenssit antavat toteuttajille myös tiettyjä velvollisuuksia. Jotkut lisenssit esimerkiksi velvoittavat, että kaikki bugikorjaukset, jotka kehittäjä tekee ohjelmistokoodiin, tulee lähettää myös alkuperäiselle ohjelmakoodin tekijälle. Toinen yleinen vaatimus on, että esimerkiksi joku patentti, jota on käytetty ohjelmistokoodissa, pitää olla tarjolla myös muille kehittäjille jotka käyttävät tuota koodia. Usein tuo oikeus pitää olla tarjolla ilman maksua [10].

Ilmaiset lisenssit sisältävät yleensä velvollisuuden, että ohjelmakoodin pohjalta tehty uusi ohjelmakoodi, tai ohjelmakoodi, jossa käytetään avoimen lähdekoodin ohjelmakoodia osana järjestelmää, pitää olla myös avoimen lähdekoodin mukaista ohjelmakoodia. Tämä on vastavuoroisuutta juuri sen takia, että jos vaikka GPL -lisenssin mukaista Linuxia muokataan tai kehitetään uusilla ominaisuuksilla ja jaetaan käyttöjärjestelmää edelleen, koko järjestelmän ohjelmakoodi pitää myös jakaa edelleen [13].

Toinen lisenssi, GNU Lesser General Public License (LGPL), muuttaa tätä vastavuoroisuusominaisuutta lähinnä ohjelmointikielten kirjastojen kohdalta. Jos kirjasto käyttää LGPL -lisenssiä ja joku toinen kehittää tätä kirjastoa edelleen, sen tulee olla myös LGPL -lisenssin mukainen ja ohjelmakoodi on laitettava muiden saataville. Jos joku kehittäjä taas pelkästään käyttää tämän kirjaston funktioita eli linkittää ohjelmansa kirjastoon, eikä suoranaisesti peri kirjaston ohjelmakoodia, tämän ohjelman ei tarvitse enää olla LGPL -lisenssin mukainen [7].

Avoimet lisenssit eivät sisällä vastavuoroisuusvelvoitetta, joten ne sallivat, että avoimen lähdekoodin ohjelmasta, jota muutetaan ja kehitetään edelleen, voi tulla niin sanottu suljettu ohjelma. Tämä tarkoittaa esimerkiksi sitä, että Apple käyttää FreeBSD -koodia osana Mac OS X -käyttöjärjestelmää ilman, että sen tulisi levittää tai jakaa Max OS X -käyttöjärjestelmän lähdekoodia edelleen [13]. FreeBSD on vapaa BSD-Unixiin perustuva käyttöjärjestelmä [25].



Kuva 15: Projektien käyttämät lisenssit [13].

Avoimen lähdekoodin lisensseistä yli kaksi kolmasosaa on GPL -lisenssejä, noin yksi kuudesosa käyttää LGPL, BSD, Apache, Mozilla tai Massachusetts Institute of Technology (MIT) -lisenssejä. Kuva 15 ilmentää lisenssien jakautumaa projektien välillä. Yksi alkuperäisistä vapaan ohjelmistokoodin lisensseistä on GPL -lisenssi. Linux ja monet muut ydintyö-

kalut käyttävät juuri tätä lisenssiä. Mozilla Public Lisenssi on samansuuntainen kuin GPL, mutta se yksinkertaistaa sääntöjä liittyen vapaaseen käyttöön tulevaisuudessa. Loput lisenssistä eivät ole vastavuoroisia, kuten esimerkiksi LGPL, BSD, Apache ja MIT. Nämä lisenssit ovat jakelijoille vähemmän rajoittavia kuin GPL, koska seuraava käyttäjä voi käyttää, muokata ja uudelleen jakaa koodia, ilman että he julkaisisivat omaa koodiansa. Tämä rajoittavuuden puute estää seuraavaa ohjelman käyttäjää näkemästä ohjelmakoodia [13].

7 Avoimen lähdekoodin kehitys- ja tukityökalut

Tässä luvussa perehdytään avoimen lähdekoodin *kehitystyökaluihin*, joita voidaan käyttää mobiilien ohjelmien kehittämisen apuna. Käsittelyssä ovat myös kehityksen apuvälineet ja kehitystä tukevat avoimen lähdekoodin työkalut, kuten toimistotyökalu OpenOffice. Työkaluista kuvataan perustietoja ja ominaisuuksia ja myöhemmin tässä tutkielmassa tarkastelen ohjelmia käytännössä, miten ne soveltuvat mobiilien ohjelmistojen kehityksen apuvälineiksi.

7.1 Kehitysympäristöt

Mobiilien ohjelmistojen kehittäminen tapahtuu monesti jotain *kehitysympäristöä* tai kehitystyökalua käyttäen. Kehitysympäristöihin on monesti integroitu tai saatavilla *lisäosina* erilaisia lisätyökaluja, kuten suunnittelu- tai käännöstyökaluja. Kehitystyökaluja on olemassa myös avoimen lähdekoodin periaattella toteutettuja ja niitä kehitysympäristöjä tarkastellaan lähemmin tässä kohdassa. Kehitysympäristöistä kerrotaan esittely ja ominaisuuksia, tietoja laajennettavuudesta, sopivuus sovellusten kehittämisestä mobiililaitteisiin sekä hiukan historiaa.

7.1.1 Eclipse

Eclipse on avoimen lähdekoodin -yhteisö, jonka projektit keskittyvät rakentamaan avoimia kehitysalustoja koostuen laajennettavista kehitysympäristöistä, työkaluista ja ajoympäristöistä, joita voi käyttää projektin ylläpitoon ja apuna koko projektin elinkaaren ajan [23]. Eclipse on myös IDE (Integrated Development Environment), joka voidaan ajatella paikkana, missä eri työkalut voivat asua sulassa sovussa yhdessä. Se yhdistää mallinnus-, suunnittelu-, ohjelmointi- ja testityökalut yhteen paikkaan. Tämän on tarkoitus helpottaa toteuttajan työtä. Eclipsen avulla projektin jäsenet voivat työskennellä yhdessä ja luoda, muuttaa ja hallita kokonaisuutta. Projektin jäsenillä on pääsy toistensa tuotoksiin erilaisilla versionhallinnan apuvälineillä, joita voi integroida Eclipseen. Eclipseen perustuvat työkalut toimivat keskenään yhdenmukaisella tavalla [5].

Eclipse voidaan nähdä monella eri tavalla. Jotkut näkevät Eclipsen normaalina Java IDE:nä, jossa on normaalit ominaisuudet kuten editori, debuggeri ja projektinluontityökalu. Toiset näkevät Eclipsen yleisenä kehitysympäristönä, johon voi lisätä työkaluja plug-in API:n avulla. Eclipsen kehityksen jatkuessa jotkut näkevät Eclipsen suunnittelutyökaluna, yrityspro-

sessien suunnittelijana, kehitysympäristönä sulautetuille C++ -kielellä tehdyille järjestelmille tai vaikka HTML -dokumentin hallintatyökaluna. Kun käytetään Rich Client Platform (RCP) -kokoonpanoa, Eclipse voi olla alustana mille tahansa loppukäyttäjän asiakasohjelmalle [3].

Eclipse tarjoaa arkkitehtuurin ja kehykset, jotka on tarkoitettu helpottamaan kehittäjiä tekemään lisää integroitavia työkaluja Eclipseen. Eclipseä voi käyttää lukemattomien olemassaolevien työkalujen integroimiseen ja myöhemmin kokemuksen karttuessa, sitä voi käyttää uusien työkalujen luomiseen siihen. Kehitysympäristö tarjoaa myös yhtenäisen tavan käyttöliittymän tekemiseen ja funktioiden järjestämiseen ja paketoimiseen. Kehittäjän on mahdollistä käyttää Java Development Tools (JDT) -pakettia uusien Java -työkalujen kehittämiseen [5]. JDT -paketti tarjoaa työkalulisäosia, mitkä mahdollistavat minkä tahansa Java-ohjelman kehittämisen ja tällöin sen avulla voi kehittää myös Eclipsen lisäosia. JDT mahdollistaa, että Eclipseä voi käyttää Eclipsen itsensä kehittämiseen [23]. Eclipse on kehitetty juuri tällä tavalla suuressa, hajautetussa kehittäjien joukossa, jotka kääntävät ja testaavat toistuvasti koko projektin joka ilta. Lopputuloksena Eclipse sopii mihin tahansa projekteihin, jotka käyttävät samanlaisia työskentelymenetelmiä [3].

