Heidi Lehtevä

Loihdin-työvälineen ja perinteisen ohjelmoinnin erot Android-sovellusten kehittämisessä

Metropolia Ammattikorkeakoulu
Insinööri (AMK)
Tietotekniikan koulutusohjelma
Insinöörityö
16.4.2012

Metropolia

Tiivistelmä

Tekijä(t) Otsikko Sivumäärä Aika	Heidi Lehtevä Loihdin-työvälineen ja perinteisen ohjelmoinnin erot Android-sovellusten kehittämisessä 43 sivua + 2 liitettä 16.4.2012
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjemistotekniikka
Ohjaaja(t)	Lehtori Juha Kämäri

Android on tällä hetkellä suosittu käyttöjärjestelmä älypuhelimiin ja tablet-tietokoneisiin. Tämän insinöörityön tavoitteena oli tutkia perinteisen ohjelmoinnin ja Loihdin-nimisen työkalun eroja Android-sovellusta kehittäessä. Työ tehtiin yritykselle Adensy oy, joka on kehittämässä Loihdinta. Adensy oy myy verkkopohjaisia asiakkuudenhallintajärjestelmiä, jotka on kehitetty Loihtimella. Loihdinta on laajennettu luomaan asiakkuudenhallintasovelluksia Android-alustalle. Työssä keskityttiin tähän laajennokseen. Loihtimella voidaan kehittää sovelluksia ilman, että kehittäjän tarvitsee itse koodata.

Työ aloitettiin perehtymällä Android-käyttöjärjestelmään ja asioihin, joita kehittäjän on hyvä Androidista tietää. Tässä käytiin läpi sekä Android-järjestelmän toimintaa että Android-ohjelmoinnin rakennuspalikoita. Työssä huomioitiin myös Loihtimen malliperustaisuus. Loihdinta verrattiin nopean kehityksen malliin ja todettiin sen muistuttavan kyseisen menetelmän työkaluja.

Tutkimus toteutettiin luomalla kaksi samanlaista asiakkuudenhallintasovellusta. Toinen sovellus ohjelmoitiin käsin käyttämällä Eclipse-kehitysympäristöä ja toinen luotiin Loihdinta käyttäen. Sovelluksissa pitäydyttiin Loihtimen tarjoamien toimintojen rajoissa. Työvaiheissa pidettiin kirjaa niihin kuluneesta ajasta ja työn lopussa näitä vertaillaan.

Huomattiin, että Loihdin oli perinteistä ohjelmointia huomattavasti nopeampi. Nopeuteen vaikutti se, ettei kehittäjän tarvitse ohjelmoida eikä syntyneessä koodissa ole kirjoitusvirheitä eikä kehittäjän tarvitse suunnitella käyttöliittymää. Loihtimessa ei myöskään kehittäjän kokemuksella ollut vaikutusta kehitykseen kuluneeseen aikaan.

Todettiin, että Loihdin sopii erityisesti sellaisiin tilanteisiin, joissa sovellus pitää julkaista ja ottaa käyttöön mahdollisimman nopeasti. Sitä voi käyttää sekä kokemattomat että kokeneet kehittäjät, sillä sen käyttö ei vaadi ohjelmointiosaamista.

Avainsanat	Android, Loihdin, Malliperustainen ohjelmointi

Metropolia

Abstract

Author(s) Title Number of Pages Date	Heidi Lehtevä The difference between Loihdin and traditional programming while developing Android Applications 43 pages + 2 appendices 16 April 2012
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Juha Kämäri, Senior Lecturer

At the moment Android is a popular operating system for smartphones and tablet computers. The purpose of this thesis was to examine the differences between developing Android applications by traditional coding and by using a tool named Loihdin. This thesis was made for a company called Adensy Ltd that developed Loihdin. The company sells web-based Customer Relationship Management systems that are created using Loihdin. Loihdin has been expanded to develop Android applications and this thesis concentrates on that expansion. Loihdin allows the creation of applications without the programmer actually having to engage him/herself in programming.

The thesis provides an introduction to the Android operating system and also to aspects that a developer should know about Android including both Android system's operations and the basic components of Android programming. The study also considers the model-driven nature of Loihdin. Loihdin was compared to the method Rapid Application Development and it was noticed that it resembles this method.

The study was carried out by creating two similar Customer Relationship Management applications. One application was programmed using Eclipse IDE and the other with the Loihdin development tool. The applications were designed to match the functions available in Loihdin. The time used on each stage was noted down and at the end of the study the results were compared.

Loihdin was found to be significantly faster than traditional programming. The speed was influenced by not needing to write code, the fact that the generated code is typo-free and that the developer does not have to design the user interface. The developer's experience does not have an impact on the speed of development.

It was concluded that Loihdin fits well situations where an application needs to be published and introduced as fast as possible. It can be used by developers with or without experience because it does not require programming expertise.

Keywords	Android, Loihdin, Model Driven Engineering

Sisällysluettelo

Lyhenteet

RAD – Rapid Application Development eli Nopean kehityksen malli on ohjelmiston kehitysprosessi. Siinä pyritään luomaan sovelluksia mahdollisimman nopeasti.

MDD – Model-Driven Development eli Malliperustainen ohjelmistokehitys hyödyntää Malliperustaista arkkitehtuuria. Siinä pyritään käyttää malleja eri ohjelmistokehityksen vaiheissa nopeuttamaan ohjelmistokehitystä sekä lyhyellä että pitkällä tähtäimellä.

MDA – Model-Driven Architecture eli malliperustainen arkkitehtuuri on standardi, jossa malleja voidaan käyttää kaikissa ohjelmistokehityksen vaiheissa. Sen tarkoituksena on vähentää ohjelmistojen kehityksen monimutkaisuutta, kustannuksia ja kehitysaikaa.

OMG – Object Management Group on kansainvälinen tietotekniikan alan liitto, joka kehitti mm. UML:n ja MDA:n.

CRM – Customer Relationship Management eli asiakkuudenhallinta on yritysten käyttämä menetelmä pitää yllä kirjaa vanhoista asiakkaista ja niiden tiedoista sekä auttaa uusien asiakkaiden hankkimista.

Aktiviteetti – Aktiviteetti on Android-sovelluksen peruskomponentti. Aktiviteetti-luokka on yleensä oma näkymänsä.

Intentio – Intentio-olio on passiivinen tietorakenne, joka sisältää kuvauksen suoritettavasta operaatiosta. Sitä käytetään yleensä viestintään sovelluksen komponenttien välillä.

Dalvik – Android-järjestelmän käyttämä virtuaalikone, jossa sovellukset suoritetaan.

1 Johdanto

Työn tavoitteena on selvittää, minkälaisia etuja ja haittatekijöitä perinteisellä ohjelmoinnilla Android-alustalle on verrattuna malliperusteisen ohjelmistokehitykseen keskittyen Adensy oy:n Loihdin-työkaluun. Adensy oy on pieni ohjelmistoalan yritys, joka tuottaa asiakkuudenhallinnan työkaluja pienille ja keskisuurille yrityksille Loihdinta käyttäen. Samalla yritys kehittää Loihdinta eteenpäin.

Internet on nykypäivänä osa arkea. Sitä saattaa tarvita myös silloin, kun lähettyvillä ei ole käytettävissäolevaa tietokonetta, kuten esimerkiksi kirjakaupassa. Internetissä voi lukea kirja-arvosteluja, mutta tämä ei hyödytä silloin, kun lukija on jo kaupassa miettimässä, voisiko kirja olla hyvä ostos. Tässä tilanteessa älypuhelin internet-yhteydellä voi auttaa lukijaa tekemään ostospäätöksensä, sillä hän pääsee lukemaan kyseisestä kirjasta tehtyjä kirja-arvosteluja. Monet sivut, kuten Amazon-nettikirjakauppa, ovat jo luoneet sivuistaan mobiililaitteille sopivat versiot, josta voi nopeasti ja helposti selata kirjoja ja niiden tietoja älypuhelimella. Älypuhelimet ja tablet-tietokoneet mahdollistavat internetiin pääsyn käyttäjän ollessa liikkeellä. Näihin mobiililaitteisiin voidaan asentaa erilaisia sovelluksia yrityssovelluksista peleihin. Sovellukset ovat yleensä pieniä johtuen mobiililaitteen rajallisesta muistista ja hinnaltaan alhaisia.

Kännyköiden ja tietokoneiden valmistajat ovat laajentaneet mobiililaitemarkkoinoille. Tutut kännyköiden ja tietokoneiden valmistajat valmistavat älypuhelimia ja käyttöjärjestelmiä, joista suurimmat nimet ovat Applen IPhone ja Googlen kehittämä Android-käyttöjärjestelmä Android-älypuhelimiin. Android-älypuhelimet ovat kilpailijaansa Iphonea halvempia, joka voi kenties olla syynä sen suurempaan suosioon, ainakin nuorten keskuudessa.

Älypuhelinten yleistyessä myös yritysmaailmassa tulee tarve saada laajennettua erilaisia yritysten hallinnan mahdollistavia sovelluksia näille markkinoille nopeasti. Loihdin-työvälinettä on aloitettu laajentamaan vastaamaan tähän tarpeeseen Android-rintamalla. Loihdin on jo kykeneväinen kehittämään web-sovelluksia, joilla yrityksen jäsenet voivat pysyä ajan tasalla asiakkaista ja yrityksen tapahtumista. Loihtimen idea pohjau-

tuu malliperustaiseen ohjelmointiin. Loihdinta verrataan myös *Nopean kehityksen mallin* ideaan tässä työssä.

Työ toteutetaan kehittämällä yrityssovellus Androidille ohjelmoiden Java-ohjelmointikielellä sekä Loihdinta käyttäen. Siinä tarkastellaan eroja tekniikoiden välillä ohjelmoijan näkökulmasta ja pohditaan, kumpi menetelmä tuntuu tässä tilanteessa paremmalta. Alussa pyritään luomaan käsitystä siitä, kuinka Android toimii, ja raapaistaan pintaa siitä, minkälaisia asioita kannattaa ottaa huomioon kirjoittaessaan sovelluksia Android-älypuhelimiin ja -tabletteihin. Myöhemmässä kappaleessa esitellään, minkälaiset sovellukset tässä työssä luodaan. Sovellusten on molempien vastattava tiettyihin tarpeisiin, jotka esitellään yrityssovelluksen määrittelyn yhteydessä. Sovellusten ohjelmointia käsittelevissä kappaleissa käydään läpi työvaiheet, joiden kautta yrityssovellukset saatiin valmiiksi. Huomiota kiinnitettiin erityisesti aikaan, joka niiden kehitykseen meni. Näiden lukujen jälkeen tehdään itse vertailu työmenetelmien välillä.

2 Android pähkinänkuoressa

2.1 Android – perusteet

Android on Googlen kehittämä käyttöjärjestelmä älypuhelimiin ja tablet-tietokoneisiin. Tässä luvussa tutustutaan hieman Androidiin järjestelmänä ja kuinka se toimii. Kappaleessa kiinnitetään huomiota asioihin, joita ohjelmoijan on hyvä tietää ohjelmoidessaan Android-sovellusta.

2.1.1 Prosessit ja tiedostot

Android-alustalle ohjelmoidaan käyttämällä Java-ohjelmointikieltä. Android on käyttöjärjestelmä. Se on monen käyttäjän Linux-systeemi, jossa jokainen sovellus on oma käyttäjänsä. Käyttäjärjestelmä antaa jokaiselle sovellukselle uniikin käyttäjätunnuksen. Sovellus ei tiedä omaa käyttäjätunnustaan. Ainoastaan käyttöjärjestelmä tietää sovelluksien käyttäjätunnukset. Järjestelmä asettaa oikeudet kaikkiin tiedostoihin sovelluk-

sessa, joiden perusteella sovellus, jolla on vastaava käyttäjätunnus, voi käyttää tiedostoja. Tiedostoja voidaan siis määrittää yksityisiksi, jolloin ainoastaan tietty sovellus voi käyttää tiedostoa. Näin ollen sovelluksella on pääsy ainoastaan niihin komponentteihin, joita se tarvitsee toimiakseen. Sovellus tarvitsee erikseen luvan, jotta se pääsee käsiksi Android-puhelimen muihin resursseihin, kuten esimerkiksi kontakteihin. Lupa annetaan sovellukselle asennuksen yhteydessä AndroidManifest.xml-tiedostossa. [1.]

Android-ympäristössä jokainen prosessi suoritetaan omassa virtuaalikoneessa erillään muista sovelluksista. Prosessi käynnistetään, kun jotakin sovelluksen osista tarvitaan suoritettavaksi. Android-järjestelmä huolehtii prosessin lopettamisesta, kun sitä ei enää tarvita tai kun järjestelmän on vapautettava muistia. [1.]

Eri sovellusten välillä on mahdollista jakaa tiedostoja mm. sillä tapaa, että sovelluksille annetaan sama Linux-käyttäjätunnus. Tällöin ne pääsevät toistensa tiedostoihin. Näitä sovelluksia voidaan suorittaa samassa virtuaalikoneessa. Prosessien suoritus samassa virtuaalikoneessa säästää järjestelmän resursseja. Dataa sovellukset voivat jakaa muille käyttämälle erilaisia sisällöntarjoajia, kuten puhelimen tiedostojärjestelmää tai SQLite-tietokantaa. Sisällöntarjoajan välityksellä sovellukset voivat muokata tallennettuja tietoja. [1.]

Android SDK-työvälineet kääntävät koodin sekä tiedostot että resurssit pakettitiedostoksi, joka tunnistetaan .apk-loppupäätteestä. Jotta käyttöjärjestelmä voi käyttää pakettitiedostoa, sen tulee sisältää AndroidManifest.xml-tiedosto, jossa lueteltuna sovelluksen komponentit ja luvat. [2, s. 28.]

2.1.2 Dalvik-virtuaalikone

Java VM-virtuaalikone ei sovellu Android-puhelimiin. Tämän takia Google kehitti Dalvik-virtuaalikoneen, jossa Android-sovellukset suoritetaan. Kyseinen virtuaalikone on kehitetty toimimaan hitaallla suorittimella ja vähäisellä määrällä RAM-muistia järjestelmässä. Dalvik-virtuaalikone ottaa myös huomioon tilanteet, joissa virtaa on vain vähän käytössä. Dalvik-virtuaalikone luo .DEX-tiedoston .CLASS-tiedoston sijaan, jonka voi ladata

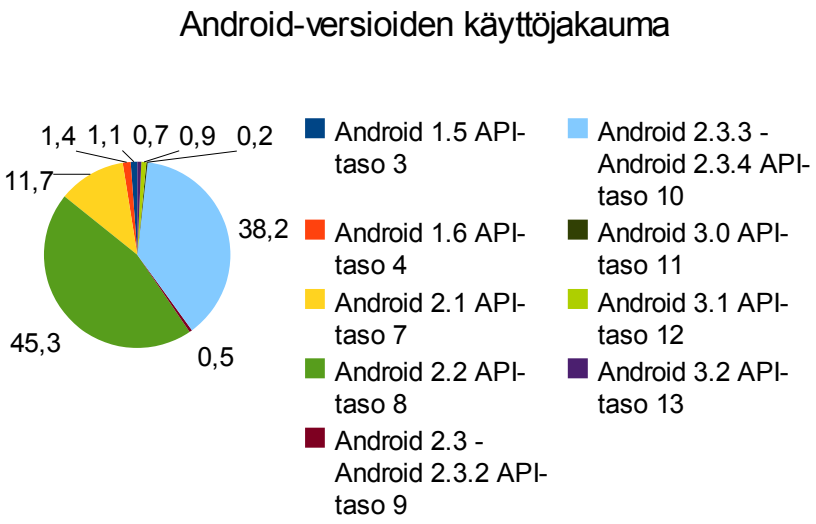
mobiililaitteeseen ja suorittaa. [3.]

