VAASAN AMMATTIKORKEAKOULU
VASA YRKESHÖGSKOLA
UNIVERSITY OF APPLIED SCIENCES

Jussi Nyholm

Ohjelmistoprojektin analyysi ketterien menetelmien kannalta

Case StarSoft Oy

Liiketalous ja matkailu

2011

VAASAN AMMATTIKORKEAKOULU
Tietojenkäsittelyn koulutusohjelma

TIIVISTELMÄ

Tekijä	Jussi Nyholm
Opinnäytetyön nimi	Ohjelmistoprojektin analyysi ketterien menetelmien kannalta. Case StarSoft Oy
Vuosi	2011
Kieli	suomi
Sivumäärä	68
Ohjaaja	Mika Tamminen

Tämän opinnäytetyön tavoitteena on selvittää vertailun avulla ketterien ohjelmistokehitysmenetelmien hyödyllisyyttä perinteisiin menetelmiin verrattuna. Toimeksiantajayritys StarSoft Oy tarvitsi uuden järjestelmän yrityksen tukipalvelun puhelutietojen tallentamiseksi ja tilastoimiseksi. Tietojärjestelmä toteutettiin käyttäen inkrementaalista evo-mallia, jossa kukin kehityskierros toteutetaan vesiputousmallin mukaisesti.

Opinnäytetyön teoriaosuus esittelee ketterän ohjelmistokehityksen ja ketterän manifestin pääperiaatteet, sekä muutaman yleisimmän ketterän menetelmän. Empiirinen osuus kuvaa yksityiskohtaisesti tietojärjestelmän teknisen toteutuksen. Työn johtopäätöksissä todetaan tehdystä työstä ja prosessista ilmenneet ketterät piirteet ja pohditaan, miten ketteriä menetelmiä hyödyntämällä oltaisiin päästy parempiin tuloksiin tietojärjestelmää toteutettaessa.

Asiasanat Scrum, agile, XP, Lean, AM, Crystal

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Tietojenkäsittelyn koulutusohjelma

ABSTRACT

Author	Jussi Nyholm
Title	Analysis of a software development project on the basis of Agile. Case StarSoft Ltd.
Year	2011
Language	Finnish
Pages	68
Name of Supervisor	Mika Tamminen

The aim of this thesis is to examine what kind of benefits the agile methods provide in comparison to the traditional program development methods. The client company, StarSoft Ltd., needed a new data system in order to save and compile statistics from the phone call data of the customer support of the company. The data system was created using an incremental evolutionary model in which every cycle is executed using the waterfall model.

The theoretical study introduces the main aspects of the agile program development as well as the agile manifesto. It also gives an overview of some of the most common agile methods. The empirical study explains the technical implementation of the data system in detail. The conclusion of the thesis discovers the agile aspects found in the process executed and reflects on in which way the use of agile methods could have given better results in the project.

Keywords Scrum, agile, XP, Lean, AM, Crystal

SISÄLLYS

1 Johdanto

Työn toimeksiantajayritys StarSoft Oy on kouluhallinnon ohjelmiin erikoistunut ohjelmistotalo. Ohjelmat ovat erittäin kattavia, toimintojen määrä on suuri ja asiakkaita on paljon ympäri Suomea. Tämän vuoksi yrityksen tukipalvelu on asiakkaiden ahkerassa käytössä. Puheluita saattaa tulla etenkin kiireaikoina satoja yhden päivän aikana. Puheluiden määristä ym. tilastoja ottamalla voidaan mm. varautua kiireaikoihin ja huolehtia, että tukipalvelussa on tarvittava miehitys paikalla.

Toimeksiantaja tarvitsee tietojärjestelmän, joka mahdollistaa puhelinkeskuksen tuottaman datan tallennuksen tietokantaan, sekä tilastojen koostamisen tallennetuista tiedoista. Yrityksen käyttämään sähköpostiohjelmaan tehdään laajennus, jonka kautta voidaan tarkastella erilaisia puhelutilastoja ja koostaa niiden perusteella graafinen esitys pylväskuvaajan muodossa.

1.1 Tutkimuksen tarkoitus

Tutkimuksen tarkoituksena on arvioida tehtyä työtä, löytää ohjelmistoprojektista mahdolliset ketterät ominaisuudet, sekä selvittää, minkälaisia hyötyjä ketterien ohjelmistokehitysmenetelmien käytöstä olisi ollut tässä projektissa siinä käytettyihin perinteisiin kehitysmenetelmiin verrattuna.

Tutkimuksen teoriaosuus käsittelee ketteriä menetelmiä yleisesti, yleisiä käsitteitä, kuten iteratiivisuus ja inkrementaalisuus, sekä ketterille menetelmille yhteisiä arvoja ja periaatteita, jotka löytyvät *Agile manifestosta*, eli ketterästä manifestista. Lisäksi luodaan katsaus muutamiin tavallisimpiin ketteriin menetelmiin ja tuodaan esille niiden ominaispiirteet.

Empiirinen osuus keskittyy yksityiskohtaiseen kuvaukseen luodusta tietojärjestelmästä ja siinä käytetyistä tekniikoista enimmäkseen kerronnallisesti, mutta osittain myös koodiesimerkkien avulla.

1.2 Tutkimusmenetelmä ja –materiaali

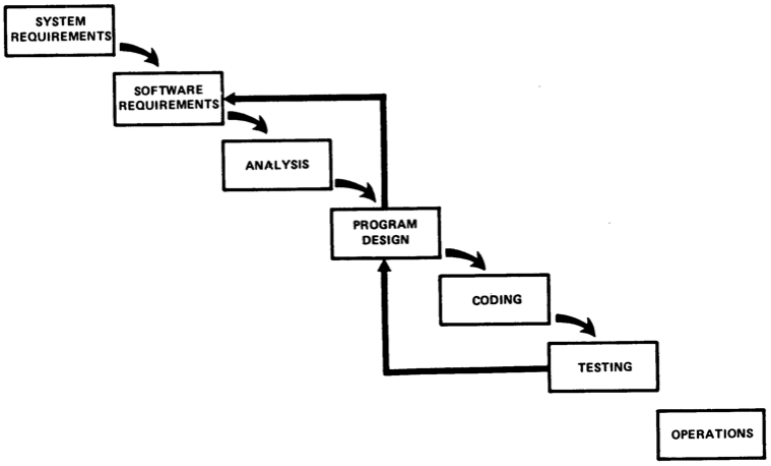
Tutkimusmenetelmänä käytetään vertailua; toteutuneen projektin ominaisuuksia verrataan teoriaosuudessa esiteltyjen ketterien menetelmien esimerkinomaisiin tilanteisiin sekä ketterien projektien ominaisuuksiin ja projektiryhmien toimintaan. Toteutuneesta projektista etsitään yhtäläisyyksiä teoriassa esiteltyihin menetelmiin, sekä pohditaan, mitä ketteriä menetelmiä tai niiden periaatteita käyttämällä olisi päästy parempiin tuloksiin.

2 Ketterät menetelmät

Ketterän kehityksen ideologia lähtee tehtyjen suunnitelmien muuttumisen mahdollisuuden ja jopa todennäköisyyden olemassaolon tiedostamisesta ja myöntämisestä. Ohjelmistoprojektin alussa tehty vaatimusmäärittely ei useimmiten vastaa kaikilta osin todellisuutta, mikä huomataan projektin edetessä. Perinteisissä malleissa on tavallista, että toteutetaan ensiksi helpot ominaisuudet, jolloin projekti näyttää edistyvän ja jätetään vaativammat ja suurimman riskin sisältävät ominaisuudet projektin loppupuolelle. Ketterissä menetelmissä keskitytään sen sijaan aina korkeimman prioriteetin ominaisuuksien toteuttamiseen ensiksi ja vasta sen jälkeen vähäpätöisempien ominaisuuksien tekemiseen.(Poimala & Heikniemi & Blåfield 2010.)

Myös ohjelmiston elinkaari käydään läpi useampaan kertaan käytettäessä ketteriä menetelmiä. Perinteisillä menetelmillä törmätään usein ongelmiin ohjelmiston käyttöönottovaiheessa. Ketterissä menetelmissä jaetaan projekti osiin, jolloin kaikki tarvittavat vaiheet käydään läpi useampaan kertaan, mikä pienentää ikävien yllätyksien mahdollisuutta projektin loppuvaiheessa. Projektin jakaminen pienempiin osiin merkitsee myös testauksen ja laadunvarmistuksen toteuttamista vaiheittain, sen sijaan että se jätettäisiin projektin loppuun. Tämä takaa sen, että kaikki, mitä on valmiina on testattua ja hyväksi havaittua.(Poimala ym. 2010.)

Ketteriä menetelmiä hyödynnettäessä jakaantuu em. osiin jakamisesta johtuen myös ohjelmiston arvon lisäys pidemmälle aikavälille, huipun sijoittuessa suhteellisen aikaiseen vaiheeseen, kun toteutettavana ovat korkeimmalle priorisoidut ominaisuudet (Berteig 2007). Ei-iteratiivinen menetelmä, kuten esim. perinteinen vesiputousmalli(Kuvio 1) puolestaan tuottaa kaiken vastineen ohjelmiston tilanneen asiakkaan sijoituksille projektin loppupuolella, eli toteutusvaiheessa. Riskit ovat tässä tapauksessa paljon suuremmat, kun pelissä saattaa olla koko ohjelmiston kohtalo. Ketteriä menetelmiä käytettäessä sen sijaan ohjelmiston arvon lisäys sekä olemassaolevat riskit ovat projektin loppuvaiheessa alhaisimmillaan.



Kuvio 1. Vesiputousmalli. (Ketteryys.fi)

2.1 Agile manifesto, ketterä manifesti

Agile Manifesto, eli ketterä manifesti kirjoitettiin helmikuussa 2001 17:n eri ohjelmointimetodologioiden harjoittajien kokouksessa. Kokoukseen osallistujat eivät olleet samaa mieltä kaikesta, mutta havaitsivat olevansa yhtä mieltä neljän arvon osalta.(Agile Alliance 2010.)

> "Me etsimme parempia tapoja ohjelmistokehitykseen tekemällä sitä itse ja auttamalla muita sen tekemisessä. Tämän työn kautta olemme päätyneet arvostamaan:
>
> **Yksilöitä ja vuorovaikutusta** enemmän kuin prosesseja ja työvälineitä
> **Toimivaa ohjelmistoa** enemmän kuin kattavaa dokumentaatiota
> **Yhteistyötä asiakkaan kanssa** enemmän kuin sopimusneuvotteluja
> **Muutokseen reagointia** enemmän kuin suunnitelman seuraamista
>
> Vaikka oikealla puolella olevat asiat ovat arvokkaita, me arvostamme enemmän vasemmalla puolella olevia asioita."
>
> (Manifesto for Agile Software Development 2001.)

Ketterän manifestin taustalla ovat seuraavat perusperiaatteet:

> "**Tyydytä asiakas:** Tärkein tehtävämme on tyydyttää asiakas toimittamalla nopeasti ja jatkuvasti arvokas sovellus.
>
> **Hyväksy muutos:** Toivotamme tervetulleeksi muuttuvat vaatimukset, myös projektin loppupuolella. Ketterät prosessit hyväksyvät muutokset asiakkaan kilpailueduksi.
>
> **Julkaise aikaisin ja usein:** Toimita toimiva sovellus usein, parin viikon - parin kuukauden välein, pyrkien mahdollisimman nopeaan sykliin.

Asiakkaan läheisyys: Toteuttajien ja liiketoiminnan tuntijoiden on toimittava yhdessä päivittäin koko projektin ajan.

Luottamus tekijöihin: Rakenna projektit motivoituneiden yksilöiden ympärille. Anna heille ympäristö ja heidän tarvitsemansa tuki ja luota että he tekevät työnsä.

Suora keskustelu välillisen viestinnän sijaan: Tehokkain kommunikointikeino on kasvokkain keskustelu.

Toimiva sovellus on ensisijainen edistymisen mittari.

Tasainen tahti: Ketterät prosessit suosivat kestävää kehitystä. Sponsoreiden, kehittäjien ja käyttäjien on pystyttävä ylläpitämään tasaista työtahtia jatkuvasti.

Tekninen loistokkuus: Jatkuva tekniseen laatuun ja hyvään suunnitteluun panostaminen parantaa ketteryyttä.

Yksinkertaisuus – tekemättä jätetyn työn määrän maksimointi – on olennaista.

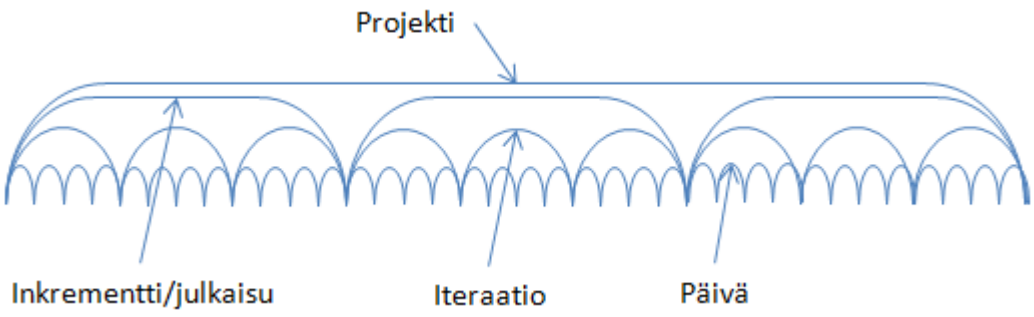
Itseohjautuvuus: Parhaat arkkitehtuurit, vaatimukset ja suunnitelmat kehittyvät tiimeistä, jotka organisoivat itse toimintansa.

Itsetarkastelu: Tiimi pysähtyy miettimään säännöllisin väliajoin, kuinka tulla vielä tehokkaammaksi ja säätää toimintatapojaan sen mukaisesti.”

(Poimala ym. 2010.)

2.2 Iteratiivinen ja inkrementaalinen ketterä kehitys

Ketterät menetelmät sisältävät usein termit *iteratiivinen* ja *inkrementaalinen*. *Iteraatiolla* tarkoitetaan projektin yhtä sykliä (Kuvio 2), jonka aikana toteutetaan osittain tai kokonaan yksi *inkrementti*, eli toimiva osakokonaisuus, joka on tarvittaessa mahdollista ottaa käyttöön. (Poimala ym. 2010.)



Kuvio 2. Projektin jakautuminen inkrementteihin ja iteraatioihin. (Poimala ym. 2010)

Iteraation pituuden on oltava riittävä, jotta sen aikana ehditään suorittaa seuraavat toimenpiteet:

- Asiantuntijat suunnittelevat iteraatiossa työstettävien ominaisuuksien yksityiskohdat.
- Ohjelmoijat toteuttavat ne.
- Käyttäjät, asiantuntijat ja testaajat tarkistavat tehtyjen ominaisuuksien oikeellisuuden ja kommentoivat, mitä asioita tulee muuttaa.
- Ohjelmoijat tekevät tarvittavat muutokset.
- Kahta edellistä kohtaa toistetaan, kunnes asiantuntijat, käyttäjät ja rahoittajat hyväksyvät iteraatiossa tehdyt ominaisuudet käyttöönotettaviksi.

Jos iteraation pituus määritetään liian lyhyeksi ja kaikkia edellämainittuja kohtia ei ehditä toteuttaa sen aikana, saatetaan pahimmassa tapauksessa päätyä seuraavaan ratkaisuun:

- Ensimmäisessä iteraatiossa suunnitellaan ensimmäinen ominaisuus.
- Toisessa iteraatiossa suunnitellaan toinen ominaisuus ja ohjelmoijat koodaavat ensimmäisen ominaisuuden.
- Kolmannessa iteraatiossa suunnitellaan kolmas ominaisuus, ohjelmoidaan toinen ominaisuus ja testaajat testaavat ensimmäisen ominaisuuden.

Tästä tilanteesta seuraa se ongelma, että eri tahot työstävät eri ominaisuuksia ja keskinäinen keskustelu on enemmänkin häiriötekijä kuin hyöty, koska nyt esim. suunnittelijoiden on muistettava kahden syklin takaiset suunnitelmansa keskustellessaan testaajien kanssa näiden testatessa ominaisuutta, joka suunniteltiin kuukausi sitten. (Cockburn 2007, 244-245.)

Jos suunnittelijat tällaisessa tilanteessa päätyvät häiriöiden välttämiseksi dokumentoimaan työnsä ohjelmoijia ja testaajia varten, luovutaan suorasta keskustelusta ja korvataan se välillisellä viestinnällä, mikä ei ole mm. ketterän manifestin taustalla olevien periaatteiden mukaista toimintaa. Toisaalta myöskään liian pitkäkestoiset iteraatiot eivät ole toivottuja. Tällöin on olemassa mahdollisuus, että syklin alkuvaiheessa tehdään liian vähän töitä, koska aikaa koetaan olevan yllin kyllin,

jolloin loppuvaiheessa joudutaan kiirehtimään, jotta saataisiin valmiiksi iteraation aikana tehdyksi suunniteltavat työt. (Cockburn 2007, 245.)

2.3 Ketterän kehityksen metodit

1990-luvun loppupuolella useat metodologiat alkoivat saada osakseen lisääntynyttä kiinnostusta. Ne koostuivat erilaisista yhdistelmistä vanhoja, uusia ja mukautettuja ideoita, mutta niille kaikille oli yhteistä se, että ne suosivat ohjelmointitiimin ja liiketalousasiantuntijoiden läheistä yhteistyötä, suullista kommunikointia kirjallista kommunikaatiota tehokkaampana, säännöllisiä käyttöönottovalmiita toimituksia, tiiviitä itseohjautuvia tiimejä sekä tapoja kehittää ohjelmakoodia ja ohjelmointitiimiä vaatimusten edellyttämällä tavalla. (Agile Alliance 2010.)

Ketteryydellä viitataan ketterän manifestin julistamien arvojen mukanaan tuomaan reagointivalmiuteen tehtyjen suunnitelmien orjallisen seuraamisen sijaan, sekä tiimien itseohjautuvuuteen ja vapauteen byrokratiasta. Muutoksiin voidaan sopeutua tarvittaessa sen sijaan, että yritettäisiin kaikissa tilanteissa palata alkuperäiseen suunnitelmaan.

Ketterän metodin käyttö ei kuitenkaan tarkoita sitä, että suunnitelmia ei pitäisi olla. Crystal-metodologiaperheen(ks. 2.3.3) isä ja yksi ketterän manifestin laatijista, Alistair Cockburn, kertoo törmänneensä lukemattomia kertoja tapauksiin, joissa ohjelmoijat ovat uskotelleet esimiehilleen, että riittävän ketteryyden avulla suunnitelmia ei tarvita ollenkaan. Tavallaan tällaista ajattelua voi tukea se, että ketterän manifestin laatijat arvostavat ”muutokseen reagointia enemmän kuin suunnitelman seuraamista”, mutta toisaalta manifestissa mainitaan heti tämän jälkeen, että oikealla puolella olevat asiat ovat nekin arvokkaita. Kyseisen lausunnon tarkoitus onkin verrata kahta tapaa hyödyntää suunnittelua sen sijaan, että siitä luovuttaisiin täysin:

- Tehdään suunnitelma ja harkitaan ajoittain, onko nykytilanteessa parempi pysyä suunnitelmassa, vai luoda uusi suunnitelma.
- Tehdään suunnitelma ja yritetään jatkuvasti pysyä siinä.

(Cockburn 2007, 247-248.)

2.3.1 AM

Agile Modeling –menetelmän tarkoitus on toimia tehokkaana ja ketteränä mallinnus- ja dokumentointityökaluna. AM on kokoelma arvoja, periaatteita ja ohjelmistojen mallinnuskäytäntöjä, jotka voidaan ottaa käyttöön ohjelmistokehitysprojektissa tehokkaalla ja ketterällä tavalla.

AM ei ole itsessään kokonainen menetelmä, vaan sen käytäntöjä tulee soveltaa muihin menetelmiin. Tällaisia ovat mm. XP, Scrum ja RUP. XP-menetelmää(Extreme Programming, ks. 2.3.4) voidaan pitää luonnollisena valintana haettaessa menetelmää, johon Agile Modelingia voidaan soveltaa, koska XP itse sisältää mallinnusta, jonka toteuttamiselle ei kuitenkaan ole tarkkoja vaatimuksia.(Ambler 2007.)

AM:n pääperiaatteita ovat:

- Tarkoituksenmukainen mallinnus ja asiakkaan tarpeiden tuntemus
- Rahoittajan investoinnin kannattavuuden maksimointi
- ”Kevyesti matkustaminen”; tarpeettoman monen mallin käyttö vähentää ketteryyttä
- Useamman mallin käyttö mallien tehtäväkohtaisten aspektien takia
- Välittömän asiakaspalautteen tarpeellisuus
- Yksinkertaisimman ratkaisun suosiminen ohjelmoinnissa ja mallinnuksessa
- Muutosvalmius ja myönteinen suhtautuminen muutoksiin
- Inkrementaalinen kehitys
- Korkean laadun tuottaminen
- Toimiva ohjelmisto on päätavoite
- Seuraavan tehtävän mahdollistaminen, esim. tehdyn ohjelmiston laajennus, tulevaisuuden huomioon ottaminen.

(Ambler 2007.)

2.3.2 Lean

Lean-johtamistavan filosofia on lähtöisin 1950-luvun Toyota-yhtiön toimintatavasta, joka on sittemmin levinnyt laajalti muiden yhtiöiden ja teollisuudenalojen käyttöön. Toyotan tavan tehokkuus, laatu, henkilöstön tyytyväisyys ja reilu lähestymistapa alihankkijoihin ovat toivottuja tekijöitä myös ohjelmistokehityksessä. John ja Mary Poppendieck ovat

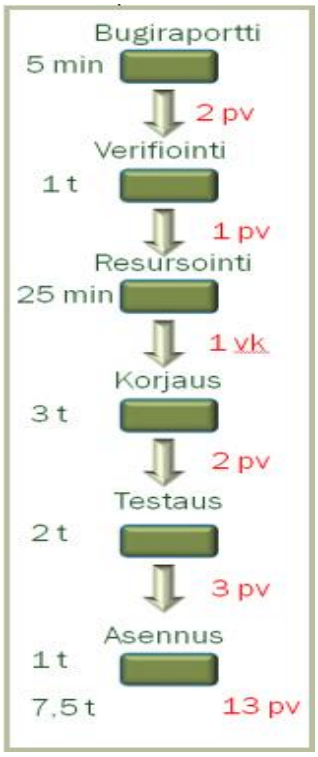
tutkineet Toyotan tapaa tuodakseen nämä tekijät ohjelmistokehityksen käyttöön ja tämän seurauksena on syntynyt Lean Software Development.(Poimala ym. 2010.)