Eclipseen on saatavilla työkalu, joka helpottaa mobiilien ohjelmien tekemistä. EclipseMe on Eclipsen lisäosa, joka auttaa kehittämään J2ME MIDlettejä esimerkiksi kännyköihin. EclipseMe yhdistää erilaiset Wireless Toolkit -työkalut Eclipsen kehitysympäristöön, joka mahdollistaa sen, että kehittäjä voi keskittyä ohjelman tekemiseen eikä kehittäjän tarvitse huolehtia J2ME -kehityksen erikoistarpeista [24].

Eclipse syntyi vuonna 2001 kun IBM ja seitsemän muuta yritystä käynnistivät Eclipsen avoimen lähdekoodin projektin. Perustamisen jälkeen Eclipse on ylittänyt perustajiensa odotukset ja sitä on ladattu kymmeniä miljoonia kertoja. Eclipse.org sponsoroi omilla sivustoillaan useita projekteja, jotka tukeutuvat Eclipsen alustaan, ja useita avoimen lähdekoodin työkalu- ja teknologiaprojekteja [5].

7.1.2 NetBeans

NetBeans on avoimen lähdekoodin yhteisö, joka ylläpitää kehitystyökalua, mitä ohjelmisto-kehittäjät tarvitsevat luodakseen alustariippumattomia Java -sovelluksia. NetBeans toimii useissa käyttöjärjestelmissä, joita ovat Windows, Linux, MaxOS ja Solaris. NetBeans -alusta on yleinen työpöytäsovellus, joilla on yleiset ominaisuudet, kuten valikot, dokumenttien-hallinta, asetukset ja niin edelleen. Sen sijaan, että kehittäjät kirjoittaisivat samaa koodia

uudelleen ja uudelleen, he voivat luoda tarvitsemansa moduulin ja niputtaa sen NetBeans:iin, jolloin heillä on alustariippumaton sovellus [31].

NetBeansiin pohjautuvat sovellukset ovat alustariippumattomia, eli ohjelmakoodi kirjoitetaan kerran ja sen jälkeen se voidaan suorittaa missä tahansa. Siinä on ilmaiset komponentit, joita uudelleenkäyttämällä ja sekoittamalla on mahdollista ratkaista yleisiä ongelmia. NetBeans -alustaa ja -moduuleja käyttämällä on mahdollista tehdä erilaisia ohjelmia, jotka jakavat saman perustan ja logiikan. NetBeans -alustaan perustuvat ohjelmat voivat asentaa moduuleja dynaamisesti, joten käyttäjän ei tarvitse enää ladata koko ohjelmaa uudestaan saadakseen päivityksen tai uuden version ohjelmasta [31].

Ohjelma voidaan koota jo olemassaolevista moduuleista ja näin on mahdollista hyödyntää aikaisemmin muiden tekemää avoimen lähdekoodin työtä. On olemassa paljon käyttö-kelpoisia NetBeans -yhteisön tekemiä moduuleita, jotka ovat valmiina käyttöön [31].

NetBeans alustaa voi käyttää myös ohjelmien tekemiseen mobiileihin laitteisiin. Alustaa voi laajentaa asentamalla NetBeans Mobility Pack -laajennuksen. Laajennuksen avulla on mahdollista tehdä ohjelmia laitteisiin, jotka tukevat Connected Limited Device Configuration (CLDC) 1.0 tai 1.1 versiota ja Mobile Information Device Profile (MIDP) 1.0 tai 2.0 versiota. Laajennus tukee muun muassa SVG -grafiikkaa (JSR-226), JMUnit -testausta, MIDletin signausta, sertifikaattien hallintaa, over-the-air (OTA) -tekniikan emulointia ja Wireless Messaging ja Multimedia API:a. Laajennuksen avulla on mahdollista tehdä ohjelmia myös kommunikaattorityyppisiin laitteisiin, kuten Sony Ericsson UIQ, Ricoh, SavaJe tai Nokia S80 tai PDA -tyyppisiin laitteisiin [31].

NetBeans Mobility Pack -laajennus mahdollistaa ohjelmien porttauksen eri laitteiden välille. Sen avulla on mahdollista lisätä ja suorittaa laitekohtaista koodia ja testata tunnetuimpien eri laitetyyppien emulaattoreiden avulla. Debuggaaminen suoraan laitteessa on myös mahdol-lista. Laajennuksen mukana tulee Visual Mobile Designer -apuväline, millä saa tehtyä nopeasti tuotteita tai prototyyppejä. Sillä on mahdollista lisätä drag and drop -tekniikalla yksinkertaisia komponentteja, kuten splash -näyttöjä tai taulukoita. Sen avulla on mahdollista myös tehdä käyttöliittymä eri kielille [31].

NetBeans projekti sai alkunsa vuonna 1996 Tsekissä ja tarkoituksena oli luoda Delphin tyylinen Java IDE. Projekti sai kiinnostusta valmistuneiden opiskelijoiden keskuudessa ja sen ympärille muodostettiin yritys kaupallistamistarkoituksessa. Sun kiinnostui projektista

vuonna 1999 ja osti yrityksen loppuvuodesta. Puolen vuoden kuluttua kesäkuussa vuonna 2000 Sun Microsystems teki NetBeansista avoimen lähdekoodin periaatteen mukaisen ja jäi projektin pääsponsoriksi. NetBeansin perustuotteet ovat NetBeans IDE ja NetBeans Platform. Molemmat ovat ilmaisia ja ohjelmakoodi on kaikkien saatavilla. NetBeans IDE:ä on ladattu yli 12 miljoonaa kertaa ja yli 500 000 kehittäjää on osallistunut netbeans.org projektiin [31].

7.2 Suunnittelun apuohjelmat

Avoimen lähdekoodin mukaisia suunnittelutyökaluja on olemassa joko irrallisina ohjelmina tai lisäosina, joita voi asentaa kehitysympäristöihin kuten Eclipseen. Tässä kohdassa käsitellään muutamia kummallakin periaatteella tehtyjä ohjelmia. Yleensä näillä suunnittelutyökaluilla voi tehdä useita eri asioita ja luoda useita eri kaavioita[4]. Ohjelmista kerrotaan esittely ja ominaisuuksia sekä tietoja laajennettavuudesta ja lisenssistä.

7.2.1 ArgoUML

ArgoUML on johtava avoimen lähdekoodin periaatteella luotu UML -suunnittelutyökalu, joka tukee täysin UML -standardin 1.3 versiota ja tietyin poikkeuksin UML -standardin 1.4 versiota. Siinä on graafinen käyttöliittymä, jolla voi suunnitella, kehittää ja dokumentoida olioperustaisesti tehtäviä ohjelmia. Sillä voi luoda kaikkia UML -standardin 1.3 mukaisia kaavioita ja siinä on muun muassa koodingenerointimahdollisuus. Ohjelma toimii Java-alustan päällä ja on saatavilla kymmenellä eri kielellä [42].

ArgoUML itsessään on ilmainen ohjelma ja siihen on saatavilla myös laajennuksia, jotka lisäävät ominaisuuksia perusversioon. Laajennuksina voidaan asentaa koodin generointi ja reverse-engineering -ominaisuuksia eri ohjelmointikielille. ArgoUML on BSD -lisenssin alainen, mikä mahdollistaa sen, että sitä voi laajentaa ja käyttää laajennusta myös kaupalliseen tarkoitukseen. Esimerkkinä kaupallistamisesta voidaan mainita Poseidon for UML -työkalu, jota kehittää Gentleware sekä MyEclipse -työkalu, joka on saatavilla Eclipseen lisäosana [42].

4 On olemassa suunnitteluohjelmia, joilla voi tehdä vain tietyn suunnittelukaavion. Tällainen ohjelma on esimerkiksi Quick Sequence Diagram Editor, jolla voi luoda sekvenssikaavion tietyn syntaksin omaavasta tekstistä [29].

Taulukko 1: ArgoUML -ohjelman ominaisuuksien vertailu

Rational Rose Modeler -ominaisuus	**ArgoUML -tuki**
Käyttöjärjestelmät: Windows, Linux, HP Unix	Windows, Linux, Unix
UML 1.x	UML 1.3 ja 1.4 poikkeuksin
Kaaviot	**X**
Web -julkaisu ja raportin generointi	**-**
Kaavioiden tulostaminen	**X**
Muutosten yhdistäminen	**-**
Kuvauskanta/Kokoonpanon hallinta	**-**
Suunnittelumallit	**-**
SoDA -Integrointi (tai vastaava dokumentointiominaisuus)	**-**

ArgoUML:n ominaisuuksia voidaan verrata kaupalliseen Rational Rose -ohjelmistoon. Taulukossa 1 on kuvattuna Rational Rose -ohjelman Modeler -version ominaisuudet ja tutkittu, löytyykö ArgoUML -ohjelmasta samat ominaisuudet [27].