Android-sovelluksia ohjelmoitaessa käytetään Java-kieltä, joka on olio-ohjelmointikieli. Olioiden käyttö varaa muistia varsinkin siinä tilanteessa, kun olio on iso. Android-sovelluksien roskien keruu on Dalvik-virtuaalikoneen vastuulla. Kun muistinvarauksia on tapahtunut liian paljon, esimerkiksi peligrafiikkan piirtyessä ruudulle, käynnistyy Dalvikin roskien keruu. Useimmissa tapauksissa tämä ei välttämättä näy käyttäjälle, mutta joissain tilanteissa, joissa nopeus on kriittinen elementti, käyttäjä huomaa viiveen. Roskien keruu Androidissa ei ole optimoitu niin, että se sujuisi huomaamattomasti. Siksi vastuu olioiden luonnista järkevästi on ohjelmoijan harteilla. [4.]

2.1.3 Android-käyttöjärjestelmäversiot

Android-käyttöjärjestelmistä on useita versioita käytössä. Tämä on hyvä huomioida silloin, kun aloittaa kehittämään sovellusta Android-ympäristöön. Sovellus, joka on kehitetty käyttämällä uusimman Android-version ohjelmointirajapintaa, ei välttämättä toimi puhelimessa, jossa käyttöjärjestelmä on vanhempi. Ongelmia tulee myös silloin, kun kehittäjä haluaa toiminnallisuutta uudemmasta versiosta sovellukseen, jota hän kehittää vanhemmalle käyttöjärjestelmälle. Esimerkiksi Androidin käyttämä xml-jäsennin ei tue API-tasolla 7, joka on Android 2.1-version ohjelmointirajapinta, tekstidatan lukua xml-elementistä. Tämä on lisätty myöhemmässä versiossa, mutta mikäli kehittäjä haluaa käyttää vanhempaa versiota, on tämä puute kierrettävä.

Android-käyttöjärjestelmän päivittäminen on jätetty käyttäjien kontolle. Tästä johtuen kaikissa puhelimissa ei ole aina käytössä uusinta versiota. Taulukossa 1 on ympyrädiagrammi kuvaamassa eri Android-versioiden määriä aikavälillä 20.9.2011-3.10.2011. Taulukossa kuvataan kaikkien käytössäolevien Android-versioiden prosentuaalista määrää. Taulukossa lukee Android-version lisäksi API-taso eli ohjelmointirajapinta-taso.



Taulukko 1: Android-versioiden käyttöjakauma ympyrätaulukossa

Vanhemmat Android-versiot ovat yhteensopivia uusien kanssa. Täten sovellus, joka on kehitetty Android-versiolle 1.5, on käytettävissä kaikissa Android-puhelimissa. Tässä työssä toteutetut yrityssovellukset ovat kehitetty versiota 2.1 vasten. Siksi 97,5 % käytössäolevista Android-puhelimista voi käyttää näitä sovelluksia. [5.]

Eclair-nimellä tunnettu Android 2.1 toteuttaa ohjelmointirajapintatasoa 7. Se julkaistiin tammikuussa 2010 ja sisälsi muutoksia ohjelmointirajapintaan ja vikojen korjauksia. Android 2.1 ei lisännyt erityisiä toimintoja käyttöjärjestelmään. Edelliset isot lisäykset tulivat Android 2.0:ssa. [6.]

2.2 Android – aktiviteetit

Android-ohjelman luonnissa rakennuskomponentit on ymmärrettävä. Nämä komponentit eivät ole tavallisia Java-rajapintoja, vaan ne ovat Androidille spesifejä, jotka löytyvät vain Android-kirjastoista. Jokainen komponentti on oma kokonaisuutensa, jolla on oma rooli ja toiminto. Erilaisia komponentteja on Android-ympäristössä neljä: aktiviteetti (*activity*), palvelu (*service*), sisällöntarjoaja (*content provider*) ja lähetysten vastaanotta-

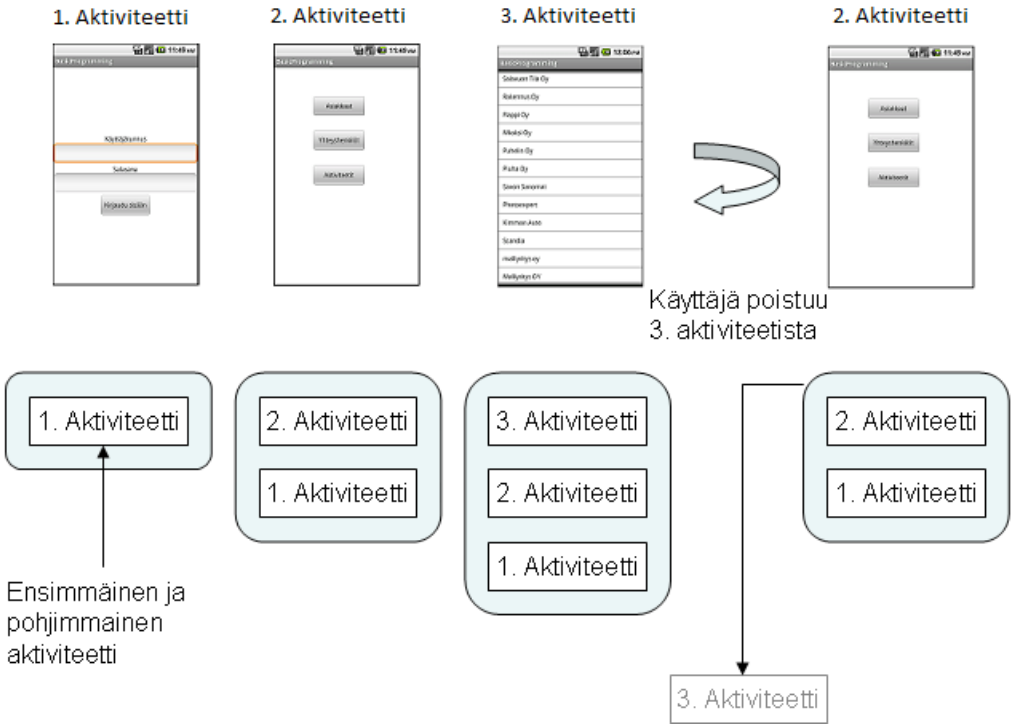
ja (*broadcast receiver*). Tässä työssä esiintyvissä tekniikoissa ja ohjelmakoodissa on käytetty aktiviteetti-rajapintaa, sillä muille ei ole tarvetta. [1.]

2.2.1 Aktiviteetti sovelluksessa

Aktiviteetti-luokka on muista ohjelman komponenteista riippumaton ikkuna, jolla on oma käyttöliittymä. Android-sovellus muodostuu useista komponenteista, jotka liittyvät löyhästi toisiinsa. Näistä johtuen yksi sovellus voi käynnistää toisen sovelluksen tietyn aktiviteetin, mikäli tämä on sallittu sovellusten AndroidManifest.xml-tiedostoissa. Esimerkiksi jonkinlainen kaverilista-sovellus voisi kutsua sähköpostisovellusta, kun käyttäjä haluaa lähettää kaverillensa sähköpostia, jolloin sähköpostisovelluksen aktiviteetti aukeaa kaverilistasovelluksen päälle. [1.]

Sovelluksen käynnistyessä käyttäjä näkee yleensä ensimmäiseksi pääaktiviteetin (*main activity*). Tämä toimii pääikkunana, josta käyttäjä lähtee navigoimaan ohjelmassa. Jokainen aktiviteetti pystyy luomaan uuden aktiviteetin. Uusi aktiviteetti peittää sen luoneen aktiviteetin, joka siirtyy pysähtyneeseen tilaan. Järjestelmä säilyttää aktiviteetin aktiviteettipinossa (*back stack*) , ellei muistin vähyyden vuoksi tuhoa sitä. Uusin aktiviteetti on aina päällimmäisenä aktiviteettipinossa ja siten näkyvillä käyttäjälle. Käyttäjän poistuessa uusimmasta aktiviteetista, järjestelmä poistaa sen pinosta ja tuhoaa sen. Tällöin pinossa seuraava aktiviteetti on päällimmäisenä ja siten käyttäjälle näkyvillä.

Kuvassa 1 on edellämainittu prosessi. Tässä kuvassa 1 ensimmäinen aktiviteetti on käynnistänyt 2. aktiviteetin, joka puolestaan käynnisti 3. aktiviteetin. Nämä ovat lopulta pinossa. Käyttäjä poistuu 3. aktiviteetista, jolloin järjestelmä tuhoaa sen muistista. Aktiviteettipinoon jäljelle jäävät 1. ja 2. aktiviteetti, joista 2. aktiviteetti on päällimmäisenä. [7.]

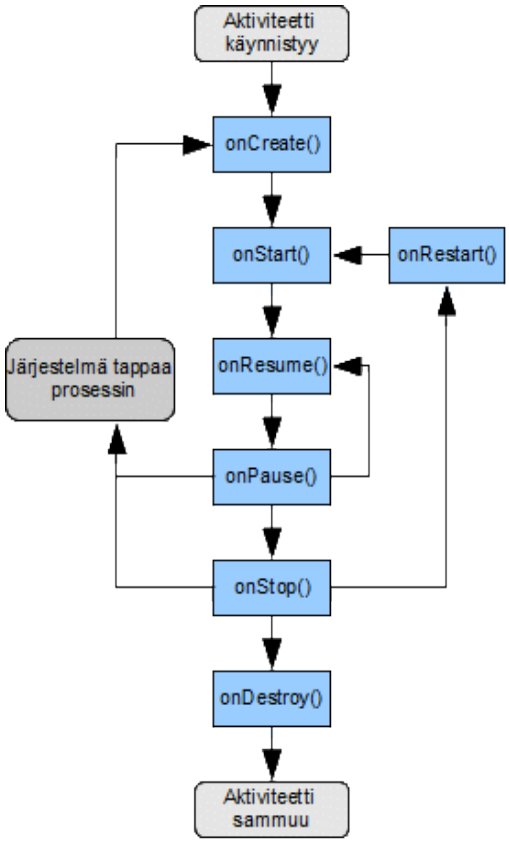


Kuva 1: Aktiviteettipino

2.2.2 Aktiviteetin elinkaari

Aktiviteetilla voi olla elinkaarella useita tiloja. Aktiviteetti voi olla tilassa, jossa se on päällimmäisenä pinossa ja käyttäjän näkyvillä. Tällöin se on aktiivisena. Aktiviteetin ollessa toisen näkymän alla niin, ettei käyttäjä näe sitä ollenkaan, on se pysähtyneessä tilassa. Aktiviteetti on tällöin vielä puhelimen muistissa tallessa, ja se on säilyttänyt tila- ja muuttujatiedot. Mikäli muistia tarvitaan muualla, järjestelmä tappaa ensin pysähtyneessä tilassa olevat aktiviteetit. Tilanteessa, jossa aktiviteetti on toisen alla, mutta näkyy vielä osittain (päällimmäinen aktiviteetti ei vie koko näyttöä), on aktiviteetti taukotilassa. Tällöin järjestelmä saattaa tappaa kyseisen aktiviteetin, mutta ainoastaan, jos sillä ei ole muuta vaihtoehtoa. [7.]

Kuvassa 2 on esitetty aktiviteetin elinkaari. Aktiviteetin elinkaari alkaa onCreate()-metodista. Tässä metodissa on hyvä tehdä raskaat staattiset alustukset ja käyttöliittymän luonti. OnCreate()-metodia seuraa joko onStart()-metodi, jota järjestelmä kutsuu aktiviteetin tullessa käyttäjälle näkyville. Kun aktiviteetti on valmis kommunikoimaan käyttäjän kanssa, kutsuu järjestelmä onResume()-metodia. Tämän ja onPause()-metodikutsun välillä aktiviteetti on aktiivisessa tilassa. OnPause-metodia kutsutaan, kun aktiviteetti siirtyy taukotilaan tai taukotilan kautta pysähtyneeseen tilaan. Pysähtyneeseen tilaan pääsy vaatii onStop()-metodin kutsun. Taukotilasta vaihto aktiviteetin aktiiviseen tilaan onnistuu taas onResume-metodilla. OnPause()-metodin tulee sisältää vain kevyitä operaatioita, jotta ohjelma ei näy käyttäjälle liian raskaana. Järjestelmä kutsuu onRestart()-metodia, mikäli aktiviteettia tarvitaan vielä. Tässä metodissa voidaan asettaa onStop-metodissa pysäytetyt operaatiot takaisin käyntiin. Mikäli aktiviteetti olikin tarkoitus sammuttaa, kutsuu järjestelmä onDestroy-metodia, jonka jälkeen aktiviteetti lopulta kuolee. Täällä on hyvä hoitaa muistin siivoaminen ja viimeistäänkin tietojen tallennus. Järjestelmän tapettua aktiviteetin, joka oli taukotilassa tai pysähtyneenä, joutuu se kutsumaan aktiviteetin onCreate()-metodia uudestaan, jos käyttäjä haluaa jatkaa tämän aktiviteetin parissa. [7.]



Kuva 2: Aktiviteetin elinkaari

2.3 Aktiviteetin intentio

Android-sovellusten ydinkomponentit, joita ovat aktiviteetit, palvelut ja lähetysten vas-

taanottajat, aktivoidaan intentioiden (*Intent*) kautta. Intentioviestintä mahdollistaa dynaamisen sidonnan sovelluksien ydinkomponenttien välille. Intentio-olio itsessään on passiivinen tietorakenne, joka sisältää kuvauksen suoritettavasta operaatiosta tai lähetysten kohdalla kuvauksen jostain, joka on tapahtunut ja siitä ilmoitetaan. [8.]

2.3.1 Intentioiden käyttö

Muille aktiviteeteille intentioiden lähetys tapahtuu käyttämällä joko *Context.startActivity()*-metodia tai *Activity.startActivityForResult()*-metodia. Context-luokka on rajapinta, joka sisältää tietoa sovelluksesta ja sillä voi myös suorittaa sovellustason operaatioita, kuten uuden aktiviteetin luonnin. Edellämainituilla metodeilla voi myös saada jo olemassa olevan aktiviteetin tekemään jotain uutta. Mikäli ohjelmakoodissa uuden aktiviteetin luontiin on käytetty *Activity.startActivityForResult()*-metodia, voidaan sen mukana lähetetty intentio-olio palauttaa takaisin *Activity.setResult()*-metodin kautta. [8.]

Palveluille intentio-olioiden lähetys tapahtuu samaan tyyliin lähetykselle tarkoitetuilla metodeilla. Lähetysten vastaanottajille intentio-olio lähetetään käyttämällä mitä tahansa lähetysmetodia, jotka kuljettavat intentio-olion kaikille siitä kiinnostuneille lähetysten vastaanottajille. Jokaisessa tapauksessa Android-järjestelmä löytää sopivan aktiviteetin, palvelun tai lähetysten vastaanottajat tarvittaessa luoden niistä ilmentymän. [8.]

Intentio-oliot menevät aina ainoastaan sille komponentille, jolle se on tarkoitettu. Tämä tarkoittaa sitä, että aktiviteetille lähetetty intentio-olio kuljetetaan ainoastaan aktiviteetille, ei koskaan palvelulle tai lähetysten vastaanottajille. Komponenteilla on omat intentiosuodattimensa (*Intent filter*), jotka auttavat järjestelmää selvittämään intentioille vastaanottajat, jotka suostuvat käsittelemään lähetettyä intentiota. Komponenteilla voi olla yksi tai useampi intentiosuodatin. [8.]

Intentio-olio sisältää sen tiedon, josta vastaanottava komponentti on kiinnostunut, esimerkiksi tietoa siitä, mitä vastaanottajan tulee tehdä tai dataa, jolle sen odotetaan tekevän jotain. Intentio sisältää myös Android-järjestelmää kiinnostavaa tietoa, kuten minkälaiselle komponentille intentio on tarkoitettu. Intention voi lähettää jollekin tietyl-

le komponentille antamalla sille komponentin nimen, mutta se ei ole pakollista. Android-järjestelmä voi käyttää muitakin tapoja saadakseen selville kohdekomponentin. [8.]

