Janne Olli

HALLITTU OHJELMISTOKEHITYS VOITTOA TAVOITTELEMATTOMASSA VAPAAEHTOISTYÖSSÄ

Opinnäytetyö
KESKI-POHJANMAAN AMMATTIKORKEAKOULU
Tietotekniikka
Huhtikuu 2011

KESKI-POHJANMAAN AMMATTIKORKEAKOULU
MELLERSTA ÖSTERBOTTENS YRKESHÖGSKOLA

TIIVISTELMÄ OPINNÄYTETYÖSTÄ

Yksikkö Kokkola	**Aika** Huhtikuu 2011	**Tekijä/tekijät** Janne Olli
Koulutusohjelma Tietotekniikan koulutusohjelma		
Työn nimi Hallittu ohjelmistokehitys voittoa tavoittelemattomassa vapaaehtoistyössä		
Työn ohjaaja Sakari Männistö		**Sivumäärä** 53
Työelämäohjaaja Jaakko Tahkola		

Suomen Rauhanyhdistysten Keskusyhdistyksen järjestötoiminnassa tarvittiin erilaisia yhdistyksen toimintaa ja yhdistyksen kautta tehtävää uskonnollista lähetystyötä helpottavia ohjelmistoja ja tietoteknisiä ratkaisuja. Vapaaehtoisvoimin toteutettavassa ohjelmistokehitystyössä kohdattiin ongelmia, jotka johtuivat pääosin huonosta työn organisoinnista ja projektinhallinnasta. Näihin ongelmiin pyrittiin löytämään ratkaisu.

Opinnäytetyössä kuvataan organisaatiorakenteessa ja ohjelmistokehitystyössä havaitut puutteet ja ongelmat sekä verrataan havaittuja ongelmia organisaatiorakenteiden ja ohjelmistokehitystyössä yleisesti tunnettuihin ominaisuuksiin ja ongelmiin. Näiden tietojen pohjalta lähdettiin rakentamaan sellaista organisaatiota ja ohjelmistokehitysmallia, jotka istuisivat hyvin uskonnollista toimintaa harjoittavan yhdistyksen olemassa olevaan perusorganisaatioon samalla tukien mahdollisimman hyvin talkoopohjaista ohjelmointikehitystyötä. Haasteen sopivien mallien löytymiseen toi se, että ohjelmistokehitystyö ei kuulunut yhdistyksen varsinaiseen toimenkuvaan. Opinnäytetyön aikana erilaisia malleja arvioitiin ja kokeiltiin sen perusteella, mikä toimisi parhaiten tilanteessa, jossa kaukana toisistaan asuvat organisaation jäsenet voivat osallistua ohjelmointityöhön vain muutamia tunteja viikossa. Eri malleja tuli tarkastella myös siten, että niitä voitaisiin hyödyntää myös silloin, kun ohjelmiston vaatimukset muuttuvat nopeasti ja ohjelmoijien vaihtuvuus on suuri.

Opinnäytetyön aikana organisaation pohjaksi tarkentui matriisiorganisaatio, jossa ohjelmointityö oli jaettu pääosin yhdistyksen eri sidosryhmien mukaisiin linjoihin. Ohjelmistojen kehitysvaiheen nopeisiin muutoksiin apua toi parhaiten Ketterä-ohjelmistokehitysmalli. Perusmallista räätälöitiin kahden sprintin Scrum-prosessi, jossa otettiin paremmin huomioon talkootyön luonne. Uusien mallien ja työkalujen avulla saatiin talkootyöllä syntyvien ohjelmistojen laatua parannettua sekä pystyttiin paremmin arvioimaan työkuormaa ja aikatauluja. Ohjelmistokehitykseen yhtenä osana kuuluva ylläpito annettiin pääosin käyttäjien vastuulle, jolloin vähäiset talkootuntimäärät voitiin hyödyntää mahdollisimman hyvin itse ohjelmiston määrittelyyn ja toteutukseen. Käyttäjien vastuulle annettiin uusien käyttäjien koulutukset ja opastukset sekä käyttäjäpalautteen keruu. Käyttäjien vaihtuessa osa tietotaidosta saattaa kadota, joten tulevaisuudessa voi olla tarpeen palkata yksi henkilö ylläpidon vastuuhenkilöksi.

Asiasanat
ohjelmistokehitys, organisaatiot, vapaaehtoistyö, yleishyödylliset yhteisöt

CENTRAL OSTROBOTHNIA
UNIVERSITY OF APPLIED SCIENCES

ABSTRACT

CENTRAL OSTROBOTHNIA UNIVERSITY OF APPLIED SCIENCES Kokkola	**Date** April 2011	**Author** Janne Olli
Degree programme Degree programme of information technology		
Name of thesis Controlled software developing in non-profit voluntary work		
Instructor Sakari Männistö		**Pages** 53
Supervisor Jaakko Tahkola		

The Central Committee of Conservative Laestadian Congregations needed various software and solutions to help its mission work. Voluntary based software development work had met problems mostly due to work organisation and project management. The purpose of this thesis was to find solutions for those problems.

This thesis describes defects and problems that were observed in the structure of the organisation and software development work. Those defects and problems were compared with generally known attributes and problems in this area of study. The structure of the organisation and software development model was based on this data. These were easy to adapt onto the current organisation and development model, and it was to support the voluntary based software development work as far as possible. The identification of suitable models was a challenge because software development work was not a part of the main activities of the association. During the thesis work various models were evaluated and tested against the course of action in which members of organisation can participate and invest only a few hours per week. The selected models were also tested in a situation where the requirements of software are changing fast and the turnover of developers is high.

During the thesis work, the model for the organisation was based on a focused matrix organisation, where programming work was allocated according to the various reference groups of the association. The best outcome for software development work was achieved using the Agile software development model. The scrum process was established with two different sprints, where the character of voluntary work was best taken into account. The quality of software and the work load of the software project can be better evaluated with these new models. Maintenance work, which is part of the software development process, was allocated to users of the software. This helped developers to focus their work better than earlier. The tasks of the users included collecting user feedback and training new users for the software. In the future, some know-how may be lost due to the turnover of users, so it may be necessary to hire one person as the status manager for software maintenance.

Key words
non-profit, organisation, software development, voluntary work

TIIVISTELMÄ
ABSTRACT
SISÄLLYS

KUVIOT

1 JOHDANTO

Tämän päivän ohjelmistokehitys on jatkuvan muutospaineen alla. Ohjelmistot ovat siirtyneet internetiin ja sitä kautta sekä kehittäjät että käyttäjät ovat hajautuneet ympäri maapallon. Globalisaation ja tekniikan kehitys tekee ohjelmistoista päivä päivältä monimutkaisempia, ja kuitenkin samaan aikaan vaaditaan nopeampaa kehittämistä entistä halvemmalla. Tämän päivän ohjelmisto koostuu usein eri palasista, joista osa voi olla kaupallisia valmiita ohjelmia tai ohjelman osia, osa on rakennettu jonkin pienen paikallisen tiimin toimesta ja osa työstä on saatettu tehdä hajautetusti ympäri maailman.

Internetin käytön yleistyessä myös yhä useammat yhdistykset ja vapaaehtoisjärjestötkin haluavat siirtää toimintaansa enenevässä määrin internetiin toivoen tavoittavansa sitä kautta helpommin ja paremmin jäsenensä, samalla tavoin ajattelevat ihmiset ja toiset yhteisöt. Tietotekniikan yleistyessä monen yhdistyksen ja järjestön jäsenistöstä löytyy myös ohjelmistoalan osaajia. Näiden osaajien innoittamana monessa järjestössä on kehitetty ja on lähdetty kehittämään sovelluksia, joiden tarkoituksena on helpottaa päivittäistä järjestötyötä. Järjestöissä käytettävät sovellukset ovat usein olleet pieniä, muutaman miehen talkootyönä tehtyjä ohjelmia, www-sivuja ja lomakkeita. Tietoteknisen tuntemuksen kasvaessa on kuitenkin syntynyt tarve myös järjestöpuolella yhä monimutkaisemmille sovelluksille.

Ohjelmistokehitystyöhön ja sen etenemisen seurantaan on rakennettu useita erilaisia prosessimalleja. Mallit ohjeistavat ohjelmiston kehitystyötä siten, että se pysyisi sovitussa aikataulussa ja talousarviossa. Prosessimallit antavat myös työkaluja kehitystyön tuloksena syntyvä ohjelmiston hyvyyden varmistamiseen siten, että ohjelmisto täyttäisi tietyt ennalta sovitut toiminnalliset ja laadulliset vaatimukset. Laajan ohjelmiston hajautettu kehitystyö on aina haaste. Jos ohjelmistokehitys toteutetaan talkootyönä oman työn ohessa erittäin pienellä työpanoksella, on sen hallittu läpivienti monesti vielä tavallista suurempi haaste. Tässä opinnäytetyössä tutustutaan ohjelmistokehitystyöhön talkootyön näkökulmasta. Talkootyönä toteutettaessa ohjelmiston kehittämiseen käytetty viikoittainen työmäärä on usein erittäin vähäistä ja kehittäjät saattavat vaihtua moneen otteeseen kesken projektin esimerkiksi kehittäjien oman elämäntilanteen tai palkkatyötilanteen vuoksi. Toisaalta kehittäjät

saattavat tuntea toisensa järjestönsä kautta erittäin hyvin, mikä helpottaa yhteistyötä projektin sisällä. Kehittäjät, jotka tekevät työtä vapaaehtoisesti talkoovoimin, ovat yleensä erittäin motivoituneita ja joskus jopa liiankin innokkaita ohjelmoimaan.

Tämän opinnäytetyön lähtökohtana on suomalaisen uskonnollisen yhdistyksen ja sen amerikkalaisen sisarjärjestön järjestötoiminnassa tarvittavien ohjelmien talkoopohjaisessa kehitystyössä havaitut ongelmat. Samat ongelmat tulevat vastaan usein missä tahansa ohjelmistoprojektissa, jossa kehitystyöltä puuttuu selkeät toimintatavat. Tässä opinnäytetyössä perehdytään ongelmien ratkaisuihin ja etsitään ohjelmiston kehittämiseen mallia, joka toimisi hyvin myös hajautetussa talkootyössä. Ohjelmistokehitys toteutettuna hajautettuna talkootyönä on kehitystyötä ympäristössä, jossa yksittäiset ohjelmoijat ovat ympäri maailmaa ja pystyvät osallistumaan ohjelmointityöhön vain muutaman tunnin ajan viikossa ja joiden osaamistaso ja päivittäin ansiotyössä tai opiskelussa käyttämät työkalut saattavat olla täysin erilaiset. Kehitysmalli, joka toimisi talkootyössä, toimisi varmasti missä tahansa ohjelmistoprojektissa, jossa työ tehdään hajautetusti eritasoisten ohjelmoijien toimesta.

Tämän opinnäytetyön kirjoittaja on ollut mukana erilaisissa ohjelmistoprojekteissa sekä projektin jäsenenä, ohjelmoijana että projektin vetäjänä yli 15 vuoden ajan, ja hänelle on kertynyt näiden projektien kautta kokemusta niin huonoista kuin hyvistäkin ohjelmistoprojekteista. Työtehtävien lisäksi hän on ollut mukana järjestöjen talkoovetoisissa ohjelmointiprojekteissa niin ohjelmoijana kuin vastuunkantajanakin. Vuosien kokemus on tuonut esille sen, että vaikka ohjelmiston kehitysmalli olisi kuinka hyvä tahansa, ei se toimi, jos sitä ei käytännössä toteuteta. Oleellista on myös löytää ne työkalut, joilla hyvät toimintatavat saadaan siirrettyä helposti aina edellisestä projektista seuraavaan projektiin. Myös tähän pyritään löytämään vastauksia tässä lopputyössä.

Aihe tähän opinnäytetyöhön nousi esille, kun pyrittiin löytämään ratkaisua talkoopohjaisessa ohjelmointikehitystyössä ilmenneisiin haasteisiin. Näihin haasteisiin syvennytään todellisen taustakertomuksen avulla luvussa kaksi. Luvuissa kolme ja neljä haetaan sopivia organisaatio- ja ohjelmistokehitysmalleja, jotka toimisivat myös yhdistyspohjaisessa vapaaehtoistyössä. Viimeisessä luvussa kootaan löydetyt ideat yhteen, kokeillaan niitä käytännössä sekä pohditaan ratkaisujen toimivuutta tulevien haasteiden edessä.

2 TALKOOPOHJAISEN OHJELMISTOKEHITYSTYÖN HAASTEET

Ohjelmistokehitystyön ei tarvitse muuttua millään tavalla sen mukaan tehdäänkö se talkoilla vai saadaanko siitä palkkaa. Talkootyössä palkan ohessa muuttuu usein myös moni muu reunaehto, jotka ovat vaikuttamassa kehitysympäristöön ja kehitysmenetelmiin. Ohjelmiston toteuttaminen ympäristössä, jossa jokainen ohjelmoija tekee työnsä vapaaehtoisesti oman kokemuksen ohjaamana, kohtaa monenlaisia haasteita. Haasteet muodostuvat hyvin nopeasti ongelmaksi, jos niitä ei tunnisteta ajoissa tai ne jätetään huomiotta. Suomen Rauhanyhdistysten Keskusyhdistyksen (SRK) talkoilla toimiva ohjelmointitiimi kohtasi omissa projekteissaan useita ohjelmistokehityksen ja projektinhallinnan yleisesti tunnettuja perusongelmia. Steve McConnell kuvaa hyvin selkeästi nämä perusongelmat omassa kirjassaan. Näitä perusongelmia, joita voidaan kutsua myös klassisiksi virheiksi, ovat muun muassa riittämätön suunnittelu, yksittäisten ohjelmoijien sankariteot, riittämätön riskien hallinta sekä käyttäjäpalautteen puute (McConnell 2002, 29–50). Haasteelliseksi ohjelmistokehitystyön teki myös se, että jokaisen tiimin jäsenen tausta, ansiotyössään käyttämät työkalut ja menetelmät olivat erilaiset.

Tässä luvussa tutustutaan talkoopohjaisen ohjelmistokehityksen haasteisiin todellisen tapauksen avulla. Alaluvussa 2.1 tuodaan taustakertomuksen avulla esille niitä ohjelmistokehitystyöhön liittyviä ongelmia, joihin on törmätty SRK:n ohjelmistoprojekteissa ja joihin törmätään missä tahansa huonosti johdetussa ohjelmistoprojektissa. Taustakertomuksessa esille nousevien ongelmien yhteisenä tekijänä voidaan pitää vahvan ja selkeän organisaation puutetta, jota käsitellään tarkemmin alaluvussa 2.2. Heikko organisaatio näkyy myös itse ohjelmistokehitysmallin puutteena. Alalukuun 2.3 on poimittu talkoopohjaisessa ohjelmistotyössä havaitut ohjelmistokehitystyön hallintaan liittyvät ongelmat.

2.1 Taustalla todellinen yhdistys -case

Kun innokkaat ohjelmoijat tapaavat toisensa vapaa-ajallaan, syntyy usein hullujakin ohjelmointiin liittyviä ideoita. Näin kävi myös kesällä 2004, kun useampi ohjelmointia työkseen ja harrastuksekseen tekevä nuori mies tapasi toisensa Suomen Rauhanyhdistysten

Keskusyhdistyksen (SRK) järjestämissä Suviseuroissa (KUVIO 1). SRK on Suomen evankelis-luterilaisen kirkon alla toimiva vanhoillislestadiolainen herätysliike, ja Suviseurat on sen jokavuotinen nelipäiväinen kesäjuhla, johon osallistuu noin 70 000 seuravierasta (Suomen Rauhanyhdistysten keskusyhdistys ry. 2010; Suviseurat 2010). Suviseuroilla olivat omat internetsivut olleet jo muutaman vuoden käytössä, mutta niiden graafinen ja tekninen taso vaihteli sen mukaan, millaista osaamista oli kyseisen vuoden seuraorganisaatio saanut hankittua omalta alueeltaan. Tämä nousi puheeksi alan osaajien kesken kahvitauolla ja esille tuli idea lähteä rakentamaan toimivampia www-sivuja talkoilla. Alkuun oli tarkoitus kehittää vain www-sivuja ja siihen läheisesti liittyviä toimintoja, kuten kuvagalleriaa ja nettiradiota. Koska innokkuutta riitti ja talkooryhmän osaaminen tuli tietoon yhdistyksen sisällä, syntyi uusia tarpeita ja ideoita.