Kaikki Lean-lähestymistavat liittyvät arvoa tuottamattomien tekijöiden eliminointiin. Ohjelmistokehityksessä tällaisia tekijöitä ovat:

- Osittain tehty työ
- Ylimääräiset prosessit
- Ylimääräiset ominaisuudet
- Tehtävien vaihdot, useamman tehtävän suorittaminen samanaikaisesti
- Odotus
- Tehtävien luovutukset henkilöltä toiselle, vastuun siirto
- Viat

(Poppendieck 2003.)

Lean-menetelmä ei ota kantaa ohjelmoijan päivittäiseen työskentelyyn(vrt. XP, ks. 2.3.4), eikä projektin organisoimiseen(vrt. Scrum, ks. 2.4), mutta se tarjoaa joukon periaatteita, joita on hyvä noudattaa ohjelmistokehityksessä. Lean-ajattelun selkärankana toimii arvovirtakartta (Kuvio 3), joka kuvaa prosessin vaiheita ja niihin käytettyä aikaa, sekä eri vaiheiden välistä aikaa, jolloin ei tapahdu mitään asiakkaan näkökulmasta katsottuna hyödyllistä.(Poimala ym. 2010.)



Kuvio 3. Arvovirtakartta. (Poimala ym. 2010)

Arvovirtakartan avulla saadaan selvitettyä, miten paljon missäkin prosessin vaiheessa tehdään hyödyllistä työtä ja kuinka paljon prosessissa on tuottamatonta osuutta esim. viiveiden muodossa. Tehollisen työn ja tuottamattoman työn suhde on prosessin hyötysuhde, jota organisaation kannattaa pyrkiä optimoimaan em. tuottamattomien tekijöiden eliminoinnilla. (Poimala ym. 2010.)

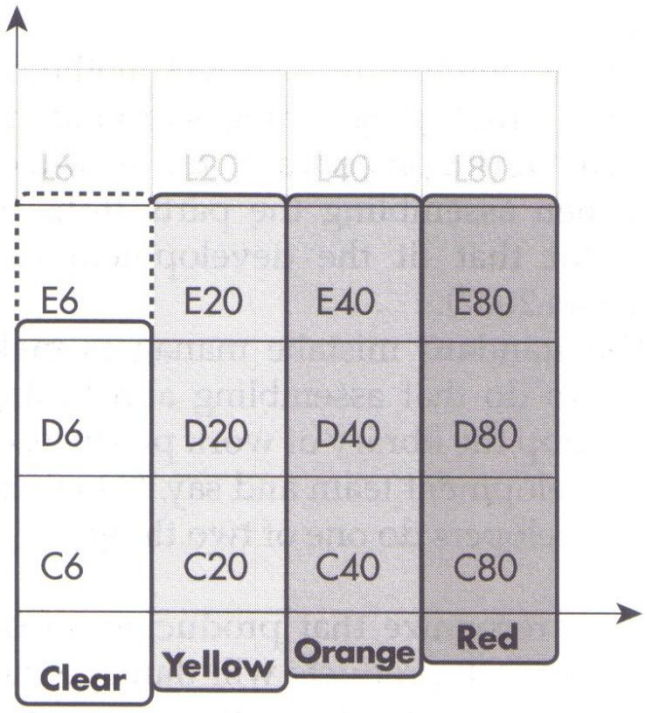
2.3.3 Crystal

Alistair Cockburnin esittelemän Crystal-metodologiaperheen avulla voidaan muodostaa sopiva menetelmä projektin tyypistä, projektiryhmän koosta ym. olosuhteista riippuen. Metodologiaa kuvataan projektiruudukon(Kuvio 4) avulla. Ruudukossa oikealle liikkuminen merkitsee isomman ryhmän koordinoimista, mikä edellyttää raskaampaa menetelmää. Tätä kuvataan sitä tummemmalla värillä, mitä enemmän ruudukossa mennään oikealle. Ruudukon vaaka-akselin lukemat ilmaisevat projektiryhmän henkilölukumäärän. Pystyakseli kuvastaa mahdollisten vahinkojen vakavuusastetta projektin epäonnistuessa (Cockburn 2007, 335-338). Pystyakselin kirjaimet tarkoittavat:

- C = Mukavuuden heikkenemistä (loss of Comfort)
- D = Rahan menetystä (loss of Discretionary money)

- E = Merkittäviä taloudellisia menetyksiä (loss of Essential money)
- L = Ihmishenkien menetyksiä (loss of Life)

(Poimala ym. 2010).



Kuvio 4. Crystal-menetelmät on nimetty värin mukaan. (Cockburn 2007: 338)

Kaikki Crystal-menetelmät edellyttävät inkrementaalista kehitystä. Crystal Clear on menetelmistä kevyin ja se soveltuu pääasiassa D6-kategorian projekteihin, joskin sitä voidaan venyttää ulottumaan E8- ja D10-kategorioihin saakka sillä edellytyksellä, että kiinnitetään erityistä huomiota kommunikaation ja testauksen toimivuuteen.(Cockburn 2007, 340.)

Lyhyt Crystal Clear –määritelmä:

- Yksi ryhmä samassa toimistotilassa, välitön kommunikointi.
- Inkrementaalinen ja säännöllinen julkaisu 2-3 kuukauden välein
- Edistystä seurataan etapeilla, jotka koostuvat ohjelmatoimituksista
- Tehdään jonkin verran ohjelman toimivuuden automaattista regressiotestausta
- Käyttäjän välitön mukanaolo
- Kaksi käyttäjäesittelyä julkaisua kohden
- Julkaisun valmistelu alkaa heti kun ohjelma on riittävän vakaa esiteltäväksi

- Tarvittava tuotteen ja menetelmien säätö jokaisen inkrementin alussa ja puolivälissä

(Cockburn 2007, 340-342.)

2.3.4 XP

XP eli Extreme Programming on vahvasti ohjelmointikeskeinen, iteratiivinen ja inkrementaalinen menetelmä. XP-projekti alkaa tutkimusjaksolla, jonka aikana selvitetään, arvioidaan ja priorisoidaan tietojärjestelmän vaatimuksia. Kun vaatimuksia on tiedossa riittävästi, jotta voidaan tuottaa pienin asiakkaalle arvoa tuottava järjestelmä, suunnitellaan ensimmäinen julkaisu. Samanaikaisesti tehtävät tutkimukset antavat uusia vaatimuksia, joiden pohjalta voidaan suunnitella uusia julkaisuja. Suunnitellut julkaisut jaetaan iteraatioihin, joista jokainen tuottaa sisältöä, jolla on arvoa asiakkaan näkökulmasta. (Newkirk & Martin 2001, 9.)

XP-projektin tutkimusjakso korvaa perinteisen dokumentoidun vaatimusmäärittelyn. Ohjelmoijat ja asiakkaan edustajat kokoontuvat keskustelemaan asiakkaan tarpeista, joita kuvaamaan asiakas kirjoittaa lyhyitä käyttäjätarinoita, joita tullaan myöhemmin hyödyntämään iteraatioiden aikana tapahtuvissa hyväksymistesteissä. Projekti jaetaan julkaisuihin, joista kukin vie aikaa yhdestä kolmeen kuukautta. Kukin julkaisu jaetaan iteraatioihin, joiden kesto on yhdestä kolmeen viikkoa. (Newkirk & Martin 2001, 10-11.)

Iteraatioon budjetoituja tehtäviä ei määrätä henkilöiden tehtäviksi, vaan ohjelmoijat saavat itse valita haluamansa tehtävän edellyttäen, että henkilökohtainen työmäärä pysyy samana iteraatiosta toiseen. Vaikka yksittäiset ohjelmoijat ovatkin vastuussa valitsemastaan tehtävästä, suoritetaan varsinainen ohjelmointi *pariohjelmointina*; kaksi ohjelmoijaa toimii samalla työasemalla yhden kirjoittaessa koodia ja toisen avustaessa (Newkirk & Martin 2001, 12). Pariohjelmoinnin kannattavuudesta tehtyjen tutkimusten perusteella koodia syntyy n. 1,5-kertaisella nopeudella, eli hitaammin kuin kaksi ohjelmoijaa saisi aikaan omilla tahoillaan, mutta laadullisesti pariohjelmointina toteutettu koodi on moninkertaisesti yksin tehtyä parempaa (Poimala ym. 2007).

Ohjelmistoa kehitetään testitapaus kerrallaan. Ohjelmoijapari valitsee osan käsillä olevasta tehtävästä työstettäväkseen, suunnittelee testitapauksen ja lopuksi kirjoittaa koodin, joka läpäisee testin. Tämän jälkeen siirrytään seuraavaan testitapaukseen.

Testaus-/kehitysjaksot ovat lyhyitä, n. 5-10 minuuttia. Kaikki testitapaukset on koottu yksikkötestausympäristöön ja testit ajetaan aina koodin muututtua. Kaikki testit on läpäistävä ennen ja jälkeen kirjoitetun koodin integroimista pääohjelmaan. (Newkirk & Martin 2001, 13.)

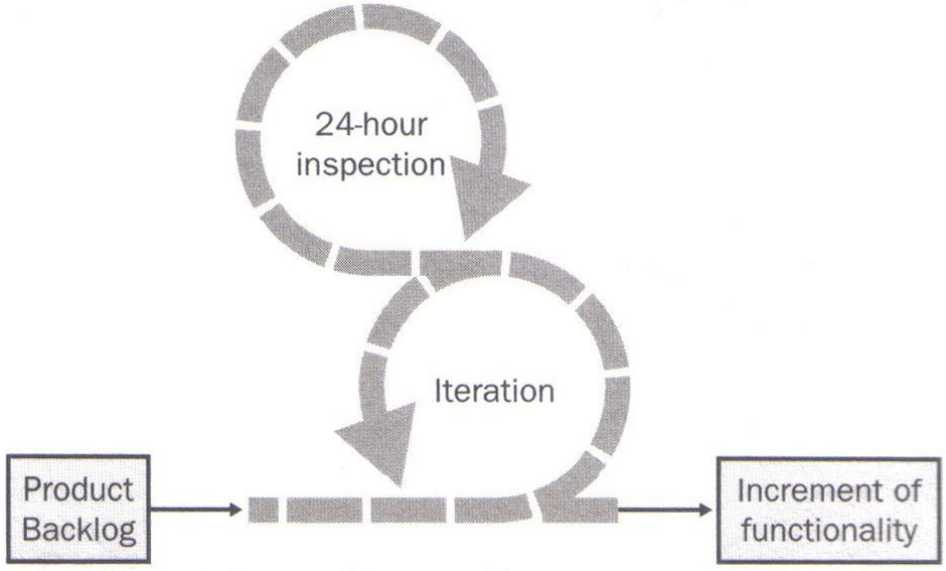
XP-ohjelmoijat noudattavat seuraavia periaatteita:

- Ohjelmoijat eivät ennakoi tulevia ominaisuuksia, vaan keskittyvät sen ohjelmakoodin kirjoittamiseen, jota tarvitaan nykyisen tehtävän suorittamiseen.
- Ohjelmoijat harjoittavat säännöllistä refaktorointia, eli olemassaolevan koodin yksinkertaisuuden ja ylläpidettävyyden parantamista. Refaktoroinnin yhteydessä testauksen merkitys korostuu.
- Koodin integroiminen pääohjelmaan on tehtävä mahdollisimman usein, vähintään kerran päivässä.
- Ohjelmoijat eivät omista kirjoittamaansa tai muokkaamaansa koodia; kaikki modulit ovat kaikkien ohjelmoijaparien saatavilla.
- Ohjelmoijat säilyttävät tasaisen työtahdin. Ylitöitä ei tehdä liikaa, jotta ajatustyötä vaativan ohjelmoinnin laatu ei heikkenisi.

(Newkirk & Martin 2001, 13.)

2.4 SCRUM

Scrum on projektin ohjaukseen soveltuva malli, joka ei ota kantaa matalan tason käytäntöihin, kuten esim. ohjelmointiteknisiin asioihin, vaan keskittyy projektin vaiheistamiseen ja sen etenemisen kontrollointiin (Poimala ym. 2007). Kaikki Scrumin käytännöt sisältyvät iteratiiviseen ja inkrementaaliseen prosessikehykseen (Kuvio 5), joka koostuu iteraatiosta eli *sprintistä* ja iteraation aikana päivittäin toistuvasta 24:n tunnin syklistä.



Kuvio 5. Scrum-prosessikehys. (Schwaber 2004: 6)

2.4.1 Scrumin roolit

Scrum-rooleja on ainoastaan kolme; tuotteen omistaja(Product Owner), tiimi(Team) ja ScrumMaster. Projektinhallinnallinen vastuu jaetaan näiden kolmen roolin kesken. Projektin henkilöt ovat Scrumissa *sikoja* tai *kanoja*. Sioiksi luetaan henkilöt, joilla on jokin Scrum-rooli(Product Owner, Team, ScrumMaster) ja kanoiksi henkilöt, jotka ovat jollain tavalla kiinnostuneita projektista, mutta eivät ole missään Scrum-roolissa, eivät kanna vastuuta projektin kulusta, eivätkä näin ollen saa osallistua projektiin muuten kuin sivusta seuraamalla.(Schwaber 2004, 7.)

Tuotteen omistaja hankkii rahoituksen projektille laatimalla vaatimusmäärittelylistan ja suunnitelman/arvion projektin eri vaiheiden vaatimasta ajankäytöstä. Arviot tehdään päivän tarkkuudella. Tämä lista on tuotteen työlista, eli *Product Backlog*. Lista kehittyy ja muuttuu liiketaloudellisten olosuhteiden tai teknologian muutoksien sitä vaatiessa (Schwaber 2004, 142). Tuotteen omistaja priorisoi jatkuvasti tuotteen työlistan sisältämiä ominaisuuksia varmistaakseen, että arvokkaimmat ominaisuudet toteutetaan ensiksi.

Tiimi vastaa tuotteen toiminnallisuuden kehittämisestä. Tiimit ovat itseohjautuvia ja niiden vastuulla on selvittää, kuinka tuotteen työlistasta voidaan tuottaa toiminnallisuutta sisältävä inkrementti yhden iteraation(ks. 2.2) aikana, sekä organisoida oma työskentelynsä tämän saavuttamiseksi. Tiimin jäsenet kantavat kollektiivisen

vastuun iteraation tavoitteen saavuttamisesta ja projektista kokonaisuudessaan.(Schwaber 2004, 6-7.)

ScrumMasterin vastuulla on Scrum prosessina, Scrumin opettaminen projektiin osallistuville henkilöille sekä Scrumin käyttöönotto organisaatiossa vallitsevat käytännöt ja kulttuuri huomioiden niin, että sen edut saadaan hyödynnettyä. Lisäksi ScrumMaster vastaa siitä, että kaikki projektin henkilöt noudattavat Scrumin sääntöjä ja käytäntöjä. (Schwaber 2004, 7.)

2.4.2 Scrum-projektin kulku

Scrum-projekti alkaa visiosta, joka ei välttämättä ole aluksi täysin selkeä. Tuotteen omistaja, *Product Owner*, vastaa vision esittelystä rahoittaville tahoille tavalla, joka vakuuttaa nämä investoinnin palautumisesta. Tämän vision pohjalta tuotteen omistaja työstää tuotteen työlistan, eli *Product Backlogin*, jonka priorisointi tehdään eniten arvoa tuottavien ominaisuuksien mukaan; nämä muodostavat erillisiä julkaisuja. Priorisoitu tuotteen työlista mahdollistaa projektin aloittamisen. On kuitenkin tavallista, että työlistan sisältö, priorisointi ja julkaisujen ryhmittely muuttuu jo projektin aloitusvaiheessa.(Schwaber 2004, 7-8.)

Työ tehdään *sprintteinä*, jotka ovat 30:n kalenteripäivän pituisia iteraatioita. Jokaisen sprintin alussa järjestetään sprintin suunnittelukokous, jonka aikana tuotteen omistaja ja tiimi selvittävät seuraavan sprintin aikana tehtävät työt. Tuotteen omistaja esittää tiimille tuotteen työlistan korkeimman prioriteetin ominaisuuden mukaan määräytyvän tavoitteen sprintin ajaksi. Tiimi puolestaan kertoo tuotteen omistajalle oman arvionsa siitä, kuinka paljon toivotusta tavoitteesta saadaan tehdyksi. Sprintin suunnittelukokouksille on määrätty aikarajaksi kahdeksan tuntia, jotta vältyttäisiin pohdiskelemasta liikaa sprintin aikana tehtävän työn määrän mahdollisuuksia ja päästäisiin mahdollisimman nopeasti käsiksi varsinaiseen työhön. (Schwaber 2004, 8.)

2.4.3 Päivittäinen Scrum

Nimi *Scrum* tarkoittaa *rugbyssa* käytettyä aloitusmuodostelmaa. Sprintin jokaisena järjestetään päivittäinen Scrum, kokous, jonka pituus on enintään 15 minuuttia huolimatta paikalla olevien tiimin jäsenten lukumäärästä. Kokouksen aikana seistään. Jokaisen tiimin jäsenen on osallistuttava, poissaolijoiden on joko osallistuttava puhelimen välityksellä, tai annettava toisen tiimin jäsenen raportoida puolestaan. Kokouksesta ei saa myöhästyä; myöhässä saapuva jäsen maksaa välittömästi ScrumMasterille 1 € suuruisen sakon. Kokouksen aikana jokainen tiimin jäsen vastaa kolmeen kysymykseen:

- Mitä olet tehnyt viime kertaisen Scrumin jälkeen?
- Mitä aiot tehdä seuraavaan päivittäiseen Scrumiin mennessä?
- Mitkä seikat estävät sinua saavuttamasta sprintin ja projektin tavoitteita?

Kanat (ks. 2.4.1) saavat osallistua kokoukseen, mutta eivät saa vaikuttaa millään tavalla kokouksen kulkuun. He eivät myöskään saa esittää kysymyksiä tai ehdotuksia tiimin jäsenille kokouksen jälkeen (Schwaber 2004, 135-136). Päivittäisen Scrum-kokouksen tarkoituksena on pitää tiimin jäsenet päivittäin ajan tasalla toistensa työtehtävien suhteen, sekä päättää mahdollisista tulevista kokoontumisista tiimin edistyksen eteenpäinvälittämiseksi (Schwaber 2004, 8).

2.4.4 Tiimin toiminta Sprintin aikana

Tiimi voi sprintin aikana hakea ulkopuolista apua, neuvoja, informaatiota ja tukea. Kukaan ulkopuolinen ei kuitenkaan saa itse tarjota apua, kommentteja tai neuvoja tiimille sprintin aikana. Tiimi sitoutuu tuotteen työlistaan sprintin suunnittelukokouksen aikana. Kukaan ei saa muuttaa työlistaa ennen sprintin loppua. Jos sprintti osoittautuu epäonnistuneeksi, voi ScrumMaster keskeyttää sen ja järjestää uuden sprintin suunnittelukokouksen aloittaakseen seuraavan sprintin. ScrumMaster voi tehdä päätöksen sprintin keskeyttämisestä oman harkintansa mukaan, tai tiimin pyynnöstä. Jos tiimin jäsenet tuntevat olevansa kykenemättömiä saamaan valmiiksi jonkin tai useamman sprintin aikana tehtäväksi suunnitelluista ominaisuuksista, voivat he ottaa yhteyttä tuotteen omistajaan karsiakseen nämä pois. Jos ominaisuuksia joudutaan karsimaan niin paljon, että sprintti menettää merkityksensä, voi ScrumMaster keskeyttää sprintin em. tavalla. Tuotteen omistaja voi vastaavasti, tiimin niin halutessa,

mahdollisuuksien mukaan lisätä ominaisuuksia sprintin aikana tehtäväksi. (Schwaber 2004, 136-137.)

2.4.5 Sprintin katselmointi

Sprintin lopulla järjestetään sprintin katselmointi, eli kokous, jonka aikarajaksi on asetettu neljä tuntia. Katselmoinnin aikana tiimi esittelee valmiit tuotokset tuotteen omistajalle ja muille asianomaisille. Ainoastaan toiminnallisuutta sisältäviä tuotoksia voidaan esitellä, ei esim. dokumentteja. Toiminnallisuus esitellään jonkun tiimin jäsenen työasemalta ja toimintaympäristön on oltava mahdollisimman paljon lopullista tuotantoympäristöä vastaava.(Schwaber 2004, 137.)