Intention sisältämä tieto siitä, mitä vastaanottajan tulee tehdä, on joko valmiiksi Android-järjestelmässä määritetty tai ohjelmoijan luoma. Intentio-olioita voi käyttää myös yksinkertaisen datan lähetykseen lähettäjältä vastaanottajalle käyttämällä intention extra-osiota. Extra-osio koostuu avain-arvo-pareista. Nämä arvot voidaan asettaa intentiolle käyttämällä erilaisia *put...()*-metodeja, kuten *putInt()*- tai *putString()*. Vastaanottavassa komponentissa nämä tiedot saadaan ulos intentiosta käyttämällä vastaavia *get...()*-metodeja. Intentio sisältää myös flagejä, jotka antavat järjestelmälle ohjeita, kuten kuinka aktiviteetti käynnistetään tai mitä sille tehdään käynnistyksen jälkeen. Flagit on kaikki määritelty Intentio-luokassa. [8.]

2.3.2 Eksplisiittinen ja implisiittinen intentio

Intentiot voidaan jakaa kahteen ryhmään: eksplisiittisiin ja implisiittisiin. Eksplisiittiset intentiot ovat niitä intentioita, joille on annettu vastaanottavan komponentin nimi. Näitä intentioita käytetään tyypillisesti sovelluksen sisäiseen viestintään, sillä ohjelmoija tietää tarkalleen sen komponentin nimen, jolle hän intention haluaa lähettää. Tällöin muulla informaatiolla ei ole merkitystä järjestelmälle. Implisiittiset intentiot eivät tiedä tarkkaan vastaanottajaansa. Näissä vastaanottavan komponentin nimi on jätetty pois. [8.]

Implisiittisiä intentioita käytetään yleensä viestittämään sovellusten välillä. Järjestelmä joutuu käyttämään komponenttien intentiosuodattimia selvittääkseen, mikä komponentti voi käsitellä lähetettävää intentiota. Komponentti ilman suodattimia voi vastaanottaa ainoastaan eksplisiittisiä intentioita. Ne kolme osaa, joita järjestelmä vertaa intention ja intentiosuodattimen välillä ovat toiminto, jonka vastaanottajan odotetaan tekevän, data ja kategoria, joka määrittää, minkälaiselle komponentille intentio on tarkoitettu. Extroilla ja flageillä ei ole merkitystä. [8.]

2.4 Android - käyttöliittymäkomponentit

Android-sovellusten käyttöliittymät koostuvat näkymä- (*view*) ja näkymäjoukko-olioista (*viewgroup*). Näkymä-oliot ovat niitä käyttöliittymäkomponentteja, joilla käyttäjä hallitsee sovellusta, kuten esimerkiksi painikkeet ja tekstikentät. Näkymäjoukot sen sijaan pitävät sisällään näkymäolioita tai muita näkymäjoukkoja. Käyttöliittymäkomponentin sijoittelu käyttöliittymässä riippuu hyvin pitkälle näkymäjoukon tyypistä. Joissain näkymäjoukoissa näkymiä voidaan sijoittaa käyttöliittymään ainoastaan vertikaaliseen tai horisontaaliseen suuntaan, kun toisentyyppisessä näkymäjoukossa näkymille voidaan tarkoin määritellä niiden sijainti suhteessa muihin komponentteihin. Näitä näkymäjoukkoja kutsutaan *layouteiksi*. Android-ohjelmointiympäristö mahdollistaa käyttöliittymien luonnin xml-tiedostoissa, joissa yhden aktiviteetin käyttöliittymä on yksi tiedosto. Käyttöliittymiä on myös mahdollista luoda dynaamisesti aktiviteetin luonnin aikana Java-koodilla. Tämä mahdollistaa Loihdinin laajennuksen Androidille. [2, s. 82-83]

Käyttäjä kommunikoi sovellusten kanssa käyttöliittymien välityksellä. Käyttöliittymien tulee siis reagoida käyttäjän syötteisiin, ja tämä tapahtuu Android-puhelimissa kosketustapahtumien (*touch event*), puhelimen näppäinten painallusten ja valikkonavigoinnin kautta. Puhelimen omilla näppäimillä on järjestelmässä valmiiksi kuuntelijat, mutta on mahdollista, joskaan ei suositeltavaa, ylikirjoittaa ne. Käyttäjillä on jo iskostunut käsitys, mitä näitä näppäimiä painamalla tapahtuu. Näppäimestä tapahtuvat toiminnon muutos saattaa hämätä käyttäjää pahasti. Käyttöliittymäkomponenteille voidaan myös määrittää tapahtumakuuntelijoita, jotka reagoivat tietynlaisiin käyttäjän antamiin syötteisiin. Koska komponentit käyttöliittymässä rakentuvat hierarkisesti, järjestelmä välittää tapahtumaa ylhäältä asti alas sille käyttöliittymäkomponentille, joka tapahtuman käsittelee. Yhdellä käyttöliittymäkomponentilla voi olla useampi eri kuuntelija, jotka tekevät eri asioita. Esimerkiksi painikkeella voi olla OnClickListener- ja OnLongClickListener-kuuntelijat. Tällöin painiketta painettaessa kosketusnäytöllä järjestelmä välittää painikkeelle onClick-tapahtuman, jolloin painike reagoi, kuten OnClickListener-kuuntelijassa on määritetty. Vastaavasti pitkän painalluksen tapahtuessa, järjestelmä välittää painikkeelle OnLongClick-tapahtuman, jolloin tuo kuuntelija reagoi. Aktiviteettiin voi myös määrittää onGestureListener-kuuntelijan. Tämä kuuntelija kuuntelee kosketusnäytöllä tapahtuvia liikkeitä. Android mahdollistaa myös multitouch-ominaisuuden, jos-

sa kosketusnäytöllä on kaksi pistettä. Näiden yhtäaikaisen kuuntelun seurauksena voidaan tuottaa toiminnallisuuksia, joita käyttäjän on helppo syöttää kahdella sormella, kuten esimerkiksi zoomaus kuvanäkymässä liikuttamalla kahta sormea erilleen toisistaan. [2, s. 117-133.]

3 Loihdin ja malliperustainen ohjelmointi

3.1 Loihdin

Suomessa sijaitsevan Adensy Oy:n tuottama Loihdin on ohjelmointityökalu. Loihtimen avulla käyttäjä kykenee luomaan monimutkaisia web-sovelluksia yrityksille ilman, että sen käyttö vaatii ohjelmointiosaamista. Loihdin mahdollistaa ohjelmistokehityksen kustannusten ja kehitysajan vähentämisen. Myös ohjelmistoprojektien epäonnistumisten määrän tulisi vähentyä Loihdinta käytettäessä. Sovelluksella luotuja järjestelmiä pystyy myös muokkaamaan helposti tarpeen vaatiessa.

Loihdin-sovellukset sopivat hyvin ERP-, CRM-, laskutus- ja henkilöstöhallinnan tehtäviin. ERP tulee sanoista Enterprice Resource Planning, ja se tarkoittaa toiminnanohjausjärjestelmää. Tällainen järjestelmä helpottaa yrityksen sisäisten tehtävien kirjanpitoa. CRM, eli Customer Relationship Management, tarkoittaa asiakkuudenhallintaa. Asiakkuudenhallinta on yrityksille hyvä tapa pitää kirjaa henkilöistä ja muista yrityksistä, joihin yritys on yhteydessä. Näitä ovat asiakkaat ja mahdolliset myyntikohteet. Päätavoitteena CRM-järjestelmässä on helpottaa uusien asiakkaiden hankkimista samalla pitäen huolta vanhoista asiakkaista.

Loihdinin vaatimat tietojärjestelmät tallennetaan pilvipalvelimelle. Yhteydet palvelimelle tukevat salattua SSL-yhteyttä. Palvelinkapasiteetti kasvaa asiakkaan tarpeiden mukaan.

Loihdin eroaa perinteisestä ohjelmistokehityksestä sillä tavalla, että se käyttää malliperustaisen ohjelmistokehityksen lähestymistapaa. Tämän takia myös laadunvalvontaan käytettävät metodit eroavat perinteisestä. Perinteisessä ohjelmistokehityksestä laatua

valvotaan luomalla vaatimusmäärittelyt kehitysprosessin alussa ja suorittamalla ohjelmistotestaus kehitysprosessin lopussa. Malliperustaisessa ohjelmistokehityksessä laadunvalvonnan voi suorittaa kehityksen aikana. Tässä lähestymistavassa pyritään virheettömään koodiin ja avustetaan kehittäjää välttämään epäloogisuuksia sovelluksessa.

Loihdinta on laajennettu Android-alustalle. Android-laajennus ei vielä kata kaikkea, mitä Loihtimella pystyy tekemään web-sovelluksiin, mutta sillä pystyy luomaan muutaman tärkeän toiminnon, joita tässä työssä käytetään. Tässä työssä ei huomioida Loihtimen web-sovellusten kehitystä.

3.2 Malliperustainen ohjelmointi

OMG (*Object Management Group*), kansainvälinen tietotekniikan alan liitto, on perustettu vuonna 1989. Se on tunnettu erilaisten standardien, kuten mm. UML:n (Unified Modelling language) ja MDA:n kehityksestä. [9.]

Vuonna 2001 OMG kehitti malliperustaisen arkkitehtuurin, eli MDA:n (*Model-Driven Architecture*), jonka tarkoituksena oli vähentää ohjelmistojen kehityksen monimutkaisuutta, kustannuksia ja kehitysaikaa. Malliperustaisuus tarkoittaa sitä, että malleja käytetään pääasiallisena lähteenä dokumentoinnille, analysoinnille, suunnittelulle, rakentamiselle, käyttöönotolle ja ylläpitämiselle [10, s. 3]. Yksi MDA:n keskeisimmistä aspekteista on sen kyky huomioida koko ohjelmistokehityksen elinkaari määrittelystä suunnitteluun, toteutukseen, testaukseen ja lopulta käyttöönottoon ja ylläpitoon. Uusien alustojen ja tekniikoiden ilmestyessä MDA mahdollistaa nopean kehityksen. Näin MDA tarjoaa laajan ja jäsennetyn ratkaisun sovellusten väliseen tiedonsiirtoon ja laajennukseen muille alustoille helpottamalla koodin uudelleenkäyttöä tulevaisuudessa. [10, s. 2.]

MDA lupaa helpottaa mallien luomista, jotka kykenevät nopeasti laajentamaan järjestelmiä uusille alustoille, parantamaan tuottavuutta automatisoimalla yksitoikkoiset ja työläät kehitystehtävät sekä helpottamaan testausta, sillä malleja voidaan validoida suoraan vaatimuksia vasten. Mallit parantavat myös laatua lisäämällä johdonmukaisuutta ja luotettavuutta tuotoksen osiin. [10, s. 5-6.]

MDD (Model Driven Development) on ohjelmistokehitystä, joka käyttää malliperustaista arkkitehtuuria. Kun sitä käyttää oikein, kehittäjien lyhyen tähtäimen tuottavuus paranee lisäämällä pääasiallisen ohjelmistotuotoksen toiminnallisuutta. Mitä enemmän toiminnallisuutta malli tarjoaa, sitä tuottavampi se on. Pidemmällä tähtäimellä pääasiallisen ohjelmistotuotoksen elinaika pitenee. Kunnollinen malli takaa sen, että pääasiallinen ohjelmistotuotos pysyy käyttökelpoisena, jolloin se palauttaa siihen sijoitetut kustannukset tuotoksen kehitysvaiheessa. Siksi yksi malliperustaisen ohjelmoinnin tärkeimmistä aspekteista on pääasiallisen ohjelmistotuotoksen muutosherkkyyden vähentäminen.

Ohjelmistokehitystä hankaloittavina tekijöinä ovat tiedon katoaminen ja vaatimusten muuttuminen. Näiden huomioonottoa on pyritty helpottamaan malliperustaisessa ohjelmoinnissa. Tietoa voi kadota henkilöstön vaihtuessa. Jotta ohjelmisto elää pidempään kuin mitä kehittäjät tekevät töitä sen parissa, on tiedon oltava saavutettavissa ja käytettävissä mahdollisimman laajasti. Tämän takia ohjelmistotuotosten tulisi olla kuvattuna tarkasti ja räätälöidysti niin, että kiinnostuneet sidosryhmät voivat ne ymmärtää ilman teknistä osaamista. Tähän sopii graafinen mallinnus, kuten esimerkiksi UML-mallinnus. Ohjelmistotuotannossa kehittäjien on myös pystyttävä lisäämään uusia toiminnallisuuksia mahdollisimman pienillä muutoksilla valmiiksi luotuihin osiin. Mallinnus auttaa havainnollistamaan, kuinka uusi ominaisuus voidaan sijoittaa tuotokseen. Malliperustaisen ohjelmoinnin yksi tärkeimmistä teknisistä perustoista on visuaalinen mallinnus. Tämän takia hyvän MDD-työkalun avulla saadaan helposti luotua dokumentaatio luodusta tuotteesta, sillä siitä on jo valmiina graafinen mallinnus. Dokumentoinnin helppous auttaa siirtämään kehittämisprosessin aikana saadut ideat muillekin kuin prosessiin osallistuville kehittäjille.

Ohjelmistot, jotka on sidottu tiettyyn kehitysalustaan, eivät pysy elossa sitä kehitysalustaa kauempaa. Tästä syystä kehitystyökalujen tulee olla sellaisia, jotka mahdollistavat alustariippumattomuuden. Tämä on yksi malliperustaisen ohjelmoinnin tärkeistä ominaisuuksista. Tuotoksen elinaikaa pidennetään myös turvaamalla tuotos alusta-tason muutoksilta. [11.]

3.3 Loihdin ja nopean kehityksen malli

3.3.1 Nopean kehityksen malli

Ohjelmistotuotannossa RAD (*Rapid Application Development*), eli Nopean kehityksen malli, on kehitysprosessi, jolla saadaan aikaan käyttökelpoisia järjestelmiä lyhyessä ajassa ilman laajamittaista suunnittelua. Sen kehitetti James Martin, joka julkaisi kirjansa *Rapid Application Development* vuonna 1991, jossa malli virallisesti luotiin. Nopean kehityksen malli on sen jälkeen vielä kehittynyt nykyiseen muotoonsa. [12.]

Nopean kehityksen mallin käyttö tarjoaa toimivan sovelluksen yli puolessa ajassa vähemmän kuin tavallisesti kehitykseen menee. Ajan puutteesta johtuen aikaansaatu tuote ei aina ole laadukas ja viimeistelty, vaan keskeneräinen tuote, jota hienosäädetään vielä käyttöönoton jälkeen. Käyttäjät voivat erilaisten nopean kehityksen mallin työkaluilla luoda valmiita sovelluksia ja interaktiivisia verkkosivuja nopeasti, vaikka heillä ei ole kokemusta ohjelmistokehityksestä. Verkkokehitykseen sijoittuva nopean kehityksen malli on melkein välttämätön nykymaailmassa, sillä pysyäkseen kilpailukykyisenä yrityksillä on oltava toimiva ja vuorovaikutteinen verkkosivusto. [12.]