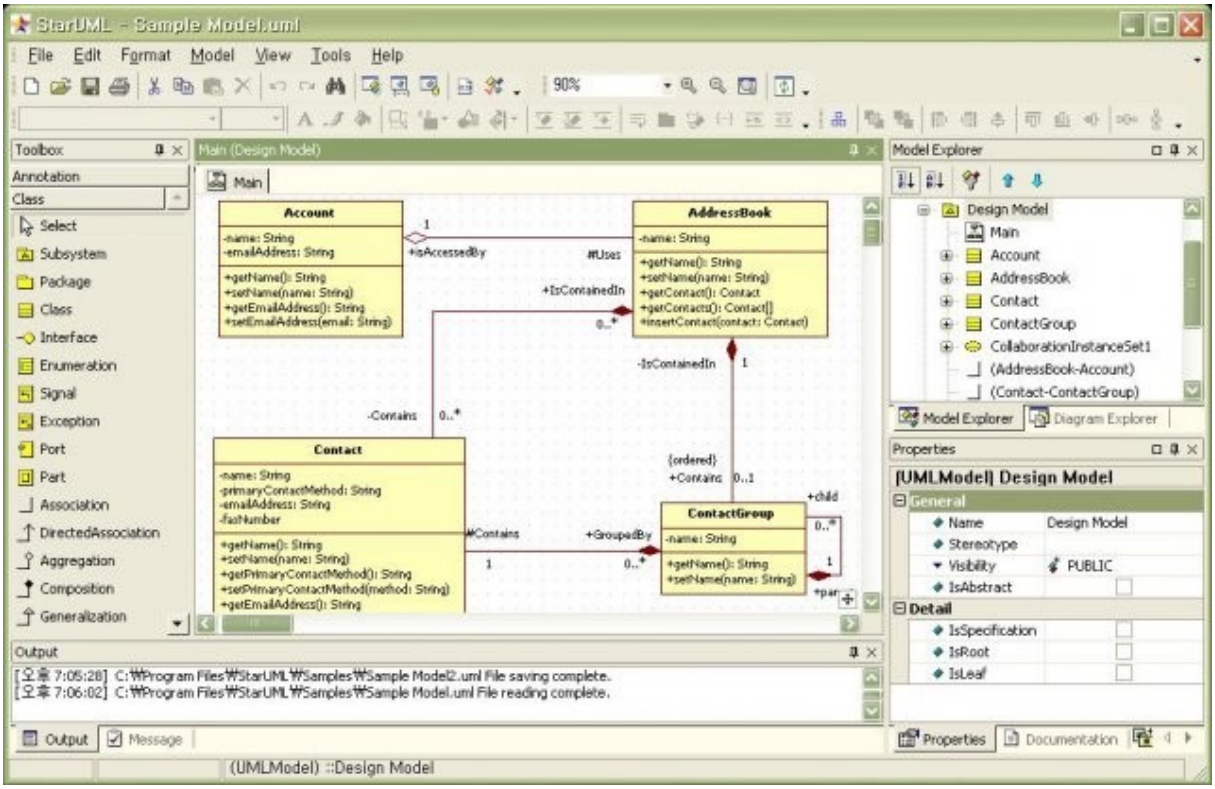
7.2.2 StarUML

StarUML on avoimen lähdekoodin projekti, jonka tarkoitus on kehittää nopeaa, joustavaa, laajennettavaa, paljon ominaisuuksia omaavaa ja ilmaista UML/MDA -alustaa Windows -käyttöjärjestelmään. Projektin tarkoituksena on rakentaa mallinnustyökalu sekä alusta, joka korvaa kaupalliset UML -työkalut, kuten esimerkiksi Rational Rosen. StarUML tukee UML 1.4 ja 2.0 versioita sekä Model Driven Architecture (MDA) -teknologiaa.

Taulukko 2: StarUML -ohjelman ominaisuuksien vertailu

Rational Rose Modeler -ominaisuus	**StarUML -tuki**
Käyttöjärjestelmät: Windows, Linux, HP Unix	Windows
UML 1.x	UML 2.0/1.4, UML 1.3 importtaus
Kaaviot	**X**
Web -julkaisu ja raportin generointi	**X**
Kaavioiden tulostaminen	**X**
Muutosten yhdistäminen	**-**
Kuvauskanta/Kokoonpanon hallinta	**-**
Suunnittelumallit	**X**
SoDA -Integrointi (tai vastaava dokumentointiominaisuus)	**X**

StarUML:n ominaisuuksia voidaan verrata kaupalliseen Rational Rose -ohjelmistoon. Taulukossa 2 on kuvattuna Rational Rose -ohjelman Modeler -version ominaisuudet ja tutkittu, löytyykö StarUML -ohjelmasta samat ominaisuudet [27].



Kuva 16: StarUML ohjelman näkymä ja luokkakaavio [39]

StarUML tukee yhtätoista UML:n kaaviotyyppiä, joista luokkakaavio näkyy kuvassa 16. StarUML tukee koodin generointia ja reverse engineering -menetelmää Javalle, C++ -kielelle sekä C# -ohjelmointikielelle. Sillä voi luoda Microsoft Office -dokumenttipohjia Wordiin, Exceliin ja PowerPoint -ohjelmaan. Se on tehty yhteensopivaksi siten, että siihen voi ladata Rational Rosen kaavioita [39].

Ohjelma tukee lisäosien asennusarkkitehtuuria ja siihen on mahdollista kehittää lisäosia COM-yhteensopivilla kielillä, joita ovat C++, Delphi, C#, Visual Basic, Visual Basic Script, VB.NET, Java Script ja Python. StarUML ohjelma on pyritty tekemään käyttäjäystävälliseksi. Se on pääsääntöisesti kirjoitettu Delphillä, mutta se on myös monikielinen projekti, joten muitakin kieliä voi käyttää StarUML:n kehittämisessä. StarUML on tehty GPL -lisenssin alaisuudessa [39].

7.2.3 Design Pattern Automation Toolkit

Design Pattern Automation Toolkit (DPAToolkit) on työkalu, jolla voi suunnitella ohjelmia suunnittelumalleja apuna käyttäen. Ohjelman ominaisuuksina on koodin generointi, reverse-

engineering -menetelmä sekä drag and drop -tyylinen UML -luokkakaavioiden luonti-mahdollisuus [22].

DPAToolkit -ohjelman mukana tulee kaikki Gamman et al. [9] kuvaamista suunnittelu-malleista, mitkä on mahdollista lisätä suunnitelmaan. Suunnittelumallit on tallennettu XML-formaattiin. Ohjelmalla pystyy generoimaan C#, C++, Java sekä VB.NET -tyylistä koodia. Ohjelman kehitystyö jatkuu koko ajan ja kaikki omaisuudet kuten reverse engineering -menetelmä eivät ole vielä täysin tuettu. Ohjelman avulla on mahdollista luoda kuvia GIF, JPEG, PNG ja TIFF -formaatteihin ja sen lisäksi luokkakaavioita voi kopioida leikepöydälle [22].

Ohjelma tukee omien lisäosien tekemistä koodin generointiin ja reverse-engineering -menetelmään. Ohjelman käyttäjän on mahdollista luoda omia suunnittelumallejaan lisäosina ja lisätä niitä ohjelmaan. Ohjelmaan voi lisätä lisäosana myös muiden ohjelmointikielien koodingenerointimahdollisuuksia. Ohjelmaa levitetään GPL -lisenssillä [22].

7.3 Testaustyökalut

Projektin testauksessa tarvitaan erilaisia apuvälineitä testaamisen helpottamiseksi. Tässä luvussa käymme läpi testaukseen kuuluvia avoimen lähdekoodin ohjelmistoja. Testaustyöka-luista on kerrottu esittely ja ohjelman ominaisuuksia sekä minkä lisenssin alaisuudessa ohjelma on tehty.

7.3.1 JUnit ja J2MEUnit -testausympäristöt

JUnit on yksinkertainen avoimen lähdekoodin mukainen testausympäristö, millä voi kirjoittaa ja suorittaa automatisoituja testejä Java -ohjelmointikielelle. JUnit -ympäristön ovat kehit-täneet Kent Beck ja Erich Gamma ja se on tärkeä kolmansien osapuolten kehittämä Java -kirjasto, mikä on koskaan luotu. JUnitin ansiosta Java -koodista on saatu vakaampaa, luotettavampaa ja virheettömämpää. JUnit on inspiroinut luomaan muille ohjelmointikielille monia xUnit -tyyppisiä yksikkötestausympäristöjä, kuten nUnit (.NET), pyUnit (Python), CppUnit (C++) ja dUnit (Delphi). JUnit on kehitetty CPL -lisenssin alaisuudessa [33].