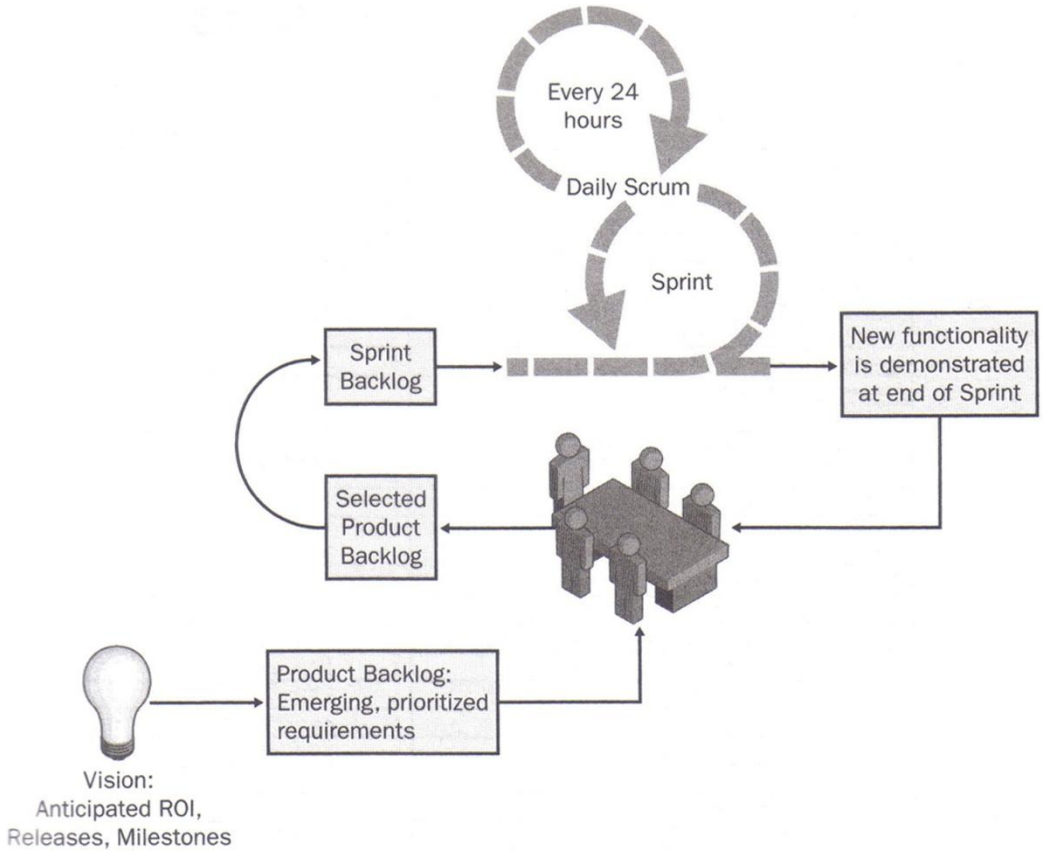
Katselmoinnin aluksi tiimin jäsen esittelee sprintin tavoitteen, työstettävänä olevan tuotteen työlistan, sekä valmiiksi saadun osuuden tuotteen työlistasta. Muut tiimin jäsenet voivat esittää kommentteja sprintin sujumisesta ja asioista, jotka menivät hyvin sekä niistä asioista, jotka menivät heikommin. Suurin osa katselmoinnille varatusta ajasta käytetään toiminnallisuuden esittelyyn, kysymyksiin vastaamiseen ja muistiinpanojen tekoon toivottujen muutosten osalta. Esittelyn jälkeen haastatellaan paikalla olevia sidosryhmien edustajia, jotta saadaan selvitetyksi heidän vaikutelmansa sprintin onnistuneisuudesta sekä muutostoiveet ja niiden priorisointi. Kokouksen lopuksi ScrumMaster ilmoittaa seuraavan sprintin katselmoinnin ajankohdan tuotteen omistajalle ja sidosryhmien edustajille.(Schwaber 2004, 137-138.)

Katselmoinnin jälkeen ja ennen seuraavan sprintin suunnittelukokousta järjestetään sprintin jälkitarkastelu, johon osallistuvat tiimin jäsenet, ScrumMaster, sekä mahdollisesti tuotteen omistaja. Jokainen tiimin jäsen vastaa kahteen kysymykseen:

- Mikä meni hyvin kuluneen sprintin aikana?
- Mitä voitaisiin parantaa seuraavaan sprinttiin?

ScrumMaster tekee yhteenvedon tiimin jäsenten vastauksista, jonka jälkeen tiimi asettaa parannusehdotuksensa tärkeysjärjestykseen. Merkittävät muutokset priorisoidaan korkealle seuraavan sprintin työlistalla.(Schwaber 2004, 139.)

Sprintin suunnittelukokous, päivittäinen Scrum, sprintin katselmointi ja jälkitarkastelu pohjustavat yhdessä Scrumin empiirisen havainnoinnin ja muodostavat sen sopeutuvat käytännöt (Kuvio 6).(Schwaber 2004, 9.)



Kuvio 6. Scrum-prosessi kokonaisuudessaan. (Schwaber 2004, 9)

3 Puhelujen tilastointi

3.1 Tilastoitavan tiedon kerääminen

StarSoft Oy:llä on käytössä Siemens Hicom-130 –puhelinkeskus, jonka tuottama puhelukohtainen data on mahdollista tallentaa yhdistämällä keskus tietokoneeseen sarjaportin kautta. Nykyisillä asetuksilla keskus antoi puhelukohtaisesti seuraavat tiedot puhelun päätyttyä: päivämäärä, kellonaika, käytetty puhelinlinja, vastaamiseen kulunut aika sekunteina, alanumero(ts. puheluun vastanneen henkilön puhelinnumero), puhelun kesto, puhelinnumero(ts. ulkoinen numero, josta soitettiin) ja koodinumero puhelun tyypille(soitettu, vastattu, käännetty).

Puhelinkeskuksen uudelleenkonfiguroinnilla olisi voitu saada käyttöön muitakin tietoja, mutta nykytilanteessa saatavien tietojen katsottiin riittävän kattavan tilastointijärjestelmän rakentamiseksi. Tiedot, joita ei saatu suoraan keskuksen antamasta datasta pystyttiin luomaan tai muokkaamaan ohjelmallisesti tietokantaan tallennusta varten. Tällaisia olivat lähinnä käännettyihin puheluihin liittyvät tiedot, sekä mm. viikonloppujen ja arkipyhien tunnistaminen.

Puhelinkeskuksen tietokoneeseen yhdistämisen lisäksi tarvittiin sovellus, joka lukee tietokoneen sarjaportin vastaanottaman datan ja tallentaa sen tietokantaan. Lisäksi tallennettujen tilastojen tarkastelua varten tarvittiin toinen sovellus, joka tehtäisiin omaksi osiokseen yrityksessä käytössä olevaan www-ympäristössä toimivaan sähköpostiohjelmaan. Tämän vuoksi puhelutilastointiin liittyvät tiedot olisi tallennettava omiin tauluihinsa sähköpostiohjelman käyttämään MySQL-tietokantaan.

3.2 Vanha tilastointijärjestelmä ja kehitystarpeet

Aikaisemmin käytössä ollut tilastointijärjestelmä oli toteutettu yhdistämällä puhelinkeskus ns. ”mustaan laatikkoon”, eli lisälaitteeseen, joka tallensi keskuksen tuottaman datan. Tämä laite oli puolestaan yhdistetty tietokoneeseen, johon oli asennettu Siemensin valmistama sovellus, jonka avulla voitiin ladata ”mustan laatikon” tiedot koneen kiintolevylle sekä tarkastella puheluiden kokonaismääriä ym. Sovelluksen avulla voitiin myös tulostaa tilastot csv-tiedostoon, jolloin niitä pystyttiin mm. tarkastelemaan taulukkolaskentaohjelmassa. Lisäksi StarSoftilla oli käytössä itse tehty graafi-sovellus, joka muodosti pylväskuvaajan csv-tiedostojen perusteella.

Vanha tilastointijärjestelmä sisälsi kuitenkin heikkouksia, joiden johdosta oli ajankohtaista siirtää tilastointi uudelle alustalle. Tällaisia olivat mm:

- Tietokannan puute
- Henkilö-/alanumerokohtaisten tilastojen puute
- Puhelujen keston, ajankohdan, tai vastausajan perusteella rajatun tilaston puute
- Riippuvuus ”mustasta laatikosta” ja ulkoisen tahon tekemästä ohjelmasta
- Erillisen ohjelman tarve graafisten kuvaajien muodostamista varten
- Reaaliaikaisen tietojen tallennuksen puute
- Puhelukohtaisen seurannan puute

Uuden tilastointijärjestelmän olisi sisällettävä jo aiemmin käytössä olleet toiminnot sekä halutut parannukset.

4 Puheluloggeri

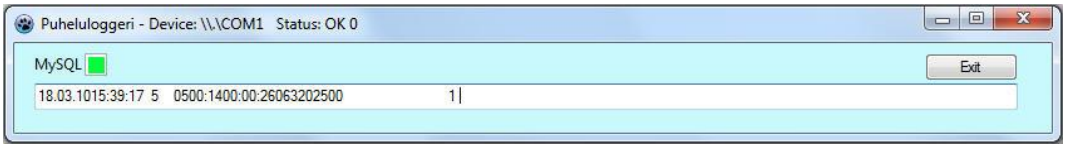
Sarjaportin lukemiseen tarkoitettuja sovelluksia löytyi Internetistä ilmaisohjelmina, mutta niiden hyödyntämistä ei pidetty parhaana ratkaisuna, koska lisäksi olisi löydettävä erillinen sovellus tietokantaan tallennusta varten. Tämän vuoksi päädyttiin tekemään itse puheluloggeri, joka lukisi tiedot reaaliaikaisesti tietokoneen sarjaportista, tekisi tarvittavat muokkaukset tietoihin ja tallentaisi ne MySQL-tietokantaan.

4.1 Lazarus/Free Pascal –kehitysympäristö

Puheluloggeri päädyttiin tekemään Lazarus/Free Pascal -kehitysympäristössä. Free Pascal on avoimen lähdekoodin Pascal-kääntäjä joka tukee TurboPascalin/Object Pascalin/Delphin syntaksia, jonka ansiosta koodiesimerkkejä ja muita ohjeita oli helppo löytää Internetistä mm. Delphi-aiheisilta foorumeilta. Lazarus-projekti puolestaan sisältää luokkakirjastoja Free Pascalille. Koko paketti on siis periaatteessa ilmainen Delphi-klooni (Baeseman & Miller & Hess 1999).

4.2 Puheluloggerin käyttöliittymä

Puheluloggerin käyttöliittymä on hyvin yksinkertainen. Puheluloggeri on varustettu tekstikentällä, jonka ainoa tehtävä on näyttää puhelinkeskuksen tuottama merkkijono, joka on luettu tietokoneen sarjaportista. Lisäksi käyttöliittymässä on paneeli, jonka tarkoitus on ilmoittaa MySQL-yhteyden tila, sekä painike ohjelman lopetusta varten (Kuvio 7).



Kuvio 7. Puheluloggerin käyttöliittymä.

4.3 Sarjaportin luku

Puheluloggeria ohjelmoitaessa oli ensin saatava puhelutiedot ohjelman käyttöön käsittelyä ja tietokantaan tallennusta varten, joten ensimmäinen askel ohjelmointityössä oli sarjaportin lukeminen. Tässä hyödynnettiin Ararat Synapse –projektia, joka on Windowsin verkkoliikennettä varten rakennettu luokkakirjasto (Gebauer 2007). Projekti

sisältää Synaser-laajennuksen, jota käytettiin tietokoneen sarjaportin kanssa kommunikoimiseen.

Puheluloggeri ohjelmoitiin lukemaan sarjaporttia puolen sekunnin välein ja tallentamaan luettu data merkkijonomuuttujaan. Muuttujan arvon muuttuminen merkitsee sitä, että sarjaportista on luettu uuden puhelun tiedot. Kun näin tapahtuu, on viimeksi luetut tiedot tallennettava kutsumalla tallennustoiminnon suorittavaa funktiota ja lähettämällä sille parametrina em. merkkijonomuuttuja joka sisältää puhelinkeskuksen antamat puhelukohtaiset tiedot; päivämäärä, kellonaika, käytetty puhelinlinja, vastaamiseen kulunut aika sekunteina, alanumero, puhelun kesto, ulkoinen puhelinnumero ja koodinumero puhelun tyypille.

4.4 Puheludatan käsittely ja tallennus

Kun sarjaportista on luettu uuden puhelun tiedot, tallennetaan ne tekstitiedostoon sekä tietokantaan. Tallennuksen suorittavan funktion parametrina saamaan merkkijonoon tallennettu sarjaportista luettu data pilkotaan omiin muuttujiinsa, jotta niiden käsittely ja tietokantaan tallennus olisi mahdollista. Jos puheluloggerin sijaintihakemistosta ei löydy tiedostoa nimeltä puheluloki.txt, luodaan sellainen. Tähän tiedostoon tallennetaan uusi rivi, joka koostuu puheludatamuuttujista. Muuttujat erotetaan toisistaan puolipisteellä. Tämä mahdollistaa tarvittaessa puhelulokitiedoston avaamisen taulukkolaskentaohjelmassa.

Ennen tietokantaan tallennusta on selvitettävä kyseessä olevan päivän tyyppi, ts. onko päivä arkipäivä, viikonloppu, vai arkipyhä. Päivän tyypin selvitys tehdään, jotta voitaisiin myöhemmin seuloa tilastoinnista pois viikonloput ja pyhäpäivät. Päivätyypin selvitystä varten on tehty erillinen funktio, jolle lähetetään parametrina päivämäärä ja saadaan paluuarvona päivätyyppi, joka tässä vaiheessa tallennetaan omaan muuttujaansa.

Päivätyypin selvityksen jälkeen muunnetaan päivämäärä- ja kellonaika-muuttujat Unix-aikaleimaksi tätä varten tehdyn funktion avulla. Aikamuuttujat muutetaan aikaleimamuotoon, jotta esim. päivämäärän perusteella tehdyt sql-kyselyt toimisivat luotettavammin.

Puhelutyyppimuuttuja sisältää vielä tässä vaiheessa puhelinkeskuksen koodin puhelutyypille. Koodi on joku seuraavista:

- **1** (vastattu puhelu)
- **2 (**soitettu puhelu)
- **35 (**vastattu, käännetty puhelu)
- **36** (soitettu, käännetty puhelu)

Koodit muutetaan seuraavasti:

- **1** -> ”v”
- **2** -> ”s”
- **35** -> ”k”
- **36** -> ”z”

Tämän jälkeen tallennetaan tiedot MySQL-tietokantaan kutsumalla siihen tarkoitettua funktiota. Funktion parametreiksi annetaan kaikki puheludatamuuttujat, paitsi linjan numeron sisältävä muuttuja, koska sen tallennusta tietokantaan ei pidetty tarpeellisena.

4.5 MySQL-tietokannan käyttö

Puheluloggeri tallentaa puhelutiedot StarSoftin sähköpostiohjelman käyttämään MySQL-tietokantaan, koska puhelutilastosovellus toteutetaan sähköpostiohjelman laajennuksena. Puheluloggerin käynnistyksen yhteydessä muodostetaan yhteys tietokantaan ja tehdään yksinkertainen Puhelut-tauluun kohdistuva SHOW COLUMNS –kysely, jolla selvitetään virheenkäsittelyn avulla ainoastaan, onko kyseinen taulu olemassa (Taulukko 1). Jos taulua ei ole, se luodaan ja indeksoidaan samalla taulun sarakkeet tulevaisuudessa tehtävien hakujen nopeuttamiseksi.

Taulukko 1. TableExists()-funktio tarkistaa, onko puhelut-taulu olemassa.

```
function tableExists():boolean;
var
 query: string;
 sql: tSQL_Query;
begin
 try
      query:='show columns in puhelut';
      sql.init(query, conn);
      sql.select_query_init;
      sql.select_query_close;
      sql.destroy;
      result:=true;
 except
 on E:exception do result:=false;
 end;
end;
```

Puheluloggerin käynnistyttyä tehdään tietokantaan toistuvasti, kolmen sekunnin välein, SHOW TABLES –kyselyä, jotta saadaan selville MySQL-yhteyden tila (Taulukko 2).

Taulukko 2. MySQLisConnected()-funktio tutkii, onko yhteys tietokantaan olemassa.

```
function MySQLisConnected():boolean;
var
query: string;
sql: tSQL_Query;
begin
 try
      query:='show tables in '+dbname;
      sql.init(query, conn);
      sql.select_query_init;
      sql.select_query_close;
      sql.destroy;
      result:=true;
 except
 on E:exception do result:=false;
 end;
end;
```

Jos kysely ei aiheuta virhettä, tarkoittaa se, että yhteys on kunnossa. Tämän merkiksi annetaan puheluloggerin käyttöliittymän MySQL-paneelille (ks. 4.2) vihreä taustaväri, joka ilmaisee yhteyden olevan olemassa. Virhetilanteessa annetaan paneelille punainen taustaväri ja lisäksi lähetetään haluttuun sähköpostiosoitteeseen vikailmoitus.

4.6 Vikatilanneilmoitukset sähköpostitse

Puheluloggeri-ohjelma on jatkuvasti käynnissä tietokoneella, joka sijaitsee suljetussa kaapissa puhelinkeskuksen läheisyydessä. Tämän takia puheluloggerin toiminnan lakkaaminen saattaa jäädä huomaamatta. Puheluloggeri saattaa tällaisen tilanteen ylläpitäjän tietoon lähettämällä vikailmoituksen sähköpostitse puheluloggerin ini-tiedostoon määritettyyn osoitteeseen, jolloin ohjelman tiedetään tarvitsevan uudelleenkäynnistystä ja vian aiheuttajan selvittämistä. Puheluloggeri lähettää sähköpostiviestin seuraavissa tilanteissa:

- Ohjelma suljetaan
- Yhteys MySQL-tietokantaan katkeaa
- Sarjaportista luettujen tietojen käsittelyssä tapahtuu virhe
- Tietokantaan tallennuksessa tapahtuu virhe

Sähköpostin lähetyksessä hyödynnettiin em. Synapse-projektin valmiita funktioita.

4.7 Käännettyjen puhelujen päivitys tietokantaan

Puhelinkeskus toimii käännettyjen puhelujen osalta seuraavalla tavalla:

1. Henkilö A vastaa saapuvaan puheluun, puhuu 30 sekuntia ja kääntää sitten puhelun henkilö B:lle. Tässä vaiheessa puhelinkeskus lähettää puheluloggerille tiedot kyseisestä puhelusta, mukaanlukien sen keston, joka on 30 sekuntia ja tyyppikoodin, joka on 1, eli vastattu puhelu.
2. Henkilö B vastaa A:n kääntämään puheluun, puhuu 5 minuuttia ja katkaisee sitten puhelun. Puhelinkeskus lähettää puheluloggerille taas puhelun tiedot, jolloin kestoksi ilmoitetaan 5 minuuttia ja tyyppikoodiksi 35, eli käännetty puhelu.

Edellisestä tilanteesta seuraa, että saman puhelun tiedot kirjataan tietokantaan kahdesti; ensin vastattuna puheluna ja sitten käännettynä puheluna. Molemmissa tietueissa on luonnollisesti sama soittajan puhelinnumero. Jos henkilö B kääntäisi puhelun vielä kerran eteenpäin, tulisi tietokantaan kolme merkintää samalle puhelulle.

Sarjaportin kautta saatujen tietojen perusteella tiedetään siis puhelun tyyppi, eli onko kyseessä vastattu vai käännetty puhelu. Näiden tietojen avulla ei kuitenkaan pystytä näkemään, kenelle henkilö on kääntänyt vastaamansa puhelun, tai keneltä käännetty puhelu on tullut. Tämän selvittämiseksi on verrattava viimeisimmän puhelun tietoja aikaisemmin tietokantaan tallennettuihin; jos viimeisimmän puhelun tyyppi on 35 (käännetty), tulee tietokannasta löytyä aiemmin tallennettu tietue, jossa soittajan puhelinnumero on sama ja puhelun tyyppi on 1 (vastattu).

Viimeisimmän puhelun tietojen vertaaminen aiempiin puheluihin toteutetaan käyttämällä taulukkoa, jossa on 300 paikkaa. Kun uuden puhelun tiedot luetaan sarjaportista ja tallennetaan tietokantaan, tallennetaan ne myös taulukon viimeiselle paikalle. Ennen tätä käydään taulukko läpi toistolauseen avulla ja siirretään jokainen taulukossa oleva tieto taulukon edelliselle paikalle; taulukko[1] -> taulukko [0] ... taulukko[299] -> taulukko[298]. Tällä tavalla taulukosta löytyy aina 300 uusimman puhelun tiedot.

Kun taulukon viimeiselle paikalle on tallennettu uusimman puhelun tiedot ja puhelun tyyppi on ’k’(käännetty), käydään taulukko uudestaan läpi toistolauseen avulla, mutta tällä kertaa lopusta alkuun. Kun taulukosta löytyy puhelu, jossa on sama puhelinnumero kuin uusimmassa puhelussa, tiedetään tämän olevan alkuperäinen puhelu, joka on käännetty eteenpäin. Tämä puhelu löytyy myös tietokannasta ja kyseiselle tietueelle päivitetään nyt to_alanumero –tiedoksi viimeisimmän puhelun tiedoista poimittu alanumero, eli tieto henkilön, jolle puhelu on käännetty, alanumerosta. Päivitys tehdään MySql:n UPDATE-lauseella jolle annetaan WHERE-ehdoksi alkuperäisen puhelun puhelinnumero ja kellonaika. Viimeisimmän puhelun, eli käännetyn puhelun, tietueelle päivitetään from_alanumero –tiedoksi alkuperäisen puhelun tiedoista poimittu alanumero, eli sen henkilön alanumero, joka on vastannut alkuperäiseen puheluun. UPDATE-lauseen WHERE-ehdoksi annetaan nyt viimeisimmän puhelun puhelinnumero ja kellonaika (Taulukko 3). Tässä vaiheessa toistolauseesta poistutaan, koska etsitty puhelu on löytynyt.

Em. toimenpiteiden jälkeen löytyy tietokannasta tieto jokaisen vastatun puhelun osalta siitä, kenelle se on käännetty, mikäli puhelu on käännetty eteenpäin. Käännettyjen puhelujen osalta löytyy tieto siitä, keneltä käännetty puhelu on tullut.

Taulukko 3. Käännettyjen puhelujen päivityksen suorittavat funktiot.

```
procedure addToArray(const puhelurivi:Tpuhelurivi);
begin //puhelutaulun indeksien siirto ja puhelurivin tietojen tallennus puhelutaulun viimeiselle paikalle
    for indeksi := Low(puhelutaulukko) to (High(puhelutaulukko)-1) do begin
        puhelutaulukko[indeksi] := puhelutaulukko[indeksi + 1];
    end;
    puhelutaulukko[High(puhelutaulukko)] := puhelurivi;

if puhelurivi.puhtyyppi = 'k' then begin //jos puhelun tyyppi on ’k’, käydään puhelutaulu läpi lopusta alkuun
  for indeksi := (High(puhelutaulukko)-1) downto (Low(puhelutaulukko)) do begin
    if puhelutaulukko[indeksi].puhnro = puhelurivi.puhnro then begin
//jos puhelutaulusta löytyy sama puhelinnumero kuin viimeisestä puhelusta, päivitetään tietokannan
//tiedot updateForwarded()-funktion avulla ja poistutaan toistolauseesta.
// updateForwarded()-funktiota kutsutaan kahdesti ja sen parametrit määräävät päivitettävät tiedot.
    updateForwarded('to', puhelurivi.alanro, puhelutaulukko[indeksi].klo, puhelutaulukko[indeksi].puhnro);
    updateForwarded('from', puhelutaulukko[indeksi].alanro, puhelurivi.klo, puhelurivi.puhnro);
     Exit;
     end;
    end;
  end;
end;


procedure updateForwarded(suunta, alanro, klo, puhnro:string);
 var
 query: string;
 sql: tSQL_Query;
begin
    if suunta = 'to' then query := 'UPDATE puhelut SET to_alanro = "'+alanro+'" WHERE klo = "'+klo+'" AND
puhnro="'+puhnro+'";'
    else if suunta = 'from' then query := 'UPDATE puhelut SET from_alanro = "'+alanro+'" WHERE klo =
"'+klo+'" AND puhnro="'+puhnro+'";';
    sql.init(query, conn);
    sql.do_query;
    sql.destroy;
end;
```

4.8 Ini-tiedoston hyödyntäminen

Puheluloggeri tarvitsee toimiakseen eräitä määrityksiä, joita saattaa olla tarpeellista muuttaa. Tämän takia niitä ei ole ns. ”kovakoodattu”, vaan määritykset on mahdollista muuttaa puheluloggeri.ini –nimiseen tiedostoon. Tällaisia määrityksiä ovat:

- Sarjaporttiasetukset, jotka saattavat muuttua jos puheluloggeri siirretään toiselle työasemalle
- Vikailmoitusten sähköpostiosoite
- Käytettävän SMTP-palvelimen IP-osoite ja portti
- Käytettävän tietokannan tunnus ja salasana, sekä IP-osoite ja portti

Puheluloggeri.ini-tiedostoon määritetyt arvot tallennetaan puheluloggerin käynnistyessä omiin muuttujiinsa, joita käytetään ohjelman ollessa käynnissä. Jos ini-tiedostoon tehdään muutoksia, on puheluloggeri käynnistettävä uudelleen, jotta nämä muutokset saadaan käyttöön. Jos puheluloggeri.ini –tiedosto(tai jokin sen määrityksistä) puuttuu, tai se poistetaan, luodaan ohjelman sijaintihakemistoon uusi ini-tiedosto oletusmäärityksillä varustettuna puheluloggerin käynnistyksen yhteydessä (Taulukko 4).