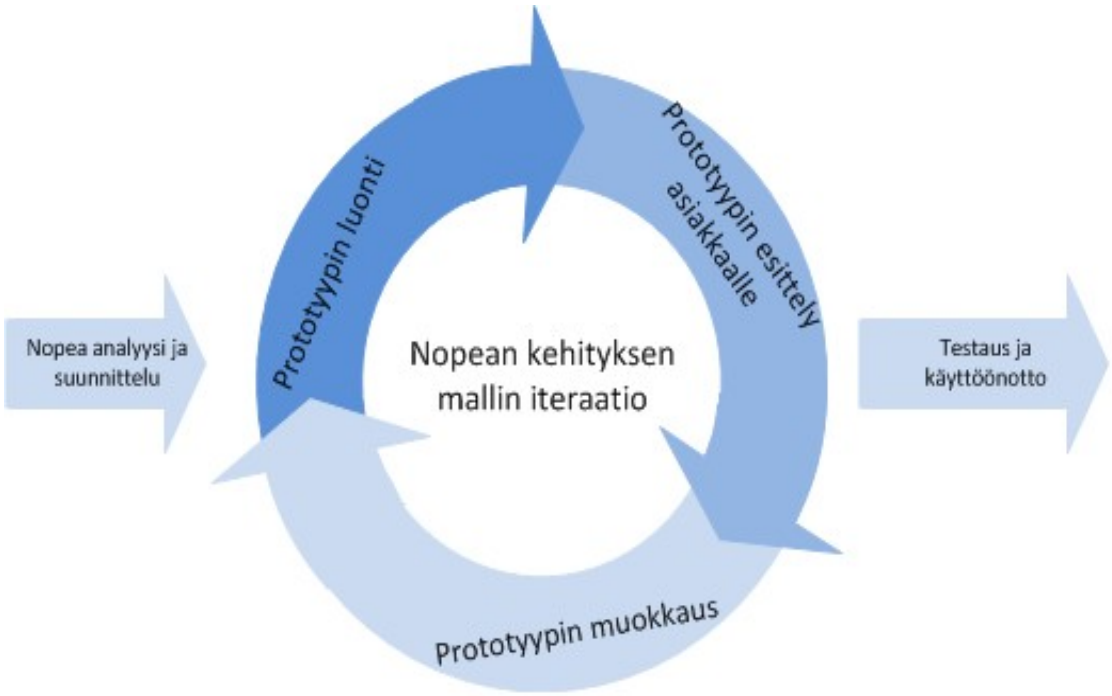
Nopean kehityksen malli on parhaimmillaan silloin, kun sovellus on itsenäinen, sovelluksella on tarkka aikataulu ja sen performanssi ei ole kriittinen. Nopean kehityksen mallin etuina verrattuna perinteiseen ohjelmointiin voivat oikein käytettynä olla parempi joustavuus, asiakkaan suurempi osallistuminen kehityksen aikana palautetta antamalla ja lyhyempi kehitysaika. Nopean kehityksen mallin haittapuolina saattaa aikaansaatu järjestelmä olla vähemmän tehokas ja siinä voi olla enemmän virheitä. Jopa järjestelmän ominaisuuksia on saatettu joutua karsimaan. Haittapuolet korostuvat etenkin siinä tapauksessa, jossa nopean kehityksen mallia on käytetty vääriin tavoitteisiin projektissa. [12.]

Eräs esimerkki, jolla nopean kehityksen mallin teho saattaisi vaarantua, on seuraavanlainen tilanne: prototyyppiä luotaessa syklin aikana keskitytään vähemmän tärkeää toimintoon, jolloin toimivan prototyypin valmistuminen ajallaan vaarantuu, sillä kriittisem-

pi toiminto saattaisi jäädä toteuttamatta. Toinen esimerkki on tilanne, jossa lopullisen tuotteen laatua aloitetaan uhraamaan heti alusta alkaen. Tällöin saattaa käydä niin, että prototyyppi ei enää toimikaan myöhemmässä vaiheessa ja aikaa kuluu laadun puutteesta aiheutuvien virheiden korjaamiseen.

Nopean kehityksen malli on lyhennetty ohjelmistokehitysprosessi, jossa suunnitteluvaiheeseen käytettyä aikaa vähennetään. Pohjimmainen periaate nopean kehityksen mallin takana on se, että tietyissä tilanteissa tuotteen valmistuminen mahdollisimman nopeasti on tärkeämpää kuin valmiin tuotteen laatu tai muut puutteet, joita syntyy epätarkan suunnittelun tuloksena. Tilanne, jossa tuotteen nopea valmistuminen on tärkeämpää, voi olla esimerkiksi sellainen, jossa kilpaileva yritys on myös luomassa samankaltaista tuotetta. Tällöin ensimmäisenä markkinoille päässyt tuote saa enemmän huomiota. Mahdollisia puutteita voidaan korjata julkaisun jälkeen. Lyhyt kehitysaika melkein aina johtaa laadun ja toimintojen vähenemiseen. [13.]

Nopean kehityksen mallin käyttäjille aikataulussa pysyminen on tärkeää. Nopean kehityksen mallissa käytetään iteratiivista prototyyppimenetelmää, jossa prosessi toistetaan, kunnes valmis sovellus on luotu. Prosessi koostuu seuraavista askelista. Analysoidaan asiakasvaatimukset ja tehdään niiden perusteella nopea suunnitelma. Luodaan toimiva prototyyppi. Valmistuneen prototyypin jälkeen tarkistetaan, että prototyyppi on sellainen, kuin asiakas on halunnut. Tämän jälkeen prototyyppiä testataan käyttäjillä. Testauksen jälkeen kehittäjät ja asiakkaat tapaavat keskustellakseen prototyypin ongelmista tai mahdollisista parannusehdotuksista. Ongelmien löytymisen jälkeen osapuolet sopivat ajalliset puitteet, jossa muutokset on tehtävä prototyyppiin. Tämä prosessi on kuvattu kuvassa 3. [13.]



Kuva 3: Tuotteen luonti prototyypin avulla nopean kehityksen mallissa

Prototyyppi on ennen lopullista tuotetta luotu mallikappale, jossa on tarkoitus testata tuotetta ja sen toimivuutta. Ohjelmistotuotannossa se soveltuu erityisesti uusien ratkaisujen ja asiakasvaatimusten testaamiseen. Valmis prototyyppi voidaan joko muokata lopulliseksi tuotteeksi tai sen avulla voidaan määritellä seuraava luotava prototyyppi. [14, s. 42-43.]

Nopean kehityksen mallissa suhde nopeuden ja laadun välillä riippuu lopullisista projektin määritelmistä. Mitä nopeammin valmista tuotetta tarvitaan, sitä enemmän laadusta joudutaan tinkimään. Yleensä nopeampi ohjelmistokehitys vaatii myös enemmän rahallista kustannusta. [13.]

Nopean kehityksen mallissa lopulliset käyttäjät ovat mukana kehityksen aikana, kun perinteisissä prosesseissa käyttäjäpalautetta saadaan usein vasta valmiista tuotteesta. Prototyyppimallin asiakaskatselmukset tapahtuvat prototyypin valmistuttua, jonka jäl-

keen alkaa uusi sykli, jossa luodaan taas prototyyppi halutuin muokkauksin tai muokataan prototyyppi valmiiseen muotoon. Nopean kehityksen mallin avulla kehitetyt tuotteet ovat usein helpompia muokata johtuen siinä käytetystä yksinkertaisesta suunnitteluprosessista. Myös asiakaspalautteessa saadut ongelmat ja ehdotukset ovat helpompia korjata ja toteuttaa prototyyppimallin käytön takia. [13.]

Nopean kehityksen mallilla luodut sovellukset saattavat olla vähemmän tehokkaita ja niissä on enemmän ohjelmointivirheitä. Niiden toiminnallisuudet ja tehokkuus ovat saattaneet vähentyä alkuperäisestä suunnitelmasta. Ammattimainen näkymä ja käyttäjäkokemus voi olla puutteellinen johtuen lyhentyneestä ajasta, jota on käytetty kehitykseen ja erityisesti suunnitteluun. [13.]

3.3.2 Nopean kehityksen malli vs ketterät menetelmät

Ketterät menetelmät ja nopean kehityksen malli muistuttavat toisiaan suuresti. Molemmissa kehitys tapahtuu iteraatioissa ja niissä käytetään tarkkaa aikataulutusta. Näissä on kuitenkin seikkoja, jotka erottavat menetelmät toisistaan.

Ketterissä menetelmissä ei ole prototyyppejä, joita käytetään nopean kehityksen mallissa. Nopean kehityksen mallissa suunnitellaan ja muokataan prototyyppejä käyttökelpoiseksi koodiksi iteraatioiden aikana. Tämän onnistuminen ei ole taattavissa. Ketterissä menetelmissä lopullinen tuote jaetaan toimiviksi osakokonaisuuksiksi, jotka toteutetaan iteraatioissa. Nopean kehityksen mallissa kehittäjät pyrkivät tuottamaan prototyypin sovelluksesta, jossa on kaikki haluttu toiminnallisuus. Toiminnallisuuksien koodi on kuitenkin toteutettu nopeasti ja se saattaa olla huolimatonta. Seuraavissa iteraatioissa sitä pyritään parantamaan käyttökelpoiseksi. [15.]

Ketterissä menetelmissä tuotantotiimin jäsenet saavat äänensä kuuluville lopullisen tuotteen toteutukseen liittyen, mutta sillä on silti selkeä johtaja, joka pitää huolen kehityksen etenemisestä. Tiimin jäsenet vastaavat pitkälti itsestään ja työnteosta. Nopean kehityksen mallissa kommunikointi tiimin jäsenten välillä on jätetty tiimin harteille. Ketterissä menetelmissä tiimi kommunikoi keskenään tiiviisti ja suunnittelee tuotetta yh-

dessä. Nopean kehityksen mallissa tiimiläisillä on tapana työskennellä itsekseen, jolloin lopullinen ohjelmakoodi on sekavaa ja vaikeasti ylläpidettävää. Nopean kehityksen mallissa tiimillä on tekninen johtaja, joka on hieman kokeneempi ohjelmoija, ja projektimanageri, joka vastaa tiimin ei-teknisistä asioista. Heidän tulee projektin alussa sopia vastuista, sillä jotta ei syntyisi aluieta, joista molemmat tuntevat olevansa vastuussa ja alueita, joista kumpikaan ei koe olevansa vastuussa. [15; 16, s. 313-314.]

Ketterissä menetelmissä suositaan erilaisia praktiikoita, kuten esimerkiksi testivetoinen kehitys, jossa virheet suunnittelussa tai ohjelmakoodissa korostuvat niin, että ne voidaan korjata mahdollisimman aikaisessa vaiheessa. Tiimin jäsenet voivat tällöin tehdä suuriakin muutoksia koodiin ilman pelkoa siitä, että tuote ei enää toimikaan halutulla tavalla. Vastaavia menetelmiä ei käytetä Nopean kehityksen mallissa. [15.]

Ketterien menetelmien tuotokset esitellään valmiina tuotteena. Nopean kehityksen mallissa esitellään prototyyppejä. Tässä on eräs prototyyppimallin kompastuskivistä. Esiteltäessä prototyyppiä, joka toimii päällisinpuolin niin kuin halutaan, asiakkaalle jää kuva, että tuote on valmis. Asiakas ei välttämättä ymmärrä, että pinnan alla sovellus on vielä kriittisesti kesken. [14, s. 314 ;15.]

Ketterissä menetelmissä tiimeihin voi kuulua testaajia ja käyttökokemukseen keskittyviä henkilöitä. Nopean kehityksen mallissa tiimit koostuvat yleensä ohjelmoijista, jotka voivat olla erikoistuneita johonkin tiettyyn alueeseen, kuten tietokannat tai käyttöliittymät. Tällöin valmiissa tuotteessa esimerkiksi käyttöliittymä, vaikkakin toimiva, ei välttämättä käyttäjäystävällisesti suunniteltu. Käyttäjät eivät halua käyttää tylsää tai epäselvää sovellusta, mikäli vain voivat asiaan vaikuttaa. [15.; 16, s. 304-313.]

Nopean kehityksen mallissa tehdään paljon kompromisseja toiminnallisuuteen ja asiakkaan haluamiin toimintoihin, jotta tuote saadaan valmiiksi nopeammin. Ketterissä menetelmissä sen sijaan iteraatioissa pyritään lisäämään muutoksia ja parannuksia, eikä käytössä ole prototyyppejä. [15.]

3.3.3 Nopean kehityksen mallin työvälineet

Nopean kehityksen mallin työvälineet pyrkivät auttamaan kehittäjää luomaan sovelluksia nopeasti automatisoimalla mahdollisimman paljon kehitysprosessia ja poistamalla toistuvia työtehtäviä. Joissakin työvälineissä tämä tarkoittaa sitä, että ohjelmointikoodin kirjoittamista vähennetään. Esimerkkinä tällaisesta ovat kehitysympäristö Qt, jossa voidaan esimerkiksi käyttöliittymää luodessa asettaa valmiita toimintoja käyttöliittymäkomponentteihin signaalien ja slottien avulla. Joissain ohjelmointikoodin kirjoittamisesta päästää kokonaan eroon. Tällainen on esimerkiksi LyteCube-nimisen yrityksen LyteRAD-ohjelma (http://www.lytecube.com/index.html). LyteRAD muistuttaa lupauksiltaan hieman Loihdinta.

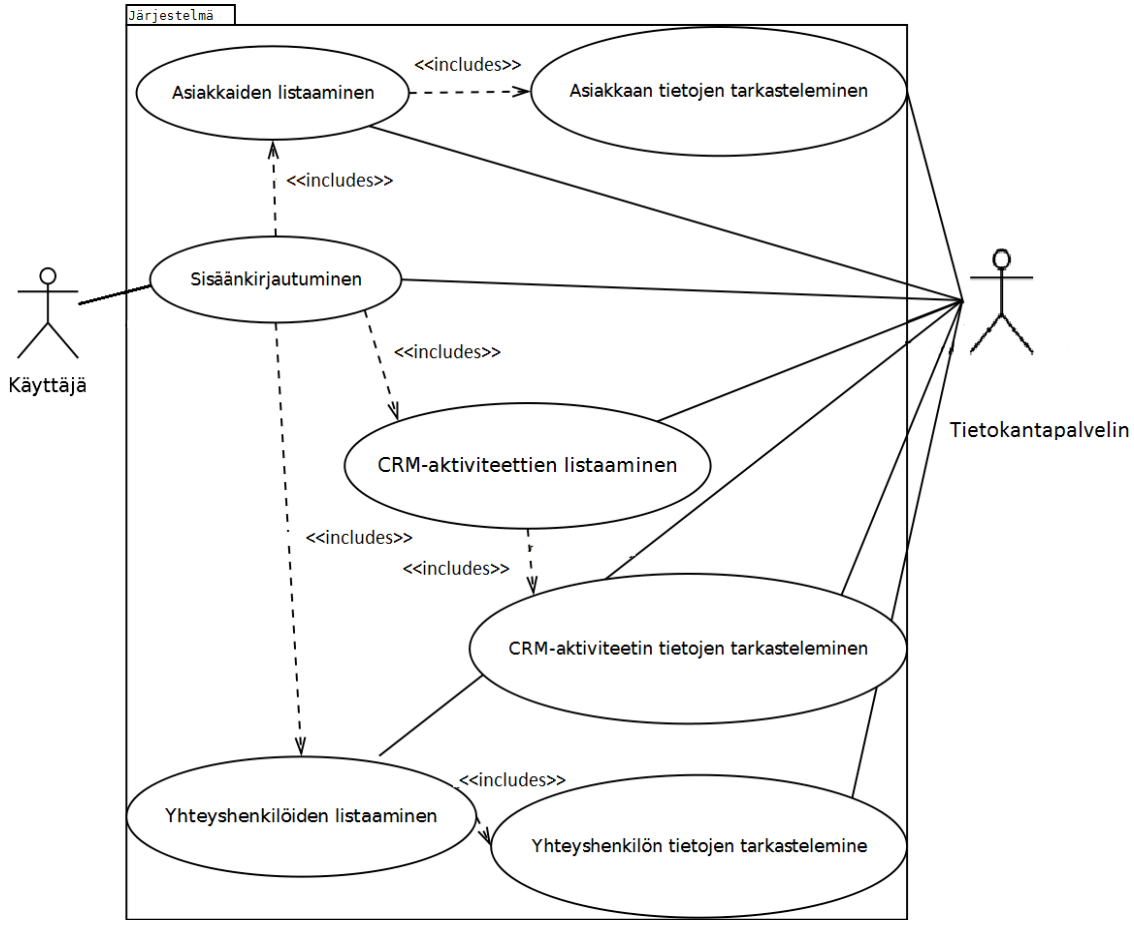
Nopean kehityksen mallissa on yhtäläisyyksiä Loihdin-työkalun kanssa. Loihtimen on luvattu lyhentävän kehitysaikaa, mikä on nopean kehityksen mallin päätavoite. Kuten aiemmin on kerrottu, nopean kehityksen mallin työkaluilla on mahdollista luoda interaktiivisia verkkosivuja ja sovelluksia ilman kokemusta ohjelmoinnista, jota myös Loihdin lupaa.