KUVIO 1. Yleiskuva SRK:n suviseuroista (Suviseurat 2010)

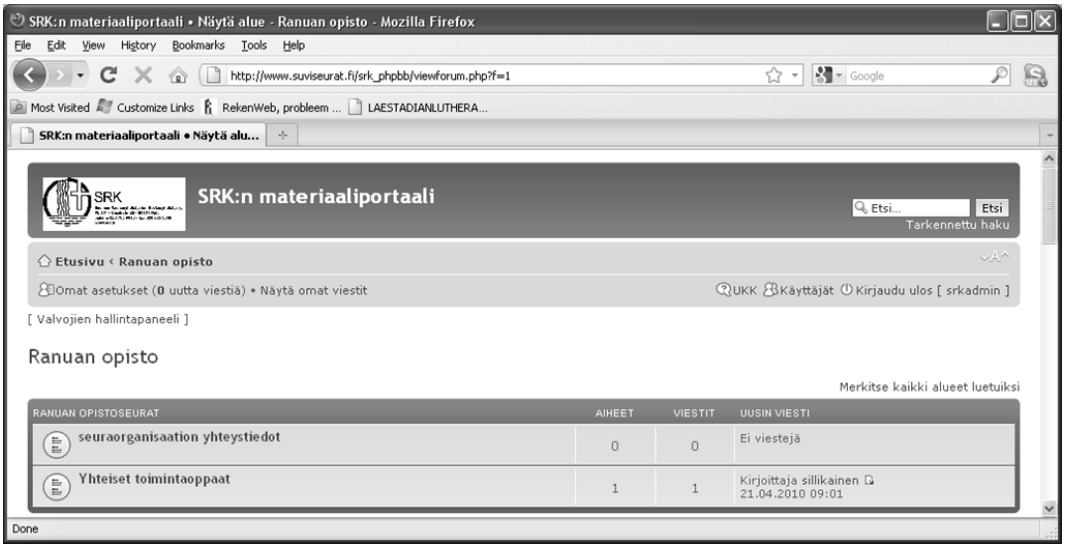
2.1.1 Innostuksen paloa

Uudessa talkootiimissä innokkuutta riitti, mutta koska kaikki olivat enemmän tai vähemmän ohjelmoijia, unohtui projektin kokonaisvaltainen ohjaus kokonaan. Kenellekään ei tullut projektinhallinta tai ohjelmiston määrittely mieleen, vaikka jokainen oli tottunut käyttämään niitä työkaluina omassa ansiotyössään. Kyseessähän oli vain vapaaehtoinen talkootyö, ja tuntui tärkeältä päästä tekemään jotain kivaa. Aikataulut venyivät, ja seuraavan kesän Suviseurat lähestyivät. Vasta kahta viikkoa ennen varsinaista seuratapahtumaa

yksi tiimiläisistä sai hankittua uuden palvelimen. Palvelimelle asennettiin Linux-ympäristö, ja se konfiguroitiin valmiiksi viikossa. Viikkoa ennen seurojen alkua Suviseuroille oli saatu uusi palvelin järeän verkkoyhteyden taakse, mutta siitä puuttuivat vielä www-sivut. Moni tiimiläinen oli ajatellut mielessään, että helppoahan se on siirtää vanhasta ja vähän muokata. Järjestelmä oli sen verran erilainen, ettei siirto onnistunut järkevästi, ja siksi päätettiin rakentaa sivusto uudestaan. Viimeistään tässä vaiheessa projektin vetäjä, jos sellainen olisi ollut, olisi varmasti keskeyttänyt työn ja käskenyt käyttää edellisen vuoden ongelmallista, mutta valmista järjestelmää. Koska tiimissä oli vain innokkaita ohjelmoijia, päätettiin rakentaa koko www-sivusto yhden viikonlopun aikana, viikkoa ennen varsinaista seuratapahtumaa. Tiimi kokoontui koko juhannusviikonlopuksi yhden ohjelmoijan kotiin, ja projekti, jos sitä sellaiseksi voi nimittää, puristettiin valmiiksi kolmen vuorokauden yhtäjaksoisella ohjelmointityöllä.

Tiimi oli onnistunut tuottamaan jotakin sellaista, joka näytti hyvältä. Seuraorganisaatio oli suorastaan mykistynyt tiimin taidoista, ja seuraavan kesän järjestäjät ehdottivat uusia toimintoja www-sivuille. Samalla toivottiin jotain internetin yli toimivaa arkistointijärjestelmää seuraorganisaatiossa tarvittavien dokumenttien ja pöytäkirjojen tallentamiseen. Tiimiläiset lupautuivat uusiin haasteisiin, vaikka jokainen heistä tiesi, että kiireessä kokoon kyhätty järjestelmä sisälsi monia ongelmakohtia, jotka olisivat suorastaan pommeja ylläpidon kannalta. Aikaa näytti olevan melkein vuosi, ja sen uskottiin riittävän vanhan järjestelmän parantamiseen ja uusien ohjelman osien rakentamiseen. Juhannuksesta viisastuneena päätettiin tiimin toimintaa hieman organisoida paremmin ja jakaa tehtävät tiimin sisällä muutaman hengen pienryhmille. Ryhmäyttäminen näytti toimivan, sillä pienryhmät syntyivät automaattisesti parhaimpien ystävien välille – yhteistyö ja kommunikointi tehostuivat. Yksi ryhmä keskittyi uudistamaan www-sivut ja toinen ryhmä miettimään järjestelmää dokumenttien tallentamiseen. Suviseurojen internetsivustosta ja sen päivitystyön haasteista seurojen aikana oli jo kokemusta, joten jokaisella ryhmän jäsenellä oli vaatimusmäärittely jo olemassa. Vaatimusmäärittelyä ei ollut kirjattu yhteisesti paperilla, vaan se oli tallennettuna jokaisen muistiin jokaisen henkilökohtaisen kokemuksen kautta. Verkkosivuprojekti eteni hienosti, ja sivut toimivat loistavasti seuraavan kesän tapahtumassa. Myös sivujen seurojen aikana tapahtuvaan päivitykseen oli saatu rakennettua työkalut, joiden helppous mahdollisti päivitystyön siirron itse seuraorganisaatiolle. Myös toinen pienryhmä onnistui tehtävässään - dokumenttien tallennus toteutettiin hyödyntämällä val-

mista keskustelufoorumin koodia (KUVIO 2). Valmis internetistä löydetty asennuspaketti oli nopea asentaa palvelimelle, ja samalla todettiin ylläpidon helpottuvan – tarvitsee vain ladata internetistä aina uusin koodiversio edellisen päälle. Ohjelmoijien tarvitsi ainoastaan keskittyä piilottamaan turhat ominaisuudet pois näkyvistä ja tehdä pieniä lisäyksiä joihinkin koodiosioihin. Sekä www-sivusto, että tämä uusi dokumentointiportaali saivat kiitosta.



KUVIO 2. Dokumenttiportaalissa hyödynnettiin keskustelufoorumin ohjelmakoodia

2.1.2 Ongelmien kasautuminen

Koska sovellusten määrä kuten myös käyttäjien määrä olivat kasvaneet, olivat talkootiimin taidot tulleet useamman tietoisuuteen. Uusia ideoita ja pyyntöjä alkoi sadella kehittäjille. Kaikessa työssä motivaatio on asia, joka vie työtä eteenpäin. Yleensä työtehtävät, joissa työntekijä pääse käyttämään omia taitojaan mahdollisimman tehokkaasti samalla oppien jotain uutta, motivoivat eniten. Tällaisessa innostuneessa ilmapiirissä uusia ideoita syntyy ja niitä lähdetään toteuttamaan vauhdilla. Kaikenlaiset muissa yhteyksissä hyväksi havaitut kehitysprosessit ja projektinhallinta tuntuvat vain jarruttavilta tekijöiltä. Tosiasiassa juuri tällaisessa tilanteessa tarvittaisiin selkeää prosessia ja tiukkaa projektinhallintaa. Talkootiimin jäsenet toteuttivat ideoita sitä mukaa, kun niitä tuli sähköpostilaatikkoon. Valmis kaaos oli syntymässä. Suviseurojen seuraorganisaatio vaihtuu joka vuosi, ja suviseurat pidetään joka vuosi eri paikassa. Seuraavan vuoden organisaation pyynnöt ja tarpeet saattoivat olla ristiriidassa edellisen vuoden organisaation kanssa tehtyjen muutosten kanssa.

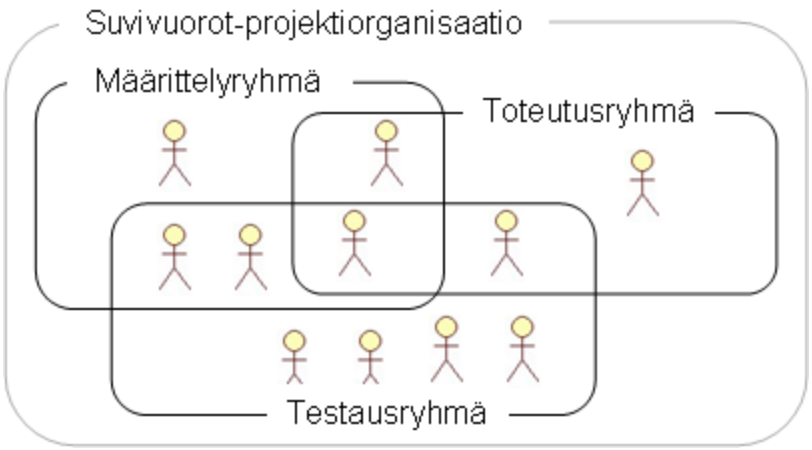
Tiimi oli kuin tuuliajolla – sännättiin aina sinne, mistä eniten huudettiin apua. Ongelmat kasautuivat, ja samalla talkootuntien määrä kasvoi. Tiimiläisten silmistä alkoi innostuksen palo hiipua – hauska talkootyö oli muuttunut puurtamiseksi.

Kehitystyö oli edennyt tähän asti valtavalla innolla, ja jokainen oli saanut käyttää juuri niitä menetelmiä, joita oli oppinut käyttämään työelämässä. Mitään yhteistä toimintatapaa, ohjelmakoodin kommentointiohjetta tai ohjelmointikieltä ei ollut päätetty. Osa internetpalveluista oli rakennettu useammalla ohjelmointikielellä, koska osa ohjelmoijista oli mieltynyt PHP-ohjelmointikieleen, kun taas toiset eivät luopuneet Perl-ohjelmointikielestä. Osa käytti kommentointiin ja muuttujien nimiin suomen kieltä, kun taas toiset tekivät samat asiat englannin kielellä. Kaikesta huolimatta järjestelmät olivat pysyneet – kumma kyllä – pystyssä. Ylläpitovaikeuksia ilmeni sitä mukaa, kun jokin tiimiläisistä ei jaksanut tai työkiireiden takia ehtinyt enää osallistua ohjelmointiin ja hänen osuus siirrettiin jollekin toiselle jo tiimissä olleelle tai uudelle innokkaalle talkootyöntekijälle. Koska mitään dokumentaatiota ohjelman rakenteesta tai edes vaatimuksista ei ollut olemassa, oli uuden ohjelmoijan opastaminen haasteellista. Ohjelmistojen käyttäjien ja seuraorganisaation näkökulmasta työ oli kuitenkin edennyt edelleen hienosti, ja siksi uusien ideoiden määrä vain kasvoi. Harrastuksekseen ja työkseen ohjelmoinnin parissa työskentelevät tiimiläiset eivät osanneet sanoa ”ei”, vaan yrittivät selvitä kaaoksesta tekemällä työtä entistä enemmän.

2.1.3 Kaaoksesta projektiksi

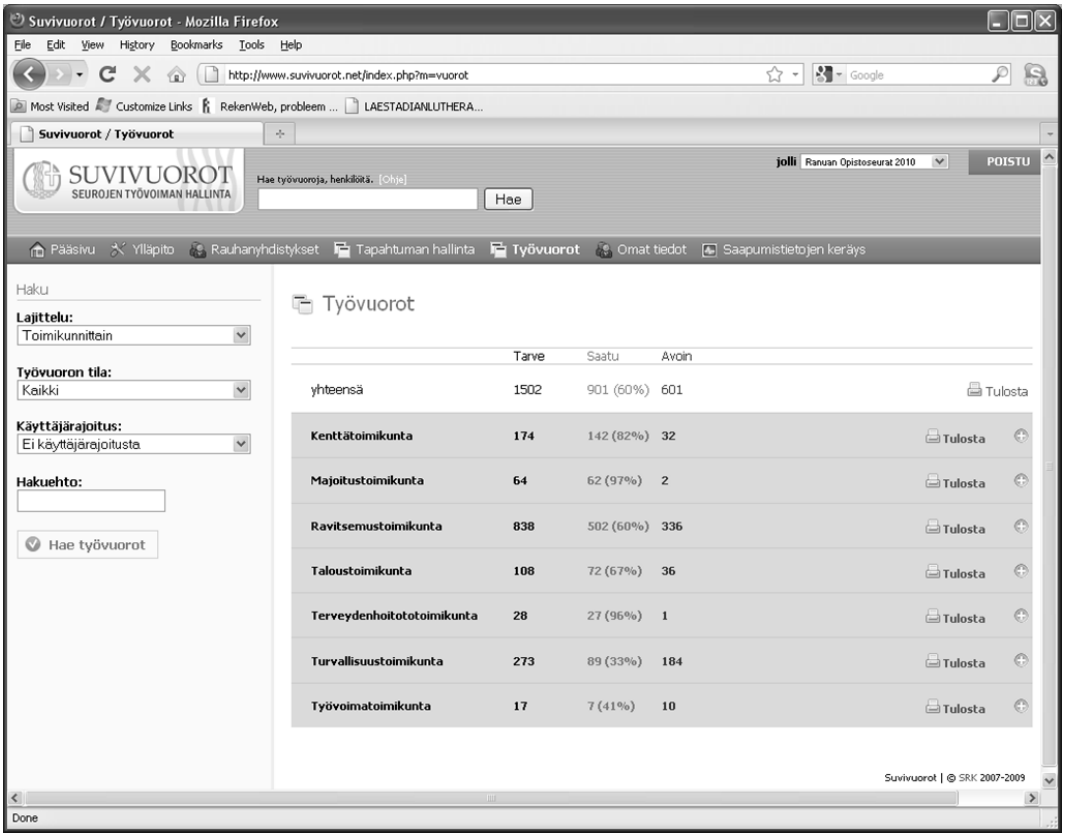
Kesken pahimman kaaoksen tiimiläisiltä kyseltiin mahdollisuutta lähteä rakentamaan internetsovellusta, jolla voisi pitää hallinnassa kaikki Suviseuroissa tarvittavat talkootyövuorot. Suviseurojen aikana tehdään joka vuosi noin 8500 talkootyövuoroa. Järjestelmään tulisi pystyä syöttämään kaikki tehtävät ja työvuorot, jakaa ne paikallisille yhdistyksille rekrytointia varten sopivan algoritmin mukaisesti ja tallentaa paikallisyhdistyksistä saatujen työntekijöiden yhteystiedot. Järjestelmän kehitystyön todettiin olevan mielenkiintoinen mutta erittäin haasteellinen. Onneksi myös suviseuraorganisaatio oli ymmärtänyt järjestelmän haasteellisuuden ja osasi ehdottaa kunnollista määrittelyvaihetta ja väljää aikataulua. Ohjelmoijien oli helppo hyväksyä tämä ja suorastaan vaatia määrittelyä kaiken sen kaaoksen jälkeen.

Suviseuroissa seuraorganisaation ydinjoukko nimetään noin kaksi vuotta ennen varsinaisia Suviseuroja. Tästä vasta nimetystä joukosta valittiin vastuuhenkilöitä mukaan määrittelemään uutta Suvivuorot-työvuorojärjestelmää ohjelmointitiimin jäsenten tueksi (KUVIO 3). Tavoitteena oli saada ensimmäinen versio ohjelmasta käyttöön vajaassa kahdessa vuodessa kesäksi 2008. Seuraorganisaation kautta saatiin määrittelyyn mukaan myös muutama ohjelmistoalan konkari sekä koko projektille oma vetäjä. Projektin vetäjän päävastuuna oli määrittelyvaiheen ja järjestelmän käyttöliittymän suunnitteluvaiheen ohjaus ja aikataulutus. Itse ohjelmoinnin aikataulutus jäi ohjelmoijille itselleen. Lähtöasetelma vaikutti hyvältä, ja tiimiläisten oli helppo nyökytellä päätään. Projektiorganisaatiolle avattiin oma blogi-sivusto, jossa määrittelyssä mukana olevat ja ohjelmoijat vaihtoivat ajatuksia. Projekti eteni mallikkaasti ja vauhdilla. Projektin kehitysmalliksi muotoutui spiraali-kehitysmalli. Spiraalimallissa projekti pilkotaan osiin ja lähdetään liikkeelle yhdestä tehtävästä tai riskistä ja kun se on saatu suoritettua tai selvitettyä, otetaan mukaan seuraava tehtävä tai riski kasvattaen näin tuote valmiiksi vaihe vaiheelta. Spiraalimalli tuntui luontevimmalta valinnalta kehitystiimissä, jonka jokainen jäsen asui eri puolella Suomea. Malli mahdollisti kehitystyön etenemisen prototyypistä toiseen. Ensimmäisen prototyypin kehitysvaiheen aikana tiimi kokoontui yhteen useamman kerran määritelläkseen ohjelmiston toiminnan pääsuuntaviivat. Ohjelmoijat tekivät näiden kokoontumisien pohjalta ensimmäisen vaiheen prototyypin internetiin omalle kehityspalvelimelle. Jo projektin alkuvaiheessa oli valittu ne henkilöt, joiden tuli testata kehitysvaiheen aikana syntyviä versioita internetissä mahdollisimman paljon, ja kaikki palaute tuli raportoida blogi-sivustolle (KUVIO 3). Palautteen pohjalta ohjelmoijien oli helppo edetä omassa työssään oman talkootyölle sopivan aikataulun mukaisesti.