Taulukko 4. Oletusmäärityksin luotu ini-tiedosto.

```
[RS232]
DefaultPort=COM1
BaudRate=9600
Bits=8
Parity=N
Stopbits=1

[Notifications]
Sender=puhelinkeskus@starsoft.fi
Receiver=jussi@starsoft.fi
SMTPHost=10.0.0.1
SMTPPort=25

[MySQL]
dbIP=localhost
dbName=kanta
dbUser=root
dbPasswd=root
```

5 Puhelutilastosovelluksen käyttö

Kun puheluloggeri on toiminnassa, löytyvät jokaisen puhelun tiedot MySQL-tietokannasta. Tarvitaan erillinen sovellus, jotta näistä tiedoista saataisiin kerättyä halutut tilastot. Sovelluksella tulisi voida tehdä erilaisia hakuja, joiden avulla saataisiin hakukriteerien mukaiset puhelujen lukumäärät, puhelukohtaiset tiedot, henkilökohtaiset tilastot sekä voitaisiin tuottaa graafinen esitys puhelumääristä. Käyttäjäoikeuksista riippuen tulisi olla mahdollista seurata omaa, kaikkien, tai vain jonkun tietyn henkilön puheluhistoriaa.

5.1 Tukijuhta-sähköpostiohjelma

Koska yrityksen käyttämien tietojärjestelmien lukumäärää ei haluttu turhaan kasvattaa, päädyttiin tekemään puhelutilastoja varten laajennus yrityksen käyttämään Tukijuhta-sähköpostiohjelmaan. Tämän ratkaisun ansiosta esim. käyttäjänhallintaa ei tarvinnut toteuttaa erikseen, vaan voitiin hyödyntää Tukijuhdan käyttäjätietoja, koska jokaisella yrityksen työntekijällä oli jo olemassa käyttäjätunnus tähän ohjelmaan. Myös tietokannan käsittelylle löytyi Tukijuhdasta valmis php-luokka, jota voitiin osittain hyödyntää myös puhelutilastoihin liittyvissä hauissa ja tallennuksissa. Tukijuhta on toteutettu www-sivustona. Kirjautumiseen, istuntokohtaisten tietojen käsittelyyn ja tietokannan käsittelyyn liittyvät ominaisuudet on toteutettu PHP-ohjelmointikielellä. Käyttöliittymän toiminnallisuus on koodattu Javascriptillä. Tietokannasta haut ja tallennukset tehdään asynkronisesti AJAX-tekniikkaa hyödyntämällä.

Tukijuhta käyttää iframe-tekniikkaa, jonka ansiosta useita eri toimintoja on saatu koottua saman sovelluksen alle omille välilehdilleen. Ohjelma on alunperin yrityksen yhteinen ”sähköpostilaatikko”, johon tukipalvelun osoitteeseen saapuvat sähköpostit ilmestyvät kaikkien Tukijuhta-tunnuksen omistavien henkilöiden luettavaksi ja käsiteltäväksi. Sittemmin ohjelmaan on lisätty myös yrityksen myyntituotteiden, www-sivujen, dokumentaation, ym. tuotannon kehityslista, eli ns. ”nakkilista”, johon kirjataan itse havaitut tai asiakkailta tulleet kehitysehdotukset, bugikorjaukset jne.

Kaikki Tukijuhdan käsittelemä tieto löytyy tietokannasta, mukaanlukien käyttäjätiedot, joiden muokkausta varten on olemassa erillinen käyttäjänhallinta-toiminto. Tietokantaan tallennettuja käyttäjäkohtaisia tietoja ovat mm. käyttäjätunnus, joka on sama kuin henkilön etunimi, salasana, sähköpostiosoite ym. Users-tauluun, josta

käyttäjätiedot haetaan, lisättiin uudeksi tiedoksi alanumero, eli jokaisen käyttäjän henkilökohtainen puhelinnumeron loppuosa. Puhelinkeskus antaa tietokantaan tallennettavaksi tiedoksi juuri alanumeron, joten sen löytyessä käyttäjätiedot sisältävästä taulusta saadaan puheluhakuja tehdessä käyttöön henkilön etunimi pelkän alanumeron sijaan.

Käyttäjänhallinta-toiminnolla voidaan myös määrittää käyttäjäoikeudet rastikenttien avulla. Puhelutilastojen kannalta olennainen asia on käyttäjän oikeus nähdä tilastot. Tämä oikeus on myönnetty ainoastaan esimiesasemassa oleville käyttäjille ja sen myötä käyttäjä voi tarkastella puhelutilastoja henkilökohtaisella tasolla muidenkin käyttäjien kuin itsensä osalta. Lisäksi tilasto-oikeuksilla käyttäjä näkee henkilöittäin koostetun puhelumäärätilaston, kun taas ”tavallinen” käyttäjä näkee ainoastaan puhelujen kokonaismäärät.

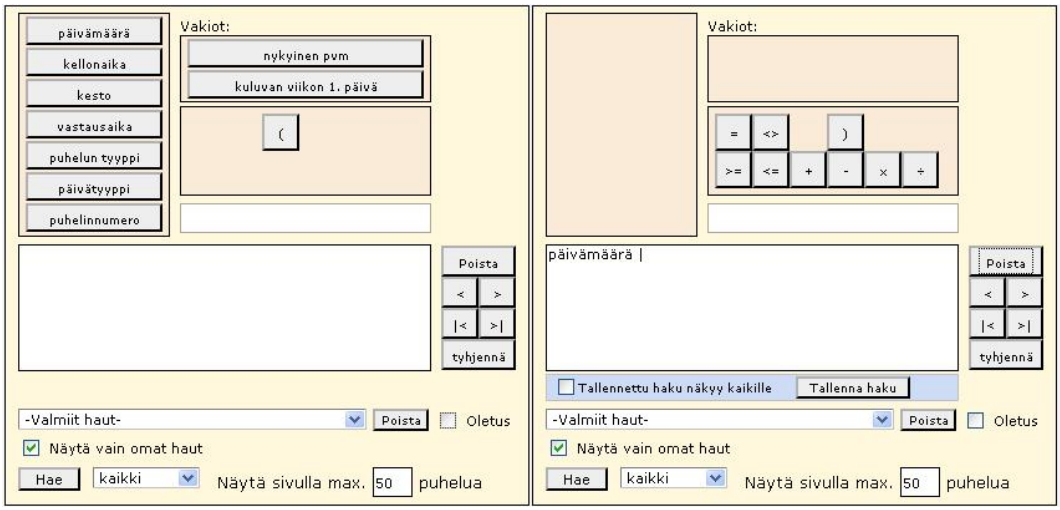
5.2 Hakulauseen rakentaminen

Toimiva ja täysin käyttäjän määritettävissä oleva tietokannasta haku osoittautui ensisijaisen tärkeäksi pohdittaessa, millä tavalla tietokannasta löytyvistä puhelutiedoista saataisiin koostettua tilastot riittävän kattavalla tavalla. Hakutoiminto toteutettiin StarSoftin päätuotteen, oppilashallinto-ohjelma Primuksen hakutoimintoa mukaillen. Henkilöt, jotka tulevat käyttämään hakutoimintoa, ovat Primus-asiantuntijoita ja tämän ohjelman hakutoiminnon logiikan ollessa entuudestaan heille tuttu, ei käytettävyysasioita tarvitsisi pohtia hakutoiminnon logiikan osalta, kun siitä tehtäisiin Primus-ohjelman haun kaltainen.

Hakulause rakennetaan siten, että se noudattaa SQL-kielen logiikkaa, mutta on hieman ”selkokielisempi”. Rakennettu hakulause käännetään syntaktisesti oikeamuotoiseksi SQL-lauseeksi ennen sen lähettämistä palvelimelle haun tekemiseksi.

Hakulause muodostetaan pala kerrallaan käyttöliittymän painikkeita, tekstikenttää, pudotusvalikoita ja kalenteria käyttämällä. Käyttöliittymän vasemmassa laidassa sijaitsevat hakuaiheet(päivämäärä, kellonaika, jne.), joita napsauttamalla kyseinen määre ilmestyy keskiosan tekstialueelle.(Kuvio 8), Ruudun keskiosan tekstialueelle muodostuvan hakulauseen perässä on kursori, joka osoittaa, mihin kohtaan seuraava pala lisätään. Tekstialueen oikealla puolella sijaitsevien nuolipainikkeiden avulla voi kursoria siirtää eteen- tai taaksepäin yhden palan verran, tai koko lauseen alkuun tai

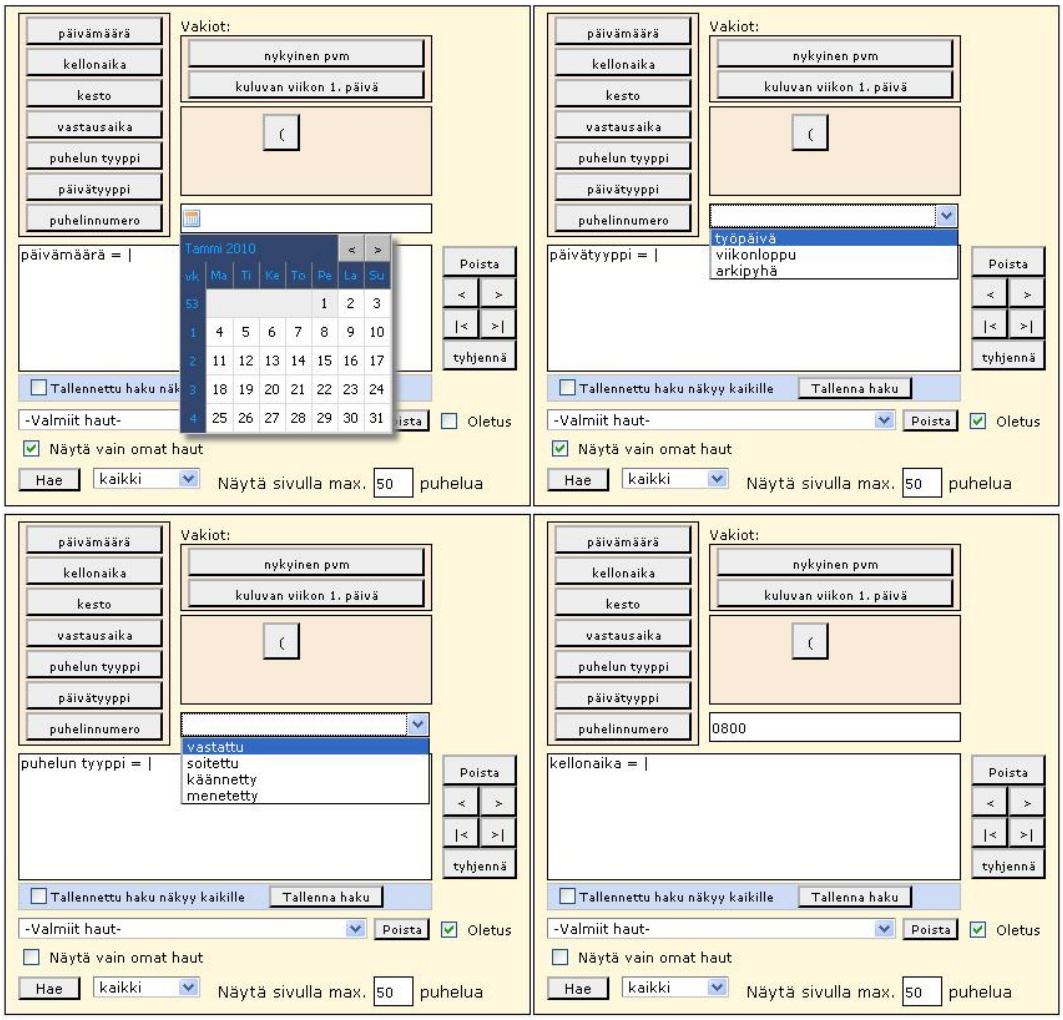
loppuun. Poista-painiketta napsauttamalla voi poistaa hakulauseesta kursoria edeltävän palan. Tyhjennä-painike poistaa koko hakulauseen.



Kuvio 8. Hakulauseen käyttöliittymä alkutilassa ja hakuaiheen lisäyksen jälkeen.

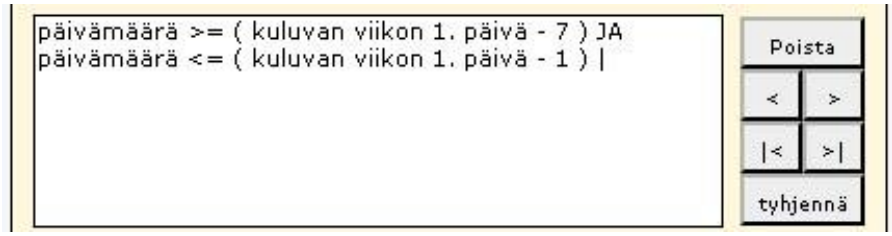
Ruutuun, jossa alkutilanteessa näkyy pelkkä avautuva sulkumerkki, ilmestyy hakulauseen rakentamisen edetessä mm. laskutoimituksiin tarkoitettuja operaattoripainikkeita, sekä JA- ja TAI-painikkeet. Painikkeet ja hakuaiheet ovat näkyvissä riippuen kursorin paikasta hakulauseessa, eli sen mukaan, miten hyvin niiden käyttö sopii valittuun hakulauseen paikkaan hakulauseen loogisuuden kärsimättä. Käytännössä tämä tarkoittaa, että esim. kahta tai useampaa hakuaihetta tai operaattoria ei voi lisätä peräkkäin, eikä toisaalta operaattoria voi jättää pois hakuaiheen ja haettavan määreen välistä. Näin ehkäistään syntaksivirheiden muodostumista ja parannetaan käytettävyyttä ”pakottamalla” käyttäjä syöttämään hakulause oikean muotoisena.

Kun hakuaihe ja esim. yhtäsuuri kuin -operaattori on syötetty, voidaan syöttää haettava arvo tekstialueen yllä olevaan tekstikenttään. Jos hakuaiheena on päivämäärä, avautuu tekstikenttää napsauttamalla kalenteri-objekti, josta voi valita päivämäärän. Jos hakuaiheena on esim. päivätyyppi, muuttuu tekstikenttä pudotusvalikoksi, josta voi valita haluamansa päivätyypin. Pudotusvalikkoa käytetään myös kun hakuaihe on puhelutyyppi, mutta tällöin valikon sisältö on erilainen. Muissa tapauksissa, eli käytettäessä kellonaikaa, kestoa, vastausaikaa, tai puhelinnumeroa hakuaiheena, on käytössä tavallinen tekstikenttä, johon voi syöttää haettavan arvon (Kuvio 9). Kellonaika ja kesto syötetään kirjoittamalla tunnit ja minuutit yhteen (esim. 0800 tarkoittaa 08:00), tai erottamalla tunnit minuuteista kaksoispisteellä. Vastausaikaa käsitellään sekunteina.



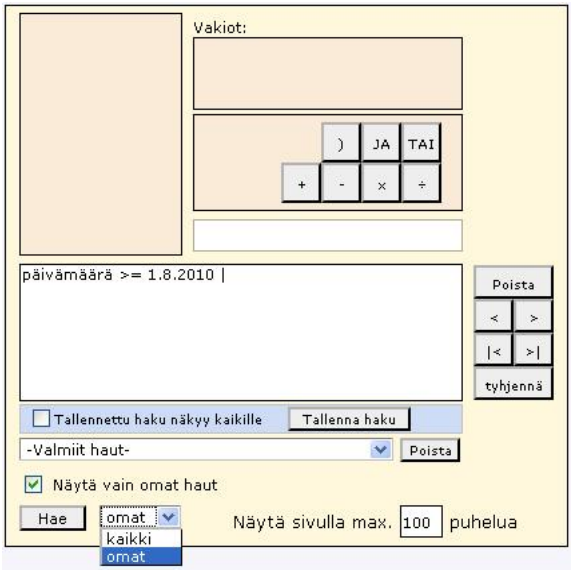
Kuvio 9. Hakulauseen valikkovaihtoehdot hakuaiheesta riippuen.

Päivämäärähaun tapauksessa voi käyttää ruudun ylälaidasta löytyviä vakioita *nykyinen pvm* ja *kuluvan viikon 1. päivä*, joista edellisen napsautus lisää hakulauseeseen senhetkisen päivämäärän ja jälkimmäisen napsauttaminen käsillä olevan viikon maanantain päivämäärän. Päivämäärävakioiden avulla voidaan hakea vaikka kaikki kuluvan viikon puhelut, tai edellisviikon puhelut vähentämällä seitsemän päivää *kuluvan viikon 1. päivä* -vakiosta ja asettamalla tämä arvo haettavan päivämäärän pienimmäksi arvoksi käyttämällä *suurempi tai yhtäsuuri kuin* –operaattoria sekä asettamalla päivämäärän suurimmaksi arvoksi kuluvan viikon ensimmäistä päivää edeltävän päivän (Kuvio 10). Vakiot ovat käteviä valmiita hakuja tehdessä (ks. 5.3).



Kuvio 10. Edellisviikon puhelujen haku.

Kun hakulause on muodostettu, voidaan tarvittaessa lisätä siihen lisää ehtoja napsauttamalla JA- tai TAI-painiketta ja syöttämällä uusi hakuehto vastaavalla tavalla kuin ensimmäinen syötettiin. Ruudun alareunassa sijaitsevasta pudotusvalikosta voi valita, koskeeko haku vain omia, vai kaikkien henkilöiden puheluita. Käyttäjille, joilla on tilasto-oikeudet, näkyy alasvetovalikossa kaikkien käyttäjien nimet, joten he voivat hakea puhelutilastot myös henkilöittäin. Ennen haun tekemistä voi lisäksi valita, montako puhelua näytetään sivulla. Oletuksena määrä on 50 (Kuvio 11).



Kuvio 11. Omat/kaikkien puhelut ja sivutus.

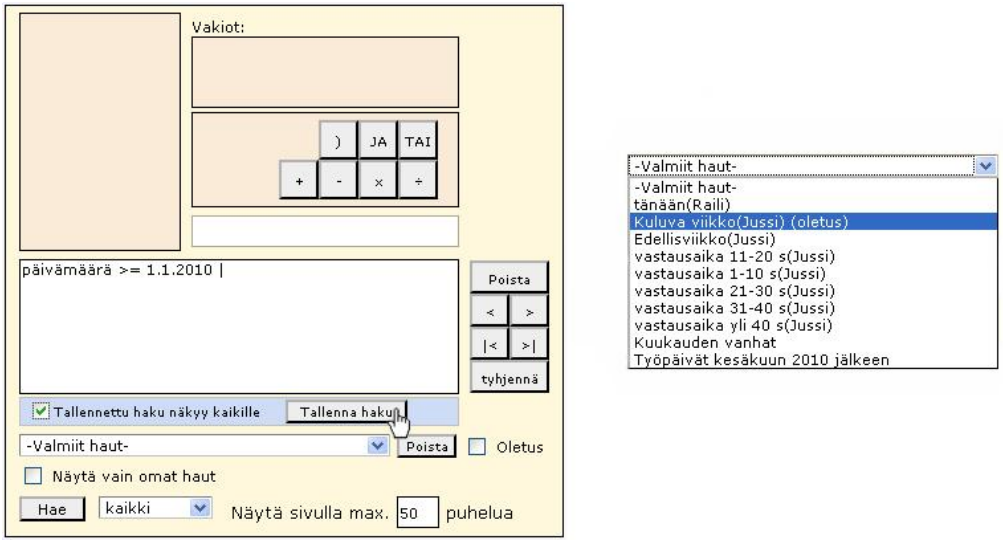
Kun hakulause on rakennettu valmiiksi, toteutetaan haku napsauttamalla Hae-painiketta, jolloin haku lähetetään AJAX-pyyntönä palvelimelle.

5.3 Valmiiden hakujen tallennus ja oletushaku

Rakennettu hakulause voidaan tallentaa, jos sitä halutaan käyttää uudelleen joutumatta rakentamaan sitä uudestaan alusta asti. Hakulause voidaan tallentaa siten, että se näkyy

vain sen tallentaneelle käyttäjälle, tai siten että se näkyy kaikille käyttäjille. Valinta tehdään *Tallennettu haku näkyy kaikille* -rastikentän avulla. Valmiin haun tallennus tapahtuu napsauttamalla *Tallenna haku* –painiketta, jonka jälkeen annetaan tallennettavalle haulle nimi. Tallennetut haut näkyvät omassa pudotusvalikossaan. Valikon alla oleva *Näytä vain omat haut* -rastikenttä määrää, näytetäänkö valikossa vain itse tallennetut haut, vai myös muiden mahdollisesti tallentamat haut, jotka on määritetty näkymään kaikille. Valikossa näkyvät toisen henkilön tallentamat haut erottaa omista hauista siitä, että niiden perässä lukee suluissa haun tallentaneen henkilön nimi .