4 Toteutettavan yrityssovelluksen määrittely

Tässä luvussa tutustutaan tarkemmin luotavien yrityssovellusten ominaisuuksiin. Käyttötapauksissa käydään läpi, mitä sovellusten odotetaan tekevän. Käyttötapaukset käydään läpi myös taulukon muodossa, joista tulisi tulla ilmi tarkemmin, mitä käyttötapaukset pitävät sisällään. Sekvenssikaavio sovelluksen käytöstä näyttää pääpiirteittäin, kuinka sovellus kommunikoi käyttäjän ja tietojärjestelmäpalvelimen välillä tapauksessa, jossa käyttäjä haluaa listan kaikista hänelle kuuluvista asiakkaista.

4.1 Käyttötapaukset

Kuvassa 4 on kuvattu käyttötapaukset, jotka esiintyvät vertailua varten luotavissa yrityssovelluksissa. Järjestelmä hakee tietonsa tietokantapalvelimelta. Palvelimelta löytyy paljon käyttäjää kiinnostavia tietoja, mutta Android-sovelluksessa kiinnostavia tietoja ovat vain käyttäjäkohtaiset asiakkaat, yhteyshenkilöt ja CRM-aktiviteetit. Muita tietoja voidaan käsitellä Loihtimen luomissa web-sovelluksissa. CRM-aktiviteetit ovat käyttäjän omia työhön liittyviä tehtäviä, kuten soitot ja tapaamiset. Seuraavassa on eriteltynä kaaviossa kuvatut käyttötapaukset. Käyttötapaukset ovat taulukkomuodossa. Niissä kerrotaan käyttötapauksen nimi, kuka sitä tulee suorittamaan, mahdolliset esiehdot, kuvauksen, mitä tarkalleen tilanteessa tapahtuu, mikä on lopputulos ja minkälaisia mahdollisia poikkeuksia saattaa esiintyä. Poikkeuksissa kerrotaan, kuinka sovelluksen tulisi toimia poikkeustilanteessa.



Kuva 4: Sovelluksen käyttötapauskaavio

Nimi	Sisäänkirjautuminen
Suorittajat	Käyttäjä
Esiehdot	Sovellus on asennettuna Android-älypuhelimeen.
Kuvaus	Käyttäjä antaa sovellukselle käyttäjätunnuksensa ja salasanansa.
Lopputulos	Käyttäjä on kirjautuneena sisään ja hän näkee toimintovalikon.
Poikkeukset	Käyttäjä ei pääse kirjautumaan sisään.

Nimi	Asiakkaiden listaaminen
Suorittajat	Käyttäjä
Esiehdot	Käyttäjän sisäänkirjautuminen on onnistunut.
Kuvaus	Käyttäjä valitsee toimintovalikosta asiakkaiden listaukseen johtavan painikkeen. Sovellus siirtyy listausnäkymään.
Lopputulos	Käyttäjä näkee listan asiakkaistaan Adensy Oy:n CRM-järjestelmässä.
Poikkeukset	Käyttäjällä ei ole Adensy Oy:n CRM-järjestelmässä asiakkaista järjestelmässä, jolloin lista on tyhjä.

Nimi	Asiakkaiden tietojen tarkastaminen
Suorittajat	Käyttäjä
Esiehdot	Käyttäjä on saanut listan asiakkaista.
Kuvaus	Käyttäjä valitsee asiakkaan listasta. Sovellus siirtyy yksittäisen asiakkaan tietojen näkymään.
Lopputulos	Käyttäjä näkee valitsemansa asiakkaan Adensy Oy:n CRM-järjestelmään kirjatut tiedot.
Poikkeukset	

Nimi	Yhteyshenkilöiden listaaminen
Suorittajat	Käyttäjä
Esiehdot	Käyttäjän sisäänkirjautuminen on onnistunut.
Kuvaus	Käyttäjä valitsee toimintovalikosta yhteyshenkilöiden listaukseen johtavan painikkeen. Sovellus siirtyy listausnäkymään.
Lopputulos	Käyttäjä näkee listan yhteyshenkilöistään Adensy Oy:n CRM-järjestelmässä.
Poikkeukset	Käyttäjällä ei ole järjestelmässä yhteyshenkilöitä, jolloin lista on tyhjä.

Nimi	Yhteyshenkilön tietojen tarkastaminen
Suorittajat	Käyttäjä
Esiehdot	Käyttäjä on saanut listan yhteyshenkilöistä.
Kuvaus	Käyttäjä valitsee yhteyshenkilön listasta. Sovellus siirtyy yksittäisen yhteyshenkilön tietojen näkymään.
Lopputulos	Käyttäjä näkee valitsemansa yhteyshenkilön tiedot.
Poikkeukset	

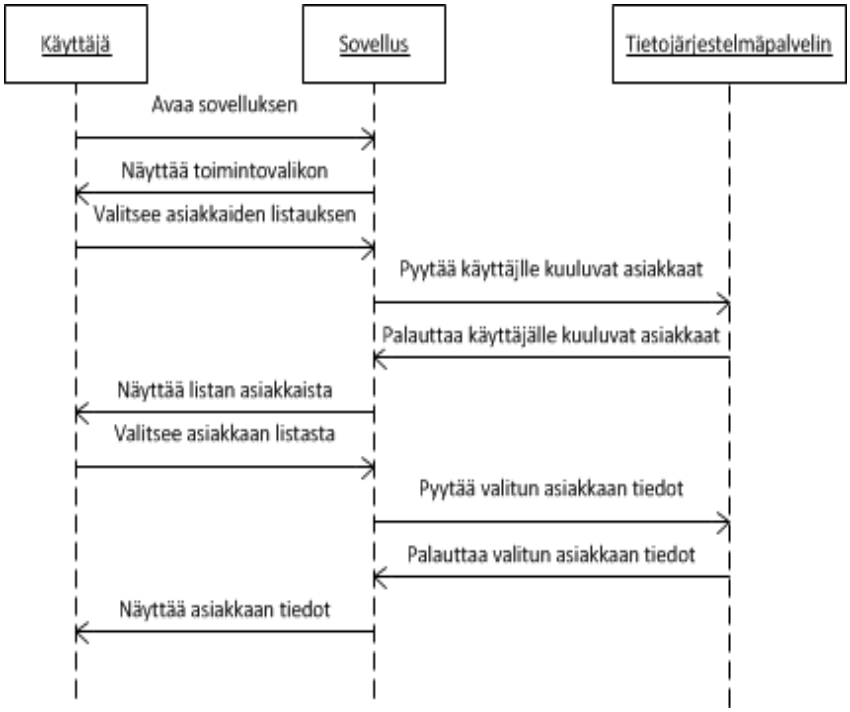
Nimi	CRM-Aktiviteettien listaaminen
Suorittajat	Käyttäjä
Esiehdot	Käyttäjän sisäänkirjautuminen on onnistunut.
Kuvaus	Käyttäjä valitsee toimintovalikosta CRM-aktiviteettien listaukseen johtavan painikkeen. Sovellus siirtyy listausnäkymään.
Lopputulos	Käyttäjä näkee listan CRM-aktiviteeteista, jotka ovat kirjattuna Adensy Oy:n CRM-järjestelmässä.
Poikkeukset	Käyttäjällä ei ole CRM-aktiviteetteja kirjattuna Adensy Oy:n CRM-järjestelmään.

Nimi	CRM-aktiviteetin tietojen tarkastaminen
Suorittajat	Käyttäjä
Esiehdot	Käyttäjä on saanut listan CRM-aktiviteeteista.
Kuvaus	Käyttäjä valitsee CRM-aktiviteetin listasta. Sovellus siirtyy yksittäisen CRM-aktiviteetin tietojen näkymään.
Lopputulos	Käyttäjä näkee valitsemansa aktiviteetin tiedot.
Poikkeukset	

4.2 Sekvenssikaavio sovelluksen käytöstä

Kuvassa 5 kuvataan kahta toimintoa yrityssovelluksessa sekvenssikaavion muodossa. Siinä tekijöinä ovat käyttäjä, sovellus ja tietojärjestelmäpalvelin. Käyttäjä on sovellusta käyttävä henkilö, jolla on käyttäjätunnukset tietojärjestelmäpalvelimelle. Kuvan 5 kaaviossa oletetaan, että käyttäjä on jo kirjautunut sisään. Kuvassa 5 käyttäjä valitsee asiakkaiden listauksen ja valitun asiakkaan tietojen tarkastelun, mutta samalla tavalla käyttäytyvät myös yhteyshenkilöt ja CRM-aktiviteetit.

Käyttäjän aloittaessa käyttämään sovellusta, hän näkee toimintovalikon, jossa on painikkeet asiakkaiden, yhteyshenkilöiden ja CRM-aktiviteettien listaukseen. Kuvassa 5 käyttäjä valitsee asiakkaiden listauksen. Tällöin sovellus lähettää pyynnön tietojärjestelmäpalvelimelle, jossa se pyytää käyttäjälle kuuluvia asiakkaita. Sovellus käyttää tähän käyttäjän sisäänkirjautuessa antamaa käyttäjätunnusta ja salasanaa. Palvelin palauttaa asiakkaiden tiedot, ja sovellus listaa asiakkaiden nimet käyttäjälle luettaviksi. Tämän jälkeen käyttäjä valitsee listasta asiakkaan, josta hän haluaa tiedot palvelimelta. Sovellus lähettää pyynnön palvelimelle, joka palauttaa yksittäisen asiakkaan tiedot. Sitten sovellus näyttää omassa näkymässään asiakkaan ne tiedot, jotka se palvelimelta sai.



Kuva 5: Sekvenssikaavio sovelluksen käytöstä

5 Yrityssovelluksen kehitys Androidille perinteisellä ohjelmoinnilla

Tämän insinöörityön tavoitteena on tutkia, millä tavalla kehitys eroaa, kun ohjelmoidaan pieni sovellus yrityksen asiakkuudenhallintaan perinteisesti ja malliperustaisen kehityksen lähestymistapaa käyttäen. Tässä luvussa käydään läpi sovelluksen kehitys perinteisellä ohjelmoinnilla. Sovellus koodattiin käyttämällä Eclipse Ganymede -kehitysympäristöä, johon on asennettu Android SDK (Software Development Kit). Ajoissa, jotka mainitaan on minuutin virhemarginaali, sillä sekunnin tarkkaa ajanlaskentaa ei katsottu tarpeelliseksi.

Ensimmäiseksi sovellusta kehittäessä luotiin aktiviteetti sisäänkirjautumista varten. Käyttöliittymän luontiin apuna käytettiin Android SDK:n mukana tullutta käyttöliittymä-editoria. Android-ympäristössä käyttöliittymät pääsääntöisesti luodaan käyttämällä xml-tiedostoja, joissa komponentit ja niiden ominaisuudet on määritelty. Editoria on tässä työssä käytetty luomaan pohja käyttöliittymälle, mutta lopulliset määritykset on muokattu xml-tiedostoon käsin. Käyttöliittymällä ei tässä vaiheessa ole vielä toiminnallisuutta, mutta sen ulkonäkö on jo valmis. Aikaa tämän luomiseen meni suunnilleen 5 minuuttia. Käyttöliittymän komponenttien alustukseen ja kuuntelijan lisäämiseen aikaa meni 25 minuuttia.

Kuvassa 6 on kuvakaappaus ensimmäisen aktiviteetin käyttöliittymästä. Kuuntelijan toimena on reagoida sisäänkirjautumis-painikkeen aktivoitumiseen. Tällöin käyttöliittymän tekstikentistä otetaan muistiin käyttäjän antamat käyttäjätunnukset. Tunnuksien avulla sovellus voi kirjautua asiakkuudenhallintapalvelimelle. Jos tunnukset ovat olemassa palvelimella ja kirjautuminen onnistuu, siirtyy sovellus seuraavaan aktiviteettiin. Jotta uusi aktiviteetti toimisi, piti kyseinen aktiviteetti lisätä AndroidManifest.xml-tiedostoon. AndroidManifest.xml-tiedostossa ovat kirjattuna kaikki Android-sovelluksen aktiviteetit ja muut komponentit. Tähän aikaa meni 30 minuuttia, eli kokonaisuudessaan ensimmäisen aktiviteetin luontiin aikaa meni yhteensä tunnin verran.

Kuva 6: Sisäänkirjautumisaktiviteetin käyttöliittymä

Kuva 7: Käyttöliittymäkuva valikkoaktiviteetista

Seuraavana työssä luotiin valikko, jolla käyttäjä voi valita, mitä hän tarkastelee kolmesta vaihtoehdosta, jotka olivat asiakkaat, yhteyshenkilöt ja CRM-aktiviteetit. Valikolle on oma aktiviteettinsa, jonka käyttöliittymään luotiin 3 painiketta kutakin vaihtoehtoa kohden. Painikkeille asetettiin yksi kuuntelija. Kuuntelija tunnistaa, mitä painiketta käyttäjä painaa, ja siten osaa käynnistää käyttäjän haluaman aktiviteetin. Valikkonäkymä on kuvassa 7.

Tämän jälkeen sovellukseen kehitettiin asiakkaan listausaktiviteetti. Tässä kohdassa tuli ratkaista, kuinka tietojärjestelmäpalvelimeen otetaan yhteyttä ja kuinka lopulta saadut tiedot, eli lista käyttäjäkohtaisista asiakkaista, saadaan asetettua Androidin ListView-käyttöliittymäkomponenttiin. Asiakkaat haetaan käyttämällä rajapintakutsua. Jotta kyseinen yhteydenotto onnistuisi, tarvitsee rajapintakutsu mukaansa käyttäjätunnuksen, salasanan sekä id-tunnuksen, jonka sisäänkirjautuminen palautti sovellukselle. Näiden tallennus ja hakeminen käsitellään vähän myöhemmin.

Tietokantahaun onnistuttua sovellukselle palautuu lista asiakkaista xml-muodossa. Saadusta xml-tiedosta muodostetaan olioita, jotka annetaan ArrayList-komponentille. Samalla kun ArrayList-komponenttia täytetään, tallennetaan asiakkaan nimi ja id-tunnus myöhempää käyttöä varten. Listausaktiviteetti perii normaalin Activity-olion sijaan ListActivity-olion. Tämä aktiviteetti on erikoistunut käsittelemään listoja. Se sisältää automaattisesti ListView-käyttöliittymäkomponentin. Tästä syystä tavallisen listan luonti on yksinkertaista. *SetListAdapter()*-metodille annetaan ArrayAdapter-olio, joka saa parametrinaan aktiviteetin kontekstin, xml-määrityksen yhden lista-alkion ulkomuodolle sekä taulukon listaan asetettavista String-arvoista. Tämän jälkeen ListView-komponentille asetetaan onItemClickListener-kuuntelijaolio, joka reagoi lista-alkioita painettaessa. Kuuntelijan onItemClick()-metodi saa parametrina mm. sen näkymä-olion, jota on pai-

nettu. Tässä tapauksessa kyseessä on TextView-olio, josta saadaan selville käyttäjän haluaman asiakkaan nimen. TextView-olio on määritelty lista-alkion ulkomuodoksi. Tämän asiakkaan id-tunnus haetaan selvillesaadun nimen perusteella ja annetaan intentio-oliolle talteen. Intentio-olion avulla käynnistetään uusi aktiviteetti.

Kuvassa 8 on näkyvillä aikaansaatu lista. Sillä sovellukseen on kirjauduttu demotunnuksilla, palvelin palauttaa näille tunnuksille kehitetyt, päästä keksityt asiakkaat. Tämän kokonaisuuden kehittämiseen aikaa kului 2 tuntia 30 minuuttia.

Kuva 8: Asiakkaiden listaus palvelimelta haetulla demodatalla

Intentio-olion avulla käynnistetyssä aktiviteetissa on tarkoitus listata asiakkaan tiedot. Rajapintakutsua varten on tiedettävä halutun asiakkaan id-tunnus. Uusi aktiviteetti saa tämän selville siitä intentio-oliosta, jolla se käynnistettiin. Jotta tietojen tarkastelu olisi järkevää, luotiin lista-alkioista kaksirivisiä. Tämä osio tuotti hankaluuksia kehityksen aikana. Kuten edellisessä aktiviteetissa, tässäkin luokka peri ListActivity-olion. Tietojen asettaminen kaksiriviseen lista-alkioon oli monimutkaisempaa. Tätä varten oli luotava lista-alkion määritystiedosto. Tässä tapauksessa se nimettiin client_list_item.xml, joka sijaitsee layout-kansiossa muiden käyttöliittymätiedostojen joukossa. Tämä kansio on Android-järjestelmän vaatima, josta se osaa etsiä käyttöliittymiä. Kansioon on myös tallennettu edellisen aktiviteetin lista-alkion määritys.