KUVIO 3. Suvivuorot-projektin organisaatio

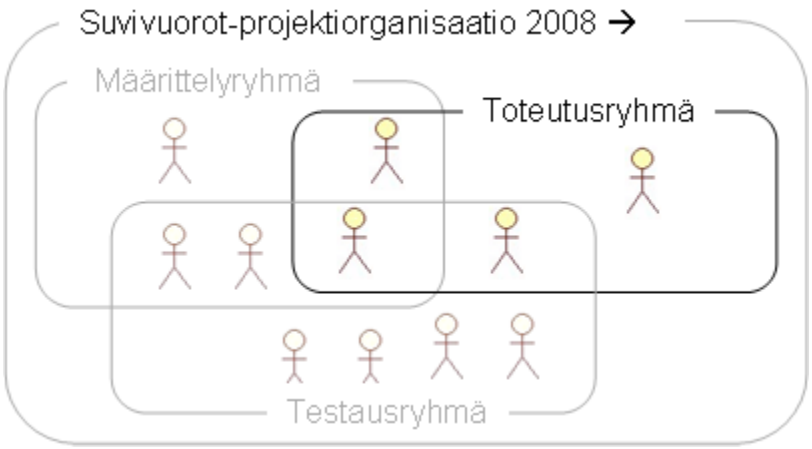
Suvivuorot-ohjelmiston ensimmäinen versio saatiin käyttöön kesällä 2008 suunnitellussa aikataulussa mutta hieman karsittuna (KUVIO 4). Tässä projektissa, kuten liian usein missä tahansa projektissa, alussa löysäksi todettu aikataulu muuttui tiukaksi ellei mahdottomaksi projektin lopussa. Aikataulussa ollut löysä osuus tuhlattiin alussa, koska aikataulun tiedettiin olevan väljä ja kiirettä ei pitänyt olla. Lopussa tekemättömien tehtävien lista oli kuitenkin niin pitkä, että listassa olevien tehtävien suorittaminen vaaditussa aikataulussa talkootyöllä tuntui mahdottomalta. Määrittelyvaiheessa ohjelmistoon oli listattu myös sellaisia ominaisuuksia, jotka eivät olleet välttämättömiä varsinkaan tässä ensimmäisessä versiossa. Ensimmäisen version avulla oli tarkoitus testata myös tulevan käyttäjäkunnan kykyä oppia tekemään aikaisemmin osin manuaalisesti tehty työ ohjelmallisesti internetin kautta. Jotta ohjelmisto saatiin pysymään aikataulussa, toteutettiin ensimmäiseen versioon vain välttämättömimmät ominaisuudet. Ohjelmiston käyttäjät olivat tyytyväisiä, ja niin oli koko projektitiimi. Tästä olisi hyvä jatkaa kehitystyötä yhteistuumin.



KUVIO 4. Suvivuorot.net järjestelmä toiminnassa

2.1.4 Päätös organisoitua

Suvivuorot-ohjelmiston kehitystyössä ja määrittelyssä alun perin mukana olleet, vuoden 2008 seuraorganisaation vastuuhenkilöt, totesivat pian ensimmäisen version käyttöönoton jälkeen, että heidän työpanoksensa on ollut riittävä ja määrittelytyötä on hyvä jatkaa seuraavan kesän seuraorganisaation kanssa (KUVIO 5). Ohjelmointitiimi lähti rakentamaan yhteistyötä uuden organisaation kanssa. Keskusteluyhteys saatiin rakennettua, mutta ongelmaksi nousi uuden seuraorganisaation sitoutuneisuuden puute. Ensimmäinen versio oli saatu toimimaan ja sillä oli pärjätty yksi kesä, joten sen ominaisuudet riittäisivät ehkä seuraavinakin kesinä. Uudella seuraorganisaatiolla ei ollut motivaatiota lähteä oman vastuullisen talkootehtävänsä ohella lukemaan edellisen organisaation aikana syntynyttä vaatimusmäärittelylistaa ja mahdollisia reunaehtoja. Uusi organisaatio lähti määrittelemään uusia ominaisuuksia omien tarpeiden mukaisesti. Osa uusista ominaisuuksista oli ristiriidassa alkuperäisen määritelmän kanssa vain siksi, että alkuperäinen määritelmä oli tehty palvelemaan kaikkia seuraorganisaatioita usean vuoden ajan. Uudella organisaatiolla oli kuitenkin näköpiirissä vain omat seuraavan kesän seurat. Ohjelmointitiimi joutui torjumaan monet uudet ideat, sillä ne olisivat tehneet järjestelmästä tapahtumariippuvaisen ja siten toimineet vain yhden kesän. Torjuminen johti lopulta siihen, että ohjelmointitiimi joutui lähes yksin jatkamaan työtä ilman seuraorganisaation tukea. Tiimin jäsenet pyrkivät omaehtoisesti tekemään valmiiksi ne tehtävät, jotka olivat jääneet toteuttamatta ensimmäisessä versiossa. Vähäisen henkilömäärän vuoksi jouduttiin hylkäämään määrittely ja testausvaiheet (KUVIO 5). Seuraava versio saatiin valmiiksi kesäksi 2009, ja sen toiminnallisuus vastasi lähestulkoon alkuperäistä tavoitetta. Koska alkuperäinen määrittelytiimi oli vaihtunut, ei ollut ketään, joka olisi virallisesti sanonut, että projekti on valmis ja voidaan siirtää ylläpitovaiheeseen.



KUVIO 5. Projektiorganisaation kuihtuminen julkistuksen jälkeen

Ohjelmistoon kuin ohjelmistoon tulee käytön aikana enemmän tai vähemmän virheitä. Suurin osa virhetilanteista saadaan pysäytettyä kenties ohjelman sisään rakennetun virheenkäsittelyrutiinin ansioista, mutta osa saattaa tulla yllättäen tilanteessa, jota ei ole pystytty ennalta estämään. Kun puhutaan internetin yli toimivasta sovelluksesta, jonka käyttäjäkunta vaihtuu ja jonka käyttäjien tietotekninen osaaminen vaihtelee, käyttäjistä johtuvien virheiden määrä moninkertaistuu. Läheskään aina ne virheet eivät johdu itse sovelluksesta vaan esimerkiksi selaimen käyttötaidoista, käyttäjän verkkoyhteydestä tai muista ohjelmisto- ja laitteistovioista ja ristiriidoista. Koska virhe ilmaantuu juuri silloin, kun käyttäjä on käyttämässä jotain tiettyä sovellusta, on käyttäjän mielestä syy automaattisesti kyseisen ohjelmiston. Yleensä tällaisten käytöstä johtuvien ongelmien ja käytössä havaittujen ohjelmistovirheiden korjaamiseen on rakennettu asiakaspalvelu- tai ylläpitotiimi. Suvivuorot-ohjelmiston kehitystiimi oli ottanut projektin alussa huomioon määrittelyvaiheen, mutta oli unohtanut yhden tärkeimmistä: projektin päätösvaiheen ja sitä seuraavan ylläpitovaiheen. Koska ohjelmointitiimi oli pysynyt lähes samana usean vuoden ja projektin ajan, oli kaikilla tiedossa heidän yhteystiedot. Lähes kaikissa ongelmatilanteissa otettiin yhteyttä suoraan ohjelmoijiin, koska seuraorganisaatio vaihtui aina vuosittain. Ohjelmointitiimin joutui vastaamaan puhelimitse ja sähköpostitse erilaisiin ongelmiin. Talkootyöhön käytettävissä olevien tuntien määrä tuhlautui tehtäviin, jotka olivat lähinnä asiakaspalvelu- ja ylläpitotehtäviä. Uusien ominaisuuksien toteutukseen ei jäänyt aikaa. Myöskään mahdollisiin uusiin projekteihin ei ollut aikaa tarttua. Ohjelmointitiimi tuskastui työkuormaansa ja lähti etsimään vaihtoehtoja erilaisista projektinkehitys- ja organisaatiomalleista.

2.1.5 Yhteistyötä globaalisti

Tieto Suomessa talkootyöllä kehitetyistä ja kehitteillä olevista ohjelmistoista levisi vuoden 2009 aikana SRK:n henkilökunnan kautta Amerikassa toimivan SRK:n sisarjärjestön tietoon. Laestadian Lutheran Church (LLC) toimii Pohjois-Amerikassa, ja sen toiminta on hyvin pitkälle samanlaista kuin sen sisarjärjestön toiminta Suomessa (Laestadian Lutheran Church 2009). LLC järjestää mm. erilaisia isoja seuratapahtumia omalla toiminta-alueellaan. LLC:llä oli omat www-sivut, mutta sitä oli alkanut kiinnostaa internetin parempi hyödyntäminen omassa toiminnassaan. Internetin laajenemisen myötä myös yhä useampi LLC:n jäsen käytti internetiä päivittäin. Koska LLC:n ja SRK:n toiminta on hyvin samanlaista ja koska jo tehtyä työtä ei kannata tehdä moneen kertaan uudestaan, nousi ajatus yhteistyöstä. Yhteistyötä puolsi myös se, että SRK:lla oli jo useampi talkoo-ohjelmoija ja jäsenmäärältään monin verroin suuremman SRK:n jäsenistöstä olisi helpompi löytää tarvittaessa lisää ohjelmointialan osaajia.

Ohjelmointitiimi oli aloittanut pohdinnat paremmasta organisoitumisesta ja paremmasta projektinhallinnasta hetkeä aikaisemmin. Globaalin näkökulman tuominen mukaan keskusteluihin toi mukanaan selkeämmän päämäärän. Tavoitteeksi tuli löytää sellaiset menetelmät, jotka toimisivat globaalisti eritasoisten ohjelmoijien ja muun projektiorganisaation välillä. Ajateltiin, että nyt käytössä olevista epämääräisistä menetelmistä, jotka toimisivat vain yhden yhteenkasvaneen tiimin sisällä, olisi päästävä eroon.

2.2 Selkeän organisaation puute

Jokaisessa yrityksessä ja yhdistyksessä on oma organisaatio. Organisaatiot pyritään rakentamaan tai ne ajan kuluessa rakentuvat siten, että ne tukisivat mahdollisimman hyvin yrityksen tai yhdistyksen päätoimialaa. Ohjelmistoalan yrityksissä organisaatio on pyritty rakentamaan siten, että se tukisi itse ohjelmistokehitystyötä. Myös SRK:lla on oma organisaatio, mutta koska yhdistys on uskonnollinen herätysliike, se on rakennettu palvelemaan mahdollisimman hyvin jäseniään hengellisistä lähtökohdista katsoen. SRK:n ohjelmointitiimin tekemä ohjelmistotyö on talkoilla toteutettua tukitoimintaa, ja siksi siihen työhön ei ole omaa selkeää organisaatiota. Toki jokaiselle ohjelmointiprojektille rakennetaan oma

projektiorganisaatio, mutta tämän organisaation olemassaolo päättyy aina projektin päättyessä.

Yritysmaailmassa projektin päättyessä projektiorganisaation henkilöt siirtyvät yleensä takaisin yrityksen perusorganisaation alaisuuteen ja projektituotos, esimerkiksi uusi ohjelmisto, siirtyy ylläpitovaiheeseen. Ylläpitovaihe tarkoittaa tässä kaikkea sitä, mitä ohjelmalle tehdään sen elinaikana: virheiden korjausta, markkinointia, käyttäjäpalautteen keruuta, uusien ohjelmaversioiden tekoa ja koulutusta. Yleensä ylläpitovaiheen hoitaa siihen erikseen nimetty henkilö tai tiimi. Tämän henkilön tai tiimin tehtävänä on toimia rajapinnassa ohjelman käyttäjien suuntaan sekä käynnistää tarvittaessa uusi projekti ja projektiorganisaatio ohjelmiston muuttamiseen, päivittämiseen tai kouluttamiseen. Täysin samanlainen organisaatiomalli ja ylläpitotiimi voidaan rakentaa toki myös talkooympäristössä, mutta siinä on haasteena nimenomaan pysyvyys. Projektiorganisaatio sidotaan tehtävään vain projektin ajaksi, mutta ylläpidon pitäisi toimia koko ohjelmiston suunnitellun elinkaaren ajan. Organisaatiossa, jossa ohjelmointikehitystyö on vain tukitoimi, haetaan tekijät itse projektin toteuttamiseen, mutta ylläpidon organisointi usein unohtuu. Ylläpidolle ei vain ole huomattu sitoa henkilöä tai tiimiä. Kun projektin tuloksena syntyneeseen ohjelmistoon tulee vikoja, ei löydykään osoitetta, mihin vikaraportit voisi lähettää. Jos käyttäjät eivät saa vastauksia omiin kysymyksiinsä tai korjauksia havaittuihin ongelmiin, he pyrkivät ohittamaan tämä ongelman. Yleensä se tarkoittaa vaihtoehtoisten ohjelmien tai menetelmien käyttöä. Ohjelmiston elinikä jää lyhyeksi siinä havaittujen virheiden tai puutteiden vuoksi. Eliniän loppumiseen vaikuttaa toki myös se, että käyttäjien tarpeet ja tekniikan vaatimukset muuttuvat hyvin nopeasti. Tämä pätee eritoten internetissä pyöriville sovelluksille.

Usein käy niin, että ylläpito jää itse projektiorganisaation vastuulle. Tämä johtaa siihen, että projektiorganisaatio ei tiedä, milloin projekti oikein päättyy, ja he joutuvat sitoutumaan samaan projektiin kenties vuosiksi. Samalla hämärtyy se, missä vaiheessa muutokset olisivat niin suuria, että siitä pitäisi käynnistää uusi projekti uusine vaatimusmäärittelyineen ja aikatauluineen. Ja jos se huomattaisiinkin, niin projektiorganisaatiohan on jo koossa, eli miksi pitäisi erikseen aloittaa uusi projekti. Projekti jää kellumaan, ja se on muuttunut huomaamatta projektista hankkeeksi. Kaiken tämän lisäksi yleensä tulee tarvetta myös uusille projekteille. Työelämässä tällaisessa tilanteessa projektinhenkilöt huo-

maamattaan tekevät pitkiä työpäiviä. Talkootyössä, kuten myös SRK:n ohjelmointityössä, käytettävä työaika on rajallinen ja yleensä ansiotyö ja perhe menevät prioriteetiltaan sen ohi. Tulee hyvin nopeasti vastaan tilanne, että uusia projekteja ei voida ottaa enää tehtäväksi.

Niin kuin selkeä organisaatiomalli tuo tukea itse projektia ennen ja sen jälkeen tapahtuvaan toimintaan, samaa selkeyttä tarvitaan myös itse projektiorganisaatiossa projektin aikana. SRK:n ohjelmistotiimi muodostui osaajien ympärille. Tiimi lähti kehittämään ohjelmistoja yhtenä tiiminä ilman erillistä projektipäällikköä tai selkeää organisaatiota. Kaikki halusivat ohjelmoida. Työ eteni Suviseurojen seuraorganisaatiosta tulleiden toiveiden pohjalta yksittäisten tiimiläisten kautta. Kenelläkään ei ollut oikein selkeää kokonaiskuvaa, ja kehitystyö poukkoili. Jos tiimi olisi tehnyt kehitystyötä koko ajan yhdessä samassa tilassa, olisi projekti saattanut jopa edetä ilman projektipäällikköä. Suuren osan ohjelmoinnista tiimiläiset tekivät kuitenkin yksin kotonaan eri puolilla Suomea, ja tämä aiheutti usein ristiriitoja niin määrittelyssä, jos sitä ylipäätänsä oli, kuin myös toteutuksessa. Projekti tarvitsee toteutuakseen projektiorganisaation ja projektiorganisaatio vetäjäkseen projektipäällikön samalla tavoin kuin laiva tarvitsee eteenpäin päästäkseen merimiehet ja kapteenin.

2.3 Ohjelmistokehitysmallin puute

Ohjelmiston kehitystyö lähtee aina liikkeelle ideasta. Jostakin tulee idea ja tarve uudelle ohjelmistolle. Ideasta keskustellaan tai sitä pohditaan mielessä enempi tai vähempi. Tätä vaihetta voidaan nimittää ohjelmiston määrittelyksi. Kun ideaa on riittävästi kypsytelty toisin sanoen uutta tulevaa ohjelmistoa on riittävästi määritelty, alkaa ohjelmointivaihe, jonka päätepisteenä on uuden ohjelmiston valmistuminen. Ohjelmiston kehitystyö kuvataan usein näin lyhyesti avaamatta sen enempää määrittely-, ohjelmointi- tai julkaisuvaihetta. Tämä aiheuttaa harhakuvitelman työn helppoudesta. Valitettavan monen projektin kohdalla sen kuvitellaan olevan näin helppoa, mutta usein niiden projektien tuotoksena syntyneiden ohjelmistojen elinikä jää lyhyeksi.