Valikkoon valitun haun poisto tehdään napsauttamalla valikon oikealla puolella sijaitsevaa *Poista* -painiketta. Kaikille näkyvät haut ovat kaikkien poistettavissa. Valmiin haun voi asettaa oletushauksi rastimalla *Oletus* -rastikentän, joka sijaitsee *Poista* -napin vieressä sen oikealla puolella. Kun haku on määritetty oletukseksi, se tehdään aina automaattisesti kun puhelusivu avataan. Hakuvalikossa on oletushaun perässä teksti *oletus* (Kuvio 12).

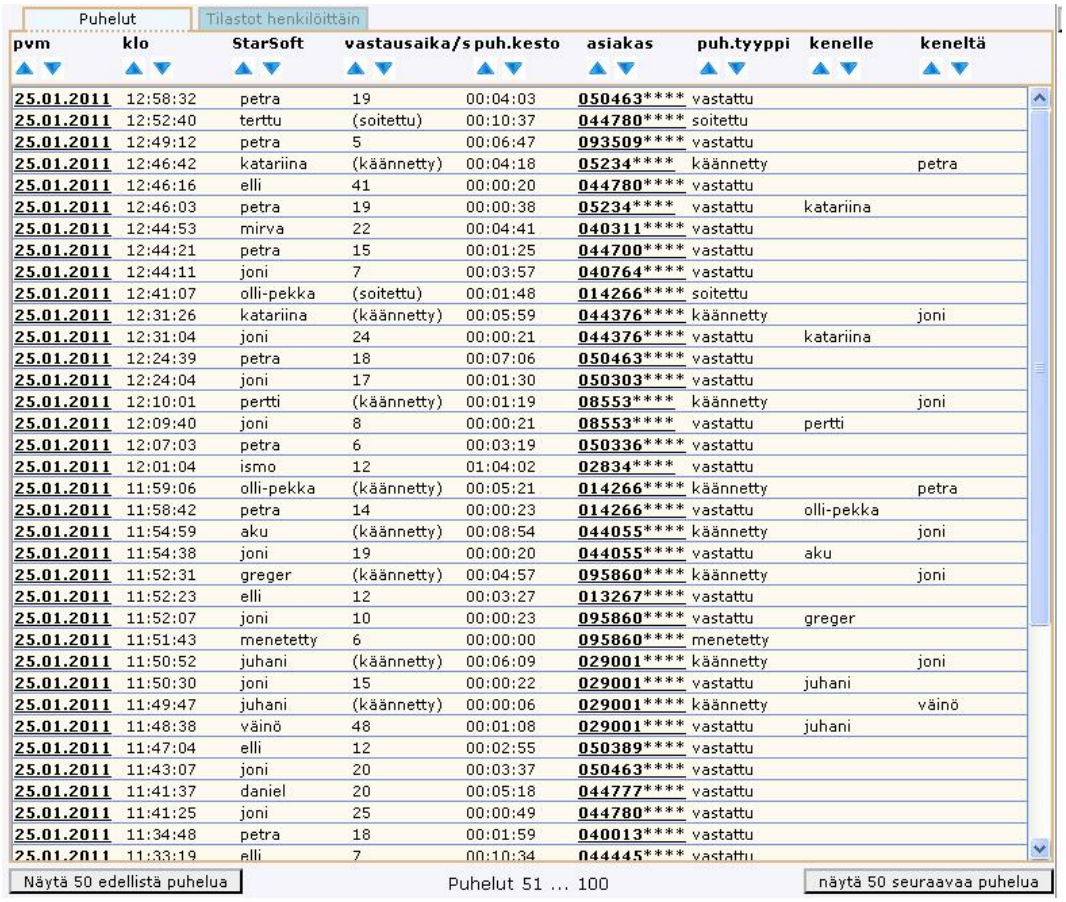


Kuvio 12. Haun tallennus ja valmiiden hakujen lista.

5.4 Hakutulokset – puhelutaulu

Hakulauseen mukaan haetut puhelut tulostuvat omaan tauluunsa(Kuvio 13). Puhelut ovat oletuksena päivämäärän ja sen jälkeen kellonajan mukaisessa järjestuksessä. Jos

taulun järjestystä halutaan muuttaa, se voidaan tehdä napsauttamalla sarakkeiden yllä olevia nuolia, jolloin taulun tiedot järjestyvät kyseisen sarakkeen tietojen mukaiseen järjestykseen nousevasti tai laskevasti. Taulun järjestystä muuttamalla löytyy helposti esim. pisin puhelu tai lyhin vastausaika jne.



Puhelut | Tilastot henkilöittäin

pvm	klo	StarSoft	vastausaika/s	puh.kesto	asiakas	puh.tyyppi	kenelle	keneltä
25.01.2011	12:58:32	petra	19	00:04:03	050463****	vastattu		
25.01.2011	12:52:40	terttu	(soitettu)	00:10:37	044780****	soitettu		
25.01.2011	12:49:12	petra	5	00:06:47	093509****	vastattu		
25.01.2011	12:46:42	katariina	(käännetty)	00:04:18	05234****	käännetty		petra
25.01.2011	12:46:16	elli	41	00:00:20	044780****	vastattu		
25.01.2011	12:46:03	petra	19	00:00:38	05234****	vastattu	katariina	
25.01.2011	12:44:53	mirva	22	00:04:41	040311****	vastattu		
25.01.2011	12:44:21	petra	15	00:01:25	044700****	vastattu		
25.01.2011	12:44:11	joni	7	00:03:57	040764****	vastattu		
25.01.2011	12:41:07	olli-pekka	(soitettu)	00:01:48	014266****	soitettu		
25.01.2011	12:31:26	katariina	(käännetty)	00:05:59	044376****	käännetty		joni
25.01.2011	12:31:04	joni	24	00:00:21	044376****	vastattu	katariina	
25.01.2011	12:24:39	petra	18	00:07:06	050463****	vastattu		
25.01.2011	12:24:04	joni	17	00:01:30	050303****	vastattu		
25.01.2011	12:10:01	pertti	(käännetty)	00:01:19	08553****	käännetty		joni
25.01.2011	12:09:40	joni	8	00:00:21	08553****	vastattu	pertti	
25.01.2011	12:07:03	petra	6	00:03:19	050336****	vastattu		
25.01.2011	12:01:04	ismo	12	01:04:02	02834****	vastattu		
25.01.2011	11:59:06	olli-pekka	(käännetty)	00:05:21	014266****	käännetty		petra
25.01.2011	11:58:42	petra	14	00:00:23	014266****	vastattu	olli-pekka	
25.01.2011	11:54:59	aku	(käännetty)	00:08:54	044055****	käännetty		joni
25.01.2011	11:54:38	joni	19	00:00:20	044055****	vastattu	aku	
25.01.2011	11:52:31	greger	(käännetty)	00:04:57	095860****	käännetty		joni
25.01.2011	11:52:23	elli	12	00:03:27	013267****	vastattu		
25.01.2011	11:52:07	joni	10	00:00:23	095860****	vastattu	greger	
25.01.2011	11:51:43	menetetty	6	00:00:00	095860****	menetetty		
25.01.2011	11:50:52	juhani	(käännetty)	00:06:09	029001****	käännetty		joni
25.01.2011	11:50:30	joni	15	00:00:22	029001****	vastattu	juhani	
25.01.2011	11:49:47	juhani	(käännetty)	00:00:06	029001****	käännetty		väinö
25.01.2011	11:48:38	väinö	48	00:01:08	029001****	vastattu	juhani	
25.01.2011	11:47:04	elli	12	00:02:55	050389****	vastattu		
25.01.2011	11:43:07	joni	20	00:03:37	050463****	vastattu		
25.01.2011	11:41:37	daniel	20	00:05:18	044777****	vastattu		
25.01.2011	11:41:25	joni	25	00:00:49	044780****	vastattu		
25.01.2011	11:34:48	petra	18	00:01:59	040013****	vastattu		
25.01.2011	11:33:19	elli	7	00:10:34	044445****	vastattu		

Näytä 50 edellistä puhelua | Puhelut 51 ... 100 | näytä 50 seuraavaa puhelua

Kuvio 13. Puhelutaulu.

Päivämäärät ja puhelinnumerot näkyvät puhelutaulussa alleviivattuina ja lihavoituina. (Tässä esitetyistä puhelinnumeroista on salattu viimeiset numerot, todellisuudessa ne näkyvät kokonaisuudessaan. Esimerkeissä käytetyt henkilöiden etunimet ovat myös tekaistuja.) Numeroita napsauttamalla siirtyy kyseinen tieto hakulauseeseen. Näin saa nopeasti haettua kaikki tietyn asiakkaan puhelut, tai jonkin tietyn päivän puhelut. Puhelutaulussa näkyy käännettyjen puheluiden osalta *keneltä* -sarakkeessa tieto, keneltä kyseinen puhelu on siirretty. Vastatuille puheluille, jotka on käännetty eteenpäin, tallentuu *kenelle* -sarakkeeseen tieto siitä, kenelle tämä puhelu on siirretty.

Puhelutauluun tulostuu haluttu määrä puheluita sivua kohden. Jos hakutuloksia on enemmän, kuin sivuille mahtuu, voidaan seuraaville ja edellisille sivuille siirtyä puhelutaulun alapuolella olevilla painikkeilla.

5.5 Hakutulokset – tilastot henkilöittäin

Puhelutaulun ylälaidassa näkyy kaksi välilehteä; puhelut ja tilastot henkilöittäin. Välilehdet näkyvät vain käyttäjille, joilla on tilasto-oikeudet - muut käyttäjät näkevät pelkän puhelutaulun ilman välilehtiä. Välilehtiä napsauttamalla voi valita tarkasteltavan näkymän; puhelutaulu puhelukohtaisine yksityiskohtineen tai tilasto puhelumääristä henkilöittäin.

Tilastot henkilöittäin -näkymä sisältää taulun, jossa on oma rivi kullekin käyttäjälle, jolle käytetty haku on löytänyt puheluita(Kuvio 14). Taulun sarakkeina ovat henkilön nimi ja vastattujen, henkilölle käännettyjen, henkilön toisille kääntämien, sekä soitettujen puheluiden lukumäärät. Taulun tiedot ovat alkutilanteessa nimen mukaisessa aakkosjärjestyksessä, mutta sarakkeiden yllä olevien nuolien avulla voi järjestää tiedot nousevaan tai laskevaan järjestykseen minkä tahansa sarakkeen mukaan, samoin kuin puhelutaulussa. Tällä tavalla on mahdollista selvittää nopeasti esim. kenelle työntekijälle on käännetty eniten puheluita jonain tiettynä aikana. Jos hakulausetta muutetaan ja haku tehdään uudestaan, päivittyy myös henkilöittäin koostettu tilasto välittömästi tietojen säilyttäessä valitun järjestyksen.

Puhelut | Tilastot henkilöittäin

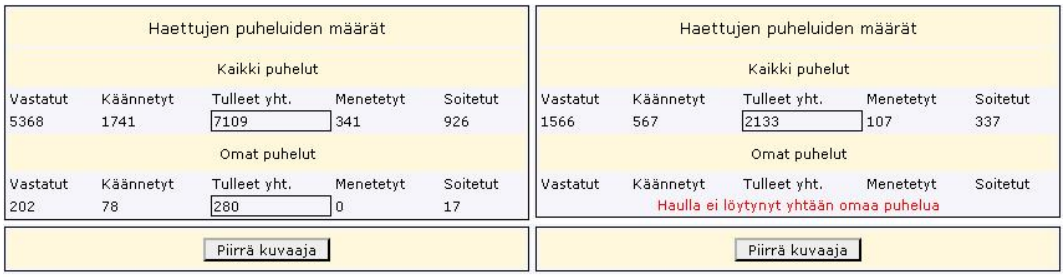
nimi	vastatut	käännetyt <-	käännetyt ->	soitetut
petra	1685	147	146	41
joni	1447	299	151	33
jarkko	1400	187	123	5
mirva	1083	37	230	134
terttu	1017	308	80	91
väinö	941	28	139	23
jyrki	934	36	92	38
daniel	879	87	93	69
simo	830	43	168	15
elli	799	122	90	61
juhani	556	280	49	41
olli-pekka	526	151	53	13
ismo	411	7	123	56
lennart	274	130	3	24
niilo	251	9	71	24
katariina	123	246	8	204
pertti	103	231	13	211
oliver	42	5	0	329
greger	33	193	3	70
laura	25	114	1	65
kalervo	20	50	0	132
heidi	18	42	1	67
reino	16	9	0	143
elisa	13	320	1	41
mikko	11	131	0	9
ville	6	12	3	11
antti	4	6	0	7
aku	3	151	0	13
pekka	1	0	0	0
tuula	0	63	5	4
hannu	0	4	0	0
pirkko	0	2	0	3

Kuvio 14. Tilastot puhelumääristä henkilöittäin vastattujen puheluiden mukaan laskevasti järjestettynä.

5.6 Hakutulokset – yhteenveto puheluiden lukumääristä

Puheluhaku tuottaa puhelutaulun ja henkilöittäin koostetun lukumäärätilaston lisäksi vielä koosteen haun palauttamasta puheluiden kokonaislukumäärästä. Nämä tiedot tulostetaan omaan osioonsa, joka sisältää tiedot puheluiden määrästä jaoteltuna vastattuihin, käännettyihin, menetettyihin ja soitettuihin puheluihin. Lisäksi näytetään kaikkien tulleiden puheluiden, eli vastattujen ja käännettyjen yhteenlaskettu määrä(Kuvio 15).

Edellämainitut tiedot tulostetaan sekä kaikkien, että käyttäjän omien puheluiden osalta. Jos käytetyllä haulla ei löytynyt yhtään käyttäjän omaa puhelua, ilmoitetaan siitä ”Haulla ei löytynyt yhtään omaa puhelua” –tekstillä. Puhelumäärien yhteenveto-osion alapuolelle ilmestyy hakutulosten myötä lisäksi *Piirrä kuvaaja* -painike, jota napsauttamalla päästään tarkastelemaan haetuista puheluista muodostettavaa graafia.

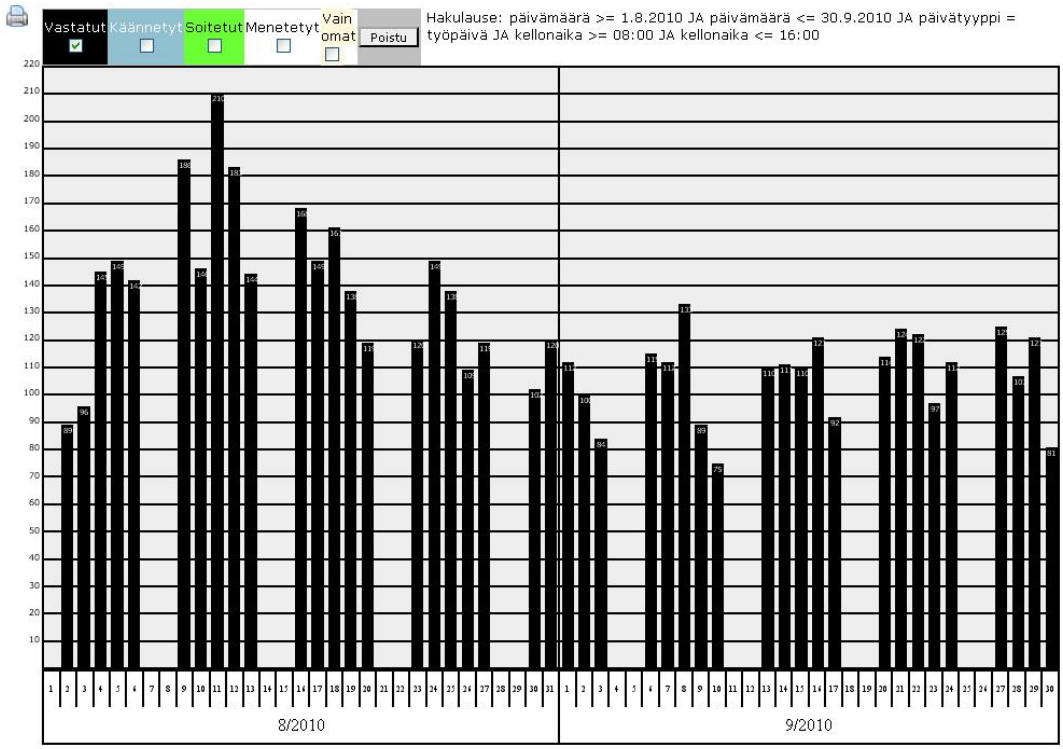


Kuvio 15. Kooste puheluiden kokonaislukumäärästä. Oikeanpuoleinen näyttö tulostuu jos haku ei palauta yhtään käyttäjän omaa puhelua.

5.7 Hakutulokset – graafi

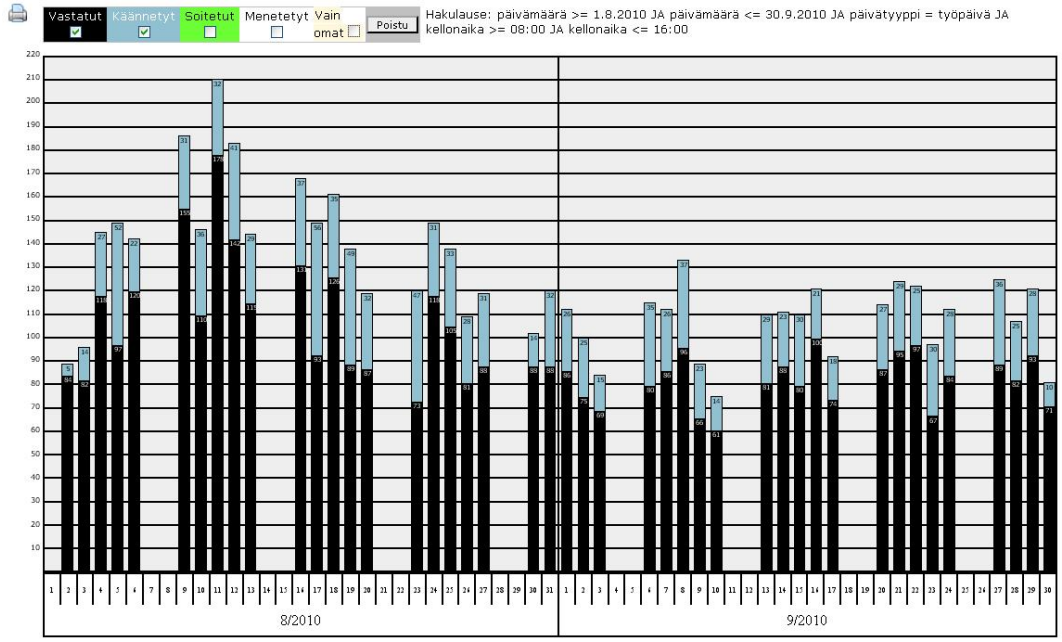
Puhelumäärien seuranta pitemmältä aikaväliltä on havainnollisinta toteuttaa graafisena esityksenä, eli pylväskuvaajana, joka sisältää puhelumäärää kuvaavan pylvään jokaiselle päivälle tehdyn puheluhaun määrittämällä aikavälillä. Graafin avulla voidaan todeta mm. vuosittain toistuvat tiettyjen ajankohtien määrälliset huiput, jotka käytännössä tarkoittavat sitä, että tukipalvelussa on näinä ajankohtina oltava mahdollisimman paljon työntekijöitä. Tällä tavalla toimiva graafi helpottaa mm. tukihenkilöstön lomien suunnittelua.

Graafin pohjana on taulukko, jossa on jokaiselle tulostuvan aikavälin kuukaudelle oma sarakkeensa ja rivejä niin monta, kuin graafin korkein pylväs tarvitsee(Kuvio 16). Yksi riviväli tarkoittaa kymmentä puhelua. Aikaväli, jolta graafi muodostetaan alkaa ensimmäiseltä kuukaudelta, jolta on tehdyllä haulla löydetty puheluita ja loppuu viimeiseen kuukauteen, jolta löytyy puheluita. Väliin mahdollisesti jäävät tyhjät(puheluttomat) kuukaudet tulostuvat nekin omina sarakkeinaan.



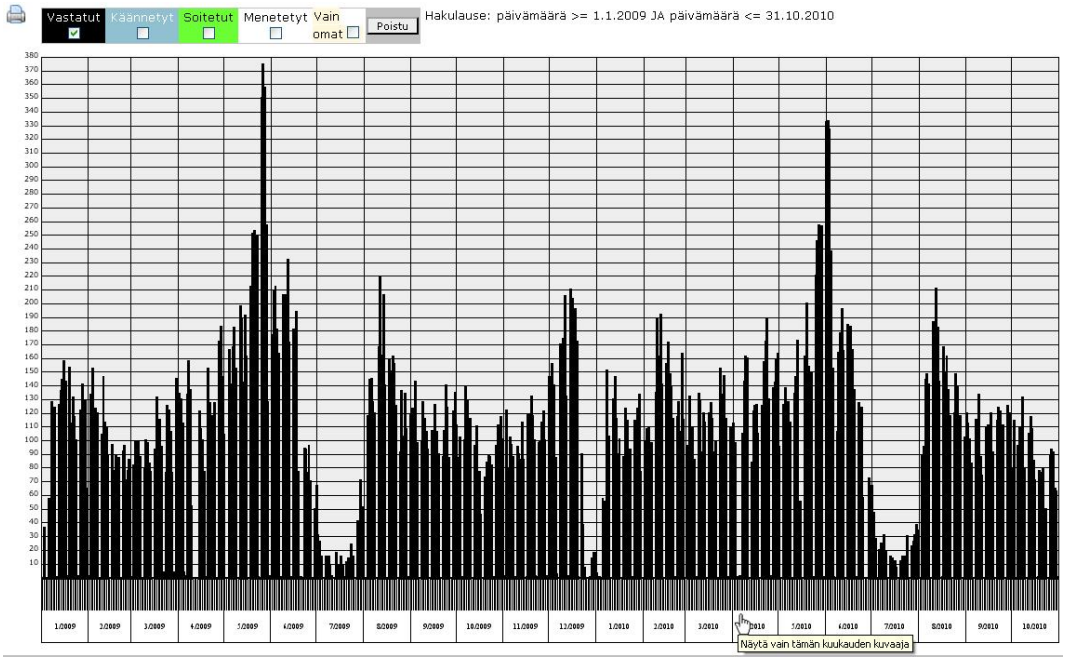
Kuvio 16. Graafi kahden kuukauden vastatuista puheluista.