J2MEUnit on Java 2 Micro Edition (J2ME) -kirjasto, joka sisältää yksikkötestausympäristön J2ME -ohjelmille. Se perustuu alkuperäisen JUnit -testausympäristön lähdekoodiin ja on hyvin samanlainen kuin alkuperäinen testausympäristö. Ympäristöjen erot johtuvat lähinnä siitä, että J2ME -alusta on erilainen ja siksi on tarvittu muutoksia alkuperäiseen JUnit

-ympäristöön. J2MEUnit -ympäristöä jaetaan Common Public License (CPL) -lisenssin alaisuudessa, mikä mahdollistaa ympäristön ilmaisen käyttämisen [28].

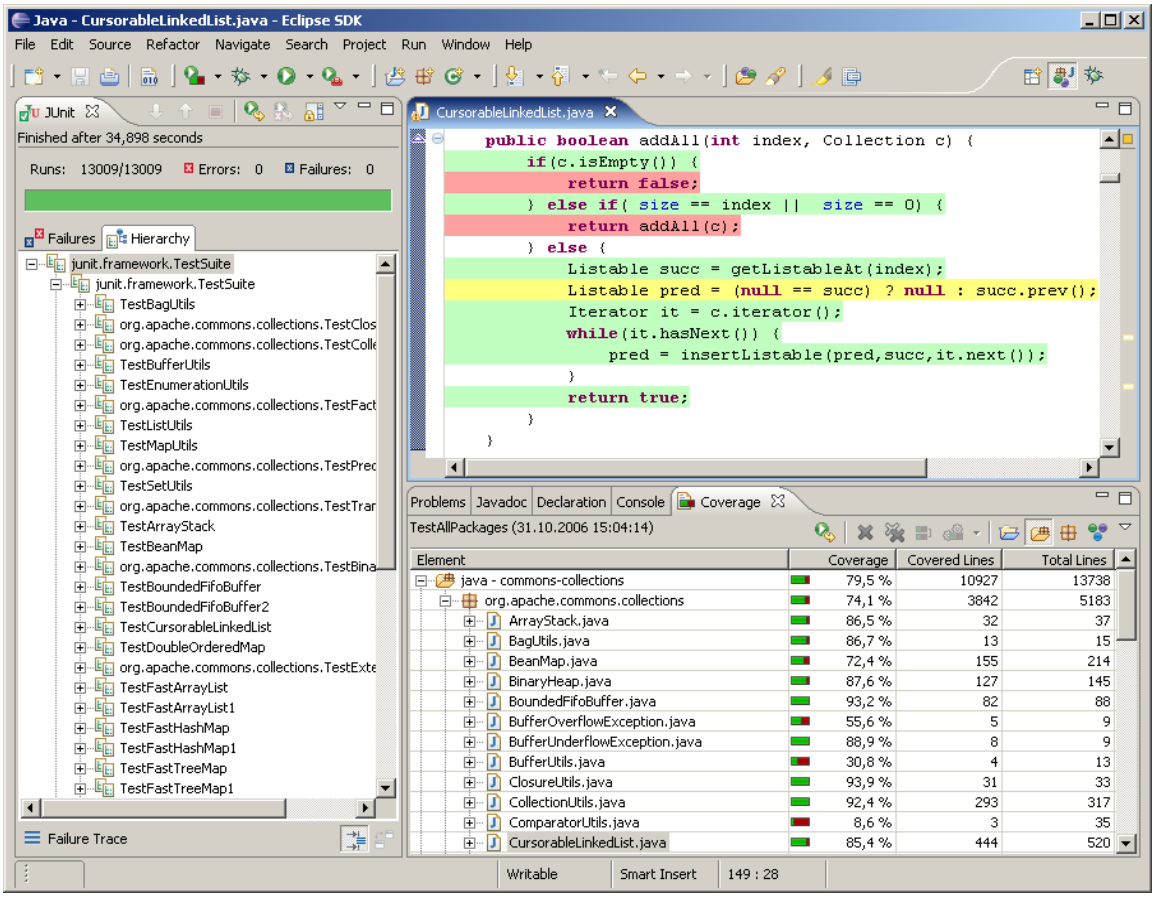
7.3.2 Software Test Automation Framework -testausympäristö

Software Test Automation Framework (STAF) on testausympäristö, joka on suunniteltu parantamaan testien uudelleenkäyttöä ja automatisointia. STAF -ympäristön tavoitteena on tarjota täydellinen end-to-end automatisointiratkaisu testaajille. Ympäristö on alusta-riippumaton ja tukee monia eri ohjelmointikieliä. Sen idea perustuu uudelleenkäytettäviin komponentteihin, joita kutsutaan palveluiksi. Sellaisia ovat esimerkiksi prosessien hyödyntäminen, resurssien hallinta ja tarkkailu. STAF auttaa monissa tuotannollisissa ongelmissa. Sen avulla on mahdollista saavuttaa muun muassa säännöllisempiä tuotantoaikoja ja vähentää valmistelu- ja testausaikaa. STAF tukee Java, C/C++, Rexx, Perl ja Tcl -ohjelmointikieliä ja se on laajennettavissa lisäosien avulla. STAF on lisensioitu CPL 1.0 -lisenssin alaisuuteen [38].

7.3.3 EclEmma -koodinkattavuustyökalu

EclEmma on ilmainen Java -koodin koodinkattavuustyökalu, jonka voi asentaa lisäosana Eclipse -kehitysympäristöön. Se lisää Eclipsen työkalupalkkiin painikkeen, millä toiminnan voi ottaa käyttöön. EclEmma tukee muun muassa paikallisesti suoritettavia Java -ohjelmia, Eclipse/RCP -ohjelmia sekä JUnit -testejä [30].

Testien suorittamisen jälkeen ohjelma näyttää kattavuustuloksen (ks. kuva 17) suoraan Eclipsen käyttöliittymässä. Tuloksista näkee kattavuuden ensin ylemmällä tasolla, jonka jälkeen käyttäjä voi tarkastella kattavuutta vaikka metoditasolla. Ohjelmaan voi myös importata kattavuustietoja tai ohjelmasta voi luoda raportteja kattavuudesta esimerkiksi XML tai HTML -muotoon. EclEmma on tarjolla Eclipse Public License 1.0 (EPL) -lisenssin alaisuudessa [30].



Kuva 17: EclEmma -lisäosan kattavuusnäkymä Eclipsessä [30]

7.4 Tukityökalut

Projektin tukitoiminnoissa tarvitaan tekstinkäsittelyohjelmia dokumentointiin, versionhallintaohjelmistoja koodin säilyttämiseen sekä projektinhallintaohjelmia projektin aikataulutus- ja tehtävänhallintaan. Tässä kohdassa tarkastelemme tukitoimintoihin liittyviä avoimen lähdekoodin mukaisia työkaluja. Ohjelmistoista on kerrottu esittely, ominaisuuksia, käyttöympäristö sekä minkä lisenssin alaisuudessa ohjelma on tehty.

7.4.1 OpenOffice -toimistosovellus

OpenOffice on johtava avoimen lähdekoodin periaatteen mukainen toimistosovellus. Ohjelman alkuunpanija on Sun, joka osti vuonna 1999 StarDivision yrityksen, joka kehitti StarOffice -ohjelmaa. Sun perusti OpenOffice.org sivuston, jossa StartOfficen pohjalta luotua Open Officea alettin ylläpitää ja kehittää avoimen lähdekoodin periaatteen mukaisesti. Saman aikaisesti Sun on kehittänyt myös kaupallista ja maksullista StarOffice -sovellusta, jossa on joitakin lisäominaisuuksia Open Officeen verrattuna. OpenOffice -toimistosovellus toimii

useilla eri alustoilla ja käyttöjärjestelmillä, joita ovat Microsoft Windows (98-Vista), GNU/Linux, Sun Solaris, Max OS X ja FreeBSD. Sovellus tukee useita eri kieliä ja on yhteensopiva muiden yleisimpien vastaavien toimistosovelluksien kanssa. OpenOfficen saa ladata ilmaiseksi ja se on ilmainen käyttää ja jakaa edelleen. OpenOffice on tehty LGPL -lisenssin alaisuudessa [41].

7.4.2 TortoiseCVS -versionhallintatyökalu

TortoiseCVS on avoimen lähdekoodin mukainen versionhallintatyökalu, jonka avulla on mahdollista käyttää CVS -versionhallintaa suoraan Windowsin Explorer -näkymästä. Käyttäjän on mahdollista suorittaa check out, update ja commit -operaatioita sekä vertailla eroja suoraan Explorer -ikkunasta käyttämällä hiiren oikeaa näppäintä tiedoston päällä. Tiedoston tila näkyy normaalien Explorer -ikonien päällä [43].