Määrittelyvaihe on ohjelmistokehityksen tärkein kulmakivi. Jos määrittelyvaihe puuttuu ja siinä syntyneet määrittelydokumentit puuttuvat, ei jälkeenpäin voida sanoa, toimiiko ohjelmisto siten, kuten sen suunniteltiin toimivan. Pienissä projekteissa määrittelynä saattaa toimia esimerkiksi sähköposti, mutta vain harvoin onnistutaan yhteen sähköpostiin samalla kertaa listaamaan kaikki ohjelmiston toimintaan liittyvät määritelmät ja rajoitukset. Määrittelyvaihe jää usein olemattomaksi projekteissa, jotka luokitellaan jo ideavaiheessa ”pikkuhommaksi” tai joiden on tarkoitus palvella erilaisia organisaation sisäisiä toimintoja. Usein näiden ”pikkuhommien” ja sisäisten projektien ympärille ei rakenneta kunnollista projektiorganisaatiota eli vaikkapa sellaista, jossa yksi henkilö on selkeästi vastuussa kokonaisuudesta, myös määrittelystä. Myös erittäin mielenkiintoinen projekti, projektijäsenten valtava halu päästä toteuttamaan ideaa käytännössä, voi saada kaikenlaiset määrittelytyöt tuntumaan turhilta hidasteilta. Tosiasia on kuitenkin se, että vaatimusten saaminen kohdalleen jo määrittelyvaiheessa voi olla jopa 50–200 kertaa edullisempaa kuin se, että niitä täydennetään vielä ohjelmiston ylläpitovaiheessa (McConnell 2002, 62).

SRK:n ohjelmointitiimistä ei yleensä ole innokkuutta puuttunut, ja koska työ tehdään vapaa-ajalla talkootyönä, on kaikki käytettävissä oleva aika yleensä pantu itse toteutusvaiheen eteenpäin viemiseen. Tämä muodostui ongelmaksi monen projektin kohdalla. Ohjelmakoodia kirjoitettiin heti, kun joku keksi senhetkiseen ongelmaan ratkaisun, ja seuraavaa vaihetta tehdessä huomattiin, ettei edellinen ratkaisu toimikaan seuraavan vaatimuksen kanssa. Ohjelmakoodia työstettiin ja ohjelmointivirheitä korjattiin vuoron perään, ja kun saatiin toimiva koodi, saatettiin todeta, ettei se ehkä vastannutkaan käyttäjän tarpeita. Juostiin kuin suunnistaja ilman karttaa ja kompassia. Määrittely tulisikin selkeästi sisällyttää omana vaiheena jokaiseen projektiin. Huolella tehty määrittely helpottaa työtä myöhemmin, sillä määrittelyvaiheessa syntyneet dokumentit toimivat usein pohjana ohjelmiston käyttöohjeelle ja kaikelle muullekin dokumentaatiolle.

Määrittelyvaihetta seuraa tai sen kanssa hallitusti rinnalla tulee itse toteutusvaihe. Toteutusvaiheeseen yleensä sisällytetään suunnittelu, ohjelmointi ja testaus. Suunnittelun tarkoituksena on tarkentaa määrittelyä itse ohjelmointia varten. Tässä vaiheessa muun muassa valitaan toteutettava ohjelmointikieli ja arkkitehtuuri. Ohjelmointi ja testaus kulkevat yleensä rinnakkain, eli samalla kun ohjelmoija saa jonkin ohjelmapätkän tehtyä, hän testaa, että kaikki toimii tässä kyseisessä ohjelman pätkässä. Ohjelmointivaiheen lopuksi kaikki

erilliset ohjelmakoodin palaset yhdistetään ja suoritetaan kokonaistestaus. Kokonaistestauksessa testataan ensin ohjelman tekninen toimivuus ja sen jälkeen verrataan sen toiminnallisuutta projektin alussa tehtyyn määrittelyyn. Jos toteutus ja määrittely kohtaavat, voidaan sanoa, että ohjelmisto toimii kuten sen pitikin toimia. Jos määrittely on jäänyt tekemättä, voidaan korkeintaan havaita ohjelmatekniset virheet eli ne virheet, jotka kaatavat kenties koko järjestelmän tai antavat virheilmoituksen. Jos määrittely puuttuu tai jos määrittelyä muutetaan toteutusvaiheen aikana, sillä on suora vaikutus toteutusvaiheen venymiseen. Jotain valmista saadaan aikaan, mutta se ei ehkä ollutkaan sitä, mitä toivottiin, ja niin tehdään taas muutoksia muutoksien perään. Samalla potentiaalisten virheiden määrä ohjelmakoodissa kasvaa. Tämä voi lopulta johtaa siihen, että toivottuja muutoksia ei voida nykyiseen ohjelmaan enää toteuttaa ja kaikki on aloitettava alusta, puhtaalta pöydältä.

SRK:n ohjelmointitiimin työmäärä on usein kuormittunut aivan turhaan liiasta innosta päästä ohjelmoimaan – tekemään jotain näkyvää. Määrittelyvaihe on puuttunut lähes tyystin, ja toteutusvaiheessa jokainen on edennyt niillä menetelmillä, joita omassa ansiotyössään on kenties tottunut käyttämään. Välttämättä ei edes ohjelmointikielen valintaa ole tehty yhdessä. Ohjelmistokomponentit on saatettu tehdä eri ohjelmointikielellä sen mukaan, kuka ohjelmointitiimistä on sen ottanut vastuulleen. Sinänsä toteutus useammalla kielellä ei ole ongelma, sillä yleensä jo käyttöympäristö tukee monikielisyyttä. On hyvin tavallista, että isoissa ohjelmistoissa tärkeimmät rutiinit toteutetaan mahdollisesti jollain toimintavarmemmalla ja nopeammalla alemman tason ohjelmointikielellä kuin toiset vähemmän kriittiset osiot. Alemman tason kielillä tarkoitetaan ohjelmointikieliä, jotka toimivat hyvin lähellä laitteiston ydintä (esimerkiksi tietokoneen prosessori) ja jotka eivät tarvitse toimiakseen tulkkia prosessorin konekielen ja kyseisen ohjelmointikielen välillä. Mutta se, että samanarvoiset ohjelmakoodin palaset ohjelmoidaan eri ohjelmointikieltä käyttäen, tekee ylläpitotyön yleensä hankalaksi. Koska projektin hallinta on ollut heikkoa, on myös projektin päättäminen hieman hämärän peitossa. Projekti luisuu huomaamatta ylläpitovaiheeseen. Ohjelmoijat eivät pysty sanomaan, milloin ohjelma on sellainen kuin sen haluttiin olevan, koska määrittely puuttuu yleensä kokonaan. Ja vaikka projekti saataisiinkin selkeästi päättymään, niin ei tiedetä, kuka hoitaa ylläpidon. Taustalta puuttuu selkeä organisaatio, jonka kautta voisi projektin tuotoksen, valmiin ohjelmiston siirtää ylläpitotiimille tai ohjelmistoa ylläpitävälle henkilölle. Toki ylläpitotiimi voitaisiin rakentaa

talkoopohjaisena kuten ohjelmointitiimikin, mutta yhteinen linkki näiden tiimien välillä on määrittelemättä.

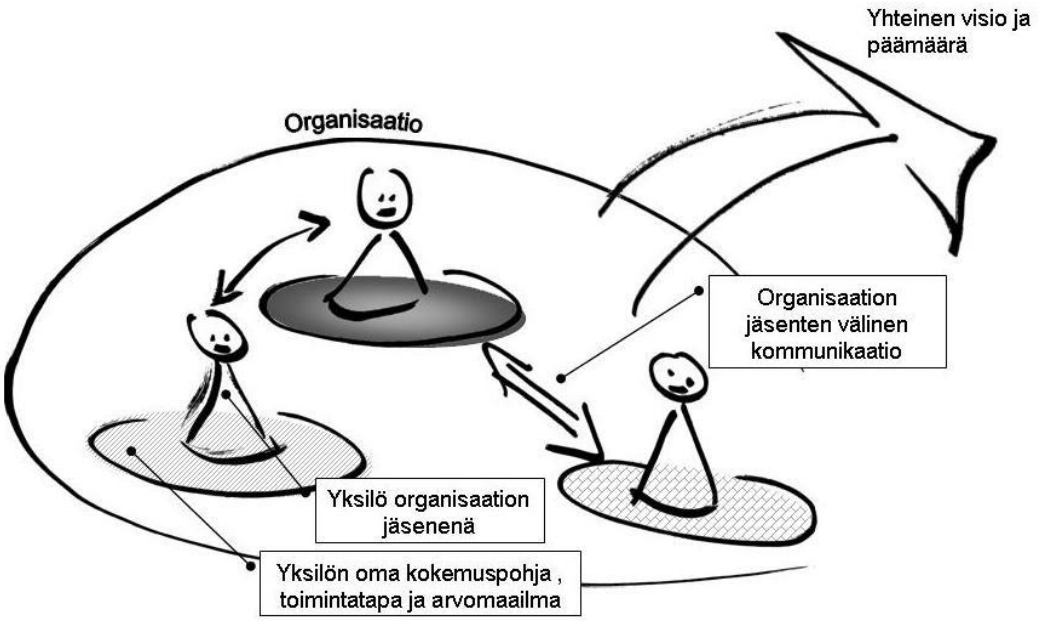
Lähes joka projektissa tulee loppua kohti kiire. Koska usein testaus on viimeisten työvaiheiden joukossa ennen ohjelmiston käyttöönottoa ja projektin päätöstä, on se usein samalla se työvaihe, josta säästetään. Ohjelmisto saatetaan julkaista heti, kun kaikki ohjelmapalaset on saatu valmiiksi ja testattua yksittäin, ennen kokonaistestausta. Kokonaistestaus jätetään käyttäjien tehtäväksi. Tämä aiheuttaa yleensä enempi harmia kuin se, että projektia olisi hieman pidennetty. Käyttäjä rakentaa ensimmäisen mielikuvan ohjelmistosta heti ensimmäisellä käyttökerralla, ja tätä ensimmäistä mielikuvaa on erittäin vaikea muuttaa myöhemmin. Toki osa kokonaistestauksesta voidaan jättää käyttäjien vastuulle, esimerkiksi uudenlaisen käyttöliittymän asiakaspalautteen keräämiseen, mutta tällöin yleensä käytetään rajattua käyttäjäkuntaa, niin kutsuttuja beta-testaajia.

Suviseurojen seuraorganisaatiolle tehtävien ohjelmistojen testaus on jäänyt usein käyttäjien vastuulle kokonaan. Huonon projektijohtamisen vuoksi aikataulut yleensä käyvät loppua kohden tiukoiksi. Suviseurojen aikataulu on ja pysyy ennalta sovittuna, ja ainut mahdollisuus on kiristää aikataulua karsimalla testauksesta. Testaus joudutaan usein tekemään puutteellisesti, vaikka aikataulu sen sallisi, sillä Suviseurojen aikana sovelluksen käyttäjiä voi olla päivän aikana tuhansia, ja tuollaisen samanaikaisen käyttäjämäärän testaaminen etukäteen on lähes mahdotonta. Tämä on riski, joka tulisi aina ottaa huomioon rakentamalla varajärjestelmiä tai kehittämällä testausmenetelmiä.

3 ORGANISAATIOMALLIT

Jokaisella, joka on työntekijänä tai jäsenenä jossain yrityksessä tai yhteisössä, on jonkinlainen käsitys kyseisen yrityksen tai yhteisön organisaatiosta. Organisaatio on tietyn periaatteen mukaisesti järjestetty henkilöstöryhmittymä (Rissanen, Sääski & Vornanen 1996, 12–13). Järjestäytymisperiaate eli organisaation hierarkia voi olla rakentunut pitkän historian kuluessa, tai se on voitu rakentaa tietoisesti jonkin tietyn mallin mukaisesti. Yritysmaailmassa taloudellinen kannattavuus on yleensä kaiken ”A ja O”, ja siksi myös organisaatio pyritään rakentamaan sellaiseksi, että se toimisi mahdollisimman tehokkaasti. Organisaatioista keskusteltaessa käytetään usein adjektiiveja raskas, kankea, joustava ja tehokas, jotka viittaavat jollain tavalla organisaation kykyyn vastata taloudellisiin haasteisiin. Yrityksen organisaatiomuoto ja se, kuinka tämä organisaatiomuoto suhtautuu asiakkaisiin ja henkilöstöön, on osaltaan ratkaisemassa yrityksen taloudellisen menestyksen (Lipiäinen 2000, 344).

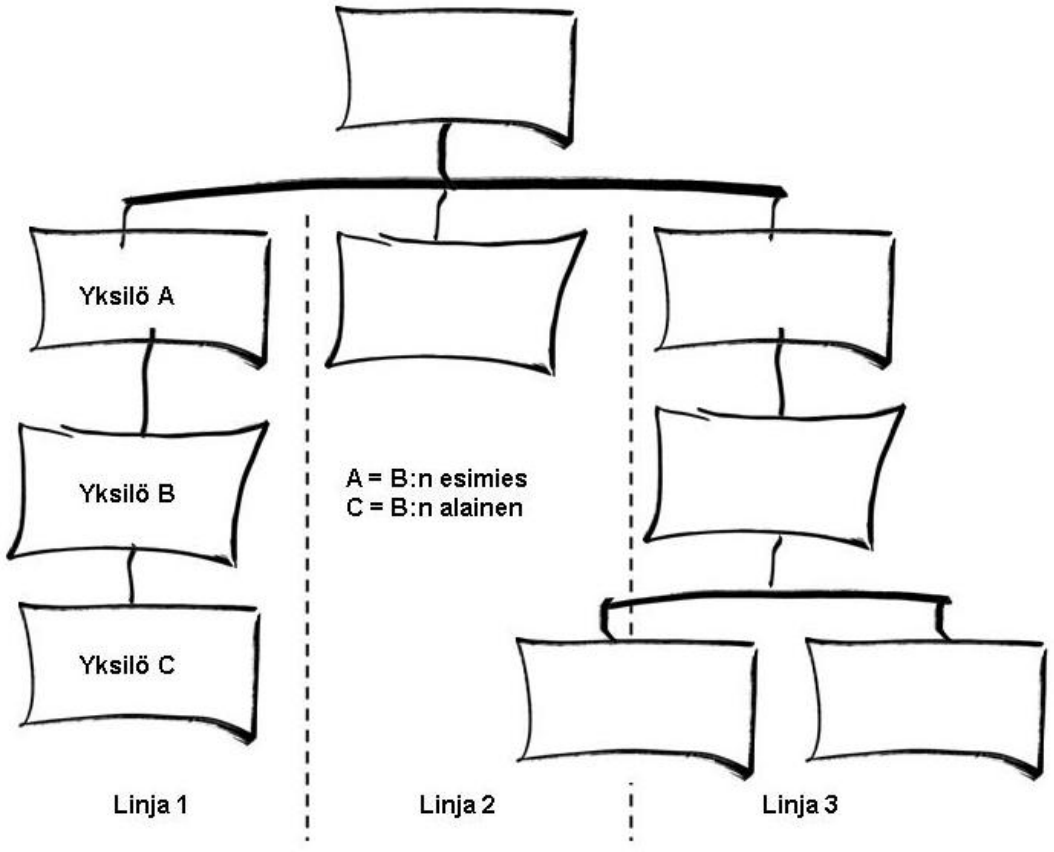
Organisaatio rakentuu yksilöistä, jotka ovat vuorovaikutuksessa keskenään omalla toimintatavallaan, kokemuspohjallaan ja arvomaailmallaan. Kuviossa 6 on näkyvissä, kuinka erilaiset yksilöt voivat muodostaa organisaation, kunhan heillä on jokin yhteinen tavoite. Organisaation jäsenten välisestä vuorovaikutuksesta (KUVIO 6) syntyy yritykselle kulttuuri eli sille ominainen tapa toimia (Kookas 2010). Toiminta voi olla jonkin piirretyn kaavion mukaista, jolloin puhutaan virallisesta organisaatiosta (Rissanen ym. 1996, 22–33). Organisaatiorakenteen mukainen toiminta vaatii organisaation yksilöiltä joustavuutta ja kykyä toimia erilaisissa rooleissa kuten esimiehen ja asiantuntijan rooleissa. Erilaisia organisaatiorakenteita ovat esimerkiksi linjaorganisaatio, linja-esikuntaorganisaatio, toimintokohtainen organisaatio (funktionaalinen organisaatio), projektiorganisaatio ja matriisiorganisaatio (Hokkanen & Strömberg 2003, 57–61).



KUVIO 6. Organisaatio rakentuu yksilöistä, joilla on sama päämäärä

3.1 Linjaorganisaatio

Linjaorganisaatio on erilaisista organisaatiomalleista se kaikkein vanhin malli. Se on organisaation perusmalli, jonka lähtökohtana on suora ohjaus. Jokaisella organisaatiossa olevalla henkilöllä on ainoastaan yksi esimies, ja tieto ja toimintaohjeet tulevat tämän esimiehen kautta ylhäältä alaspäin. (KUVIO 7.) Tässä organisaatiomallissa yrityksen toimivalta on keskitetty ylimpään johtoon, ja siinä on olemassa selkeä hierarkia.