Graafin pylväät muodostuvat oletuksena pelkistä vastatuista puheluista, mutta ruudun yläreunan rastikenttien avulla voi valita tarkemmin minkä tyyppisistä puheluista pylväät halutaan muodostaa(Kuvio 17). Jos valitaan vastatut ja käännetyt, niin käännettyjen määrä vähennetään vastatuista, koska käännetty puhelu löytyy tietokannasta myös vastattuna puheluna. Jos käyttäjällä on tehdyn haun palauttamia omia puheluita, voidaan rastia *Vain omat* –rastikenttä, jolloin pylväät koostuvat vain käyttäjän omista puheluista.

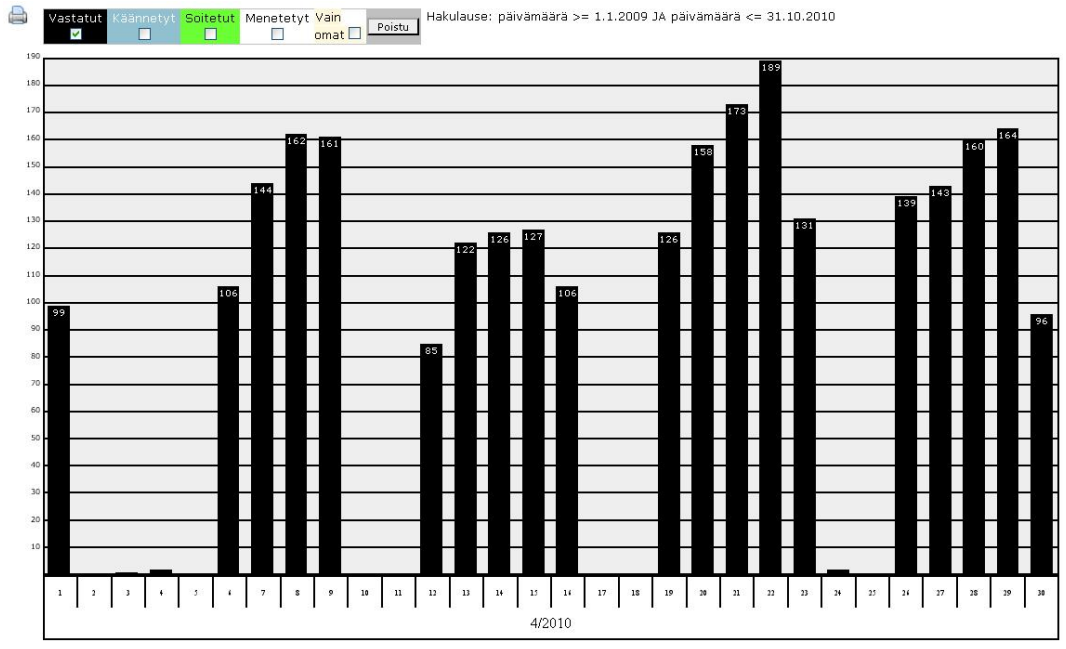


Kuvio 17. Graafi kahden kuukauden vastatuista ja käännetyistä puheluista.

Jos graafi tulostetaan pitkältä ajalta, esim. yhden tai useamman vuoden ajalta, saadaan nopeasti käsitys tukipalvelun työmäärän jakautumisesta kiireaikoihin ja hiljaisempiin kausiin eri vuodenaikoina (Kuvio 18). Tällöin on kuitenkin hankala hahmottaa tarkasti yhden tietyn kuukauden tilannetta puhelumäärien suhteen. Jos halutaan tarkastella tiettyä kuukautta lähemmin, ei graafia tarvitse sulkea ja hakua muuttaa koskemaan vain kyseistä kuukautta, vaan voidaan napsauttaa graafin alaosan kuukausi/vuosi –tiedon ilmaisevaa kenttää, jolloin ruudulle saadaan vain tämän kuukauden kuvaaja (Kuvio 19). Alkuperäiseen näkymään voidaan palata napsauttamalla uudestaan kuukausi/vuosi –kenttää.



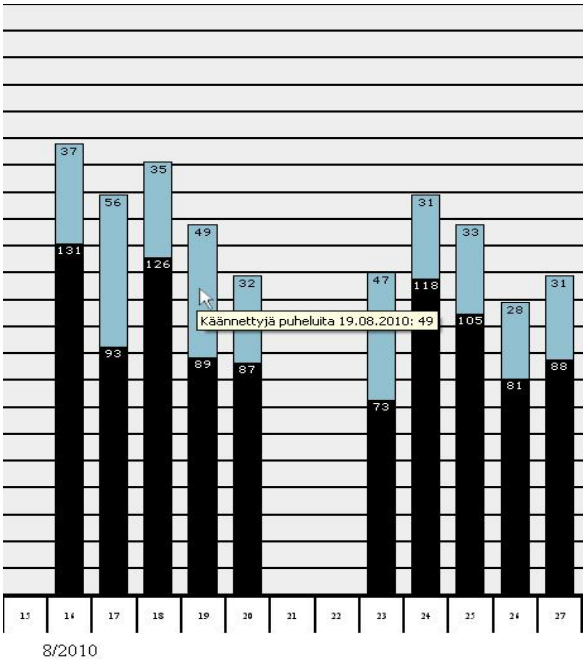
Kuvio 18. Graafi vuoden 2009 tammikuusta vuoden 2010 lokakuuhun.



Kuvio 19. Pitkän aikavälin kuvaajasta poimittu yhden kuukauden näkymä.

Jos graafi piirretään enintään kahdelta kuukaudelta, tulostuu jokaiseen pylvääseen luku, joka ilmaisee kyseisen pylvään edustaman puhelujen lukumäärän. Kun kuukausia on useampia ja lukemat eivät ole nähtävissä suoraan pylväistä, voidaan viedä hiiren osoitin

pylvään päälle, jolloin kyseinen päivämäärä ja pylvään puhelumäärä saadaan näkyviin tooltip-tekstinä (Kuvio 20).



Kuvio 20. Päivämäärä ja puhelumäärä tooltip-tekstinä.

Graafi-ikkunan yläosaan tulostuu hakulause, jonka pohjalta kuvaaja on tuotettu. Tämä havainnollistaa graafin sisältöä, jos graafi tulostetaan paperille ja sen lähtötiedot eivät enää ole nähtävissä itse tietojärjestelmästä. Graafi voidaan tulostaa napsauttamalla vasemmassa yläreunassa olevaa tulostimen kuvaa, jolloin tulostuva alue rajataan pelkkään graafin sisältävään div-elementtiin, eikä selaimen painikkeita tai Tukijuhdan välilehtiä ym. tule mukaan tulosteeseen.

6 Puhelutilastosovelluksen toteutus

Puhelutilastosovelluksen käyttöliittymän toiminta, hakulause ja graafi, on toteutettu Javascript-ohjelmointikielellä. Palvelinpuolen skriptit, joilla tehdään tietokannasta haut ja tallennukset, sekä käyttäjätietojen käsittely, on toteutettu PHP:lla. Tietokannan käsittelyyn liittyvät toiminnot toteutetaan asynkronisesti Ajax-pyyntöinä, jolloin sivun uudelleenlatausta ei tarvita missään vaiheessa.

6.1 Hakulauseen toiminta – palan lisäys ja poisto

Hakulauseen toiminnallisuus on toteutettu Javascriptilla ja sen koodi sijaitsee omassa hakulause.js –tiedostossaan. Toiminta toteutetaan Hakulause-oliona, jonka ominaisuuksiksi määritetään mm. taulukkomuuttujat textarray ja sql (Taulukko 5).

Taulukko 5. Hakulause-olion ominaisuudet.

```
var Hakulause = {
   selectedIndex: 0, //kursorin paikan määräämä indeksi
   textarray:[], //taulukko, johon hakulause tallentuu tekstinä
   textstring: '', //edelliset tiedot merkkijonomuodossa käyttöliittymään tulostusta varten
   sql:[], //taulukko, johon hakulause tallentuu sql-muodossa
   sqlstring: '', //edellisen merkkijonomuoto, jolla tehdään lopullinen haku tietokannasta
   lastsqlstring: '', //viimeisen tehdyn haun sql-lause
   offset: 0, //sivutusta varten
   order:'', //hakutulosten järjestystä varten
   savearray:[], //taulukko haun tallennusta varten
   vertailuoperaattorit:['+','-','=','<>','<=','>=','Ã·','Ã—'], // aputaulukko, vertailuoperaattorien tunnistukseen
   puheluotsikot:[], //puhelutaulun otsikot config.php:sta
   hakutyyppi:'puhelutaulu', //hakutyyppi: puhelutaulu/henk_tilasto
   hakuimport: false, //tallennetun haun tuonti: true/false
.
.
.
}
```

Textarray-taulukkoon lisätään addPart()-metodin (Taulukko 6) avulla kukin hakulauseen pala (hakuaihe, operaattori, päivämäärä jne.) käyttäjän muodostaessa hakulausetta. AddPart()-metodi saa parametreinaan metodia kutsuneen Click-tapahtuman kohde-elementin, sekä sen arvon, eli varsinaisen hakuun lisättävän tiedon. AddPart()-metodi kutsuu addToSqlArray()-metodia, joka lisää vastaavan hakulauseen palan sql-taulukkoon, mutta muuntaa ko. määreen SQL-syntaksin mukaiseksi, sekä mm. aikamuuttujat aikaleimamuotoon ennen lisäystä. Myös addToSqlArray()-metodi saa parametreikseen html-elementin lisättävän arvon lisäksi. On tiedettävä, mikä elementti on kyseessä, ts. minkä tyyppistä tietoa ollaan lisäämässä hakuun, jotta tiedetään miten

muunto SQL-muotoon tehdään. Sql-taulukkoon muodostuu lopullinen hakulause, joka lopulta lähetetään palvelimelle tietokantaan tehtävää kyselyä varten.

Taulukko 6. AddPart()-metodi ja esimerkki addToSqlArray()-metodin toiminnasta.

```
addPart: function(elm,value)
//elementti ja arvo parametreina, paitsi tallennetun haun tuonnissa elementin id ja arvo
  {
    if(elm)
       {
         //textarray-taulukkoon lisätään parametrina saatu arvo
          this.textarray.splice(this.selectedIndex,0,value);
          this.textarray[this.selectedIndex]=this.textarray[this.selectedIndex].replace('&gt;','>');
          this.textarray[this.selectedIndex]=this.textarray[this.selectedIndex].replace('&lt;','<');

          this.addToSqlArray(elm,value);
          this.addToSaveArray(elm,value);
          this.selectIndex($('eteen'));
       }
       else return;
  }

addToSqlArray: function(elm,value)
  {
  var add;
  var id = elm.id || elm; //(tallennettu haku ei tuo elementtiä vaan pelkän id-tiedon, jolloin elm = id)

//switch-rakenteessa tehdään elementistä riippuen muunto sql-muotoon ja lisätään ko. arvo sql-
taulukkoon
           switch(id)
           {
           .
           .
           .
              case 'equals':
                 add = ' =';
                 break;
             case 'grthan':
                 add = ' >=';
                 break;
             case 'lsthan':
                 add = ' <=';
                 break;
            .
            .
            .
            }
 }
```

Hakulauseen palan poistoa varten on olemassa removePart()- ja removeFromSqlArray()-metodit, jotka poistavat tiedot textArray- ja sql-taulukoista hakulauseen kursorin sijainnin määräämän indeksin kohdalta. Hakulauseen kursorin siirtyminen hakulauseen palan lisäyksen tai poiston, tai nuolipainikkeiden napsautuksen seurauksena, kutsuu

Hakulause-olion selectedIndexChanged()-metodia, joka asettaa olion selectedIndex-ominaisuudeksi nykyistä kursorin paikkaa vastaavan arvon.

Aina kursorin siirtyessä ajetaan buttonsHideShow()-funktio, joka määrää, riippuen kursorin paikasta hakulauseeseen nähden, mitkä painikkeet ovat näkyvillä hakulauseen käyttöliittymässä ja tarvittaessa kutsuu tekstikentän tai valikon focus()-metodia ohjatakseen kursorin siihen, tai vastaavasti estää tekstikentän käytön, jos kyseisessä hakulauseen osassa ei tule voida syöttää tekstiä.

6.2 Hakulauseen toiminta – hakujen tallennuksen toiminnallisuus

AddToSQLArray()- ja AddPart()-metodien lisäksi kutsutaan jokaisen hakulauseen palan lisäyksen yhteydessä addToSaveArray()-metodia, joka lisää erilliseen savearray-taulukkoon kyseisen html-elementin id-arvon, ”#”-merkin erotinmerkiksi, sekä elementin arvon. Käytännössä tämä tarkoittaa sitä, että jos käyttäjä aloittaessaan hakulauseen rakentamista napsauttaa päivämäärä-painiketta, lisätään savearray-taulukkoon seuraava tieto: ”pvm # päivämäärä”. ”Pvm” on painikkeen id-arvo, ”#” on erotinmerkki ja ”päivämäärä” on painikkeessa lukeva teksti, joka painikkeen napsautuksen myötä lisätään hakulauseen sisällöksi. Jos lopulliseksi hauksi muodostuisi ”päivämäärä >= 1.6.2010” olisi sitä vastaava savearray-taulukon sisältö seuraavanlainen (Taulukko 7):

Taulukko 7. Savearray-taulukon sisältö esimerkkitapauksessa.

savearray[0] = ”pvm # päivämäärä”
savearray[1] = ”grthan # >=”
savearray[2] = ”kalenteri # 1.6.2010”

Jos muodostettu hakulause halutaan tallentaa, käyttäjä painaa *Tallenna haku*-painiketta, jolloin kutsutaan Hakulause.saveSearch()-metodia, joka hoitaa tallennuksen. Savearray-taulukon sisältö muutetaan merkkijonoksi tallennusta varten (Taulukko 8).

Taulukko 8. Savearray-taulukon sisältö merkkijonoksi muunnettuna.

”pvm # päivämäärä, grthan # >=, kalenteri # 1.6.2010”

Lisäksi käyttäjältä pyydetään tässä vaiheessa hakulauseen nimi ja tarkistetaan onko tallennettavan hakulauseen tarkoitus näkyä kaikille käyttäjille.

SaveSearch-metodi lähettää Ajax-pyyntönä tiedot palvelimelle, jossa tallennuksen suorittava puheluhaut.php –tiedosto lisää tietokannan puheluhaut-tauluun tallennettua hakua koskevat tiedot:

- ID: Auto increment –tyyppinen integer-arvo, tallennetun haun id-numero, joka on aina yhden suurempi kuin edellinen tallennettu haku.
- Savestring: merkkijono, joka sisältää tallennettavan haun
- Owner: Haun tallentaneen käyttäjän User-ID –arvo, joka saadaan Tukijuhdan käyttäjänhallinnasta. Tämän perusteella tiedetään haun omistaja, ts. kenelle haku näkyy.
- Name: Haulle annettu nimi
- Public: “1” tai “0”. Tämä arvo määrää näkyykö haku kaikille käyttäjille.

Valmiit haut haetaan tietokannasta aina käyttäjän napsauttaessa valmiiden hakujen pudotusvalikkoa. Tällöin kutsutaan Hakulause-olion selectFill()-metodia (Taulukko 9), joka poistaa kyseisen select-elementin kaikki olemassaolevat lapsielementit, eli valikon sisällön, lukuunottamatta valikon ”-Valmiit haut-” -otsikkoa. Tämän jälkeen lisätään elementin uusiksi lapsiksi kaikki tietokannasta haetut tallennetut haut jotka saavat näkyä senhetkiselle käyttäjälle. Kunkin select-elementin lapsen, eli option-elementin value-attribuutiksi asetetaan tietokannasta haettu haku sen em. merkkijonomuodossa. Tämän jälkeen lisätää option-elementin lapsielementiksi tekstinode, jonka arvoksi annetaan tietokannasta haetun haun nimi. Tämän jälkeen pudotusvalikossa näkyvät tallennettujen hakujen nimet.

Taulukko 9. selectFill()-metodin toiminta.

```
selectFill:function(str) //parametrina hakulause-merkkijono
  {
//poistetaan select-elementin lapsielementtejä kunnes
//jäljellä on vain yksi lapsi, jonka id on ”first”. Tämä on select-elementin “otsikko” (-Valmiit Haut-)
    while($('hakuselect').childNodes.length > 1)
      {
        if($('hakuselect').firstChild.id != 'first')
          {
            $('hakuselect').removeChild($('hakuselect').firstChild);
          }
          else
          {
            $('hakuselect').removeChild($('hakuselect').lastChild);
          }
      }
    if(str != '')
    {
//muutetaan str-merkkijono taulukkomuotoon, taulukossa on tällöin tietokannasta haetut valmiit haut
    var array = str.split(';');
    var elm,text;
    for(var i=0; i<array.length; i++)
      {
//luodaan jokaista taulukon indeksiä kohden option-elementti, jonka value-attribuutti sisältää haun id-
//arvon ja itse hakulauseen merkkijonona
        elm = document.createElement('option')
        elm.setAttribute('value',array[i].split('/')[0] + ';' + array[i].split('/')[2]);

//luodaan kunkin elementin lapseksi tekstinode, jonka arvona on haun nimi sekä teksti ”oletus”
//oletushaun tapauksessa
        text = document.createTextNode(array[i].split('/')[1] + ($('userOletushaku').innerHTML ==
array[i].split('/')[0] ? ' (oletus)' : ''));
        elm.appendChild(text);
        $('hakuselect').appendChild(elm);
      }
    }
  }
```

Tallennetun haun saa ladattua valitsemalla sen pudotusvalikosta. Tässä vaiheessa kutsutaan Hakulauseen importSearch()-metodia (Taulukko 10), joka muuttaa valitun haun merkkijonomuodosta takaisin taulukkomuotoon, jotta siitä voidaan tuottaa uusi hakulause yksi pala kerrallaan. Hakulausetta ei voida tuoda merkkijonona, koska tällöin siitä ei mm. voisi poistaa yksittäisiä paloja, kuten käyttöliittymän kautta luodulle hakulauseelle on mahdollista tehdä. Kun hakulause on taulukkomuodossa, käydään sen jokainen indeksi läpi toistorakenteessa. Jokaisen indeksin kohdalla kutsutaan addPart()-metodia (ks. 6.1), jolloin hakulause rakentuu käytännössä samalla tavalla, kuin se rakennettiin alunperin.

Taulukko 10. importSearch()-metodin toiminta.

```
importSearch:function()
  {
    this.hakuimport = true;
//empty() –metodi nollaa Hakulause-olion kaikki tiedot ennen tallennetun haun tuomista
    this.empty();
//poimitaan select-elementin valitun arvon hakulause-osa merkkijonona string-muuttujaan
    var string = $('hakuselect').value.split(';')[1];
//string-merkkijono paloitellaan taulukoksi, jolloin se on samassa muodossa kuin alkuperäinen saveArray
    var array = string.split(',');
//käydään taulukon jokainen indeksi, eli hakulauseen pala läpi toistorakenteessa
    for(var i=0; i<array.length; i++)
    {
      if(i == array.length -1)
        {
          this.hakuimport = false;
        }
//kutsutaan addPart()-metodia ja lähetetään parametreina kunkin palan html-elementin id sekä arvo, sen
//jälkeen kun ne on erotettu toisistaan tallennusvaiheessa annetun #-erotinmerkin avulla
      this.addPart((array[i].split('#'))[0], (array[i].split('#'))[1]);
    }
//lopuksi toteutetaan haku simuloimalla hakunapin napsautusta kutsumalla haePuhelut()-metodia
    this.haePuhelut('hakunappi');
  }
```

Tallennetun haun poisto tapahtuu napsauttamalla tallennetun haun valinnan jälkeen *Poista*-painiketta, jolloin poistetaan haku tietokannasta valitun haun id-arvon perusteella. Poiston hoitaa Hakulause-olion deleteSearch() –metodi, jonka toimintaperiaate on sama kuin saveSearch() –metodilla. Varsinaisen tietokannasta poistamisen tekee puheluhaut.php –tiedostossa sijaitseva poistofunktio.

6.3 Hakulauseen toiminta – haun toteutus Ajax-pyyntönä

Hakulause-olion sqlstring-ominaisuus on Hakulause.sql -taulukkoon tallentuva hakulause merkkijonomuodossa. Tätä merkkijonoa käytetään tietokannasta haussa lähettämällä se palvelimelle Ajax-pyyntönä muiden tarvittavien tietojen ohella. Hakulause-oliolla on lisäksi lastsqlstring-ominaisuus, jota käytetään sqlstring-merkkijonon sijaan silloin, kun halutaan käyttää viimeksi tehtyä hakua, kun muutetaan esim. puhelutaulun järjestystä. Tällaisessa tapauksessa käyttäjä on saattanut jo rakentaa uuden hakulauseen, mutta haluaa ennen sen toteuttamista(eli *Hae*–napin napsautusta) tarkastella vielä edellisen haun tuottamaa puhelutaulua muuttamalla sen järjestystä.