TortoiseCVS -versionhallintatyökalun tarkoitus on mahdollistaa Windowsin käyttäjille helpon CVS -versionhallinnan käyttämisen yhdellä ohjelmalla. Käyttöliittymän on tarkoitus olla yksinkertainen ilman monimutkaisia asetusdialogeja. Sen sijaan asetukset on pyritty muotoilemaan kysymysten muotoon, joita kysytään silloin kun vastauksia ensimmäisen kerran tarvitaan. TortoiseCVS -projektin internetsivujen mukaan tarkoituksena on tehdä asioita mahdollisimman paljon automaattisesti, sen sijaan että käyttäjää vaivataan ylimääräisillä vaihtoehdoilla. TortoiseCVS on saatavilla GPL -lisenssin alaisuudessa [43].

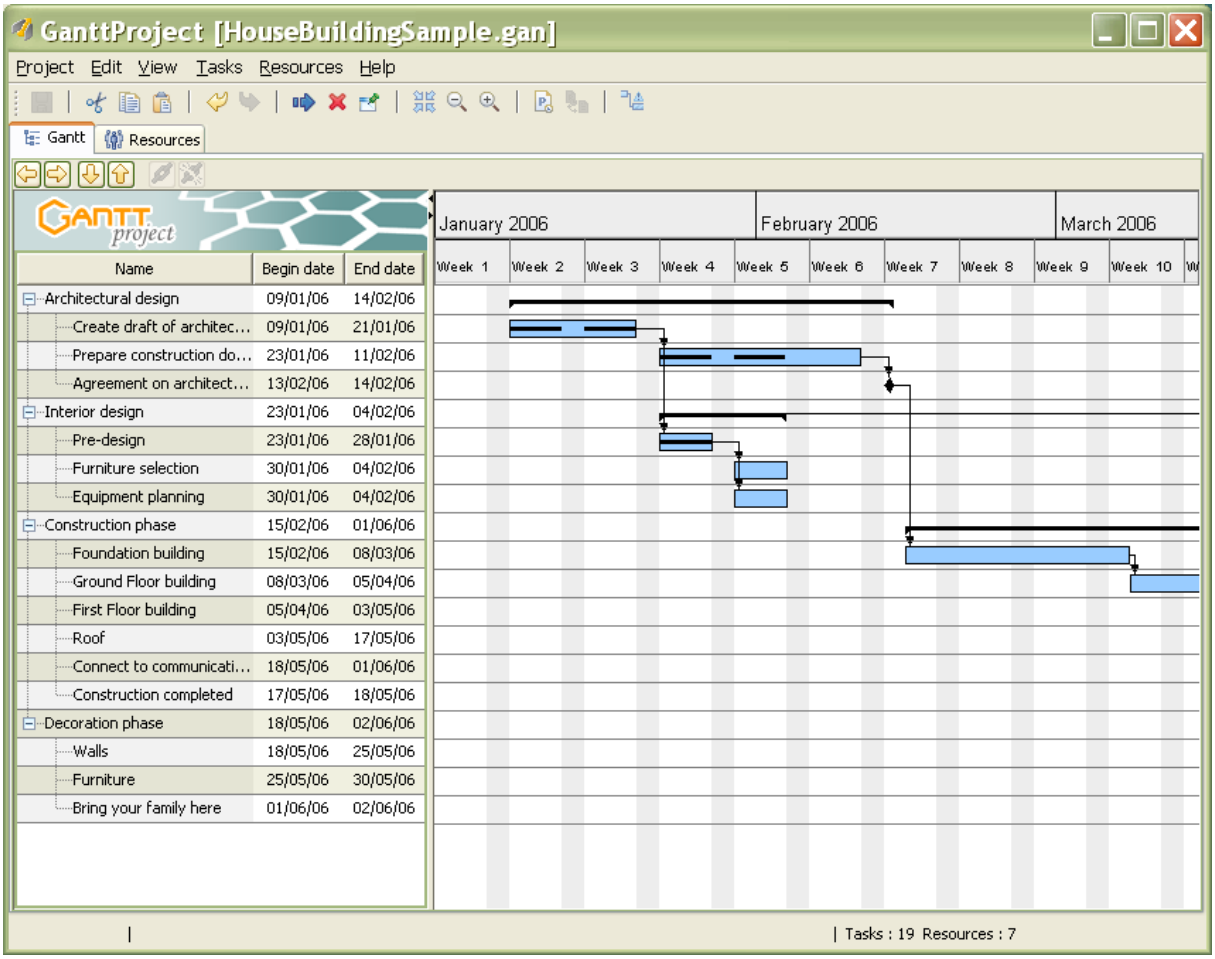
7.4.3 GanttProject -projektinhallintaohjelma

GanttProject on ilmainen avoimen lähdekoodin mukainen projektin aikataulutus- ja hallintaohjelmisto. Ohjelman ominaisuuksina ovat gantt -kaavioiden luonti, henkilöstön hallinta, kalenteri, erilaiset raportit ja MS Project -tiedostotyyppien tukeminen [26].

GanttProject -ohjelman avulla käyttäjä voi jakaa projektinsa eri tehtäviin ja liittää työntekijäresursseja työskentelemään niihin. Tehtävien välillä on mahdollista lisätä riippuvuuksia eli joku tehtävä esimerkiksi ei voi alkaa, ennen kuin joku toinen tehtävä on saatu päätökseen. Ohjelma muokkaa tiedot kahdelle erilaiselle kaaviolle, joista gantt-kaavio (kuva 18) esittää tehtäviä ja resurssienjakaumakaavio resusseja. Kaaviot on mahdollista tulostaa, muuttaa PDF tai HTML -raporteiksi tai muuntaa Microsoft Project tai esitysgrafiikka-ohjelman ymmärtämään muotoon. GanttProject -ohjelman etuja ovat sen internetsivujen

mukaan hyvä ja kasvava määrä perusominaisuuksia, helppokäyttöisyys, hinta, alustariippumattomuus ja avoimen lähdekoodin periaatteen mukaisuus [26].

Ohjelma on tehty Javalla, joten se tekee siitä alustariippumattoman ja se toimii muun muassa Windows, Linux ja MacOSX -käyttöjärjestelmissä. GanttProject ohjelman yhteyteen on saatavilla maksullisia ja ilmaisia apuohjelmia, joilla voi lisätä perusohjelman ominaisuuksia. Tälläisia lisäominaisuuksia ovat muun muassa GanttProject ohjelman tiedostotyyppien muuntamisohjelmat, joilla tiedostotyyppejä voi muuntaa toisen tyyppisiksi. Ohjelman koodi käyttää useita eri lisenssejä, pääsääntöisesti GPL -lisenssiä [26].



Kuva 18: GanttProject ohjelman näkymä gantt -kaaviosta [26]

7.4.4 OpenProj -projektinhallintaohjelma

OpenProj on ilmainen avoimen lähdekoodin mukainen projektinhallintaohjelmisto, jolla pyritään korvaamaan kaupallisia vastaavia ohjelmia, kuten Microsoft Project -ohjelmaa. Sillä pystyy avaamaan Microsoftin ja Primaveran tiedostotyyppejä. Ohjelmalla pystyy luomaan

muun muassa Gantt -kaavioita, PERT -kaavioita sekä WBS ja RBS -kaavioita. OpenProj -ohjelmiston kehittäjä Projity tarjoaa myös maksullista Project-ON-Demand -tuotetta, joka jakaa samat ydinominaisuudet ja sen lisäksi tarjoaa joitain lisäomiaisuuksia [36].

Ohjelma on saatavissa Linux, Unix, Max ja Windows -käyttöjärjestelmiin. OpenProj -ohjelmaa levitetään CPAL -linsessin alaisuudessa [36].

8 Työkalut käytännössä

Edellisessä luvussa kävi ilmi, millaisia kaikkia avoimen lähdekoodin työkaluja on olemassa ja mitä voi käyttää tukemaan ohjelmistoprosessia ja ohjelmistojen kehittämistä. Tässä luvussa eritellään työkalujen ominaisuuksia, mitä työkaluja voi käyttää missäkin ohjelmistoprosessin vaiheessa, mitkä soveltuvat suunnitteluun UML-kielellä tai tukevat suunnittelumalleja, mitkä soveltuvat erityisesti sovellusten kehittämiseen mobiileihin laitteisiin sekä miten työkaluja voi käyttää rinnakkain tai yhdessä.