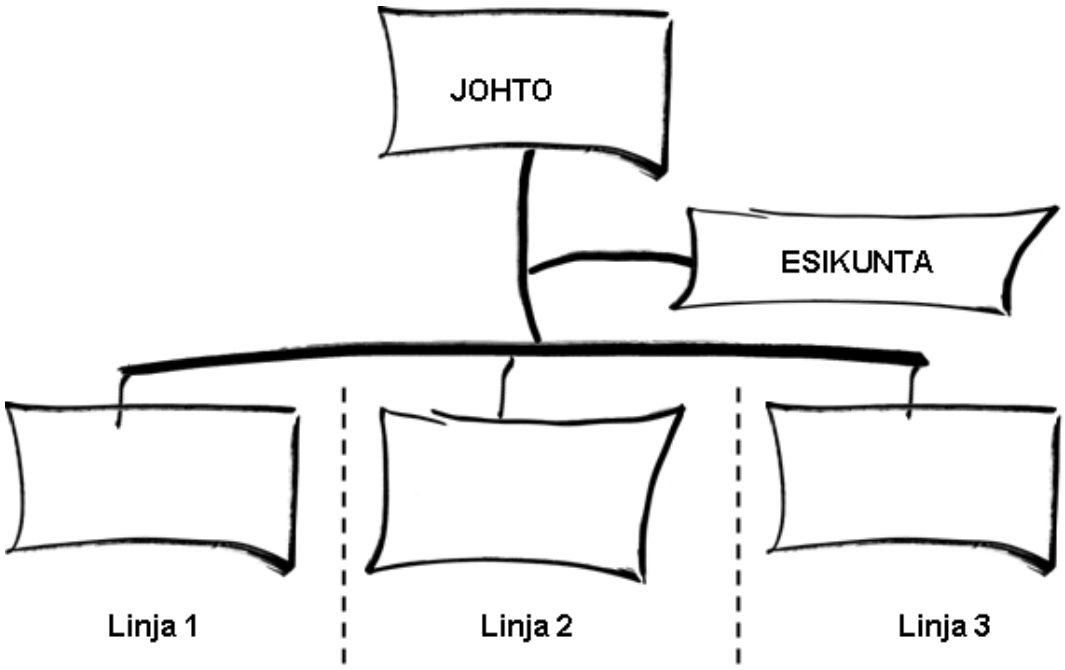


KUVIO 7. Linjaorganisaation hierarkia (mukaillen Rissanen ym. 1996, 24.)

Linjaorganisaatio on perinteisesti koettu hyvin kaavamaiseksi ja jäykäksi, mutta se kiehtoo, koska siinä valta ja vallan käyttö on keskitettyä. Organisaation koon kasvaessa, linjaorganisaatio tulee hyvin raskaaksi. Organisaation kokoa pyritään järkevöittämään yleensä muilla malleilla. (Rissanen ym. 1996, 24; Hokkanen & Strömberg 2003, 60.)

3.2 Linja-esikuntaorganisaatio

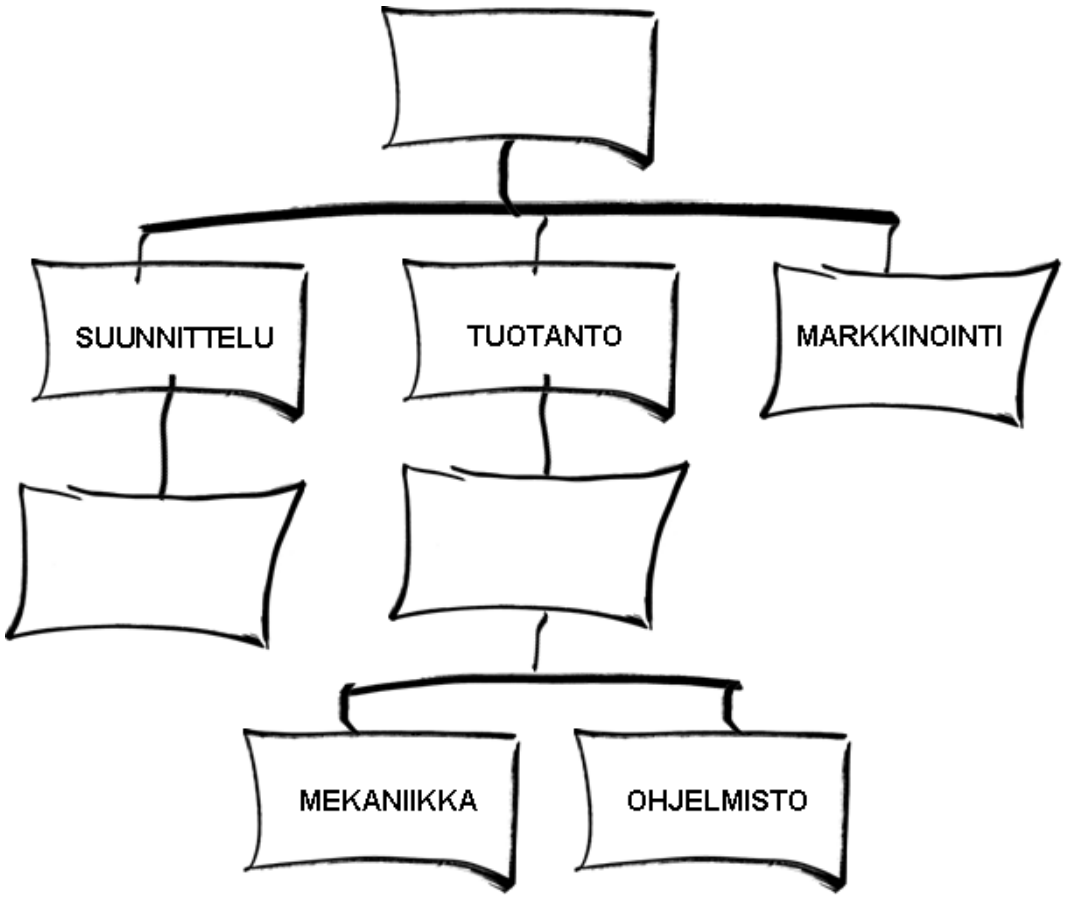
Linja-esikuntaorganisaatiomallissa perusmalliin on lisätty yksiköitä, jotka palvelevat asiantuntemuksellaan johtoa ja kaikkia muita yksiköitä. Nämä perusmalliin lisätyt yksiköt tarjoavat neuvoja ja palveluja muille organisaatiossa oleville, mutteivät itse käytä päätösvaltaa. (KUVIO 8.) Näitä yksiköitä ovat mm. palkkaosasto, talousosasto, henkilöstöpalvelut ja erilaiset asiantuntijapalvelut. (Rissanen ym. 1996, 25; Hokkanen & Strömberg 2003, 60.)



KUVIO 8. Linja-esikuntaorganisaation rakenne (mukaillen Rissanen ym. 1996, 25.)

3.3 Toimintokohtainen organisaatio

Toimintokohtaisessa organisaatiossa organisaation mallin lähtökohtana on perinteinen linjaorganisaatio. Linjaorganisaation esimies–alainen-rakenne on ryhmitelty siten, että organisaation jäsenet sijoittuvat organisaatiossa toiminnallisuuden ja työtehtävän perusteella omiksi ryhmiksi. Kuviossa 9 organisaatio on ryhmitelty suunnittelu-, markkinointi- ja tuotanto-osastoihin. Tarvittaessa osastot voivat jakautua useammaksi alaosastoksi, kuten tuotanto-osasto kuviossa 9, jossa tuotanto-osasto on jaettu mekaniikka- ja ohjelmisto-osastoiksi. Tässä organisaatiomallissa johtaja on suorassa yhteydessä eri osastojen päälliköihin, joiden tehtävänä on tuottaa ja kehittää omaa toimintokohtaista erikoisosaamista yhdessä oman osastonsa kanssa ja tarjota tätä erikoisosaamista koko organisaation käyttöön. Toimintokohtainen organisaatio, jota myös funktionaaliseksi organisaatioksi kutsutaan, on hyvin tyypillinen pienissä organisaatioissa. (Rissanen ym. 1996, 26; Hokkanen & Strömberg 2003, 60.)

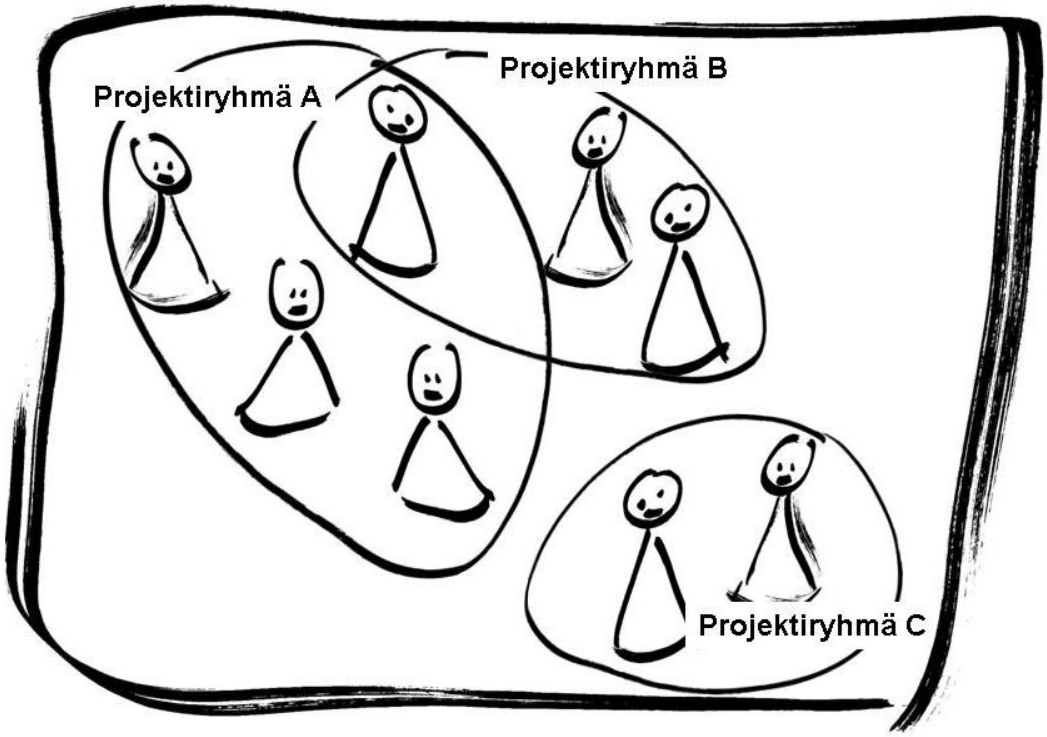


KUVIO 9. Toimintokohtainen organisaatio (mukaillen Rissanen ym. 1996, 26.)

3.4 Projektiorganisaatio

Projektiorganisaatio rakentuu nimensä mukaisesti jonkin tietyn uuden menetelmän, palvelun, tuotteen tai järjestelmän kehittämistyön ympärille. Kehittämistyötä, jolla on selkeä alku ja loppu, voidaan nimittää projektiksi. Projektin eteenpäin viemisestä vastaa projektia varten rakennettu kertakäyttöinen projektiryhmä (Hokkanen & Strömberg 2003, 61). Projektiryhmä voi olla laajuudeltaan minkä kokoinen tahansa, mutta sen koko on aina suhteessa projektituotoksen kokoon. Pienessä ohjelmistoprojektissa voi yksi mies olla riittävä, mutta esimerkiksi suuressa ydinvoimalaprojektissa projektihenkilöstö koostuu useiden eri aloilla toimivien yritysten työntekijöistä ja heitä voi olla satoja ellei tuhansia. Organisaatiota, joka toimii yksinomaan projektityöskentelyn periaatteiden mukaisesti ja jolla ei pysyviä organisaatiorakenteita juurikaan ole, voidaan nimittää puhtaaksi projektiorganisaatioksi.

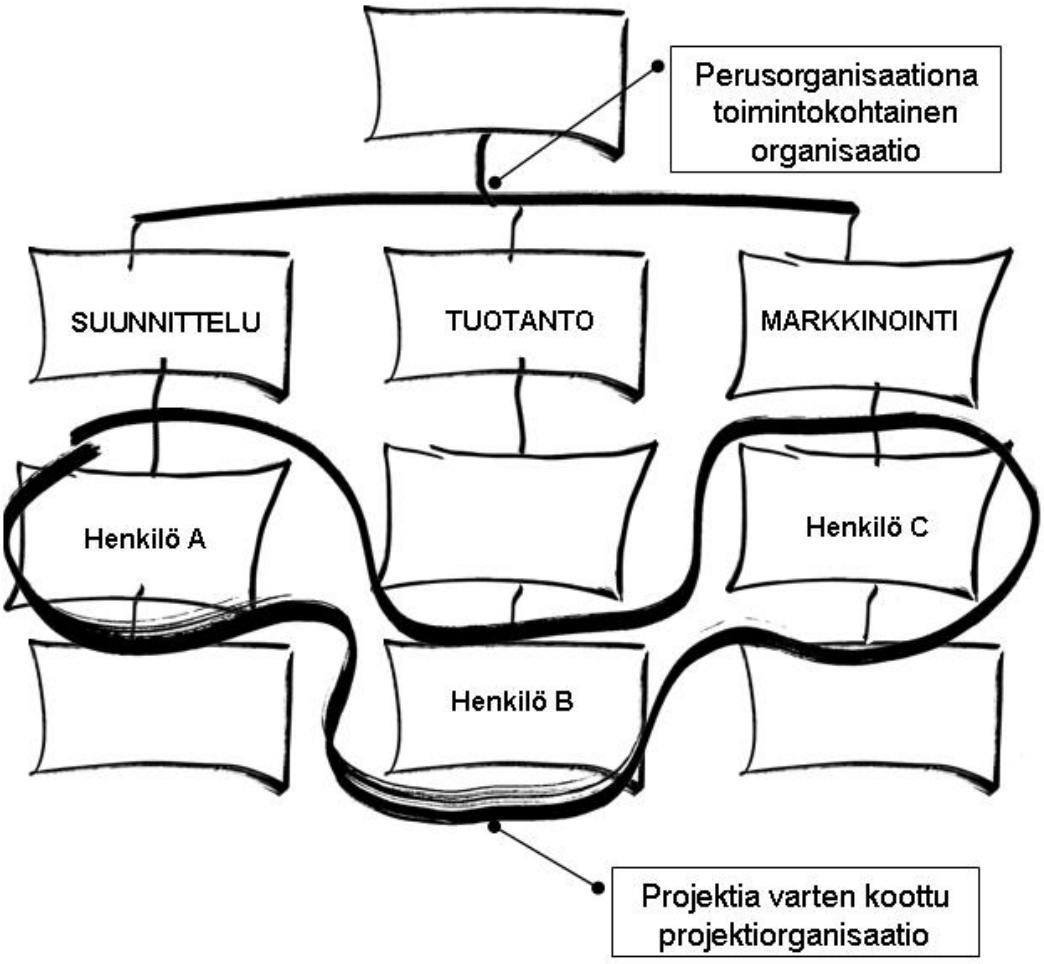
Puhtaassa projektiorganisaatiossa toiminnan edellytetään koostuvan kokonaan erillisistä projektitehtävistä, jotka eivät toistu samanlaisina (Rissanen ym. 1996, 27). Erilaiset asiakasprojektit ovat usein tällaisia projektitehtäviä. Projektin päättyessä projektissa mukana olleet henkilöt sidotaan johonkin toiseen projektiin. Jotta projektihenkilöillä olisi töitä tasaisesti, sijoitetaan heidät yleensä useampaan projektiin samalla kertaa siten, että yhteenlaskettu työmäärä on 100 prosenttia. (KUVIO 10.) Näin ollen yhden projektin päättyminen ei aiheuta täydellistä työttömyyttä. Puhtaassa projektiorganisaatiossa on kuviossa 10 esitettyjen projektiryhmien lisäksi vähintään yksi johtaja, jonka alaisuudessa kaikki nämä erilliset projektiryhmät toimivat. Yksittäinen johtaja tai organisaation johto määrittelee yhteisen vision ja päämäärän, joiden ympärille organisaatio rakentuu (KUVIO 6).



KUVIO 10. Puhdas projektiorganisaatio (mukaillen Rissanen ym. 1996, 27.)

Projektiorganisaatiossa tarvitaan projektin tuotoksen, laajuuden ja projektin tilan mukaan yleensä eri alan osaajia. Osaajien sijoittaminen projekteihin projektin luontivaiheessa ja projektituotosten ylläpito on oma työnsä, ja usein tämän työn helpottamiseksi projektien taustalla yrityksessä on olemassa pysyvämpi perusorganisaatio (Rissanen ym. 1996, 28). Tällöin projektiorganisaatio luodaan vain projektin ajaksi perusorganisaatiota täydentämään ja projektin tuotos siirtyy perusorganisaation käyttöön ja vastuulle projektin päättyessä. Pääsääntöisesti kaikki projektiin sidotut henkilöt ovat mukana myös perusorganisaa-

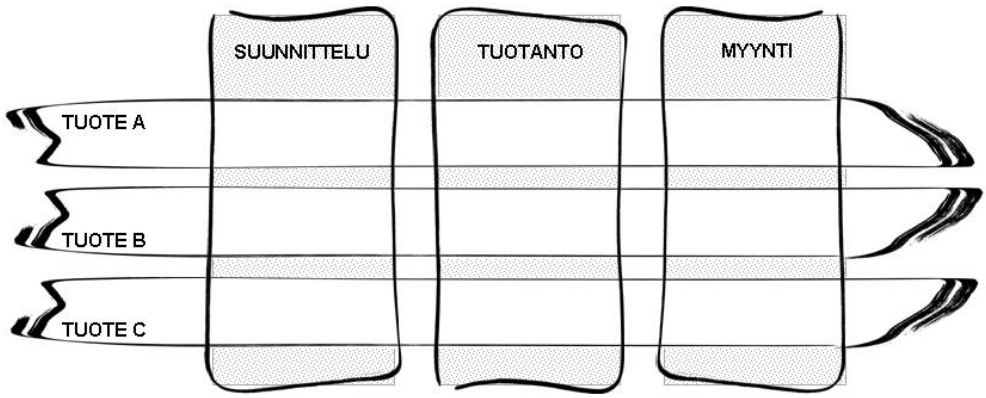
tiossa. Jos pääosa työstä tehdään projekteissa, on perusorganisaation rakenne yleensä muokattu sellaiseksi, että uuteen projektiin on helppo löytää sopivat osaajat. Esimerkiksi jos projekteissa tarvitaan aina suunnittelua, markkinointia ja itse suunnitellun tuotteen toteutus, sopisi perusorganisaation malliksi toimintokohtainen organisaatio. Organisaatio olisi osastoitu luonnollisesti suunnittelu-, markkinointi- ja tuotanto-osastoon. Uusi kolmen hengen projektiorganisaatio olisi helppo rakentaa sitomalla yksi henkilö suunnittelusta, toinen markkinoinnista ja kolmas tuotannosta. (KUVIO 11.) Projektiorganisaation toimivuus edellyttää osaavaa projektin ohjausta, josta yleensä vastaa projektipäällikkö (Rissanen ym. 1996, 28; Hokkanen & Strömberg 2003, 61). Jos projektin vastuut, valtuudet ja riittävät resurssit määritellään huonosti, perusorganisaation osastopäällikön toimintavalta-alue ja projektipäällikön toimintavalta-alue saattavat mennä päällekkäin. Tämä aiheuttaa helposti ristiriitatilanteita henkilöstön johtamisessa ja töiden priorisoinneissa.