Hakulauseen valmistuttua käyttäjä voi toteuttaa tekemänsä haun napsauttamalla Hae-nappia, tai valitsemalla pudotusvalikosta tallennetun haun. Molemmissa tapauksissa kutsutaan Hakulause-olion haePuhelut()-metodia, jolle annetaan parametriksi teksti

”hakunappi”. Puheluhaku, henkilökohtaisen tilaston haku ja puhelutaulun järjestyksen muutos tai sivunvaihto tehdään aina haePuhelut()-metodin kautta, jolloin metodin saama parametri kertoo, minkä tyyppinen haku on kyseessä. Mahdolliset parametrit ovat:

- ”hakunappi” : uusi haku, joka käyttää Hakulause.sqlstring –merkkijonoa haun muodostamiseen. Hakulause.offset –arvo on 0(nolla). Hakulause.lastsqlstring –arvoksi asetetaan Hakulause.sqlstring –arvo, eli nyt tehtävä haku.
- ”nextpage” : Tämä tarkoittaa, että käyttäjä on napsauttanut *Näytä n seuraavaa puhelua* –painiketta siirtyäkseen puhelutaulun näyttämien hakutulosten seuraavalle sivulle. Haku tehdään nyt käyttäen Hakulause.lastsqlstring –arvoa, joka on siis viimeksi tehty haku. Lisäksi Hakulause.offset –arvoa kasvatetaan yhdellä.
- ”prevpage” : Käyttäjän napsautettua *Näytä n edellistä puhelua* –painiketta, haetaan edellisen sivun puhelut käyttämällä hakuun Hakulause.lastsqlstring –arvoa ja pienentämällä Hakulause.offset –arvoa yhdellä, paitsi jos sen arvo on jo 0(nolla), mikä merkitsee, että ollaan jo ensimmäisellä sivulla.
- ”personalstat_tab” : Puhelutaulun yllä olevan *Tilastot henkilöittäin* –välilehden napsautus asettaa Hakulause.hakutyyppi –ominaisuuden arvoksi ”henk_tilasto”, joka tarkoittaa, että haetaan puhelutaulun sijaan puhelumäärien tilasto henkilöittäin(ks. 3.3.6) Haussa käytetään Hakulause.lastsqlstring –arvoa.
- ”calltable_tab” : Puhelutaulun yllä olevan *Puhelut*-välilehden napsautus tekee haun, jolla tuotetaan puhelutaulu samoin kuin Hae-nappia napsautettaessa, mutta käyttää hakuun Hakulause.lastsqlstring –arvoa Hakulause.sqlstring –arvon sijaan.
- ”pvm sortasc” : Puhelutaulun järjestystä muutettaessa napsautetaan esim. *pvm*-otsikon alla olevaa ylöspäin osoittavaa nuolta. Tällöin parametriksi tulee otsikon nimi ja nousevan tai laskevan järjestyksen mukaan ”sortasc” tai ”sortdesc”. Tästä seuraa, että Hakulause.order –ominaisuus, joka määrää puhelutaulun järjestyksen, saa arvokseen ”ORDER BY pvm ASC”. Vastaavasti esim. napsautettaessa *vastausaika/s* –otsikon alla olevaa alaspäin osoittavaa nuolta saataisiin parametriksi ”vstaika sortdesc”, jolloin Hakulause.order –arvoksi tulisi ” ORDER BY vstaika DESC”. Hakuun käytetään järjestystä muuttaessa Hakulause.lastsqlstring –arvoa.

Ajax-pyyntö lähetetään haepuhelut.php –tiedostolle varustettuna seuraavilla tiedoilla:

- sql : Käyttäjän rakentama hakulause; Hakulause.sqlstring tai Hakulause.lastsqlstring riippuen haePuhelut()-metodin saamista parametreista.
- limit : Käyttäjän määrittämä(tai oletusmäärä, eli 50) puhelutauluun yhdelle sivulle tulostuvien puhelujen enimmäismäärä(ks. 3.3.3)
- haekaikki : Käyttäjän valinta; haetaanko omat vai kaikki puhelut(ks. 3.3.3). Arvo on 1(kaikki) tai 0(omat).
- alanro : Käyttäjän oma alanumero, jonka mukaan haku tehdään, jos on valittu vain omien puheluiden haku.
- order : Puhelutaulun järjestys, Hakulause.order –arvo.
- offset : sivutuksessa käytettävä muuttuja, Hakulause.offset –arvo.
- hakutyyppi : Puhelutaulu tai tilasto henkilöittäin, Hakulause.hakutyyppi –arvo.

6.4 Haku tietokannasta

Hakulause lähetetään asynkronisesti palvelimelle Ajax-tekniikkaa hyödyntämällä, jolloin sivua ei tarvitse ladata uudestaan ja mm. tehty hakulause säilyy muuttumattomana. Pyyntö tehdään Post-metodia käyttäen. Varsinaisen tietokannasta haun suorittaa haepuhelut.php –tiedosto (Taulukko 11).

Haepuhelut.php –tiedoston suorittaminen alkaa siitä, että luetaan $_POST-taulukosta Ajax-pyyntönä lähetetyt, haussa tarvittavat tiedot omiin php-muuttujiinsa. Näitä ovat mm:

- $sqlstr : muodostettu hakulause MySQL-syntaksin mukaisesti
- $limit: sivulla näytettävien puheluiden maksimimäärä
- $offset: sivutuksesta riippuva muuttuja, joka määrää hakutulosten ”aloituspaikan”. Esim. OFFSET 50 –määre jättää pois 50 ensimmäistä hakutulosta
- $sivutus: $limit ja $offset –muuttujien määräämä sivutusmuuttuja

Taulukko 11. Haepuhelut.php-tiedoston alkuarvot.

```
if ((($_POST["sql"] != ""))&&(isset($_POST["alanro"])))
{
  date_default_timezone_set('UTC'); //asetetaan aikavyöhykkeeksi UTC/GMT aikaleimamuunnoksia varten
  $sqlstr = $_POST["sql"]; //hakulauseen tuottama sql-lause, jolla tehdään haku tietokannasta
  $alanro = $_POST["alanro"]; //alanumero kertoo, kenen puhelut haetaan

  //jos järjestystä ei saada käyttäjän syöttämänä, järjestetään hakutulokset päivämäärän ja kellonajan mukaan:
  $order = ($_POST["order"] == "" ? "ORDER BY pvm DESC,klo DESC" : $_POST["order"]);

  //hakutulosten rajoitus, eli sivulla näytettäville puheluille annettu enimmäismäärä:
  $limit = $_POST["limit"];
  $offset = $_POST["offset"]; //sivutuksessa käytettävä muuttuja

  //php:n stripslashes-funktio poistaa mahdolliset ylimääräiset kenoviivat sql-lauseesta:
  $sqlstr = stripslashes($sqlstr);

  //lopullisen sivutuksen määräävän sql-lauseen muodostus riippuu $limit- ja $offset-muuttujista
  $sivutus = " LIMIT ". $limit." OFFSET ". ($offset * $limit);

  //getCalls()-funktio suorittaa puhelujen haun
  getCalls($sqlstr,$alanro,$sivutus,$order);
}else die;
```

GetCalls()-funktio suorittaa tietokannasta haun lopullista SQL-lausetta käyttäen. Lause muodostuu funktion parametreinaan saamista $sqlstr, $alanro, $sivutus ja $order –muuttujista (Taulukko 12).

Taulukko 12. Tietokannasta haussa käytettävä SQL-lause PHP-muuttujien kanssa.

```
"SELECT * FROM puhelut WHERE ($sqlstr) AND alanro = $alanro $order $sivutus"
```

Esim. Parametrien arvot ovat seuraavat:

- $sqlstr = ”pvm >= 1262336677”
- $alanro = ”98”
- $order = ” ORDER BY pvm DESC,klo DESC”
- $sivutus = ”LIMIT 50 OFFSET 50”

Tällöin lopullinen SQL-lause, jolla haku tehdään, saadaan muodostettua, kun php-muuttujat kirjoitetaan ”selkokielisinä” (Taulukko 13).

Taulukko 13. Tietokannasta haussa käytettävä SQL-lause lopullisessa MySQL-syntaksin mukaisessa muodossaan.

```
"SELECT * FROM puhelut WHERE (pvm >= 1262336677) AND alanro = 98 ORDER BY pvm DESC,klo DESC LIMIT 50 OFFSET 50"
```

Lopullinen hakulause hakee nyt tietokannasta puhelut, joiden päivämäärä on 1.1.2010 tai myöhempi ja alanumero on 98. Hakutulokset järjestetään päivämäärän ja kellonajan mukaisesti laskevassa järjestyksessä. Hakutuloksia palutetaan 50 kpl ja jätetään pois 50 ensimmäistä, mikä käytännössä tarkoittaa sitä, että käyttäjä on napsauttanut puhelutaulun alla sijaitsevaa *Näytä 50 seuraavaa puhelua* –painiketta, eli siirtynyt järjestyksessä toiselle sivulle. Käyttäjän rakentamaa hakulausetta käytetään siis aina SELECT-lauseen WHERE-ehtona.

6.5 Puhelutaulun tuottaminen

Puhelutaulun muodostaminen tapahtuu haepuhelut.php –tiedoston puheluTaulu()-funktion suorittamana, mikäli hakutyyppinä on ”puhelutaulu” (ks. 6.3). Haetut puhelut tallennetaan haun yhteydessä kaksiulotteiseen taulukkoon. Sivulle tuotetaan uusi table-elementti, jonka jälkeen käydään hakutulokset sisältävä taulukko läpi foreach-rakenteessa ja lisätään table-elementtiin oma rivi kullekin puhelulle, sekä kullekin riville puhelutietojen tarvitsema määrä sarakkeita (Kuvio 21). Tauluun tulostuu nyt Tukijuhdan käyttäjähallinnan kautta saadut henkilöiden etunimet tietokannassa olevien alanumeroiden sijaan. Käyttäjätiedot on tallennettu istunnon alkaessa php:n $_SESSION –taulukkoon, jonka avulla etsitään kutakin alanumeroa vastaava nimi.

Puhelut | Tilastot henkilöittäin

pvm	klo	StarSoft	vastausaika/s	puh.kesto	asiakas	puh.tyyppi	kenelle	keneltä
25.01.2011	12:58:32	petra	19	00:04:03	050463****	vastattu		
25.01.2011	12:52:40	terttu	(soitettu)	00:10:37	044780****	soitettu		
25.01.2011	12:49:12	petra	5	00:06:47	093509****	vastattu		
25.01.2011	12:46:42	katariina	(käännetty)	00:04:18	05234****	käännetty		petra
25.01.2011	12:46:16	elli	41	00:00:20	044780****	vastattu		
25.01.2011	12:46:03	petra	19	00:00:38	05234****	vastattu	katariina	
25.01.2011	12:44:53	mirva	22	00:04:41	040311****	vastattu		
25.01.2011	12:44:21	petra	15	00:01:25	044700****	vastattu		
25.01.2011	12:44:11	joni	7	00:03:57	040764****	vastattu		
25.01.2011	12:41:07	olli-pekka	(soitettu)	00:01:48	014266****	soitettu		
25.01.2011	12:31:26	katariina	(käännetty)	00:05:59	044376****	käännetty		joni
25.01.2011	12:31:04	joni	24	00:00:21	044376****	vastattu	katariina	

Kuvio 21. Puhelutaulu.

Tauluun lisätään myös otsikko-selitteet ja niiden alle nuolikuvakkeet, joiden napsautus järjestää tiedot kyseisen sarakkeen mukaan nousevasti tai laskevasti. Taulun alle lisätään painike, jolla saa näkyviin seuraavan sivun puhelut, mikäli haetut puhelut eivät mahdu yhdelle sivulle. Painikkeen tekstiksi tulee *Näytä seuraavat n puhelua*, missä *n* tarkoittaa sivulle mahtuvien puheluiden enimmäismäärää, joka on käyttäjän määrittämä(ks. 5.2) tai oletusarvoisesti 50 kpl. Jos sivulle on määritetty mahtumaan 100 puhelua, tulee painikkeen tekstiksi *Näytä seuraavat 100 puhelua*. Lisäksi puhelutaulun alla ilmoitetaan tekstinä, mitkä puhelut ovat näkyvissä, esim. *Puhelut 1 ... 50*. Jos ei olla ensimmäisellä sivulla, lisätään taulun alle myös painike, jolla pääsee palaamaan edelliselle sivulle (Kuvio 22).

Kuvio 22. Sivunvaihtonapit.

6.6 Henkilökohtaisen tilastotaulun tuottaminen

Puhelutaulun muodostuttua sivulle, muodostuu sen yläpuolelle div-elementteinä kaksi välilehteä, joiden avulla voi vaihtaa näkymää puhelutaulun ja tilastot henkilöittäin sisältävän taulun välillä. Nämä välilehdet näkyvät vain käyttäjille, joille on annettu Tukijuhdan käyttäjänhallinnassa ns. tilasto-oikeudet. Kyseinen tieto tarkistetaan $_SESSION –tauluun tallennetuista käyttäjätiedoista. Kun napsautetaan *Tilastot henkilöittäin* –välilehteä, kutsutaan Hakulause.haePuhelut()-metodia ja lähetetään parametrina teksti ”personalstat_tab”, jonka ansiosta Hakulause.hakutyyppi –ominaisuuden arvoksi muuttuu ”henk_tilasto”. Haku tehdään uutena Ajax-pyyntönä haepuhelut-php –tiedostolle käyttäen Hakulause.lastsqlstring –arvoa joka tarkoittaa, että käytetään samaa käyttäjän rakentamaa hakulausetta, mitä käytettiin viimeksi tehdyssä haussa.

Koska hakutyyppi on nyt ”henk_tilasto” eikä ”puhelutaulu”, ei ajeta puheluTaulu()-funktiota, joka tulostaa puhelutaulun, vaan sen sijaan personalStats()-funktio, joka tulostaa taulun, jossa on puhelumäärät henkilöittäin jaoteltuna (ks. 5.5). PersonalStats()-funktio ei hae tietokannasta yksityiskohtaisia puhelutietoja, vaan pelkät käyttäjäkohtaiset eri tyyppisten puheluiden määrät (Taulukko 14).

Taulukko 14. Käyttäjäkohtaisten erityyppisten puheluiden haussa käytetty SQL-lause.

```
"SELECT alanro, SUM(IF(puhtyyppi = 'v', 1,0)) AS 'vastatut', SUM(IF(puhtyyppi = 'k', 1,0)) AS 'kaannetyt', SUM(IF(to_alanro IS NOT NULL, 1,0)) AS 'kaannetyt2', SUM(IF(puhtyyppi = 's', 1,0)) AS 'soitetut' FROM puhelut WHERE ($sqlstr)$removeNonUsers GROUP BY alanro $order"
```

SQL-lause kerää puhelumäärät henkilöittäin ja puhelutyypeittäin SUM-funktion avulla. Tämä koskee kuitenkin ainoastaan niitä puheluita, jotka täyttävät WHERE-ehtona olevan käyttäjän muodostaman hakulauseen.

Haetut tiedot tallennetaan kaksiulotteiseen taulukkoon, jonka avulla tuotetaan tilastotaulu (Kuvio 23) samaan tapaan kuin puhelutaulukin, tällä kertaa tauluun lisätään yksi rivi kullekin henkilölle, jolle löytyi puheluita haun avulla. Tietojen järjestäminen toimii kuten puhelutaulussakin napsauttamalla otsikoiden alla olevia nuolia. Jos hakulausetta muutetaan nyt ja painetaan *Hae*-painiketta tai valitaan pudotusvalikosta tallennettu haku, muuttuvat henkilöittäin koostetut tilastot hakua vastaaviksi sen sijaan, että muodostettaisiin uusi puhelutaulu, koska hakutyyppinä on edelleen ”henk_tilasto”.

Puhelut | Tilastot henkilöittäin

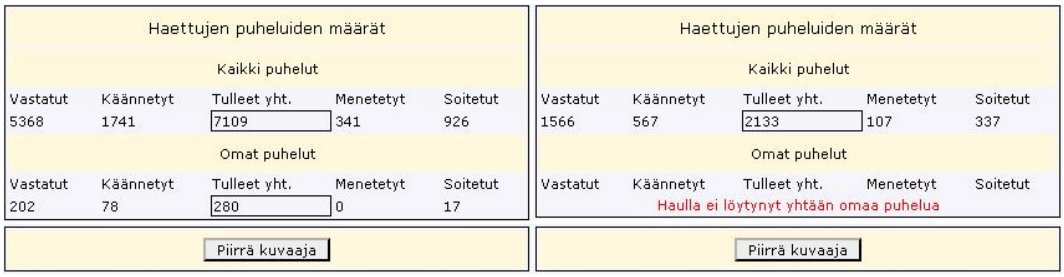
nimi	vastatut	käännetyt <-	käännetyt ->	soitetut
petra	1685	147	146	41
joni	1447	299	151	33
jarkko	1400	187	123	5
mirva	1083	37	230	134
terttu	1017	308	80	91
väinö	941	28	139	23
jyrki	934	36	92	38
daniel	879	87	93	69
simo	830	43	168	15
elli	799	122	90	61
juhani	556	280	49	41
olli-pekka	526	151	53	13
ismo	411	7	123	56
lennart	274	130	3	24
niilo	251	9	71	24
katariina	123	246	8	204
pertti	103	231	13	211
oliver	42	5	0	329
greger	33	193	3	70
laura	25	114	1	65
kalervo	20	50	0	132
heidi	18	42	1	67
reino	16	9	0	143
elisa	13	320	1	41
mikko	11	131	0	9
ville	6	12	3	11
antti	4	6	0	7
aku	3	151	0	13
pekka	1	0	0	0
tuula	0	63	5	4
hannu	0	4	0	0
pirkko	0	2	0	3

Kuvio 23. Tilastot henkilöittäin.

Jos napsautetaan *Puhelut*-välilehteä, muuttuu hakutyypiksi ”puhelutaulu”, jolloin tehdään uusi haku ja muodostetaan uusi puhelutaulu.

6.7 Puhelumäärien yhteenvedon ja graafin lähtötietojen tuottaminen

Riippumatta siitä, hakeeko käyttäjä tietokannasta vain omien vai kaikkien puheluiden tietoja, koostetaan aina puhelumäärien yhteenveto kaikista puheluista, jotka käyttäjän luoma haku palauttaa. Tämä tapahtuu haepuhelut.php –tiedoston tulostaTilastot()-funktion toimesta. Kyseinen funktio ajetaan aina haun tapahtuessa, oli kyseessä sitten puhelutaulu, tai tilasto henkilöittäin. Funktio muodostaa sivulle taulun, jossa on koottuna kunkin puhelutyypin puhelumäärät sekä kaikkien, että käyttäjän omien puheluiden osalta (Kuvio 24).



Kuvio 24. Kooste puhelumääristä.

TulostaTilastot()-funktio muodostaa sivulle myös piilotetut p-tagit, joiden sisään tulostuu $datestring-muuttujien sisältämät kunkin päivämäärän puheluiden lukumäärät sekä omien että kaikkien henkilöiden puhelujen osalta (Taulukko 15). $datestring-muuttujan sisältö voi olla esim.:”1.1.2010:35 2.1.2010:40 3.1.2010:89 jne.” Näiden tietojen pohjalta muodostetaan graafi puhelumääristä.

Taulukko 15. Graafin lähtötiedot sisältävät p-tagit.

```
//omat vastatut
echo"<p id=\"dateString_v_oma\" style=\"display:none\">$dateString_v</p>";
//omat soitetut
echo"<p id=\"dateString_s_oma\" style=\"display:none\">$dateString_s</p>";
 //omat menetetyt
echo"<p id=\"dateString_m_oma\" style=\"display:none\">$dateString_m</p>";
 //omat käännetyt
echo"<p id=\"dateString_k_oma\" style=\"display:none\">$dateString_k</p>";
.
.
.
 //vastatut
echo"<p id=\"dateString_v\" style=\"display:none\">$dateString_v</p>";
 //soitetut
echo"<p id=\"dateString_s\" style=\"display:none\">$dateString_s</p>";
 //menetetyt
echo"<p id=\"dateString_m\" style=\"display:none\">$dateString_m</p>";
 //käännetyt
echo"<p id=\"dateString_k\" style=\"display:none\">$dateString_k</p>";
 //vastattujen ja käännettyjen erotus
echo"<p id=\"dateString_vk\" style=\"display:none\">$dateString_vk</p>";
```

6.8 Graafin toiminta

Puhelumäärien kuvaaja toteutetaan erilliseen Graafi.js –tiedostoon luotavana Graafi-oliona. Graafi piirretään uuteen iframe-elementtiin, joka avautuessaan sisältää vain rastikentät, joilla valitaan tulostuvat puhelutyypit sekä *Poistu*-painikkeen, joka sulkee graafi-ikkunan(Kuvio 25). Tulostuvista puhelutyypeistä on alkutilanteessa valittuna

pelkästään vastatut puhelut. Käyttäjä voi valita tulostumaan myös käännetyt, soitetut ja menetetyt puhelut. Jokin puhelutyypeistä on oltava aina valittuna.

Kuvio 25. Graafin puhelutyyppien valinta.

Kun käyttäjä napsauttaa puhelutaulun ja koostetilaston myötä sivulle muodostunutta *Piirrä kuvaaja* –painiketta, avataan graafi-ikkuna uuteen iframe-elementtiin ja kutsutaan Graafi-olion graphData()-metodia, joka poimii piilotettujen p-tagien sisältämän datan puhelutyyppikohtaisiin taulukkomuuttujiin (Taulukko 16).

Taulukko 16. Graafi-olion ominaisuudet

```
Graafi = {
   dataVastatut:[], //taulukko vastatuille puheluille
   dataSoitetut:[], //taulukko soitetuille puheluille
   dataMenetetyt:[], //taulukko menetetyille puheluille
   dataKaannetyt:[], //taulukko käännetyille puheluille
   topValue:0, //suurin päiväkohtainen puhelumäärä
   startDate:'', //ensimmäinen päivämäärä
   endDate:'', //viimeinen päivämäärä
.
.
.
}
```

Lähtötilanteessa vain vastattujen puheluiden rastikenttä on valittuna, jolloin poimitaan sen p-tagin, jonka id on ”dateString_v”, sisältämät tiedot Graafi.dataVastatut –taulukkoon Graafin dateSort()-metodin avulla. Kyseinen metodi pilkkoo merkkijonon taulukkomuotoon järjestettyään tiedot aikajärjestykseen; taulukon ensimmäiselle paikalle tulee ensimmäisen päivämäärän puhelumäärä ja viimeiselle viimeisen päivän puhelumäärä (Taulukko 17).