8.1 Työkalujen sopivuus eri vaiheisiin

Avoimen lähdekoodin työkaluja on mahdollista käyttää ohjelmistoprosessin jokaisessa vaiheessa. Määrittelyyn ja suunnitteluun löytyy omia työkaluja kuten myös toteutukseen ja testaukseen. Tukitoimintoihin on saatavilla avoimen lähdekoodin ohjelmistoja ja jotkin työkalut sopivat useaan eri vaiheeseen. Taulukossa 3 näkyy, miten mikäkin työkalu sopii mihinkin vaiheiseen ja joitakin vaiheita on myös tarkennettu ja jaettu pienempiin osiin. Taulukon tekemisessä on käytetty seuraavia lähteitä: [43], [30], [38], [28], [33], [41], [36], [26], [22], [29], [39], [42], [31], [23].

Ohjelmistoprosessin eri vaiheet kerrotaan taulukon 3 ylimmällä rivillä yläkäsitteinä eli määrittely, suunnittelu, toteutus, testaus, ylläpito ja tukitoiminnot. Lisäksi on oma sarakkeensa sopivuudesta sovellusten kehittämiseen erityisesti mobiileihin laitteisiin sekä oma sarake lisäosien ja eri tiedostomuotojen tuelle. Toisella rivillä on joitakin näitä kohtia tarkennettu, esimerkiksi toteutusvaiheeseen kuuluu yleinen toteutus, mikä tarkoittaa käännösmahdollisuutta, ohjelman ajomahdollisuutta ja muita yleisiä toteutuksen toimintoja. Lisäksi toteutusvaiheeseen on erikseen jaoteltu koodin generointimahdollisuus sekä reverse-engineering menetelmä, joita jotkut suunnitteluohjelmat tukevat. Määrittely- ja suunnitteluvaihe on tässä yhdistetty ja selvyyden vuoksi UML ja suunnittelumallit on laitettu näiden vaiheiden alaisuuteen, vaikkakin ne voisivat kuulua myös toteutusvaiheeseen sen lisäksi. Testaus on jaoteltu koodin kattavuuden analysointiin ja yleiseen testaukseen. Tukitoiminnot on tarkennettu projektin hallintaan, dokumentointiin sekä versionhallintaan. Tuki sovellusten kehittämiseen mobiililaitteisiin on jaoteltu Java ja Symbian -ohjelmistojen kehittämisen tuelle. Lopuksi on kerrottu ohjelmistojen tuki lisäosille, sekä mitä tiedosto-muotoja ohjelmat pystyvät luomaan tai ottamaan vastaan.

Taulukko 3: Työkalujen ominaisuuksia

	Määrittely ja suunnittelu		**Toteutus**			**Testaus**		**Ylläpito**	**Tukitoim.**			**Mobiili**		**Lisäosat**	
Väline	*UML*	*Suunnittelumallit*	*Yleinen toteutus*	*Koodin generointi*	*Reverse-engineering*	*Koodin kattavuus*	*Yleinen testaus*		*Projektin hallinta*	*Dokumentointi*	*Versionhallinta*	*Java*	*Symbian OS*		**Tiedostomuodot**
ArgoUML	X	X		X	X									X	GIF, PNG, PS, EPS, PGML, SVG
DPAToolkit	X	X		X	X									X	XML, GIF, JPEG, PNG, TIFF
EclEmma						X	X								XML, HTML
Eclipse	X	X	X	X	X	X	X	X	X		X	X	X	X	
GanttProject									X						PDF, HTML, MS Project, PPT
J2MEUnit			X				X					X			
JUnit			X				X								
NetBeans	X	X	X	X	X	X	X	X			X	X		X	
OpenOffice								X		X					
OpenProj									X						Microsoft, Primavera
QSDEditor	X														PDF, (E)PS, SVG, GIF, JPEG, jne.
STAF							X								
StarUML	X	X		X	X									X	DOC, EXEL, PPT, Rational Rose
TortoiseCVS											X				

Kuten taulukosta 3 voidaan havaita, Eclipse on saanut eniten ominaisuuksia. Tämä johtuu pääsääntöisesti siitä, että Eclipse on laajennettavissa lisäosien avulla ja siihen on saatavilla laaja skaala eri lisäosia jokaiselle ohjelmistoprosessin vaiheelle. Lisäksi Eclipse sopii sovellusten kehittämiseen mobiileihin laitteisiin, koska siihen on mahdollista saada lisäosat Java- ja Symbian sovellusten tekemiselle [23]. NetBeans sai toiseksi eniten ominaisuuksia. Lähinnä erot olivat siinä, että NetBeansiin ei löytynyt projektin hallintaan sopivia lisäosia ja NetBeans on vain Java-sovellusten kehitysympäristö, joten Symbian-sovelluksia ei sen avulla ole mahdollista tehdä [31]. Sekä Eclipse että NetBeans soveltuvat siis laajasti ohjelmistojen kehittämiseen mobiileihin laitteisiin avoimen lähdekoodin periaatteen mukaisesti kyseisiin ohjelmiin saatavien lisäosien mahdollistamana. Ainoastaan dokumentointi hoidetaan muilla ohjelmilla. Koodin dokumentointiin on mahdollista käyttää myös esimerkiksi JavaDoc dokumenttimuotoa, jolla voidaan tehdä dokumentteja suoraan ohjelmakoodista ja sen kommenteista. Menetelmä toimii sekä Eclipsessä ja NetBeansissa ja on ilmainen, mutta ei kuitenkaan avoimen lähdekoodin periaatteella tehty [40].

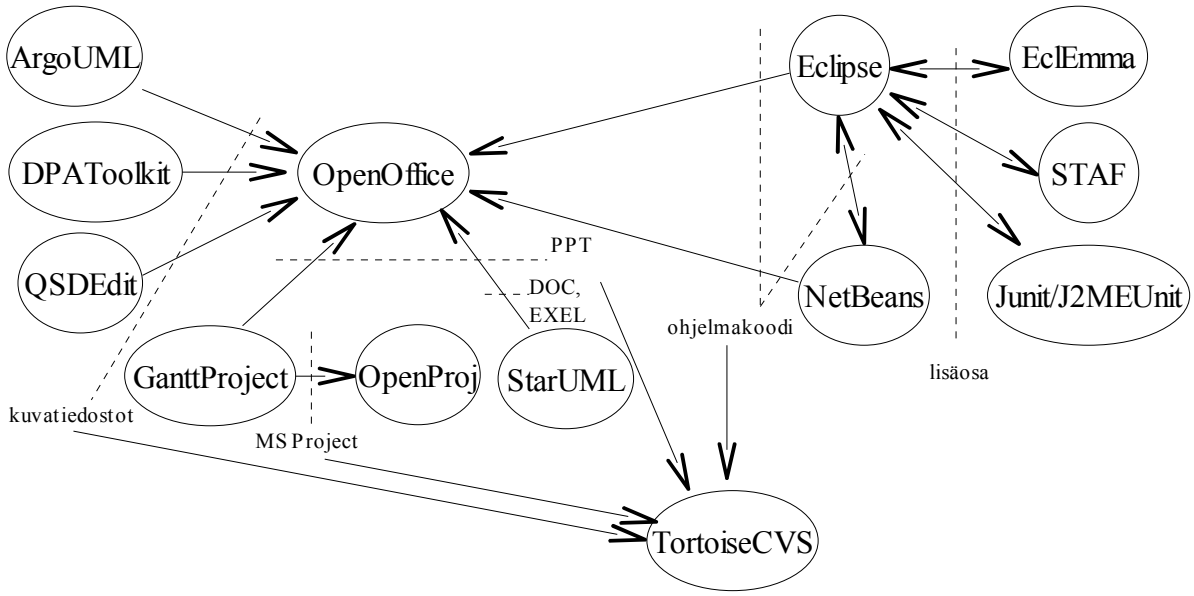
Seuraavaksi eniten ominaisuuksia saivat ArgoUML ja StarUML -suunnittelutyökalut, sekä Design Pattern Automation Toolkit -työkalu, jotka kaikki tukevat luonnollisesti UML -mallinnuskieltä ja sen lisäksi monia muita hyödyllisiä ominaisuuksia. Kaikki nämä ohjelmat käyvät mobiiliohjelmistojen suunnitteluun, mutta taulukon mobiilisarakkeessa ei siitä ole mainintaa, koska ohjelmista puuttuvat tuet koodin generoimiselle ja reverse-engineering menetelmälle Java ja Symbian -ohjelmointia varten. Ohjelmilla pystyy kuitenkin luomaan tarvittavat suunnitelmat ja luomaan niistä joko kuvia liitettäväksi suunnitteludokumenttiin tai mahdollisesti luomaan suoraan suunnitteludokumenttipohjia [42], [39], [22].