Koodiesimerkki 1 on aiemmin mainittu client_list_item.xml. Tässä näkyy, miten kaksirivinen lista-alio on koottu. Se koostuu kahdesta TextView-oliosta yhden LinearLayoutin sisällä. LinearLayout-olion ominaisuuksien huomionarvoisin osa on orientaation määritys, joka tässä on määritetty pystysuuntaiseksi. Tällöin TextView-oliot sijoittuvat lista-

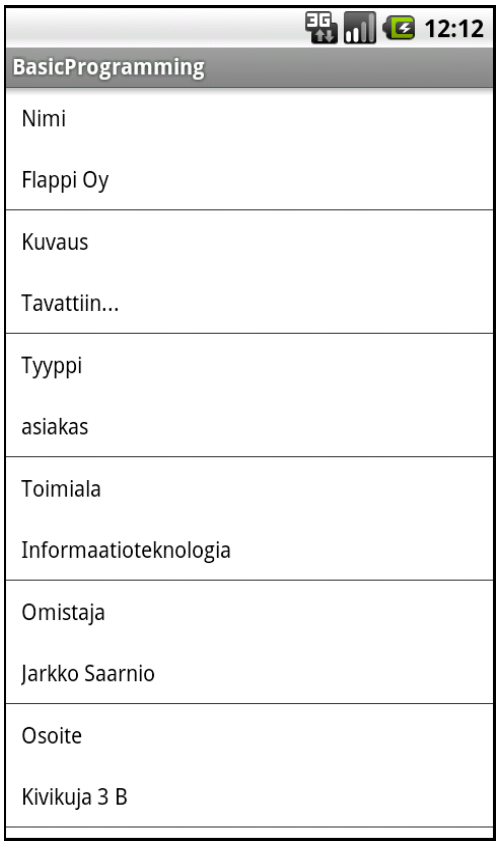
alkiossa päällekäin. TextView-olioiden määritykset sisältävät id:n asettamisen ja ulkomuodon muokkaamisen. Molempia id:itä käytetään SimpleAdapterissa, johon palataan myöhemmin. Ulkomuoto on tässä tapauksessa varsin yksinkertainen, sillä näihin ei haluttu käyttää aikaa.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.and-
roid.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent">
      <TextView android:id="@+id/first_line"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:background="#FFFFFF"
            android:padding="10dp"
            android:textColor="#000000">
      </TextView>
      <TextView android:id="@+id/second_line"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:background="#FFFFFF"
            android:padding="10dp"
            android:textColor="#000000">
      </TextView>
</LinearLayout>
```

Koodiesimerkki 1: Kaksirivisen lista-alkion xml-määritys

Halutut tiedot saatiin asetettua listaan ja lista-alkioihin käyttämällä SimpleAdapter-nimistä adapteria. Tämän hahmottaminen vei ylivoimaisesti eniten aikaa. SimpleAdapterin konstruktori haluaa parametreikseen Context-olion, Map-olioilla täytetyn ArrayList-olion, lista-alkion xml-määrityksen, lista-alkion TextView-olioiden id:t sekä String-taulukon, joka sisältää Map-olioiden avaimet. Tietojen haun aikana nämä Map-oliot siis täytettiin avain-arvo-pareilla ja lisättiin ArrayList-olioon. Map-olion avain kertoi sen, kummalle riville sen arvo tuli. Esimerkiksi ylemmälle riville sijoitettiin suomenkielinen asiakkaan attribuutti, kuten ”Nimi”, ja alemmalle riville itse attribuutti, kuten ”Asiakas Oy”, jossa Asiakas Oy on asiakasyrityksen nimi. Suomenkielisiin attribuutteihin käytettiin Androidin strings.xml-tiedostoa, johon kirjattiin string-arvoja englanniksi. Vastaavasti luotiin strings-FI.xml-tiedosto, jossa vastaavat string-arvot olivat suomeksi. Tällä tavalla attribuutit vaihtuvat puhelimen sijainnin mukaan joko suomeksi tai englanniksi. Tähän aktiviteettiin aikaa kului noin 2 tuntia.

Kuvassa 9 on esimerkki erään keksityn yrityksen tiedoista. Mikäli puhelin sijaitsisi muualla kuin Suomessa, olisi Nimi, Kuvaus ja muut kentät englanniksi. Koska nämä kuvakaappaukset on otettu sovelluksesta sen ollessa käynnissä emulaattorissa, tuli sen asetuksista vaihtaa sijainti erikseen Suomeen.



Kuva 9: Demoasiakkaan tiedot

Yhteyshenkilöiden ja CRM-aktiviteettien listaus ja tietojen tarkastelu -aktiviteettien tapahtuvat teknisesti hyvin samalla tavalla asiakkaiden listaus. Vastaavien listausten luomiseen aikaa kului 30 minuuttia. Tietojen tarkastelu -aktiviteettien luomiseen meni 1 tunti 30 minuuttia, vaikka logiikka ei muuttunut eikä uutta tapaa hoitaa tietojen hakua ja listausta tarvinnut keksiä. Intentio-oliolle välitettävässä id-tunnuksessa oli kirjoitusvirhe, jonka etsimiseen kului suurin osa ajasta.

Aiemmin mainittiin käyttäjätunnusten ja vastaavan id-tunnuksen tallentaminen. Tämä tapahtuu Android-kirjastossa olevan Application-luokan avulla. Aliluokka perii tämän. AndroidManifest.xml:n tulee lisätä kyseinen luokka application-elementin nimi-osaan. Tämän näkee koodiesimerkissä 2. Kyseisessä esimerkissä luokan nimi on ”CredsHolder”. Tällöin Android-järjestelmä luo instanssin kyseisestä luokasta, ja siihen instanssiin pääsee käsiksi kaikkialta sovelluksesta käyttämällä aktiviteetin context-olion *getApplicationContext()*-metodia. Tämä tapa vastaa Singleton-suunnittelumallia, mutta käyttää Android-järjestelmän metodeja.