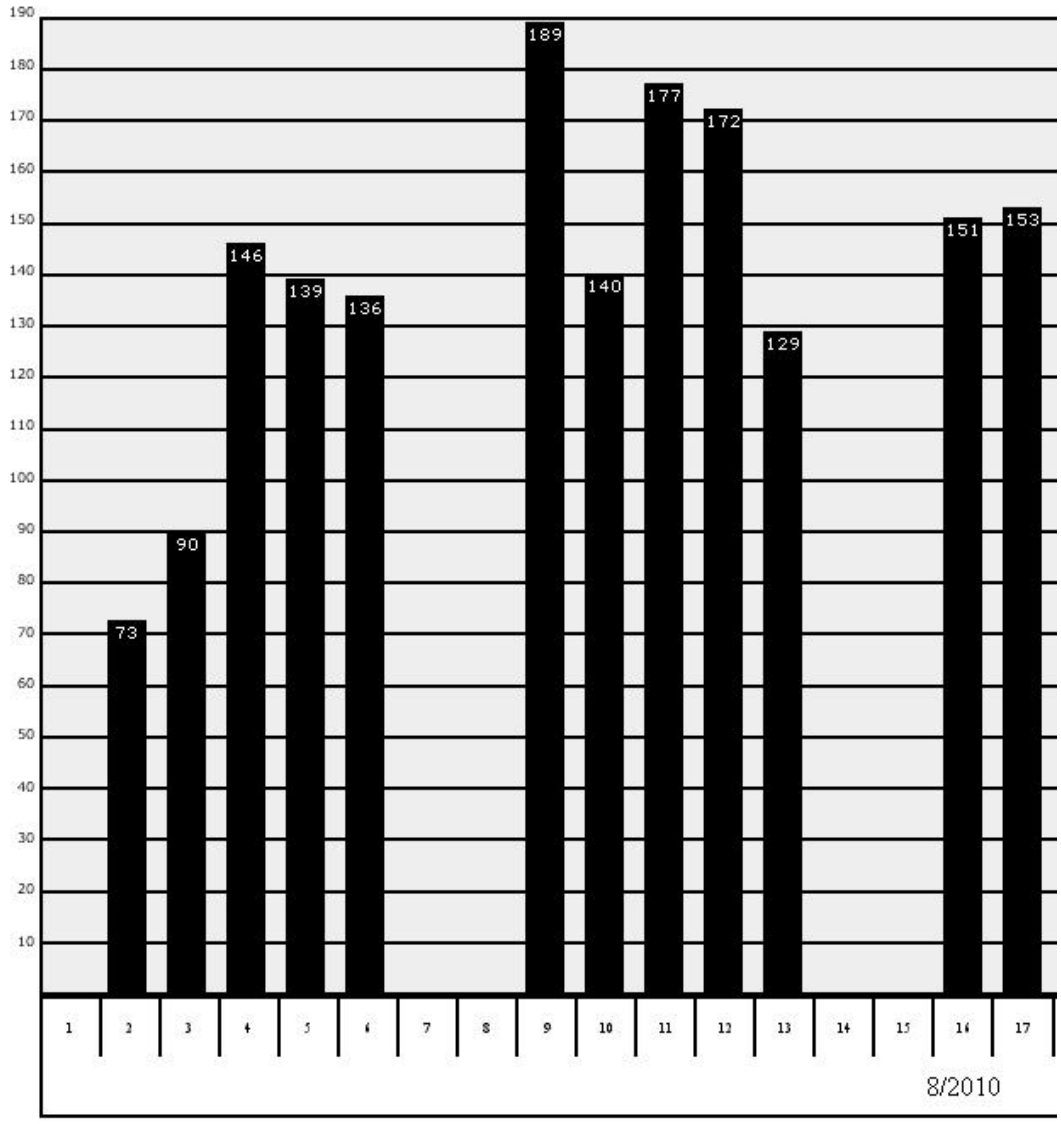
Taulukko 17. Esimerkki dataVastatut-taulukon sisällöstä.

```
dataVastatut[0] = 1.1.2010:35 ... dataVastatut[20] = 30.1.2010:68
```

Tämän jälkeen graphData()-metodi kutsuu getTopValue()-metodia, jonka tehtävä on selvittää suurin käsiteltävä puheluiden määrä, sekä startEnd()-metodia, joka selvittää ensimmäisen ja viimeisen käsiteltävän päivämäärän. Näiden tietojen perusteella tiedetään, montako riviä ja saraketta tarvitaan tauluun, joka toimii graafin pohjana.

Kukin rivi vastaa kymmentä puhelua ja kukin sarake yhtä kuukautta. Jos suurin päiväkohtainen puheluiden määrä on esim. 85, tarkoittaa se, että taulussa on oltava yhdeksän riviä. Jos ensimmäinen päivämäärä on 1.1.2010 ja viimeinen päivämäärä on 20.3.2010, tarvitaan kolme saraketta, koska puhelut jakautuvat kolmelle kuukaudelle. Rivien määrä saadaan rowCount()-metodin paluuarvona ja sarakkeiden määrä columnCount()-metodin palauttamana.

Kun graafin luomiseen tarvittavat tiedot on saatu, kutsutaan Graafi-olion drawGraph()-metodia, joka piirtää varsinaisen kuvaajan luomalla siihen tarvittavat elementit dynaamisesti. Table-elementtejä luodaan kolme kappaletta; yksi varsinaisen kuvaajan pohjaksi, toinen sen alapuolelle päiviä ja kuukausia varten ja kolmas vasempaan reunaan, johon tulostuvat numeeriset arvot kullekin riville (Kuvio 26).



Kuvio 26. Kolmesta table-elementistä koostuva graafi.

Taulujen jälkeen luodaan niiden lapsielementeiksi tarvittava määrä tr-elementtejä, eli rivejä Graafi-olion rowCount()-metodin määräämän rivilukumäärän verran toistettavan toistorakenteen avulla. Joka kierroksella kutsutaan createTableColumn()-metodia, joka puolestaan luo jokaiselle tr-elementille lapsielementeiksi niin monta td-elementtiä kuin columnCount()-metodi antaa sarakelukumääräksi. CreateTableColumn()-metodi saa parametreikseen kulloinkin kyseessä olevan rivin sekä sarakkeen numeron. Kun päästään viimeiselle riville, kutsutaan createPillars()-metodia, joka piirtää pylväät div-elementteinä kyseisen sarakkeen lapsiksi (Taulukko 18).

Taulukko 18. CreateTableRow()- ja createTableColumn()-funktiot luovat graafiin tarvittavat elementit.

```
//drawGraph()-metodin toistolause, joka lisää tarvittavan määrän rivejä taulukkoon kutsumalla rivit luovaa createTableRow()-metodia
var i;
      for(i=0; i<this.rowCount(); i++)
        {
          graphTable.appendChild(this.createTableRow(i));
          numTable.appendChild(this.createnumRow(i));
        }
//createTableRow()-metodi palauttaa tr-elementin, mutta ennen sitä lisää sille lapsielementeiksi td-elementtejä tarvittavan määrän kutsumalla ne luovaa createTableColumn()-metodia
createTableRow:function(n)
  {
     var i, row = document.createElement('tr');
     for(i=0; i< this.columnCount(); i++)
       {
              row.appendChild(this.createTableColumn(n,i));
       }
     return row;
  }

//createTableColumn()-metodi palauttaa td-elementin. Jos ollaan viimeisellä rivillä, luodaan td-elementin lapseksi kyseiselle kuukaudelle kuuluvat pylväät kutsumalla pylväät luovaa createPillars()-metodia
createTableColumn:function(rowNr,colNr)
  {
     var i, column = document.createElement('td');
     column.style.width =((100 / this.getAllDays())*this.getMonthDays(colNr)) + '%';
     column.className = 'grph';
     column.style.borderWidth=(this.columnCount()>2 ? 1 : 2)+'px';

     if(rowNr==this.rowCount()-1)
       {
          for (i=1; i<=this.getMonthDays(colNr); i++)
            {
              if(this.dataVastatut.length>0)column.appendChild(this.createPillars(colNr,i,'v'));
              if(this.dataKaannetyt.length>0)column.appendChild(this.createPillars(colNr,i,'k'));
              if(this.dataSoitetut.length>0)column.appendChild(this.createPillars(colNr,i,'s'));
              if(this.dataMenetetyt.length>0)column.appendChild(this.createPillars(colNr,i,'m'));
            }
       }

     return column;
  }
```

Kun drawGraph()-metodin toistolause on käyty läpi, on graafi piirretty valmiiksi pylväineen. Tämän jälkeen voidaan luodut table-elementit, jotka nyt sisältävät koko graafin, lisätä niille varattujen staattisesti luotujen div-elementtien lapsiksi, jolloin graafi lopulta sijoitetaan sivulle näkyviin.

7 Johtopäätökset

7.1 Prosessi

Puhelutilastointiprojekti eteni eräänlaisen inkrementaalisen evo-mallin mukaisesti. Projektin alkuvaiheessa keskusteltuani tilastointijärjestelmää eniten hyödyntävien henkilöiden kanssa, muodostettiin melko löyhä vaatimusmäärittely, jossa tulivat esille tärkeimmät vanhan järjestelmän parannusta kaipaavat kohteet (ks. 3.2). Käytännön toteutuksen sain tehdä itse parhaaksi katsomallani tavalla. Prosessia voidaan pitää inkrementaalisen evo-mallin mukaisena, koska jaoin toteutuksen pienempiin osakokonaisuuksiin, jotka testasin ja käyttöliittymävaiheeseen päästyäni myös julkaisin kollegoideni testattavaksi sekä parannusehdotuksia saadakseni. Sain tällä tavalla teetettyä käyttäjillä ns. black box –testausta, mikä ei olisi tullut kysymykseen, mikäli osakokonaisuudet olisi toimitettu ulkopuoliselle asiakkaalle.

Inkrementit julkaistiin seuraavassa järjestyksessä:

- Puheluloggeri
- Yksinkertainen päivämäärähaku tietokannasta
- Hakulauseen käyttöliittymä ja implementointi Tukijuhta-ohjelmaan
- Puhelumäärien graafinen kuvaaja
- Valmiiden hakujen tallennus tietokantaan
- Henkilökohtaisen tilaston koostaminen hakulauseen määrittämin ehdoin

Inkrementtien julkaisuille ei ollut mahdollista tehdä aikataulua, koska tein töitä puhelutilastointiprojektin parissa pääasiallisen työtehtäväni salliessa. Pystyin kuitenkin koko projektin elinkaaren aikana käyttämään sen kehittämiseen keskimäärin 60% työajastani.

7.2 Puhelutilastointiprojektin ketterät piirteet

Projektia jälkikäteen analysoidessani kykenin erottamaan sen kulusta monia aiemmin kuvailtuja ketterien menetelmien ominaispiirteitä. Eri menetelmiin sopivia havaintoja löytyi useita ja ne voidaan kuvailla Agile Modeling -menetelmän joidenkin pääperiaatteiden avulla (ks. 2.3.1), koska nämä ovat keskeisiä periaatteita useimmissa

ketterissä menetelmissä ja ketterän manifestin(ks. 2.1) taustalla olevien arvojen mukaisia. Kyseiset piirteet ilmenivät omassa projektissani seuraavalla tavalla:

AM - pääperiaatteita

- Tarkoituksenmukainen mallinnus ja asiakkaan tarpeiden tuntemus
    - Työskenneltyäni yrityksessä pitemmän aikaa ja oltuani läheisessä keskustelukontaktissa asianomaisten henkilöiden kanssa tunnen ohjelmiston vastaanottajan tarpeet.
- Välittömän asiakaspalautteen tarpeellisuus
    - Pyrin saamaan ohjelmiston käytettävyyden mahdollisimman hyvälle tasolle ja tässä käyttäjäpalaute oli erittäin tärkeässä asemassa.
- Muutosvalmius ja myönteinen suhtautuminen muutoksiin
    - Edellisestä kohdasta johtuvan asiakaslähtöisyyden vuoksi toteutin ominaisuudet loppukäyttäjien toiveiden mukaan ja luovuin joistakin omista ratkaisuistani korvatakseni ne käyttäjien toivomilla ratkaisuilla.
- Inkrementaalinen kehitys
    - Jaoin koko projektin inkrementteihin mm. pienempien osakokonaisuuksien testauksen mahdollistamiseksi.
- Korkean laadun tuottaminen
    - Em. testauksen ja käyttäjälähtöisyyden ansiosta lopputuloksena oli hyvin toimiva ohjelma.
- Seuraavan tehtävän mahdollistaminen, esim. tehdyn ohjelmiston laajennus, tulevaisuuden huomioon ottaminen.
    - Puheluiden aikana tapahtuvien asiakaskontaktien ja tukitapahtumien kirjaaminen yrityksen asiakasrekisteriin tullaan mahdollisuuksien mukaan ohjelmoimaan tulevaisuudessa puhelutilastointisovelluksen ominaisuudeksi. Yritin ottaa tämän huomioon jo tässä vaiheessa, jotta käyttöliittymän laajennus onnistuisi tarvittaessa.

7.3 Projektin parannuskohteet

Vaikkakin puhelutilastointiprojektista oli havaittavissa ketteriä piirteitä, olisi prosessin kulkua pystytty parantamaan entisestään ketteriä menetelmiä soveltamalla. Lean-ajattelua (ks. 2.3.2) ja XP:n ohjelmointi- ja testauskäytäntöjä (ks. 2.3.4) hyödyntämällä oltaisiin luultavasti päästy parempiin tuloksiin tiettyjen osa-alueiden osalta (Taulukko 19).

Taulukko 19. Eliminoitavat tekijät ja parannuskeinot.

Eliminoitava tekijä (Lean)	**Parannuskeino (XP)**
Osittain tehty työ, ylimääräiset prosessit ja ominaisuudet.	Säännöllinen ohjelmakoodin refaktorointi; puolivalmiin ja turhan koodin olisi voinut poistaa säännöllisesti. Mm. hakulauseen koodi olisi ollut hyvä refaktoroida useaan otteeseen sen jatkuvan muuttumisen takia.
Tehtävien vaihdot, useamman tehtävän suorittaminen samanaikaisesti. Tehtävien luovutukset henkilöltä toiselle, vastuun siirto. Odotus.	Ohjelmoijan iteraation ajaksi valitsemaansa tehtävään keskittyminen. Projekti olisi edennyt sujuvammin, jos olisin voinut keskittyä siihen kokoaikaisesti. Tämä ei kuitenkaan ollut mahdollista johtuen pääasiallisesta työtehtävästäni StarSoftin asiakastuessa.
Viat, bugit.	Säännöllinen ohjelmakoodin refaktorointi ja jatkuva osittain automatisoitu yksikkötestaus testausympäristössä, sekä koko osakokonaisuuden testaus ennen sen implementoimista pääohjelmaan (Tukijuhta). Käyttäjien bugihavainnot olisivat vähentyneet huomattavasti. Tämän kohdan tärkeys korostuu ulkopuolisen asiakkaan ollessa kyseessä.

XP:n käytännöistä myös pariohjelmoinnista olisi ollut hyötyä, olettaen että projektiryhmä olisi koostunut useammasta henkilöstä itseni lisäksi. Erityisesti hakulauseen ja graafin jotkut funktiot vaativat paljon pohdiskelua ja pariohjelmoinnin mahdollistamasta kahden ohjelmoijan yhteistyöstä olisi epäilemättä ollut apua näissä tapauksissa. Lisäksi vaatimusmäärittelyvaiheessa olisi voitu hyödyntää XP:n käytäntöjen mukaisia käyttäjätarinoita apuna muodostamaan selkeämpi käsitys halutuista ominaisuuksista sekä vanhan järjestelmän sisältämistä ominaisuuksista ja niille toivotuista parannuksista.

7.4 Projekti Scrum/XP-prosessina

Puhelutilastointiprojekti oltaisiin voitu toteuttaa Scrum/XP –yhdistelmänä hyödyntäen samalla Lean-ajattelua sekä AM:n periaatteita. XP:n käyttäjätarinoiden pohjalta voitaisiin laatia Scrumin määrittelemä tuotteen työlista (ks. 2.4.2). Projektin iteratiivisen ja inkrementaalisen luonteen ansiosta Scrum tuntuisi sopivalta ohjausmenetelmältä ja XP:n käytännöt helposti ohjelmointiteknisissä asioissa hyödynnettäviltä. Tämä ratkaisu edellyttäisi kuitenkin käytännössä useammasta henkilöstä koostuvaa projektiryhmää, ScrumMasterin osallistumista, sekä kaikkien osallistumista projektiin päätoimisesti.

Tuotteen työlista (Taulukko 20) laadittaisiin ajatellen, että kaikki sprintin tehtävät jaetaan kullekin ryhmän jäsenelle. Käytännössä ohjelmointityö suoritettaisiin kuitenkin pariohjelmointina XP:n mukaisesti. Ohjelmointityössä noudatettaisiin myös XP:n yksikkötestauskäytäntöjä.

Taulukko 20. Tuotteen työlista.

id	Sprintti / Kuvaus / Sprintin kokonaistyömäärä (h)	1 50	2 130	3 80	4 80
1	Sarjaportin luku	15			
2	Tietokannan taulujen luontilauseiden laatiminen	5			
3	Puheluloggerin tietokantayhteys ja taulujen luonti	10			
4	Puheludatan parseri ja tietokantaan tallennus	10			
5	Virheilmoitusten sähköpostilähetys ja ini-tiedosto	10			
6	Uuden osion lisäys Tukijuhtaan		10		
7	Hakulauseen ja sen käyttöliittymän rakentaminen		60		
8	Tietokannasta haku ja puhelutaulun sekä koontitilaston muodostus		60		
9	Graafin lähtötietojen keruu ja oletusasetukset			10	
10	Graafin muodostaminen			70	
11	Valmiiden hakujen tallentaminen tietokantaan ja lataus tietokannasta				40
12	Henkilökohtaisen puhelumääräkoosteen laatiminen				40

Tuotteen työlista on priorisoitu alkuvaiheessa siten, että välttämättömimmät ja kriittisimmät ominaisuudet (puheluloggeri) toteutetaan ensin. Tarvittaessa tuotteen omistaja priorisoi työlistaa uudelleen olosuhteiden mahdollisesti muuttuessa ja vaikuttaessa projektin osakokonaisuuksien keskinäiseen tärkeysjärjestykseen.

Tuotteen työlistan lisäksi laadittaisiin sprinttikohtaisesti myös sprintin työlista (Taulukko 21), joka sisältää sprintin aikana tuotettavan inkrementin tarkemmat yksityiskohdat. Scrum-menetelmää käytettäessä järjestettäisiin luonnollisesti päivittäin Scrum-kokous, jonka lisäksi järjestetään sprintin katselmointi ja jälkitarkastelu ja tämän jälkeen seuraavan sprintin suunnittelukokous.

Taulukko 21. Kolmannen sprintin työlista.

		Sprintin päivät									
id	Kuvaus Sprintin jäljelläolevat tunnit	1 80	2 72	3 64	4 56	5 48	6 40	7 32	8 24	9 16	10 8
9	Graafin lähtötietojen keruu ja oletusasetukset										
	Lähtötietodot merkkijonomuotoon tietokannasta	5									
	Graafin iframen ja muiden staattisten elementtien luonti	3	2								
10	Graafin muodostaminen										
	Lähtötietojen järjestys aikajärjestykseen		6								
	Maksimiarvon ja sarakemäärän määritys			8							
	Kuukausien ja päivien lukumäärien selvitysfunktiot				4						
	Dynaamisten elementtien luontifunktiot				4	8					
	Graafin pylväiden luontifunktio						8				
	Tapahtumankäsittelijöiden määritys							8			
	Graafin tyylimääritykset								8	4	
	Kuukausinäkymän poiminta koko näkymästä									4	
	Käyttäjäkohtainen graafi ja graafin tulostus										8

Tuotteen ja sprintin työlistat toimivat hyvin ajankäytön havainnollistamisvälineinä. Oheiset esimerkkityölistat eivät tosin vastaisi todellisuutta suunnitteluvaiheessa, koska ne ovat toteutuneista sovelluksen osakokonaisuuksista ja niiden osista/funktioista jälkikäteen koostettuja. Todellisessa tilanteessa tuotteen ja sprintin työlistaa laadittaessa tehtävät eivät luultavasti olisi yhtä seikkaperäisesti kuvailtuja ja aika-arvioihin olisi ehkä lisättävä enemmän pelivaraa.

7.5 Loppusanat

Perehdyttyäni ketteriin menetelmiin ja pohtiessani tehtyä työtä niiden valossa, tuntui Scrum/XP –yhdistelmä luonnolliselta valinnalta pienen ohjelmistoprojektin ohjausmenetelmäksi. Suuremman projektin ja siten suuremman henkilömäärän projektiryhmän tapauksessa voidaan valita menetelmäksi vaikkapa jokin Crystal-perheen menetelmistä (ks. 2.3.3) ryhmän koon määräämän kategorian perusteella. Molemmissa tapauksissa menetelmien taustalla olevat arvot ja periaatteet ovat kutakuinkin samat, eli ketterän manifestin mukaiset, vaikka menetelmien käytännöt saattavatkin poiketa toisistaan.

Scrumin inkrementaalisuus yhdistettynä XP:n testaus- ja ohjelmointikäytäntöjen toimivaan toteutukseen herättää ainakin itsessäni luottamusta projektin riskienhallinnan onnistumiseen. Kun näihin yhdistetään vielä tehtävälistojen jatkuva priorisointi siten, että riskialttiimmat tehtävät hoidetaan projektin alkuvaiheessa, sekä välitön kommunikointi päivittäisen Scrumin tapaan ja yleensäkin ketterän manifestin edustamien periaatteiden noudattaminen, on lopputuloksena suurella todennäköisyydellä toimiva tietojärjestelmä.

LÄHDELUETTELO

Painetut teokset

Cockburn, Alistair (2007). Agile Software Development. 2. painos. Crawfordsville, Indiana: RR Donnelly.

Newkirk, James & Robert C. Martin (2001). Extreme Programming in Practice. 1. painos. Reading, Massachusetts: Addison-Wesley.

Schwaber, Ken (2004). Agile project management with Scrum. 1. painos. Redmond, Washington : Microsoft Press.

Elektroniset julkaisut

Agile Alliance (2010). The Agile Manifesto [Viitattu 03.11.2010]. Saatavana Internetissä: <URL: http://www.agilealliance.org/the-alliance/the-agile-manifesto/>.

Ambler, Scott (2007). Agile Modeling [Viitattu 12.11.2010]. Saatavana Internetissä: <URL: http://www.agilemodeling.com/>.

Baeseman, Cliff, Shane Miller, Michael A. Hess (1999). Lazarus Free Pascal [Viitattu 14.10.2010]. Saatavana Internetissä: <URL: http://www.lazarus.freepascal.org>.

Beck, Kent, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, D. Thomas (2001). Manifesto for Agile Software Development [Viitattu 03.11.2010]. Saatavana Internetissä: <URL: http://www.agilemanifesto.org/>.

Berteig, Mishkin (2007). Agile Benefits: Early Return on Investment [Viitattu 05.11.2010]. Agile Advice. Saatavana Internetissä: <URL: http://www.agileadvice.com/archives/2007/09/agile_benefits_2.html>.

Gebauer, Lukas (2007). Ararat Synapse [Viitattu 14.10.2010]. Saatavana Internetissä: <URL: http://www.ararat.cz/synapse/doku.php/start>.

Poimala, Sami, Jouni Heikniemi, Henrik Blåfield (2010). Ketterät käytännöt[Viitattu 20.11.2010]. Saatavana Internetissä: <URL: http://www.ketteratkaytannot.fi>.

Poppendieck, Mary (2003). Lean Software Development [Viitattu 21.11. 2010]. Saatavana Internetissä: <URL: http://poppendieck.blogspot.com/2003/04/lean-software-development.html>.