Seuraavaksi eniten ominaisuuksia saanut työkalu taulukon mukaan on J2MEUnit -testausympäristö, jota käytetään toteutuksen rinnalla yksikkötestauksessa sekä yleensäkin testauksessa. Ympäristö on tehty juuri J2ME -ohjelmointia varten, joten mobiilisarakkeeseen se on saanut merkin Java-tuesta [28]. JUnit sai kaksi ominaisuutta juuri sen takia, koska se on testausympäristön alkuperäinen versio, joka ei tue ohjelmien kehittämistä mobiileihin ympäristöihin [33]. OpenOffice toimistosovellus sai kaksi merkintää ylläpidolle ja tukitoiminnoista dokumentoinnille, mutta kyseisen kaltaisen ohjelmiston tarve on erittäin tärkeä ohjelmistotuotannossa. Tukisovelluksia tarvitaan kuitenkin koko ohjelmistoprosessin elinkaaren ajan ja siinä mielessä voitaisiin ajatella, että kyseinen ohjelma voisi saada merkinnän jokaisesta prosessin vaiheesta. EclEmma -koodinkattavuustyökalua käytetään testausvaiheessa tarkastamaan, miten kattavia testit ovat. Näin pystytään löytämään ohjelmakoodista kohtia, mitä testeissä ei suoriteta ollenkaan ja korjaamaan sitä kautta testejä tai katselmoimaan koodia niiltä osin tarkemmin [30]. Loput mainituista avoimen lähdekoodin mukaisista kehityksen apuvälineistä ovatkin vain tiettyä toiminnallisuutta tai prosessin vaihetta varten kehitetty ja täten saivat taulukkoon merkinnän yhteen sarakkeeseen.

Taulukosta voidaan päätellä, että avoimen lähdekoodin ohjelmilla voidaan kattaa kaikki ohjelmistoprosessin vaiheet, mitkä liittyvät sovellusten kehittämiseen mobiililaitteisiin. Yhteenvetona mainittakoon, että kaikki määrittely- ja suunnitteluvaiheen työkalut sekä tukitoimintovaiheen työkalut käyvät mobiiliohjelmien kehittämiseen, tosin tiettyjä suunnitteluohjelmien ominaisuuksia ei pystytä käyttämään. Monia mainittuja ohjelmia pystyy laajentamaan lisäosien avulla ja se mahdollistaa sen, että on mahdollista tehdä oma lisäosa jotain omaa tiettyä tarkoitusta varten, esimerkiksi juuri laajemman mobiilikehityksen mahdollistamista varten. Monet työkalut myös tukevat useita eri tiedostomuotoja, mitkä mahdollistavat tietojen siirtämisen eri ohjelmien välillä. Tästä kerronkin enemmän seuraavassa luvussa.

8.2 Työkalujen yhteiskäyttö

Avoimen lähdekoodin työkaluja voi käyttää myös rinnakkain tai yhdessä jonkun toisen avoimen lähdekoodin työkalun kanssa. Joillakin työkaluilla on mahdollista tuottaa tuloksia, joita joku toinen työkalu voi ottaa vastaan. Suunnittelutyökalusta on esimerkiksi mahdollista tallentaa suunnitelma kuvaksi ja siirtää kuva vaikka tekstinkäsittelyohjelmaan, jolla taas voi tehdä dokumentointia. Monimutkaisempi esimerkki voisi olla, että joku ohjelma luo suoraan toisen ohjelman muotoisen dokumentin. Kuvasta 19 voi havaita, miten tässä tutkielmassa tarkastellut avoimen lähdekoodin työkalut liittyvät toisiinsa perustuen seuraaviin lähteisiin: [43], [30], [38], [28], [33], [41], [36], [26], [22], [29], [39], [42], [31], [23].



Kuva 19: Työkalujen yhteydet toisiinsa

Kuten kuvasta 19 voi päätellä, monesta työkaluohjelmasta on linkki OpenOffice toimisto-sovellukseen. ArgoUML, DPAToolkit ja QSDEdit -suunnitteluohjemista voi tuottaa kuvatiedostoja, jota voidaan siirtä OpenOfficen tekstinkäsittelyohjelmaan dokumentointia varten. Lisäksi GanttProject -projektinhallintaohjelmasta ja StarUML -suunnittelutyökalusta voi luoda PPT -tiedostoja, joita OpenOfficen esitysohjelma ymmärtää. StarUML -työkalu tuottaa myös DOC ja EXEL -muotoisia tiedostoja, joita OpenOfficen tekstinkäsittelyohjelma ja taulukkolaskentaohjelma ymmärtävät. Toinen projektinhallintaohjelma OpenProj osaa taas avata MS Project -tyyppisiä tiedostoja, joita GanttProject voi tuottaa. EclEmma -koodinkattavuustyökalu, STAF -testausympäristö sekä JUnit ja J2MEUnit -testausympäristöt voidaan lisätä Eclipseen lisäosana ja tällöin vuorovaikutuksen voidaan ajatella olevan molemminpuolinen. Eclipse -kehitysympäristön ja NetBeans -kehitysympäristön välillä voidaan vaihtaa

Java-ohjelmakoodia ja molemmista kehitysympäristöistä ohjelmakoodi voidaan kopioida myös OpenOffice -toimistosovelluksen tekstikäsittelyohjelmaan mahdollista dokumentointia varten. Kaikki nämä äsken mainitut tiedostot ja kaikki muutkin tiedostot, mitä nämä ohjelmat tuottavat, voidaan ohjelmistoprojektin aikana säilyttää TortoiseCVS -versionhallintaohjelman hallussa.

Linkitykset eri avoimen lähdekoodin työkalujen välillä ovat siis mahdollisia ja helpottavat ohjelmistokehitystä ja prosessin kulkua. Tukitoiminnot, joita ovat muun muassa dokumentointi ja versionhallinta, linkittyvät moneen muuhun työkaluun ja tässä korostuukin tukitoimintojen tärkeys ohjelmistotuotannossa. Dokumentoinnin avulla suunnitelmat, resurssilaskelmat, ohjelmakoodi ja testitulokset saadaan laitettua eri dokumenttien muotoon, mitä ohjelmistoprosessin dokumentointivaihe määrittelee. Versionhallinta taas pitää huolen, että tieto säilyy turvallisessa paikassa, useat toteuttajat voivat käyttää tiedostoja ja että vanhoihin versioihin voidaan palata tarvittaessa.

9 Yhteenveto

Mobiililaitteiden kirjo on suuri ja laitteisiin on olemassa monia eri alustoja. Ohjelmiston kehitysprosessi mobiileihin laitteisiin sisältää useita eri vaiheita ja tehtäviä. Suunnittelu on tärkeää ohjelmiston kehittämisessä, samoin myös dokumentointi ja luonnollisesti toteutus ja testaus. Sovellusten kehittäminen mobiililaitteisiin ja kehitysprosessin tukeminen avoimen lähdekoodin ympäristöjen avulla on tämän tutkielman mukaan kuitenkin mahdollista. Tarjolla on useita erilaisia avoimeen lähdekoodin perustuvia työkaluja, joista vain osaa tarkasteltiin tässä tutkielmassa. Silti havaittiin, että kaikki ohjelmistoprosessin vaiheet on mahdollista toteuttaa avoimen lähdekoodin työkalujen avulla. Lisäksi havaittiin, että Eclipse kehitys-ympäristöllä saadaan kaikki ohjelmistoprosessin vaiheet katettua varsinaista dokumentointia lukuunottamatta ja ajatellen juuri Java -sovellusten kehittämistä mobiileihin laitteisiin. Eclipsen hyvänä ominaisuutena olikin sen laajennettavuus lisäosien avulla, mikä tarkoittaa, että siihen on saatavilla paljon työkaluja ja lisää työkaluja on kehitteillä tai varmasti tullaan kehittämään tulevaisuudessa. Tutkielman avulla löytyi useita suunnitteluapuohjelmia, joilla voi korvata kaupalliset suunnitteluohjelmat. Dokumentointiin sopivaa OpenOffice toimisto-sovellusta käytettiin myös tämän tutkielman laatimiseen, joten käytännössä voitiin jo todeta, että se sopii hyvin laajojenkin dokumenttien tekemiseen. Avoimen lähdekoodin ohjelmistojen yhteiskäyttö onnistui myös, ja tietoa pystyttiin liikuttamaan hyvin eri ohjelmien välillä.