```
<application android:name=".CredsHolder"
   android:icon="@drawable/icon"
   android:label="@string/app_name">
   <activity android:name=".MainActivity"
             android:label="@string/app_name">
     <intent-filter>
       <action android:name="android.intent.action.MAIN" />
       <category
android:name="android.intent.category.LAUNCHER" />
     </intent-filter>
   </activity>
   <activity android:name=".MenuActivity" />
   <activity android:name=".ClientList" />
   <activity android:name=".ContactList" />
   <activity android:name=".CRMActivityList" />
   <activity android:name=".ClientActivity" />
   <activity android:name=".ContactActivity" />
   <activity android:name=".CRMActivityActivity" />
</application>
```

Koodiesimerkki 2: AndroidManifest.xml-tiedoston Application-elementti

Koodiesimerkissä 2 on kuvattuna koko application-elementti. Ensimmäisen tagin sisässä on määritelty nimen lisäksi myös se kuvake, joka näkyy Android-puhelimen valikossa sitten, kun sovellus on asennettu, sekä sovelluksen käyttäjälle näkyvä nimi. Application-elementin sisälle on määritelty kaikki sovelluksen aktiviteetit. Aktiviteetit pitää määrittää, jotta niitä voidaan käyttää sovelluksessa. Aktiviteetti-elementit voisivat sisältää enemmän attribuuttien määrityksiä, mutta kyseisessä sovelluksessa niitä ei tarvita. Ensimmäinen aktiviteetti-elementti esimerkissä kuvaa MainActivity-aktiviteettia, jossa käyttäjä kirjautuu sisään.

Aktiviteetilla on intentiosuodatin, johon on määritetty action- ja category-elementit. Intentio-suodatin, jonka action-elementti on määritetty ”android.intent.action.MAIN” ja category-elementti ”android.intent.category.LAUNCHER” kertovat järjestelmälle, että tämä aktiviteetti on se näkymä, jonka käyttäjä saa ensimmäisenä eteensä sovelluksen käynnistyttyä. Lopuista aktiviteetti-elementeistä puuttuu intentiosuodatin, joka tarkoittaa sitä, että näille voidaan lähettää ainoastaan eksplisiittisiä intentioita, eli lähetetyn intention on tiedettävä tarkalleen vastaanottajansa. Tässä sovelluksessa se riittää. Mikäli niistä kuitenkin haluttaisiin kykeneviä vastaanottamaan implisiittisiä intentioita, jouduttaisiin activity-elementteihin lisäämään intentiosuodattimet ja niihin category-elementit nimellä ”android.intent.category.DEFAULT”.

6 Yrityssovelluksen kehitys Androidille Loihdinta käyttäen

Loihtimen käyttöä varten kehittäjällä on oltava käyttäjätunnukset palveluun. Ensimmäiseksi on luotu MySQL-kanta niitä tietoja varten, joita valmiissa sovelluksessa tullaan käyttämään. Tämä kanta on liitetty Loihtimeen. Kanta on palvelimella, josta sovellus hakee käyttäjän pyytämät tietonsa verkon yli. Tässä työssä ei esitellä Loihtimen käyttöliittymää työnantajan pyynnöstä.

Sovelluksen kehittämisen ensimmäinen osa on palveluun kirjautuminen. Tämän jälkeen valittiin toiminto käyttöliittymästä, jolla luodaan uusi sovellus. Uutta sovellusta varten oli luotava dataluokat, joita sovelluksessa käsitellään. Näitä olivat aiemmin mainitut asiakas, yhteyshenkilö ja CRM-aktiviteetit. Dataluokka on kaikkien määritysten jälkeen malli, jota käytetään käsittelemään tietokannasta saatuja tietoja. Mallin pohjalta Loihdin tietää, minkälaisia tietoja dataluokasta halutaan nähdä ja siten osaa rakentaa tarvitunlaiset käyttöliittymät. Tähän aikaa meni 5 minuuttia. Mitään konkreettista ei vielä ole saatu aikaiseksi.

Seuraavaksi dataluokkien tietokentät piti määritellä. Tämä aloitettiin yhteyshenkilöstä. Aluksi luotiin kieliasetus, jolla saatiin määritettyä dataluokan suomen- ja englanninkielinen nimi sekä yksikössä että monikossa. Aikaa tähän meni korkeintaan 1 minuutti. Tämän jälkeen voitiin dataluokalle määrittää tietokentät, jotka vastasivat dataluokan attri-

buutteja. Tietokentissä käytettiin ainoastaan tekstitietotyyppiä. Näille annettiin myös kuvausteksti, jolloin tietokentän selite tulee automaattisesti ohjelmasta, eikä sitä tarvitse hakea tietokannasta. Tietokenttien luontiin käytettiin aikaa 9 minuuttia. Tämä tarkoittaa sitä, että yhden dataluokan määritykseen aikaa meni yhteensä 10 minuuttia.

Edelliset asetukset toistettiin sekä asiakas- ja CRM-aktiviteetti-dataluokille. Aikaa asiakkaan kieliasetusten ja tekstikenttien luontiin kului yhteensä 8 minuuttia. CRM-aktiviteetti-dataluokan asetuksiin kului sama aika. Ero kahden edellisen ja yhteyshenkilö-dataluokan luonnin ajankulun kanssa johtui siitä, että näissä oli eri määrät tietokenttiä lisättynä. Asiakaalla ja CRM-aktiviteetilla oli 10 olennaista tietoa, joita sovelluksessa on haluttu näytettävän, kun taas yhteyshenkilöllä näitä oli 12. Olennaisia tietoja olivat mm. nimi, yhteystiedot ja kuvaus. Tämän vaiheen jälkeen kolmen mallin tiedot on määritelty. Jokainen tietokannasta löytyvät asiakas, yhteyshenkilö ja CRM-aktiviteetti noudattavat sitä vastaavaa mallia. Tämä tarkoittaa sitä, että tietokannasta haetaan mallissa määritettyjä tietoja automaattisesti, vaikka niitä ei sinne olisikaan kirjattu. Loihdin osaa luoda käyttöliittymät, joissa haetaan ainoastaan asiakkaalle liittyviä olennaisia tietoja, eikä esimerkiksi yhteyshenkilölle määritettyjä olennaisia tietoja haeta asiakkaan tietojen tarkasteluun tarkoitettuun käyttöliittymään.

Jokaiselle dataluokalle tuli myös määrittää käyttäjät ja oikeudet. Tällä kertaa sovelluksessa ei ollut mitään sellaista, johon käyttäjät tarvitsisivat erityislupia tehdäkseen jotain, joten kaikille käyttäjille annettiin samat oikeudet, joilla he voivat vapaasti tarkastella tietokannan palauttamia tietoja. Oikeuksien asettamiseen aikaa kului 3 minuuttia.

Seuraavaksi luotiin ne tiedostot, joita sovellus tarvitsee toimiakseen. Näissä xml-tiedostoissa oli määritettynä aktiviteettien käyttöliittymät ja käyttöliittymäkomponentteihin liittyvät toiminnallisuudet. Loihtimessa jokaiselle dataluokalle luotiin nämä tiedostot painamalla ”Generate UI” -nappia, joka avasi valikon. Valikosta valittiin Android-vaihtoehto, jolloin Loihdin loi kaksi käyttöliittymä- ja kaksi toiminnallisuus-määritystiedostoa. Näissä olivat yhtä dataluokkaa kohti käyttöliittymä- ja toiminnallisuus-tiedostot dataluokan listausta ja tietojen tarkastelua varten. Tämä tiedostojen generointi vei kaikenkaikkiaan yhden minuutin.

Tässä kohdassa valmiina olivat ne tiedostot niitä aktiviteetteja varten, joilla käyttäjät tarkastelivat tietokantapalvelimelta saatuja tiedostoja, kuten asiakaslistaus ja yhden valitun asiakkaan tiedot. Sovelluksesta puuttui vielä sisäänkirjautumisaktiviteetti ja valikkoaktiviteetti. Tätä ei ole vielä Loihdinissa toteutettu, joten nämä tiedostot piti kirjoittaa käsin. Aiemmin luotuja käyttöliittymiä mallina käyttäen uusien käyttöliittymien kirjoittamiseen ei mennyt kuin arviolta noin 30 minuuttia. Käyttöliittymien xml-tiedostojen luonti muistuttaa suuresti Androidin vastaavien käsinkirjoittamista. Syntaksi oli hieman erilainen, mutta se ei tuottanut vaikeuksia. Toiminnallisuuksien määritykseenkään ei mennyt paljoa aikaa. Tiedoston runkoon sai esimerkkiä aiemmin luoduista tiedostoista. Itse toiminnallisuuksien määritykset tietyille painikkeille ei vienyt aikaa, sillä näitä oli vain muutama.

Sisäänkirjautumisnäkymän ”Kirjaudu sisään” -painikkeelle määritettiin sisäänkirjautumistoiminto ja annettiin sille osoite palvelimelle, jossa käytettävä tietokanta sijaitsee. Sisäänkirjautumistoiminto on valmiiksi toteutettu luokka Loihtimen käyttämässä kirjastossa. Valikkonäkymän painikkeisiin yhdistettiin käynnistystoiminto, joka aloittaa uuden aktiviteetin. Käynnistystoiminto löytyy myös Loihtimen käyttämästä kirjastosta, ja se tarvitsee tiedon siitä, mitä käyttöliittymä- ja toiminnallisuustiedostoja uudessa aktiviteetissa käytetään. Nämä määritellään käynnistystoiminnon yhteydessä. Kuvassa 10 on kuvakaappaus toteutetusta valikkonäkymästä. Siinä näkyvä vihreä yläpalkki toimii teemana Loihtimella toteutetuissa Android-sovelluksissa. Tähän aikaa kului suunnilleen 15 minuuttia.

Kuva 10: Valikkonäkymä

Kun kaikki tiedostot olivat valmiina, ne pakattiin yhdeksi tiedostopaketiksi ja asetettiin sovelluksen valmiin rungon resurssikansioon. Sovelluksen runko pysyy aina samana, sillä käyttöliittymät ja toiminnallisuudet ovat Loihdinilla luoduissa tiedostoissa. Sitten so-

vellusta ajettaessa valmiina oli ohjelma, joka käyttäjän sisäänkirjauduttua näytti käyttäjän tietokannasta löytyvät asiakkaat, yhteyshenkilöt ja CRM-aktiviteetit tietoineen juuri niin kuin määrittelyssä haluttiin.

Kuvassa 11 on kuvakaappaus erään esimerkkiasiakkaan tietojen tarkastelu-näkymästä. Ylhäällä vihreässä palkissa lukee asiakkaan nimi ja sen alla tiedot, jotka on haettu palvelimen tietokannasta. Mikäli tässä tarkasteltaisiin yhteyshenkilöä, olisi vihreässä palkissa henkilön nimi. Tämä pätee myös CRM-aktiviteetteihin.

Kuva 11: Esimerkkitiedoilla varustettu asiakkaan tietojen tarkastelunäkymä

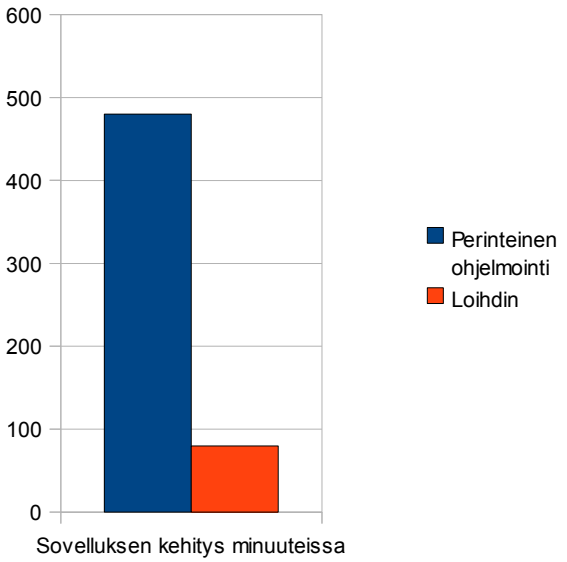
7 Perinteisen ohjelmoinnin ja Loihdinin hyödyt ja haitat

Tämän työn tutkimuksen kohteena oli tutkia eroja perinteisen ohjelmoinnin ja Loihtimen käytön välillä luotaessa samanlaiset yritysso-vellukset. Eroja käydään läpi ohjelmoijan kannalta ja yritetään pohtia, missä tilanteissa Loihdin voisi olla parempi kuin perinteinen ohjelmointi tai toisin päin.

7.1 Kehitykseen kulunut aika

Selkeimpänä näkyvänä mittana on käytetty aikaa. Jokaisesta työvaiheesta merkittiin ylös siihen kulunut aika, ja nämä ajat on ilmaistu niissä kappaleissa, joissa työvaiheita käydään läpi. Perinteisellä ohjelmoinnilla sovelluksen kehittäminen vei aikaa suunnilleen 8 tuntia. Muutaman minuutin heittoa on saattanut tapahtua. Loihtimella listaus- ja tietojentarkastelunäkymien luonti vei 35 minuuttia, josta virhemarginaali on sekunteja. Sisäänkirjautumis- ja valikkonäkymää kirjoitettiin 45 minuuttia. Yhteensä Loihdin-sovellukseen aikaa käytettiin 1 tunti 20 minuuttia, joka on selkeästi vähemmän kuin perinteisellä ohjelmoinnilla luotu yrityssovellus.

Taulukossa 2 on graafinen esitys ajoista, jotka kuluivat näiden kahden sovelluksen kehittämiseen. Siinä havainnollistuu, kuinka suuri ero ajoissa oli. Perinteisen ohjelmoinnin ajankulu ei itsessään ole kuitenkaan huono, sillä vaikka sovellus olikin pieni, oli ratkaisuissa käytetty aiemmin tuntemattomia tekniikoita. Uusien tekniikoiden oppimiseen ei kuitenkaan kulunut paljoa aikaa. Ohjelmoijalla itsellään oli mahdollisuus kartuttaa omia taitojaan. Tämä ei tietenkään poista sitä tosiasiaa, että Loihtimella sovelluksen kehitys oli huomattavasti helpompaa. Loihtimen käyttö oli huomattavasti nopeampi tapa luoda halutunlaiset sovellukset, koska siinä ei tarvinnut etsiä ja opetella uusia tekniikoita tai valita jo osatuista tavoista, joilla saada aikaiseksi esimerkiksi kaksirivisen listauksen. Myöskään tällä tavalla ei syntynyt virheitä, joita joutuisi etsimään ohjelmakoodista. Virheiden koosta ja vakavuudesta riippuen niiden korjaukseen voi mennä pitempikin aika.



Taulukko 2: Sovelluksien luontiin kuluneen ajan vertailu

Loihdinta käyttäessä ohjelmointikokemuksen määrällä ei ole väliä. Vasta-aloittelija ja kokeneempi koodaaja ovat samalla lähtöviivalla. Loihdinta käyttäessä ei ole tarvetta osata ohjelmoida optimaalisesti tai opetella uusia tapoja toteuttaa haluttuja toimintoja. Ohjelmointikokemus ei vaikuta, sillä Loihdinta käyttäessä ei ohjelmoida käsin, vaan sovelluksen luonti hoidetaan suurimmaksi osaksi hiirenpainalluksilla. Loihdin antaa vain yhden tavan luoda käyttöliittymiä, ja toiminnot se luo automaattisesti. Perinteisessä ohjelmoinnissa kokemus näkyy ajallisesti. Jos kehittäjä on luonut runsaasti Android-sovelluksia, hänellä tuskin menee paljoa aikaa yksinkertaisen sovelluksen luontiin. Kehittäjän ei tarvitse opetella uusia tapoja luoda toiminnallisuuksia, jos ne ovat jo tuttuja. Niiden kirjoittamiseen menee kuitenkin aikaa, ellei hänellä ole tallessa valmista mallikoodia halutuista toiminnallisuuksista, jota hän voisi uudelleenkäyttää. On otettava huomioon myös se, että kokeneellekin kehittäjälle voi sattua kirjoitusvirheitä, joiden löytäminen

vie aikaa.

7.2 Kehitysympäristöt

Ohjelmoimalla perinteisesti on omalla koneella oltava asennettuna jonkinlainen ohjelmointiympäristö. Koska Android-sovellukset koodataan Javalla, loogisin valinta ohjelmointiympäristössä on joko Eclipse tai NetBeans. Kumpaankin on vielä lisättävä Android SDK, jotta voidaan ohjelmoida Android-sovelluksia. Tässä työssä perinteisen ohjelmoinnin työkaluksi valittiin Eclipse IDE sen tuttuuden vuoksi. Ohjelmointiympäristön käyttöliittymä oli siis tuttu entuudestaan eikä sitä tarvinnut alkaa opetella erikseen. Eclipse IDE on Android SDK:n asennusohjeissa oletettu ohjelmointiympäristö. Tämä ei tarkoita, etteikö muita ohjelmointiympäristöjä voisi käyttää, mutta Android SDK:n lisääminen Eclipseen on helpompaa ja paremmin ohjeistettu kuin muihin ohjelmointiympäristöihin. Henkilö, joka ei aiemmin ole käyttänyt kumpaakaan aiemmin mainittua ohjelmointiympäristöä, joutuu opettelemaan alusta lähtien toisen, mikäli mielii ohjelmoida Android-alustalle. Eclipse IDE on parempi vaihtoehto tässä tapauksessa, sillä Android SDK:n lisäys sujuu siihen helpommin. Näiden ympäristöjen käytön opettelu on helppoa johtuen niiden laajasta opastusmateriaalista tuotteiden sivuilla ja internetin blogeissa. Eclipse- ja NetBeans- ympäristöistä on olemassa versiot Windows-, Mac OS- ja Linux -käyttöjärjestelmille.

Loihdin asennetaan palvelimelle. Tällöin sitä ei ole tarvetta asentaa omalle koneelle. Palvelimella on oltava asennettuna myös MySQL-ohjelmisto, sillä se tietokanta, jossa sovelluksessa käytettävät tiedot ovat, sijaitsee tällä palvelimella. Kehittäessä sovellusta Loihtimella kyseisen tietokannan sijaintia ei tarvitse erikseen etsiä eikä tietää sen tarkkaa osoitetta toisin kuin perinteisellä ohjelmoinnilla. Loihdinta voi siis käyttää eri tietokoneista ilman, että niille tarvitsee erikseen asentaa mitään. Toimiakseen kehittäjän koneella Loihdin vaatii Javan asennettuna koneeseen sekä Internet-yhteyden. Nämä nykyään löytynevät melkein kaikista tietokoneista. Loihtimen käyttöliittymä on selkeä ja looginen. Tämä helpottaa Loihtimen käytön opettelua, mikä on hyvin tarpeellista, sillä kirjallista ohjeistusta ei ollut saatavilla tämän työn aikana. Kirjallinen ohjeistus olisi hyvä olla olemassa, sillä käyttäjän on mahdotonta tietää, mitkä asetukset on välttämä-

töntä luoda, jotta saadaan aikaan toimiva kokonaisuus. Tämän työn aikana ohjeet olivat suulliset ja pikaiset, mutta hyvän käyttöliittymän ansiosta ne riittivät. Loihdin on parempi vaihtoehto silloin, kun on kehityksessä voidaan käyttää useampia tietokoneita ja aikaa on tiukasti rajattu määrä. Käyttöjärjestelmäkin voi olla tietokoneissa eri. Käyttöjärjestelmäksi Loihtimen käyttöön kelpaa mikä tahansa, jossa on mahdollista käyttää Javaa ja Internet-yhteyttä.

7.3 Kehitys

Kehitettäessä Android-sovelluksia perinteisellä tavalla on kehittäjällä hyvä olla riittävän kattava pohjatieto olio-ohjelmoinnista. Java-ohjelmointikielen osaava henkilö voi hyvin helposti oppia Android-sovellusten kehityksen, sillä Android käyttää Javaa. Tilanteessa, jossa henkilöllä ei ole mitään pohjatietoa ohjelmoinnista, hän joutuisi opettelemaan kaiken alusta lähtien. Työtä tehdessä aikataulusta ei välttämättä löydy aikaa ohjelmoinnin opetteluun. Ohjelmoinnin opiskelu ei tässä tilanteessa ole kannattavaa.

Loihtimen käytössä ei tarvitse osata ohjelmoida. Loihtimen käytön oppiminen vie huomattavasti vähemmän aikaa kuin ohjelmoinnin opiskelu. Käyttöohjeiden kirjoittaminen nopeuttaisi käytön oppimista entisestään. Henkilölle, joka osaa koodata, Loihtimen käyttö ei välttämättä ole niin mieluisaa, sillä vaikka aikaa toki säästyy, se ei tarjoa uusien tekniikoiden oppimista. Koodaaja voi myös itse haluta tehdä sovelluksen, koska tietää tekevänsä sen paremmin itse.

Loihdinta voi useampi kehittäjä käyttää kerrallaan. Tämä mahdollistaa työryhmätyöskentelyn. Näin ollen samat säännöt pätevät työryhmätyöskentelyssä sekä Loihtimella kehittämiseen että perinteiseen ohjelmointiin. Molemmissa tapauksissa sovellusta voi kehittää useampi henkilö, kunhan työstetään eri tiedostoja. Tiedostot voidaan jakaa esimerkiksi versionhallinnassa.

Yrityssovelluksia luotaessa yksi isoista eroista oli aika. Aikaeroa vielä suuremmaksi teki perinteisen ohjelmoinnin aikana syntynyt kirjoitusvirhe, jonka paikantamiseen kului melkein tunti. Tämä sijaitsi siinä kohtaa, kun listausaktiviteetista siirryttiin tietojen tar-

kastelunäkymään. Tällaisia kirjoitusvirheitä sattuu helposti, kun kehittäjä kirjoittaa yksin omaa koodiaan. Tämän takia yksikkötestien kirjoittaminen olisi suotavaa. Loihdin mahdollistaa sovellusten luonnin ilman ohjelmointia, joten vastaavia kirjoituksesta johtuvia ohjelmointivirheitä ei tapahdu, eikä niiden etsimiseen ja korjaamiseen kulu aikaa. Ohjelmakoodin virheettömyyshän on yksi malliperustaisen ohjelmoinnin tavoitteista. Loihdin ei kuitenkaan pysty korjaamaan kehittäjän omaa logiikkavirhettä. Logiikkavirheet Loihtimessa testataan kokeilemalla luotua sovellusta.

Loihtimen Android-laajennus ei ollut vielä täysin valmis halutunlaiseen kehitykseen johtuen sisäänkirjautumis- ja valikkonäkymien puuttumisesta. Näiden puutteiden korjaantuessa nopeutuu kehitysaika Loihdinta käyttäessä. Näkymien puute ei haittaa kovinkaan paljon sellaista kehittäjää, jolle XML-kieli on tuttu. Muille se aiheuttaa pulmia, sillä nuo näkymät muodostetaan XML-kielellä.

Luodessa XML-määrityksiä sisäänkirjautumis- ja valikkonäkymiin, niiden ulkonäköä olisi voinut muokata muunkinlaisiksi. Niistä tehtiin kuitenkin samanlaiset kuin mitä generoiduissa näkymissä. Generoiduissa tiedostoissa on se huono puoli, että niiden ulkonäköön ei kehittäjän ei ole tarkoitus pystyä vaikuttamaan. Tässä perinteinen ohjelmointi voittaa Loihdinin, sillä koodatessa normaalisti, kehittäjä voi itse päättää, minkälaiset käyttöliittymät hän luo. Joillekin kehittäjille käyttöliittymien luonnin automatisointi taas sopii hyvin, sillä he saattavat haluta keskittyä muihin kiinnostavampiin ohjelmointikohteisiin.

Generoituja tiedostoja on mahdollista muokata, ja siten ne sopivat hyvin uudelleenkäytettäviksi, jos kehittäjä osaa tulkita Loihtimen luomaa koodia. Tässä työssäkin on uudelleenkäytetty joitakin osia generoiduista tiedostoista, kun puuttuvat sisäänkirjautumis- ja valikkonäkymät luotiin. Loihtimen luomia tiedostoja voidaan siis käyttää uudestaan ja muokata samaan tapaan kuin kirjoitettua ohjelmakoodia.

7.4 Rajoitukset

Perinteisessä ohjelmoinnissa periaatteessa rajana ovat ainoastaan kehittäjän omat tai-

dot. Käytännössä perinteistä ohjelmointia rajoittavat myös aika ja kustannukset. Joskus tuote täytyy saada ulos ajassa, jota ei perinteisellä ohjelmoinnilla kyetä toteuttamaan ja testaamaan. Perinteinen ohjelmointi on virhealtista, ja tämä korottaa kustannuksia ja kehittämiseen kuluvaa aikaa.

Loihdinta ei rajoita samat asiat kuin perinteistä ohjelmointia. Loihdin on nopea, eikä sen kanssa tässä työssä saatu virheellistä tuotetta. Loihtimessa on muita rajoitteita. Kehittäjä ei pysty käyttämään omaa tietotaitoaan kehittämään parasta mahdollista ratkaisua tietyssä tilanteessa, vaan joutuu tyytymään Loihtimen yleispätevään ratkaisuun. Myöskään käyttöliittymiin kehittäjä ei voi vaikuttaa. Kehittäjä ei myöskään pysty jatkokehittämään Loihdin-sovellustaan omaan suuntaansa, vaan jatkokehitys voi tapahtua ainoastaan Loihtimen kautta. Tästä johtuen kehittäjä ei pysty itse muokkaamaan Android-sovellusta niin, että siinä voitaisiin tietojen tarkastelun lisäksi muokata niitä, vaan hän joutuu odottamaan, että tämä saadaan valmiiksi Loihtimen Android-laajennuksessa.

Loihtimella tuotetut sovellukset sopivat ainoastaan yrityskäyttöön, jossa tarvitaan asiakkuudenhallintaa. Loihdin-sovellukset ovat tarkoitettu yrityksille, jotta ne voivat työskennellä omien tietojensa kanssa, mutta yleisesti puhuttaessa yrityksiin rajautuminen on mainittava.

Vaikka Android-älypuhelinten muistikapasiteetti on noussut viime vuosina, eivät ne vieläkään pysty kilpailemaan oikeiden tietokoneiden kanssa. Tästä syystä Loihdin-sovelluksen viemä tila lasketaan rajoitukseksi. Loihdin-sovelluksissa käyttöliittymät ja toiminnot luodaan tiedostoihin, joiden mukaan sovellus toimii. Nämä tiedostot vievät paljon tilaa. Perinteisesti koodatut Android-ohjelmat vievät vähemmän muistia, vaikka toiminnot olisivat samanlaiset. Loihdin-sovellusten käyttämien tiedostojen ja etenkin niiden varaaman muistin määrän vuoksi Android-alustalle kyseiset sovellukset eivät voi olla yhtä monimutkaisia ja laajoja, kuin mitä Loihdinilla luodut web-sovellukset.

8 Yhteenveto

Työssä on toteutettu kaksi yrityssovellusta Android-ympäristöön, joita voidaan käyttää käyttäjän tietokantapalvelimella sijaitsevien tietojen tarkasteluun Android-älypuhelimella. Työn aikana on huomioitu kahden eri toteutustavan eroavaisuuksia. Näkyvimmäksi eroksi saatiin nopeus. Tämän työn tulokset perustuvat vain yhden henkilön kokemuksiin. Tuloksissa saattaa olla eroja riippuen henkilön kokemuksesta joko Loihtimen käytössä tai perinteisesti ohjelmoidessa.

Työssä varmistettiin, että Loihtimen käyttö on nopeaa eikä kehittäjällä tarvitse olla ohjelmoijan taitoja luodakseen toimivan sovelluksen. Jo ohjelmointia osaaville Loihdin tarjoaa kehittämisen nopeuttamista, mutta silloin ohjelmoijan oman luovuuden rooli pienenee. Vastaavissa tuotteissa, joissa sovelluksen nopealla kehityksellä on suuri merkitys, laatu yleensä kärsii ja kustannukset nousevat, jos kehittäjät eivät osaa suunnitella ajankäyttöään tehokkaasti.

Loihtimella tuotettujen sovellusten toiminnot rajoittuvat sellaisiin yrityssovelluksiin, joissa tarkastellaan käyttäjän tietokantapalvelimella sijaitsevia tietoja, sillä Loihtimen Android-laajennus ei vielä tue muokkaus-toimintoa. Tämä olisi seuraava looginen toteutettava toiminto Loihtimeen.

Perinteisellä ohjelmoinnilla on tässä työssä tarkoitettu ohjelmointia, joka suoritetaan kirjoittamalla jokainen koodirivi itse. Tämä on aikaa vievää ja tässäkin insinöörityössä on tullut todettua, että perinteinen ohjelmointi vei miltei kuusinkertaisesti enemmän aikaa kuin Loihtimen käyttö. Aikaa vievät myös mahdolliset virheet, joita syntyy helpommin, kun jokainen koodirivi kirjoitetaan itse. Loihtimen käytössä ei kirjoitusvirheitä synny, sillä koodirivit, joissa nuo virheet saattaisivat syntyä, on testattu ennen käyttöä. Kuitenkin perinteisessä ohjelmoinnissa kehittäjällä on vähemmän rajoituksia lopullisen tuotteen suhteen. Kehittäjä voi tehdä sovelluksesta itsensä näköisen.

	Loihdin-sovellus	Perinteisesti ohjelmoitu sovellus
Luotavat sovellukset	Sovellukset, joita Loihtimella luodaan soveltuvat asiakkuudenhallintaan. Tämän työn aikana tuetut toiminnot liittyvät tietokannassa sijaitsevien tietojen tarkasteluun.	Perinteisesti ohjelmoiden sovellustyyppien rajoitteena on vain Android-alustan asettamat rajoitteet.
Käytetty työaika	Kehitykseen käytetty työaika on huomattavasti lyhyempi kuin perinteisesti ohjelmoidessa. Tässä tapauksessa aikaa kului 1 tunti 20 minuuttia.	Perinteisesti ohjelmoidessa sovelluksien kehittämiseen menee enemmän aikaa. Tässä tapauksessa aikaa kului 8 tuntia.
Kehittäjän kokemus	Loihtimen käytössä kehittäjän ohjelmointikokemus ei vaikuta kehitykseen kuluvaan aikaan taikka tehokkuuteen.	Perinteisessä ohjelmoinnissa sovelluksen kehitysaikaan vaikuttaa kokemus suuresti. Vasta-alkajalla menee huomattavasti enemmän aikaa kuin kokeneella kehittäjällä, jonka ei tarvitse opetella uusia asioita.
Työryhmässä työskentely	Loihdinta käytettäessä työryhmässä työskentely on mahdollista, kun kehittäjät työstävät eri osia sovelluksesta.	Perinteisessä ohjelmoinnissa työryhmässä työskentely on mahdollista, kun kehittäjät työstävät eri osia sovelluksesta.
Testaus	Loihdin on luotu siten, että kirjoituksesta johtuvia ohjelmointivirheitä ei synny. Logiikkavirheiden syntyminen johtuu kehittäjästä ja näitä testataan käyttämällä sovellusta.	Perinteisessä ohjelmoinnissa voidaan kaikkea testata ohjelmakoodista koko sovelluksen toimintaan.
Uudelleenkäytettävyys	Loihtimella luotuja tiedostoja on mahdollista muokata käsin, jos osaa tulkita Loihtimen tuottamaa koodia. Näin Loihtimella tuotettujen sovellusten osia voidaan uudelleenkäyttää, mutta tällöin riski virheiden syntymisestä kriittisiin tiedostoihin kasvaa.	Perinteisessä ohjelmoinnissa on mahdollista uudelleenkäyttää koodia, jos se on tehty hyvin ja yleispätevästi.

Oheisessa taulukossa on koottuna vertailua työssä tutkittujen menetelmien välillä. Huomiota on kiinnitetty, minkälaisia sovelluksia menetelmillä voi luoda, kuinka kauan näihin menee aikaa ja vaikuttaako kehittäjän kokemus niihin sekä kuinka ryhmätyö onnistuu. Myös testausta ja luotujen sovellusten osien uudelleenkäyttöä on pohdittu molemmissa

tapauksissa.

Älypuhelin-sovellukset ovat yleensä kooltaan pieniä. Tämä johtuu siitä rajallisesta muistin määrästä, mitä älypuhelimissa on. Myös älypuhelinten suoritusteholla on vaikutusta sovelluksiin, joita älypuhelimilla voidaan käyttää. Älypuhelimissa suoritusteho on pienempi kuin tavallisissa pöytäkoneissa. Tässä insinöörityössä luodut sovellukset ovat pieniä. Tästä syystä ei voida tarkalleen sanoa, minkälainen laatu Loihtimen luomassa isommassa sovelluksessa voisi olla. Liian isoa sovellusta ei kuitenkaan edes voida luoda Loihtimella johtuen Loihdin-sovellusten normaalia isommasta koosta.

Lähteet

[1] Android Fundamentals. 3.10.2011. Google. Verkkodokumentti. <http://developer.android.com/guide/topics/fundamentals.html>. Luettu 5.10.2011.

[2] Steele, James & To, Nelson. The AndroidDeveloper's Cookbook Building Applications with the Android SDK. 2010. Addison-Wesley.

[3] Brian DeLacey. 2007. Google Calling: Inside Android, the gPhone SDK. Verkkodokumentti.<http://onlamp.com/pub/a/onlamp/2007/11/12/google-calling-inside-the-gphone-sdk.html>. Luettu 6.10.2011.

[4] Guy, Romain. 2009. Track memory allocations. Android Developers. Verkkodokumentti.<http://android-developers.blogspot.com/2009/02/track-memory-allocations.html>. Luettu 22.3.2012.

[5] Platform Versions. 2011. Google. Verkkodokumentti. <http://developer.android.com/resources/dashboard/platform-versions.html>. Luettu 13.10.2011.

[6] Android 2.1 Platform. 2010. Google. Verkkodokumentti. <http://developer.android.com/sdk/android-2.1.html>. Luettu 13.10.2011.

[7] Activities. 3.10.2011. Google. Verkkodokumentti. <http://developer.android.com/guide/topics/fundamentals/activities.html>. Luettu 6.10.2011.

[8] Intents And Intent Filters. 2011. Google. Verkkodokumentti. <http://developer.android.com/guide/topics/intents/intents-filters.html>. Luettu 25.10.2011.

[9] Who's OMG. 2010. Object Management Group. Verkkodokumentti. <http://www.omg.org/marketing/about-omg.htm>. Luettu 12.10.2011.

[10] Truyen, Frank. 2006. The Fast Guide to Model Driven Architecture. Verkkodokumentti.<http://www.omg.org/mda/mda_files/Cephas_MDA_Fast_Guide.pdf>. Luettu 10.10.2011.

[11] Atkinson, Colin & Kühne, Thomas. Model-Driven Development: A Metamodeling Foundation. 2003.IEEE Computer Society.

[12] What Is Rapid Application Development?. 2010. Custom Application Development Portal. Verkkodokumentti. <http://www.customapplicationdevelopment.org/rapid-application-development/what-is-rapid-application-development.html>. Luettu 26.2.2012.

[13] Understanding the Rapid Application Development Process. 2012. Custom Application Development Portal. Verkkodokumentti. <http://www.customapplicationdevelopment.org/rapid-application-development/understanding-the-rapid-application-development-process.html>. Luettu 26.2.2012.

[14] Haikala, Ilkka & Märijärvi Jukka. Ohjelmistotuotanto. 2004. Talentum.

[15] Evans, Simon. 2006. 10 Reasons why Agile is not Rapid Application Development (RAD). Verkkodokumentti. <http://consultingblogs.emc.com/simonevans/archive/2006/04/18/10-Reasons-why-Agile-is-not-Rapid-Application-Development-_2800_RAD_2900_.aspx>. Luettu 14.3.2012.

[16] McConnell, Steve. Rapid Development - Taming Wild Software Schedules. 1996. Microsoft Press.