KUVIO 11. Projektiorganisaation muodostaminen perusorganisaatiosta

3.5 Matriisiorganisaatio

Kun liikeideoiden ja toiminta-alueiden määrä ja erikoistarve lisääntyvät, linjaorganisaatio ei enää riitä. Matriisiorganisaatiossa erikoisosaaminen asetetaan palvelemaan kaikkia tulosalueita tai tulosyksiköitä (Hokkanen & Strömberg 2003, 61). Matriisiorganisaatio saa nimensä siitä, että yleensä se kuvataan siten, että erikoisosaaminen kulkee vaakarivillä kaikkien pystyyn sijoitettujen tulosalueiden yli. Syntyy matriisi, jonka eri soluissa olevat organisaation jäsenet ovat samanaikaisesti jonkin asiantuntijaryhmän tai linjan vetäjän alaisena ja samanaikaisesti jonkin tulosalueen vetäjän alaisena. (KUVIO 12.) Tämän lisäksi usein itse työ tehdään projekteissa, joilla on omat projektipäällikkönsä. Yhdistelmä tuottaa monijohtajutta, eli yhdellä toimijalla on kaksi tai useampi esimies. Monijohtajuus saattaa aiheuttaa vastuuristiriitoja, mutta toisaalta matriisiorganisaatiossa henkilön erikoisosaamista on mahdollista hallitusti kehittää projektien ulkopuolellakin (Rissanen ym. 1996, 30). Hankalimpia tilanteita ovat mm. kehityskeskustelut, joissa henkilön työsuoritusta arvioidaan ja osaamista kehitetään. Yleensä projektipäälliköt seuraavat päivittäin työntekijän työntekoa, mutta kuitenkin harvoin he ovat mukana kehityskeskusteluissa.

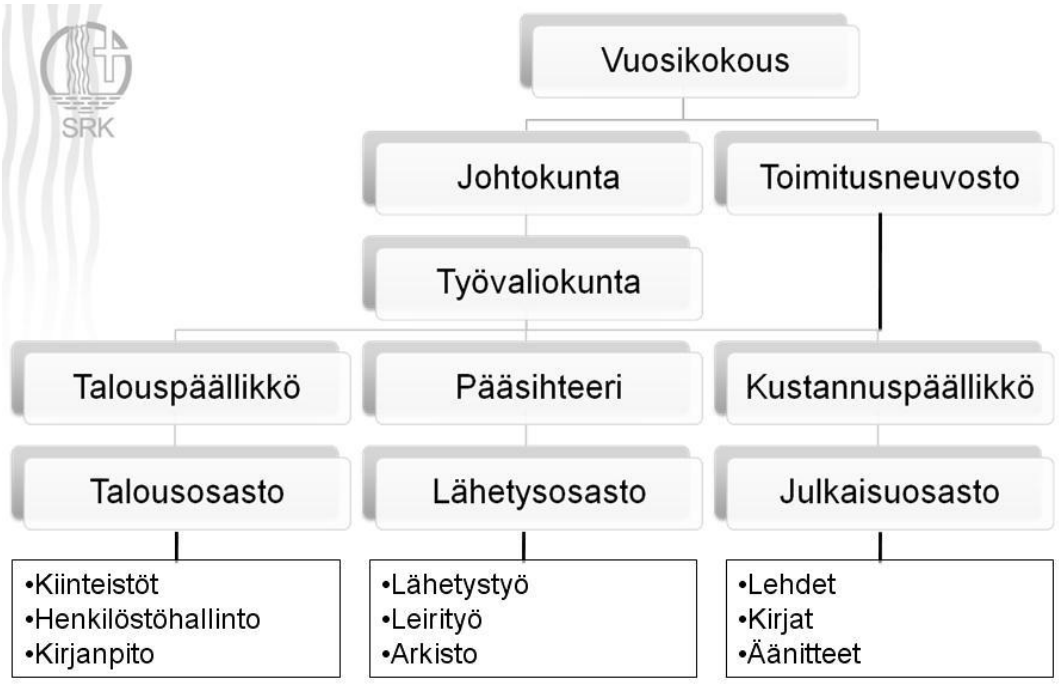


KUVIO 12. Matriisiorganisaation rakenne (mukaillen Rissanen ym. 1996, 29)

Matriisiorganisaatio on hyvin tyypillinen kansainvälisissä suuryrityksissä, joissa eri maissa on oma maajohtaja ja sen alla on linjaorganisaatio (Rissanen ym. 1996, 29). Kunkin maan paikalliset osastot kuuluvat yli maan rajojen ulottuvaan liiketoiminta-alueeseen, jolla on oma johto. Matriisiorganisaation vahvuus riippuu siitä, onko pääasiallinen johtamisvastuu linjaorganisaatiolla vai tulosalueella. Mitä enemmän vastuuta on annettu toiminnoille eli tulosalueille, sitä vahvempi on matriisi – puhutaan vahvasta matriisiorganisaatiosta.

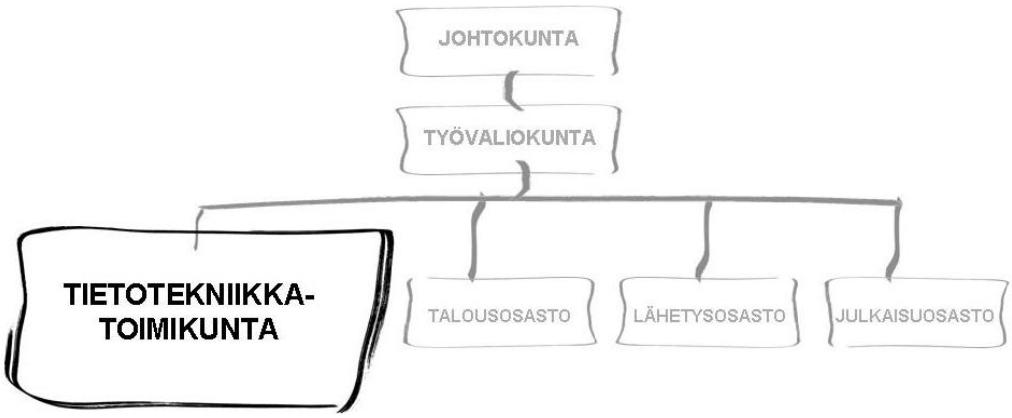
3.6 Talkootyöhön sopivan organisaatiomallin kehittäminen

SRK:n talkootyöllä toteutetuissa ohjelmistoprojekteissa suurimmaksi ongelmaksi nousi vahvan ja ohjelmistokehitystyötä tukevan organisaation puute. Myös ohjelmistokehitysmalli todettiin epäselväksi, mutta koska organisaation rakenne vaikuttaa usein myös ohjelmistokehitysmallin valintaan, lähdettiin ensimmäisessä vaiheessa rakentamaan sopivaa organisaatiota. SRK:n perusorganisaatioon ei ollut tarkoitus puuttua, vaan tavoitteena oli rakentaa sopiva organisaatio talkoilla toteutettavan ohjelmistokehityksen tukemiseen ja sellainen, joka kytkeytyisi parhaalla mahdollisella tavalla jo olemassa olevaan organisaatioon. SRK:n jo olemassa oleva organisaatio on tyypillinen yhdistyksien käyttämä organisaatio, joka muodostuu jäsenistön vuosikokouksessa valitsemasta johtokunnasta ja johtokunnan alaisuudessa toimivista toimikunnista ja muista toimielimistä. Johtokunnan alla on lisäksi pysyvä konttorihenkilöstö, joka huolehtii päivittäisistä rutiineista, kuten kirjanpidosta, SRK:n oman lehden, Päivämiehen, toimittamisesta ja arkiston hoitamisesta. SRK:n organisaation taustalla on nähtävissä linja-esikuntaorganisaatio (KUVIO 8), jonka linjoina toimivat Talousosasto, Lähetysosasto ja Julkaisuosasto. Toimitusneuvoston yhtenä organisatorisena tehtävänä on toimia esikuntaelimenä julkaisuosastolle. (KUVIO 13.)



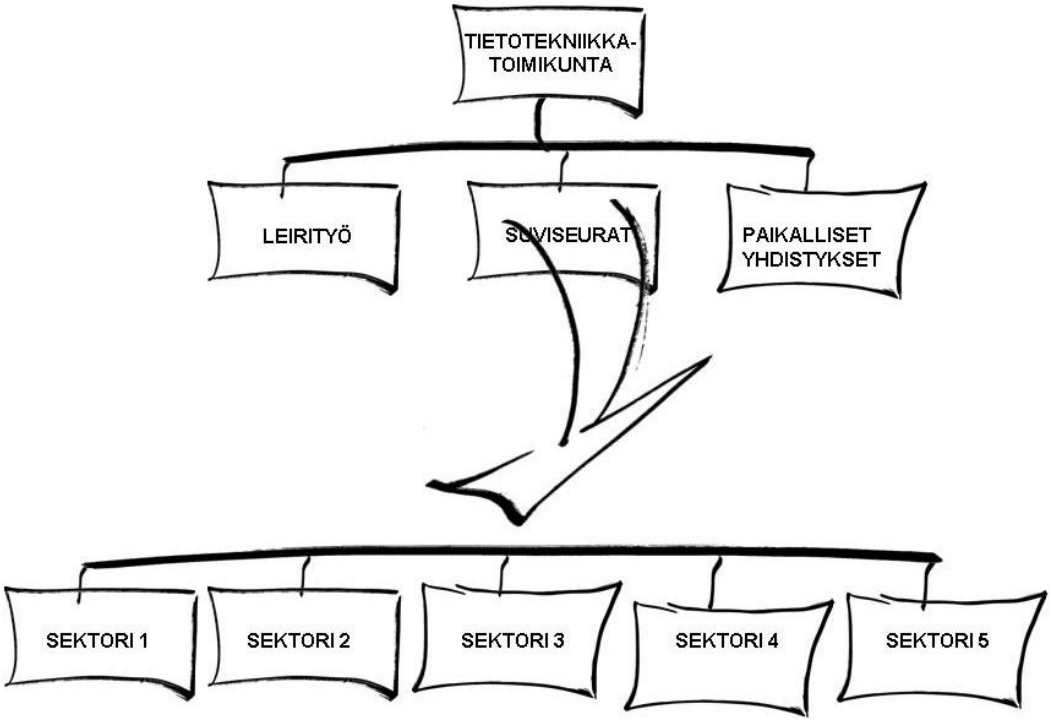
KUVIO 13. SRK:n perusorganisaatio

Organisoitumisessa lähdettiin miettimään ensin SRK:n ja ohjelmoijien välisen rajapinnan selkeyttämistä. Perusorganisaatiosta puuttui vahva tietotekninen ymmärrys, ja siksi päätettiin luoda oma toimikunta, tietotekniikkatoimikunta (KUVIO 14). Pohdittiin myös perusorganisaation vahvistamista tietoteknisellä osaamisella rekrytoinnin kautta, mutta se jätettiin pois ainakin tässä vaiheessa. Tietotekniikkatoimikunnan jäseninä oli tietotekniikka-alan osaajia ja SRK:n henkilökuntaa. Toimikunnalle annettiin valtuudet tehdä päätökset tarvittavista laitteistoista ja tehtäväksi otettavista projekteista. Tietotekniikkatoimikunnan tehtävänä oli toimia myös rajapintana ohjelmoijien ja eri sidosryhmien välillä keräten sidosryhmiltä tarvittavat kehittämistarpeet ja aikatauluttaa ne siten, että ohjelmointityö olisi mahdollista myös talkoilla. Toimikunnan ohjenuoraksi tehtävien priorisointiin annettiin se, että mitä enemmän kyseinen ohjelmisto tulisi tukemaan perusorganisaation työtä tai järjestön hengellistä työtä, sitä korkeampi prioriteetti sille annettaisiin.



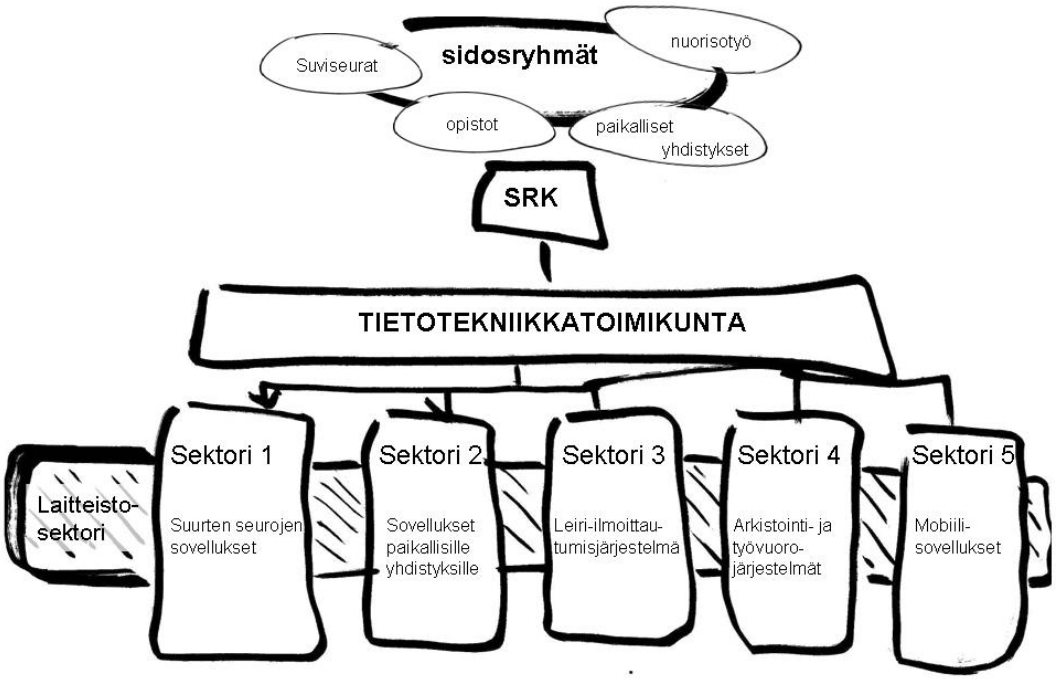
KUVIO 14. Tietotekniikkatoimikunta SRK:n organisaatiossa

Koska toiminta-alue oli laaja ja sidosryhmiä oli useampia, päätettiin tietotekniikkatoimikunnan työ jakaa sektoreihin, joissa varsinainen työ tehtäisiin. Alkuun sektorit jaettiin lähinnä sidosryhmien mukaan: Leirityö, Suviseurat ja Rauhanyhdistykset. (KUVIO 15.) Pian havaittiin, ettei jako sidosryhmittäin toiminut kunnolla, sillä sidosryhmien tarpeet työmäärällisesti olivat täysin erilaiset. Sidosryhmäjaon lisäksi tehtiin muutama ylimääräinen sektori ja tasattiin näiden sektorien avulla sektorien alla pyörivien projektien määrää. Koska yksi sidosryhmä ei tässä uudessa jaossa enää kohdistunutkaan välttämättä vain yhteen sektoriin, päätettiin sektorit numeroida yhdestä viiteen kuvion 15 mukaisesti. Jokaiselle sektorille nimettiin vastuuhenkilö, sektorivastaava. Sektorien alle listattiin meneillään olevat projektit. Ohjelmistokehitykseen oli saatu rakennettua talkoopohjainen linjaorganisaatio.