Avoimeen lähdekoodiin perustuvat ohjelmat ovat tämän tutkielman mukaan sopivia vaikka aloittelevan yrityksen toimintaa silmälläpitäen. Ohjelmistoista ei tule kuluja yritykselle ja liiketoiminnan aloittaminen helpottuu huomattavasti, kun ei tarvitse maksaa kaupallisten ohjelmien lisenssimaksuja. Yrityksen kehittäessä toimintaa, sen on mahdollista laajentaa avoimen lähdekoodin ohjelmistoja omien erityistarpeittensa mukaan ja lisenssistä riippuen ohjelmia on mahdollista myös kaupallistaa muutosten jälkeen. Lisäosien avulla yritys pystyy tarvittaessa suunnittelemaan omia ohjelmistotuotannon apuohjelmia esimerkiksi Eclipseen tai moneen muuhun työkaluun, mitä tutkielmassa käsiteltiin. Aloittelevan yrityksen kannattaakin huolellisesti perehtyä avoimen lähdekoodin ohjelmistoihin, tutkia laajaa ohjelmien tarjontaa ja valita sopivat ohjelmat kehityksen apuvälineiksi. Sen jälkeen käyttämällä, antamalla ohjelmista palautetta kehitysyhteisölle, osallistumalla mahdollisesti itse kehittämiseen tai ohjelmia laajentamalla, yritys pystyy monipuolisesti helpottamaan omaa ohjelmistoprosessin läpivientiä.

Viitteet

[1] B'Far, R (2005) *Designing and Developing Mobile Applications with UML and XML*. Cambridge University Press.

[2] Baumeister, H., Koch, N., Kosiuczenko, P., Stevens, P., Wirsing, M (2003) *Global Computing*. Springer Berlin / Heidelberg.

[3] Carlson, D (2005) *Eclipse Distilled*. Addison Wesley.

[4] Cooper, J.W (1998) *The Design Patterns, Java Companion*. Addison-Wesley.

[5] D'Anjou, J., Fairbrother, S., Kehn D., Kellerman J., McCarthy P (2004) *The Java Developer's Guide to Eclipse, Second Edition*. Addison-Wesley.

[6] Erikson, H.E., Magnus, P (1997) *UML Toolkit*. John Wiley & Sons, Inc.

[7] Feller, J., Fitzgerald, B (2002) *Understanding Open Source Software development*. Pearson Education Limited.

[8] Fowler, M (1997) *Analysis Patterns, Reusable Object Models*. Addison-Wesley.

[9] Gamma, E., Helm, R., Vlissides, J (1995) *Design Patterns, Elements of Reusable Object-Oriented Software*. Addison-Wesley.

[10] Goldman, R., Gabriel, R.P (2005) *Innovation Happens Elsewhere - Open Source as Business Strategy*. Elsevier Inc.

[11] Haikala, I., Märijärvi, J (1998) *Ohjelmistotuotanto*. Suomen Atk-kustannus Oy.

[12] Hightower, R., Lesiecki N (2002) *Java Tools for Extreme Programming*. John Wiley & Sons, Inc.

[13] Kavanagh, P (2004) *Open Source Software, Implementation and Management*. Elsevier Digital Press.

[14] Kontio, M (2002) *Mobile Java with J2ME*. Edita Publishing Inc.

[15] Kontio, M., Tervo, T., Jääskeläinen, J., Arokoski, A., Vierimaa, K. & al. (2002) *Mobiiliteknologiat*. Edita Publishing Oy.

[16] Koskimies, K (2000) *Oliokirja*. Satku-Kauppakaari.

[17] Mapelsden, D., Hosking, J., Grundy, J () *Design Pattern Modelling and Instantiation using DPML*. .

[18] Mikkonen, T (2007) *Programming Mobile Devices: An Introduction for Practitioners*. John Wiley & Sons.

[19] Mikkonen, T (2004) *Mobiiliohjelmointi*. Talentum Media Oy.

[20] Murch, R (2001) *Project Management - Best Practices for IT Professionals*. Prentice Hall PTR.

[21] Peltomäki, J (2004) *J2ME-ohjelmointi*. Docendo Finland Oy.
[22] DPAToolkit (2007) *DPAToolkit projektin internetsivusto*. WWW-sivusto, http://dpatoolkit.sourceforge.net/ (6.11.2007).
[23] Eclipse Community (2007) *Eclipse kehityksen internetsivusto*. WWW-sivusto, http://www.eclipse.org/ (10.10.2007).
[24] EclipseMe (2007) *EclipseMe kehityksen internetsivusto*. WWW-sivusto, http://www.eclipseme.org/ (29.10.2007).
[25] FreeBSD Foundation (2008) *FreeBSD -sivusto*. WWW-sivusto, http://www.freebsd.org/ (10.3.2008).
[26] GanttProject community (2007) *GanttProject yhteisön internetsivusto*. WWW-sivusto, http://ganttproject.biz/ (7.11.2007).
[27] IBM (2008) *Rational Rose -ohjelman vertailusivusto*. WWW-sivusto, http://www-111.ibm.com/software/dre/hmc/compare.wss?HMC02=N778041C31148N78 (22.4.2008).
[28] J2MEUnit project (2007) *J2MEUnit projektin internetsivusto*. WWW-sivusto, http://j2meunit.sourceforge.net/ (7.11.2007).
[29] Markus Strauch (2007) *Quick Sequence Diagram Editorin kotisivu*. WWW-sivusto, http://sdedit.sourceforge.net/index.html (30.10.2007).
[30] Mountainminds GmbH & Co. KG (2007) *EclEmma ohjelman internetsivusto*. WWW-sivusto, http://www.eclemma.org/ (7.11.2007).
[31] NetBeans Community (2007) *NetBeans kehityksen internetsivusto*. WWW-sivusto, http://www.netbeans.org/ (10.10.2007).
[32] Object Management Group (2008) *UML version 1.4.2 lataussivusto*. WWW-sivusto, http://www.omg.org/cgi-bin/doc?formal/05-04-01 (11.3.2008).
[33] Object Mentor (2007) *JUnit käyttäjien internetsivusto*. WWW-sivusto, http://junit.org/ (7.11.2007).
[34] Open Handset Alliance (2007) *Open Handset Alliance -ryhmittymän internetsivusto*. WWW-sivusto, http://www.openhandsetalliance.com (29.11.2007).
[35] Open Source Initiative (2007) *Open Source internetsivusto*. WWW-sivusto, http://www.opensource.org/ (3.10.2007).
[36] Projity (2007) *OpenProj ohjelman internetsivusto*. WWW-sivusto, http://openproj.org/openproj (7.11.2007).
[37] SourceForge (2007) *Open Source projektien säilytyspaikka internetissä*. WWW-sivusto, http://sourceforge.net/ (10.10.2007).

[38] STAF project (2007) *STAF projektin internetsivusto*. WWW-sivusto, http://staf.sourceforge.net/index.php (7.11.2007).
[39] StarUML (2007) *StarUML projektin internetsivusto*. WWW-sivusto, http://staruml.sourceforge.net/index.php (30.10.2007).
[40] Sun (2008) *Javadoc -sivusto*. WWW-sivusto, http://java.sun.com/j2se/javadoc/ (18.1.2007).
[41] Sun (2007) *OpenOffice internetsivusto*. WWW-sivusto, http://www.openoffice.org/ (10.10.2007).
[42] Tigris.org (2007) *ArgoUML kehityksen internetsivusto*. WWW-sivusto, http://argouml.tigris.org/ (30.10.2007).
[43] TortoiseCVS (2007) *TortoiseCVS projektin internetsivusto*. WWW-sivusto, http://www.tortoisecvs.org/ (20.11.2007).
[44] Wikipedia (2007) *Wikipedia internetsivusto*. WWW-sivusto, http://fi.wikipedia.org/wiki/Avoin_lähdekoodi (3.10.2007).
[45] Wikipedia (2007) *Android (mobile device platform) Wikipedia sivusto*. WWW-sivusto, http://en.wikipedia.org/wiki/Android_(mobile_device_platform) (29.11.2007).
[46] Wikipedia (2007) *Maemo sivusto Wikipediassa*. WWW-sivusto, http://en.wikipedia.org/wiki/Maemo (29.11.2007).
[47] Wikipedia (2007) *MontaVista Softwaren sivusto Wikipediassa*. WWW-sivusto, http://en.wikipedia.org/wiki/MontaVista_Software (29.11.2007).
[48] Wikipedia (2007) *Embedded LInux sivusto Wikipediassa*. WWW-sivusto, http://en.wikipedia.org/wiki/Embedded_Linux (29.11.2007).
[49] Wikipedia (2008) *Windows Mobile -sivusto Wikipediassa*. WWW-sivusto, http://en.wikipedia.org/wiki/Windows_Mobile (23.04.2008).