```
public class ClientList extends ListActivity {


  private CredsHolder holder;


  @Override

  protected void onCreate(Bundle savedInstanceState) {


      super.onCreate(savedInstanceState);

      setContentView(R.layout.clientlist);


      holder = (CredsHolder)getApplicationContext();


      String[] clients = getClients();

      setListAdapter(new ArrayAdapter<String>(this,
            R.layout.list_item, clients));

      ListView listview = getListView();

      listview.setOnItemClickListener(new OnItemClickListener() {

         public void onItemClick(AdapterView<?> parent, View
                  view, int position, long id) {

             Log.v("on item click", "position=" + position + " view="
                  + ((TextView)view).getText());

             Log.v("on item click", "rowid=" +

                        holder.getCompanyRowid(

                        ((TextView)view).getText().toString()));

             String titlevalue = ((TextView)view).getText().toString();

             if(holder.hasCompanyRowid(titlevalue)){

                  Intent intent = new Intent(getBaseContext(),
                                    ClientActivity.class);

                  intent.putExtra("companyid",

                                    holder.getCompanyRowid(
```

```
                        ((TextView)view).getText().toString()));
                startActivity(intent);
            }
        }
    });
}

public String[] getClients() {
    ArrayList<String> arraylist = new ArrayList<String>();

    String username = holder.getUsername();
    String password = holder.getPassword();
    String rowid = holder.getRowid();
    try {
      URL url = new URL(
          "http:// the address of database /get?username="
                  + username
                  + "&password="
                  + password
                  + "&xmlquery="
                  + URLEncoder.encode(
                  "<dataclass name=\"company\"><condition><field
                  useLinkId=\"true\">owner"
                  + "</field><operator>=</operator><parameter>"
                  + rowid
                  + "</parameter></condition></dataclass>",
                  "UTF-8"));
```

```
URLConnection connection = url.openConnection();
        InputStream input = connection.getInputStream();


        DocumentBuilderFactory factory = DocumentBuilderFactory

                              .newInstance();

        DocumentBuilder parser = factory.newDocumentBuilder();

        Document document = parser.parse(input);

        NodeList clients =
        document.getDocumentElement().getChildNodes();

        for (int i = 0; i < clients.getLength(); i++) {

            Node n = clients.item(i);

            if (n instanceof Element) {

              String client = ((Element) n).getAttribute("titlevalue");

              arraylist.add(client);

              String companyrowid = ((Element)
                  n).getAttribute("rowid");

              holder.saveCompanyRowid(client, companyrowid);

            }



        }

      } catch (Exception e) {

            Log.v("Connection failed..", "Something is wrong");

            e.printStackTrace();

      }

      return arraylist.toArray(new String[arraylist.size()]);

  }

}
```

```
public class ClientActivity extends ListActivity {

  private CredsHolder holder;

  @Override

  protected void onCreate(Bundle savedInstanceState) {

      super.onCreate(savedInstanceState);

      setContentView(R.layout.clientlist);


      holder = (CredsHolder) getApplicationContext();

      // create the grid item mapping

      int[] to = new int[] { R.id.first_line, R.id.second_line };

      ArrayList<Map<String, String>> data = new ArrayList<Map<String,
                                                          String>>();

      String companyid = getIntent().getStringExtra("companyid");

      data = getClientInfo(data, companyid);

      String[] from = new String[] { "key", "value" };

      setListAdapter(new SimpleAdapter(this, data,
                  R.layout.client_list_item, from, to));

  }


  private ArrayList<Map<String, String>> getClientInfo(

                  ArrayList<Map<String, String>> data, String
                                                  companyrowid) {

      String username = holder.getUsername();

      String password = holder.getPassword();

      String rowid = holder.getRowid();


      try {

        URL url = new URL(

            "http://the address of database/get?username="
```

```
    + username

    + "&password="

    + password

    + "&xmlquery="

    + URLEncoder.encode(

    "<dataclass name=\"company\"><condition><field
    useLinkId=\"true\">rowid"


    + "</field><operator>=</operator><parameter>"


    + companyrowid


    + "</parameter></condition></dataclass>",

    "UTF-8"));


URLConnection connection = url.openConnection();

InputStream input = connection.getInputStream();

DocumentBuilderFactory factory = DocumentBuilderFactory

                    .newInstance();

DocumentBuilder parser = factory.newDocumentBuilder();

Document document = parser.parse(input);

NodeList clientInfo =
document.getDocumentElement().getChildNodes();


Map<String, String> maps = null;

for (int i = 0; i < clientInfo.getLength(); i++) {

   Node n = clientInfo.item(i);

      if (n instanceof Element) {

         NodeList attributes = n.getChildNodes();
```

```
                for (int j = 0; j < attributes.getLength(); j++) {
                   Node attribute = attributes.item(j);
                   if (attribute instanceof Element) {
                      if (attribute.getChildNodes().getLength() > 0) {
                         maps = new HashMap<String, String>();
                         String local = getLocalized(attribute
                                        .getNodeName());
                         if (local != null) {
                            String attrValue = attribute
                                        .getChildNodes().item(0)
                                        .getNodeValue();
                            maps.put("key", local);
                            maps.put("value", attrValue);
                            data.add(maps);
                         }
                      }
                   }
                }
             }
          }
       } catch (Exception e) {
                e.printStackTrace();
       }
       return data;
    }

    private String getLocalized(String string) {
       String newString = "";
```

```
        if (string.equals("name")) {
                    newString = getText(R.string.name).toString();
        } else if (string.equals("type")) {
              newString = getText(R.string.type).toString();
        } else if (string.equals("wwwadress")) {
              newString = getText(R.string.wwwadress).toString();
        } else if (string.equals("phonenumber")) {
              newString = getText(R.string.phonenumber).toString();
        } else if (string.equals("industry")) {
              newString = getText(R.string.industry).toString();
        } else if (string.equals("owner")) {
              newString = getText(R.string.owner).toString();
        } else if (string.equals("billingstreet")) {
              newString = getText(R.string.billingstreet).toString();
        } else if (string.equals("officalcode")) {
              newString = getText(R.string.officalcode).toString();
        } else if (string.equals("billingcity")) {
              newString = getText(R.string.billingcity).toString();
        } else if (string.equals("description")) {
              newString = getText(R.string.description).toString();
        } else {
              newString = null;
        }
        return newString;
    }
}
```