KUVIO 15. Linjaorganisaatio ohjelmointiorganisaation pohjamallina

Käytännön työssä havaittiin organisaation puutteeksi edelleen se, että jokainen sektori käytti samoja palvelimia ohjelmistojen kehittämiseen, testaukseen ja jakeluun. Eri sektoreilla oli hieman toisistaan eriävät tarpeet, ja siksi jokainen sääti palvelimen asetukset itselleen sopivaksi. Tämä nähtiin riskiksi jo siksi, että joka sektorilla oli ylläpitotunnukset palvelimille ja myös että toisen sektorin tarpeet saattoivat olla ristiriidassa jonkin toisen sektorin tarpeiden kanssa. Päätettiin perustaa oma Palvelin-sektori, jonka tehtävänä oli huolehtia palvelimien asetuksista ja tietoturvasta. Tämä uusi sektori sijoitettiin matriisiorganisaatiomallin mukaisesti poikittain muihin sektoreihin nähden (KUVIO 16). Näin saatiin jokaisen sektorin tarpeet Palvelin-sektorin tietoon ja samalla palvelinten hallinta rajoittui vain yhdelle sektorille.

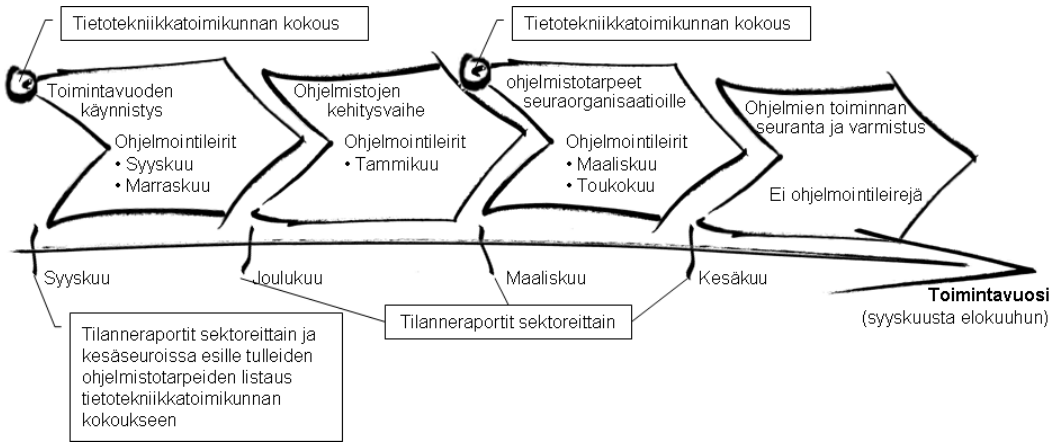


KUVIO 16. SRK:n ohjelmistokehitykseen rakennettu matriisiorganisaatio

Organisoitumisessa pohdittiin myös itse ohjelmointityötä. SRK:n ohjelmointitiimin työ oli ollut koko ajan (enempi tai vähempi) projektiluonteista, ja sen heikkoutena kehitystyövaiheen huonon projektihallinnan lisäksi oli ollut puhtaalle projektiorganisaatiolle tyypillinen ongelma: ohjelmistoprojektin käynnistys ja ylläpitovaihe oli hoidettu puutteellisesti. Kehitystyövaiheen projektinhallinnan parantaminen onnistuu määrittelemällä vastuuhenkilö jokaiselle projektille ja sopimalla yhteiset toimintatavat. Uusi organisaatiorakenne tai uudet selkeät toimintatavat eivät kuitenkaan tuo suoraan tukea ylläpitovaiheeseen. Sektorivastaavan yksi tehtävä on toimia rajapintana sidosryhmien eli tavallaan ohjelmien käyttäjien ja ohjelmoijien välissä. Sektorivastaava ei kuitenkaan voi toimia ohjelmien ylläpitäjänä, sillä ohjelmistoja kehitetään jatkuvalla syötöllä ja sektorivastaavan aika ei kerta kaikkiaan riittäisi kaikkien oman sektorin alla kehitettävien ohjelmien ylläpitoon. Ylläpitovaihe voitaisiin toteuttaa ohjelmointitiimin rinnalle rakennettavalla omalla ylläpitotiimillä, mutta ohjelmistojen määrän kasvaessa pitäisi myös ylläpitotiimin kokoa kasvattaa. On työlästä löytää henkilöä tai henkilöitä, joilla olisi jatkuva motivaatio kerätä ohjelmien käyttäjiltä palautetta, listata viat ja potkaista muutosprojektit käyntiin yhdessä sektorivastaavan kanssa. Parhaimmaksi ratkaisuksi koettiin ajatus, jossa ylläpito pyrittäisiin pilkkomaan sidosryhmien alle eli osittain käyttäjien vastuulle. Henkilöt, jotka ovat jatkuvasti tekemisissä ohjelmiston kanssa, ovat mitä parhaimpia henkilöitä listaamaan ongelmat, kouluttamaan muita käyttäjiä ja keräämään muutosehdotukset. Nämä vikaraportit ja uudet tarpeet

sektorivastaavan olisi helppo käydä läpi käyttäjien kanssa, ja tarvittaessa hän voisi vikaraporttien pohjalta käynnistää muutosprojektin. Sektorivastaavan yksi tehtävä olisi toimia oman sektorin alla pyörivien kehitysprojektien vetäjänä. Tarvittaessa projektin vetäjäksi, varsinkin laajoissa projekteissa, voitaisiin nimetä myös joku muu.

Organisaatiorakenteen yhteydessä sovittiin myös uuden talkoopohjaisen organisaation toimintajärjestelmästä. Toimintajärjestelmä on organisaation ohjaus- tai johtamisjärjestelmä, joka sisältää muun muassa toimintaprosessit, periaatteet toiminnan mittaamiseen ja sopimuksen henkilöstön hyvinvoinnin kehittämisen (Kookas2 2010). Tässä talkoopohjaisessa organisaatiossa ei niinkään tarvittu johtamisjärjestelmää vaan toiminnan ohjaamiseen kaivattiin työkaluja. Sopiviksi työkaluiksi koettiin säännölliset kokoontumiset sekä tilanneraportit. Sovittiin, että tietotekniikkatoimikunta kokoontuu kaksi kertaa vuodessa ja sen lisäksi jokainen sektorivastaava raportoi tietotekniikkatoimikunnalle aina neljännesvuosittain oman sektorinsa projektien etenemisestä. (KUVIO 17.) Koska kaikki ylimääräinen byrokratia vie aikaa varsinaiselta ohjelmointityöltä, päädyttiin kokeilemaan käytäntöä, jossa sektorivastaava antaa vastauksen oman sektorinsa tilanneraportissa vain kolmeen kysymykseen: ”Mitä teitte edellisen vuosineljänneksen aikana?”, ”Mitkä ovat tavoitteet seuraavalle jaksolle?” ja ”Onko ongelmia tai linjauksia jotka vaativat tietotekniikkatoimikunnan tai koko organisaation päätöksiä?”. Kokoontumisia, raporttien aikatauluja ja koko ohjelmointitiimin toimintavuotta ohjaavat paljolti kesällä järjestettävät Suviseurat ja muut suuret seuratilaisuudet. Tämän vuoksi toimintavuoden aloitus oli mielekäs sijoittaa syksyyn aivan kuten koulujen lukuvuosikin. Syksyn toimintaa leimaa kesällä seurojen aikana käyttäjiltä saatujen palautteiden läpikäynti ja niiden pohjalta käynnistettävät täysin uudet projektit ja mahdolliset muutosprojektit. Syksy on myös hyvä aika käynnistää mahdollisia uusia isoja projekteja, koska syksyllä kesäseuraorganisaatiot elävät hiljaiseloa. Keväällä painopiste siirtyy uusista projekteista kesällä tarvittavien, seuraorganisaatioita palvelevien jo olemassa olevien ohjelmien toiminnan varmistamiseen ja muutoksiin.



KUVIO 17. SRK:n ohjelmointitiimin toimintavuosi syyskuusta elokuuhun

Ohjelmointityö oli aikaisemmin toteutettu itsenäisesti talkootyönä ja lähinnä kokoonnuttiin yhteen vasta, kun selviä ongelmia oli olemassa ja tarvittiin kiireellistä korjausta eli niin sanottua ”tulipalon sammutusta”. Laaja ohjelmistoprojekti, jossa tekijöitä on useampia, on huomattavasti helpompi toteuttaa ja hallita, kun tekijöillä on mahdollisuus keskustella projektista ja ohjelmiston eri osasten rajapinnoista kasvotusten. Kokoontumalla riittävän usein projektin eteen tulevat haasteet ovat ratkaistavissa, ennen kuin ne kasvavat kriittisiksi ongelmiksi. Yhteisiä ohjelmointihetkiä päätettiin lisätä ja tehdä kokoontumisista säännöllisiä. Sovittiin, että kokoonnutaan kuukausittain tai vähintään kahden kuukauden välein johonkin SRK:n leirikeskukseen (KUVIO 17). Kotona tehtävän ohjelmointityön rinnalle saatiin selkeä tiimityötä tukeva toimintatapa, johon ohjelmoijien oli helppo sitoutua. Tällä toimintatavalla saadaan varmasti syntymään tuloksia ja onnistuneempia projekteja.

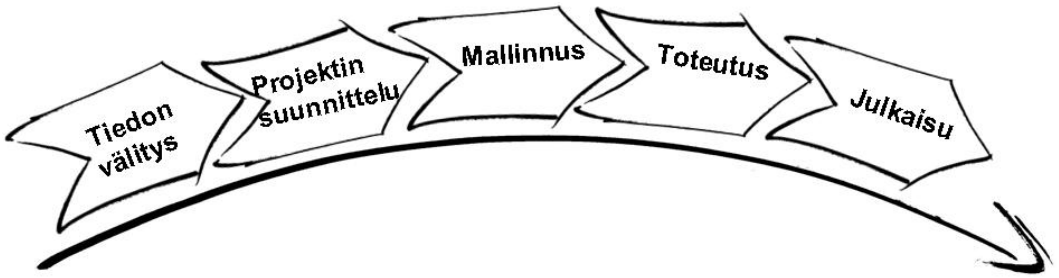
4 OHJELMISTOKEHITYSMALLIT

Jokainen ohjelmisto kulkee läpi elinkaaren, joka alkaa ideasta luoda uusi ohjelmisto ja päättyy siihen, kun ideasta kehitetty ohjelmisto pysäytetään viimeisen kerran viimeisenkin käyttäjän koneessa. Ohjelmistokehitysmalli on reseptimäinen malli siitä, mitä pitäisi tapahtua ensimmäisen ohjelmistoidean ja viimeisen pysäytyksen välillä. (McConnell 2002, 133.)

Ohjelmistokehitysmallilla tarkoitetaan yhteisiä toimintatapoja, joita pyritään noudattamaan tietyn ohjelmistoprojektin aikana (Pressman 2010, 31). Ohjelmistokehitysmallia voidaan verrata yrityksen toimintajärjestelmään tai laatujärjestelmään, joka sisältää ohjeet hyvälaatuisen tuotteen syntyyn. Ilman yhteisiä, ennalta sovittuja toimintatapoja ohjelmiston kehitystyö ajautuu ennemmin tai myöhemmin karille. Ohjelmiston päätyminen karille tapahtuu sitä nopeammin ja sitä varmemmin, mitä laajemmasta projektista on kyse ja mitä kauempana toisistaan ovat ohjelmistoprojektin jäsenet.

Ohjelmiston elinkaari voidaan jakaa Roger Pressmanin mukaan viiteen eri vaiheeseen: tiedon välitys (engl. communication), projektin suunnittelu (engl. planning), mallinnus (engl. modelling), toteutus (engl. construction) ja julkaisu (engl. deployment) (KUVIO 18). Tiedon välitys -vaihe lähtee liikkeelle ohjelmistoideasta ja päättyy, kun on saatu kerättyä kaikki toiminnalliset vaatimukset eli asiakastarpeet. Vaatimusten pohjalta määritellään projektisuunnitelma, jossa käyvät selville aikataulut ja resurssitarpeet. Projektisuunnitelmaa voidaan nimetä myös kartaksi. Minkä tahansa haastavan matkan, johon myös ohjelmistoprojektia voidaan verrata, toteutus on helpompaa, jos käytössä on hyvin suunniteltu kartta ja varusteluettelo. Projektisuunnitelman avulla voidaan alkavan projektin kustannukset helposti laskea ja ymmärtää. Mallinnusvaiheessa ohjelmistoprojektin lähtötietojen pohjalta mietitään ohjelmiston arkkitehtuuri, ohjelmiston paloittelu osiin, ohjelman eri osasten keskinäiset riippuvuudet ja rajapinnat sekä käyttöliittymät. Suunnitelmia tarkennetaan, kunnes tiedetään keinot, miten alkuperäisiin vaatimuksiin voidaan vastata. Hyvin suunniteltu on puoliksi tehty – kuten sanotaan. Mitä valmiimmaksi suunnitelmat on onnistuttu tekemään, sitä helpompi on varsinainen toteutusvaihe. Toteutusvaihe sisältää sekä varsinaisen ohjelmointivaiheen että siihen liittyvän testauksen. Toteutusvaiheessa

voidaan käyttää automaattisia koodigeneraattoreita, mikäli mallinnusvaiheessa ohjelmiston toiminta on tarpeeksi pitkälle suunniteltu. Valmiin testatun koodin julkaisu ja käyttöönotto tapahtuu julkaisuvaiheessa. Julkaisu vaihe sisältää ohjelmiston julkaisuun ja ylläpitoon liittyvät vaiheet. Siihen liittyvät kaikki mahdolliset ohjelmiston käyttökoulutukset, ohjelmiston asennukset, käyttäjäpalautteen keräämiset, mahdollisten virheiden korjaukset ja korjauspäivitykset. Uusista ohjelmistoversioista luonnollisesti luodaan aina uusi ohjelmistoprojekti. Kun elinkaaren osat sijoitetaan sananmukaisesti kaarelle kuvion 18 mukaisesti, havaitaan että työläin vaihe (kaaren nousu) ohjelmistokehityksessä on kaikki työ ennen toteutusta. Oikein tehdyillä suunnitelmilla itse toteutus ja julkaisu ovat vain suunnitelmien toteuttamista. (Pressman 2010, 15 & 31.)



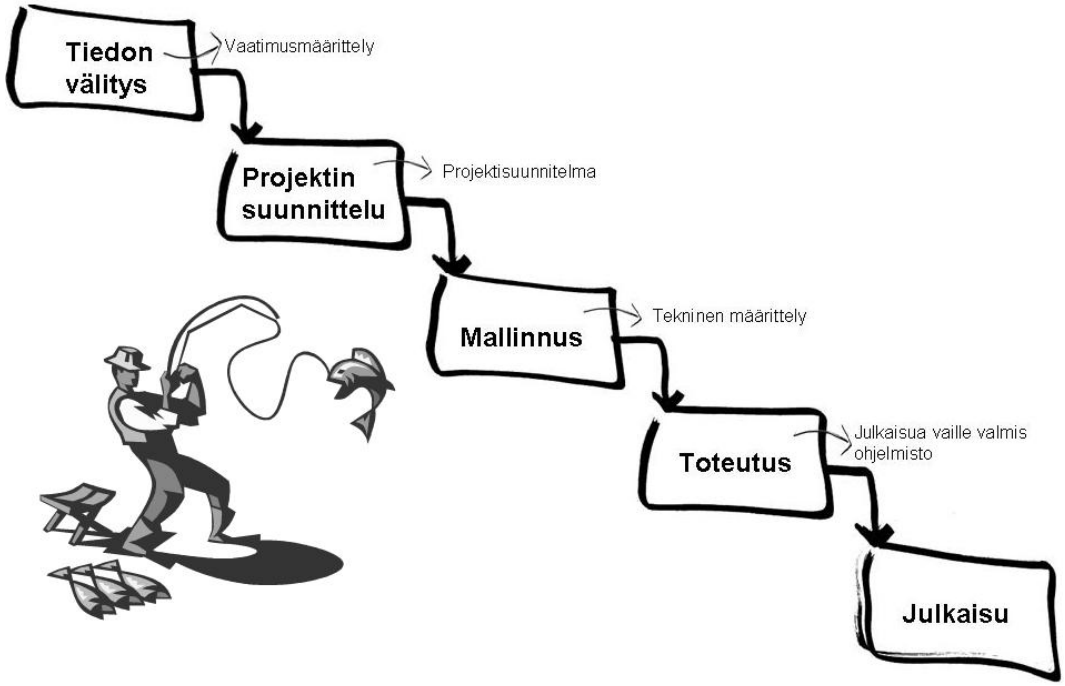
KUVIO 18. Ohjelmiston elinkaari (mukaillen Pressman 2010, 33.)

Kun mietitään ohjelmistokehitystä ja pyritään mallintamaan sitä, usein kiinnitetään huomiota vain itse ohjelmiston tekniseen toteuttamiseen. Tällöin helposti unohtuvat ohjelmiston kehitystyön reunaehdot eli projektisuunnitelma ja siihen selkeästi kuuluva aikataulutus. Näissä malleissa ohjelmiston kehitys lähtee liikkeelle ideasta ja vaatimusmäärittelystä. Määrittelyä seuraavat suunnittelu, ohjelmointi ja testaus. Tällainen työvaiheiden jakomalli on yleisesti käytetty ja sen kuvaa yksityiskohtaisesti muun muassa Steve McConnell kirjassaan (McConnell 2002, 133–139). Ohjelmiston kehitys sisältää aina nämä vaiheet riippumatta ohjelmiston koosta ja ohjelmointimenetelmistä. Myös ne ohjelmistot, joiden toteutus ei seuraa mitään hyväksi havaittua kehitysmallia, sisältävät nämä samat vaiheet. Tosin ilman ennalta sovittua kehitysmallia erilliset vaiheet voidaan havaita vasta jälkeenpäin. Projektinhallintaan ja siihen kuuluvan aikataulutuksen voidaan toki olettaa sisältyvän automaattisesti kehitettävän ohjelmistotuotteen ympärille. Teoriatasolla tämä automaatio toimii, mutta käytännössä ei. Jos itse projektinhallintaa ja varsinkin aikataulua ei ole selkeästi määritelty, kehitettävät palaset alkavat helposti elää omaa elämäänsä ja niiden kehitys-

aikataulut venyvät ja tärkeysjärjestys unohtuu. Ohjelmointityöstä tulee valmista ohjelmaa tärkeämpi.

4.1 Vesiputousmalli

Vesiputousmalli, jota myös perinteiseksi elinkaarimalliksi kutsutaan, toimii pohjana muille kehittyneemmille elinkaarimalleille (Pressman 2010, 39). Vesiputousmallissa ohjelmistokehitys etenee vaiheesta toiseen vasta, kun edellinen vaihe on saatu täysin valmiiksi, ts. ohjelmisto määritellään ja suunnitellaan ensin täysin valmiiksi, ennen kuin aloitetaan ohjelmointi, ja vasta kun kaikki on saatu ohjelmoitua, siirrytään testaukseen ja niin edelleen. Vesiputousmalli on dokumenttiohjautuva. Tämä tarkoittaa sitä, että aina edellisen vaiheen lopputuotteena syntyneet dokumentit toimivat seuraavan vaiheen lähtötietoina (McConnell 2002, 136). Esimerkiksi määrittelyvaiheessa syntynyt vaatimusmäärittelydokumentti toimii toteutusvaiheen aloitusdokumenttina. Nimensä vesiputousmalli on saanut siitä, että siinä ohjelmistokehitystyön eri vaiheet etenevät ja putoavat kuin vesiputouksessa tai koskessa tasolta toiselle kohti joen alajuoksua (KUVIO 19). Kuten oikeassa koskessa tai vesiputouksessakin, tässä mallissa takaisinpäin peruuttaminen on hankalaa tai lähes mahdotonta. Jos esimerkiksi ohjelmointivaiheessa havaitaan määrittelydokumentaatiossa virhe tai jos ohjelmointivaiheen aikana tulee tarvetta muutoksille, on koko määrittelyvaihe käytävä läpi uudestaan ja odotettava uutta versiota määrittelyvaiheen dokumentaatiosta. McConnell toteaakin muutoksien ja virheiden korjaamisen olevan työlästä ja usein aikaa vievää (McConnell 2002, 138).



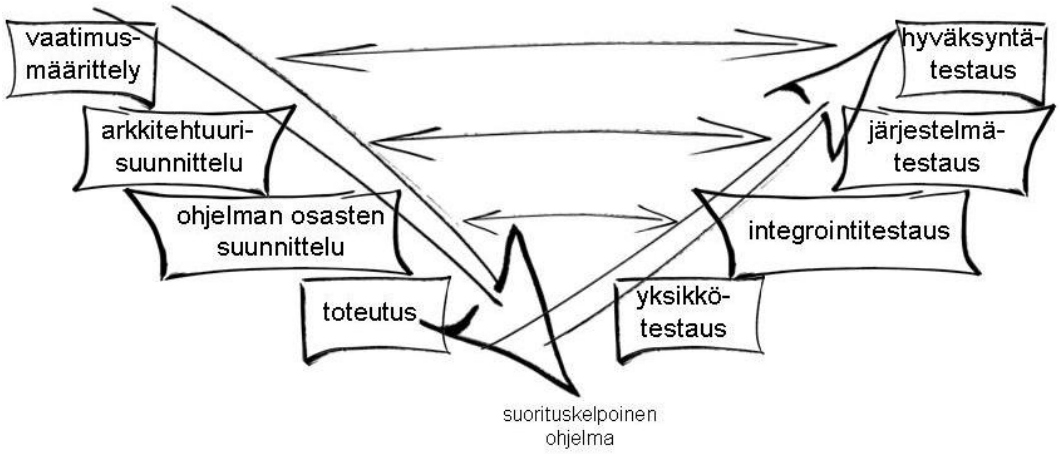
KUVIO 19. Vesiputous-ohjelmistokehitysmalli (mukaillen Pressman 2010, 39.)

Vesiputousmallin suurin haaste on suunnitella ohjelmisto täysin valmiiksi yhdellä kertaa. Ohjelmistot sisältävät usein toiminnallisuuksia, jotka saattavat olla erittäin hankalia toteuttaa, ja määrittelyä ei voida tehdä, ennen kuin on vähän päästy ohjelmoinnissa kokeilemaan erilaisia ratkaisuvaihtoehtoja. Jos määritelmä pyrittäisiin viemään väkisin eteenpäin vesiputousmallin mukaisesti, ohjelman kehitys saattaisi kaatua jossain vaiheessa johonkin ylitsepääsemättömään tekniseen ongelmaan (McConnell 2002, 138). Ongelma saattaisi aiheuttaa sen, että määrittelyvaiheessa suunniteltu perusajatus pitäisi rakentaa täysin uudella tavalla. Koko suunnitelma olisi mennyt hukkaan, ja ohjelmiston kehitys olisi jo valmiiksi kuukausia myöhässä.

Käytännössä ohjelmointiprosessi ei juuri koskaan etene selkeästi vaiheesta toiseen, vaan prosessin aikana se etenee lukuisten pienten suunnittelu- ja toteutusvaiheiden läpi (Pressman 2010, 41). Tämä jatkuva suunnittelu- ja toteutusvaiheiden tarve on synnyttänyt vesiputousmallille läheisen V-mallin sekä iteratiivisia ohjelmointiprosessimalleja, kuten spiraalimalli ja ketterät ohjelmointiprosessit.

4.2 V-malli

Yksi muunnelma vesiputousmallista on V-malli (engl. V-model). V-mallissa testaus ja tuotteen vastaavuus alkuperäisiin määritelmiin on tuotu hieman paremmin esille kuin perinteisessä vesiputousmallissa. V-mallissa perinteisen vesiputousmallin jokaisen vaiheen rinnalle on lisätty kyseisen vaiheen testaus. V-mallin nimi tulee itse mallin muodosta (KUVIO 20). Vesiputousmalli muodostaa V-kirjaimen vasemman, laskevan sakaran, ja elinkaaren eri vaiheita vastaavat testaukset muodostavat V-kirjaimen oikean, nousevan sakaran. Tällöin ohjelman toteutus ja sitä vastaava yksikkötestaus ovat V-kirjaimen pohjalla. Jos ohjelma läpäisee yksikkötestauksen, suoritetaan integrointitestaus. Integrointitestauksessa tulevat ilmi mahdolliset eri ohjelman osasten väliset ongelmat. Järjestelmätestauksella pyritään löytämään mahdolliset arkkitehtuurisuunnittelussa syntyneet virheet. Ennen kuin asiakas voi hyväksyä ohjelman olevan valmis, on ohjelman läpäistävä hyväksyntätestaus. Hyväksyntätestauksessa ohjelma tarkistetaan siten, että sen toiminta on vaatimusmäärittelyn mukainen. (Pressman 2010, 40.)



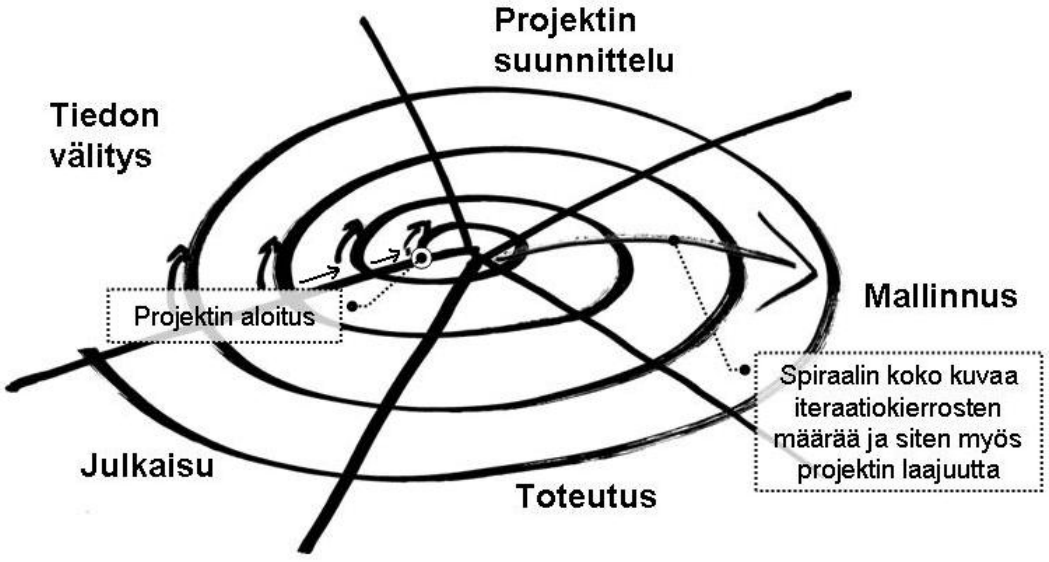
KUVIO 20. Ohjelmistokehityksessä käytetty V-malli (mukaillen Pressman 2010, 40.)

4.3 Spiraalimalli

Nykypäivän nopeasti muuttuvassa ympäristössä perinteisen vesiputousmallin vaatima ennalta suunnittelu on noussut suurimmaksi haasteeksi toteuttaa. Nykypäivän tuotteet ovat erittäin monimutkaisia, ja jos vaikka suunnitelma saataisiinkin täydellisesti tehtyä ennen ohjelmointivaihetta, ovat suunnitelmat jo yleensä vanhentuneita. On tullut tarvetta purkaa

ohjelmisto pienempiin osiin ja toteuttaa kehitystyö iteratiivisena prosessina. Spiraalimalli on yksi tällainen iteratiivinen prosessi (Pressman 2010, 45).

Spiraalimallissa, jota voidaan kutsua myös protoilumalliksi, tuotteen rakentaminen aloitetaan jostain pienestä kokonaisuudesta. Tälle pienelle kokonaisuudelle, miniprojektille, tehdään vesiputousmallista tutut vaiheet eli määrittely, suunnittelu, mallinnus, toteutus ja julkaisu. Tehdään siis tuotteesta tai sen osasta ensimmäinen prototyyppi. Kun ensimmäisen iteraatiokierroksen aikana syntynyt tuote tai tuotteen osa todetaan toimivaksi, aloitetaan uusi iteraatiokierros. Uuden kierroksen aikana tuotteeseen lisätään ominaisuuksia seuraavalla määrittely–toteutus–julkaisu-vaiheella. Tuotteen koko ja ominaisuudet kasvavat vaihe vaiheelta tarpeita vastaavaksi. Syntyy spiraalimainen kuvio, joka on esitetty kuviossa 21. Jos uuden iteraatiokierroksen aikana syntynyt tuotos havaitaan virheelliseksi, voidaan spiraalikuviossa hypätä ennen tiedonvälitysvaihetta säteensuuntaisesti kohti spiraalin keskiötä edellisten iteraatiokierroksen tuotosten tasolle. Ohjelman lopullinen ominaisuuksien määrä ei yleensä ole aivan se, mitä alusta oletettiin, sillä iteratiivisuus mahdollistaa muutostarpeisiin vastaamisen jo kehitystyön aikana. Projektin suuntaa ja kulkua voidaan muuttaa hallitusti kesken prosessin. Ohjelmisto tulee harvoin kerralla valmiiksi, joten useat ohjelmistotuotteet jatkavat kehitystään ensimmäisen julkaisun jälkeenkin. (McConnell 2002, 141–142; Pressman 2010, 43–47.)



KUVIO 21. Spiraalimallin havainnekuva (mukaillen Pressman 2010, 47.)

Spiraalimallissa yksittäiset iteraatiot on ketjutettu toisiinsa ja käärityt rullalle, spiraalimaiseksi nauhaksi. McConnellin spiraalimallissa spiraalin koko ja siinä olevien kierrosten määrä kuvaa visuaalisesti iteraatioiden määrän ja sitä kautta suuntaa antavasti ohjelman ominaisuuksien ja kustannusten määrän (McConnell 2002, 141). Iteraation määrä on esitetty kuviossa 21 säteen suuntaisella nuolella. Spiraalimalli on erittäin käyttökelpoinen suurissa projekteissa ja silloin, kun kehitetään jotakin uutta, jonka onnistumisessa on suuria riskejä. Vaihe kerrallaan edeten löydetään kenties ratkaisut ja saadaan näin ollen riskit pienemmiksi. Spiraalimallissa riskit pienenevät sitä enemmän, mitä enemmän aikaa ja rahaa on sidottuna projektiin. Projektin toteutuksen estävät tekniset tai muut syyt havaitaan mahdollisimman aikaisin, jolloin myös kustannukset ovat pienet. Projekteissa, joissa kustannuksille on asetettu jokin kiinteä katto, on spiraalimallin käyttö yleensä mahdoton. Iteraatiokierrosten aikana määritelmät ja tarpeet muuttuvat jatkuvasti, ja harvoin muutokset ovat kustannuksettomia. Yhden iteraatiokierroksen aikana löytyy kenties uusia ongelmia, jotka tulee selvittää ennen seuraavaa iteraatiota, tai asiakkaalta tulee muutosehdotuksia ensimmäisen prototyypin pohjalta.

4.4 Ohjelmoi ja korjaa -malli

Ohjelmoi ja korjaa -malli ei oikeastaan ole mikään hallittavissa oleva malli, mutta olen nostanut sen esille, koska hyvin moni ohjelmistoprojekti ja varsinkin sen viimeiset vaiheet noudattavat menetelmää, jota voidaan kutsua McConnellin kuvaamaksi Ohjelmoi ja korjaa -malliksi (KUVIO 22). Todennäköisesti ohjelmistoprojektin kehityksessä on käytössä automaattisesti Ohjelmoi ja korjaa -malli, mikäli ei ole mitään muuta elinkaarimallia valittu käyttöön (McConnell 2002, 140).

Ohjelmiston kehitystyöhön perinteisesti kuuluu mm. määrittely, ohjelmointi, testaus ja dokumentointi. Määrittely, dokumentointi ja joskus myös testaus nähdään usein kiireen keskellä vain turhana työtehtävänä. Kun projektin havaitaan olevan myöhässä ja/tai muutostarpeita tulee tiuhaan, luisutaan helposti keskittymään ohjelmointiin määrittelyn sijaan. Tavoitteena on saada valmis ohjelma niin pian kuin mahdollista. Ilman suunnitelmallista ohjelmointia työn edetessä törmätään ongelmiin, virheisiin ja ristiriitoihin, jotka kenties vaativat uudelleen ohjelmointia. Koska aikaa ei ole miettiä ongelmiin kunnon ratkaisua,

tyydytään ”purkka-liima”-ratkaisuihin. Nämä ”purkka-liima”-ratkaisut voivat olla mitä tahansa ohjelmallisia niksejä, joilla saadaan ohjelma jotenkuten toimimaan toivotulla tavalla. Ennemmin tai myöhemmin nämä hätäratkaisut unohtuvat, ja uusia muutoksia ja ohjelmaversioita tehtäessä ne nousevat ongelmaksi. Hätäratkaisu kenties rajoittaa jonkin osan toiminnallisuutta siinä määrin, että uusi versio pitää aloittaa täysin alusta. (McConnell 2002, 140.)

KUVIO 22. Ohjelmoi ja korjaa -malli (mukaillen McConnell 2002, 140.)

Ohjelmoi ja korjaa -malli saattaa olla käyttökelpoinen projekteihin, jotka ovat erittäin pieniä, yhden miehen kyhäelmiä ja jotka tiedetään väliaikaisiksi, koska mallissa ei ole yhtään ylimääräistä työtä. Ohjelman prototyypit ja varsinkin varsinaisen ohjelmiston testaamiseen tehtävät apuohjelmat toteutetaan usein tämän kaltaista mallia käyttäen, mutta vaarana on se, että aikataulu ja kustannukset karkaavat käsistä. Ohjelmoi ja korjaa -mallissa ei ole ylimääräistä työtä, mutta siinä ei ole myöskään minkäänlaisia keinoja arvioida projektin edistymistä; ohjelmointia tehdään kuumeisesti, kunnes saadaan työ valmiiksi tai kuten kuviossa 22 on näkyvissä, ohjelma ehkä saadaan valmiiksi. Pienissäkin projekteissa olisi suositeltavampaa käyttää esimerkiksi aikaisemmin esitettyä spiraalimallia. (McConnell 2002, 140.)

4.5 Ketterä-ohjelmistokehitys

Ohjelmistotuotannon aloilla, joissa on varauduttava suureen muutosnopeuteen on omaksuttu niin sanottuja ketteriä ohjelmistoprosesseja, jotka korostavat muutosten hallintaa ja nopeita iteraatiosyklejä. Aikaisemmin esillä ollut spiraalimallin iteratiivisuus mahdollistaa muutostarpeisiin vastaamisen jo kehitystyön aikana, mutta malli ei anna suoraan työkaluja

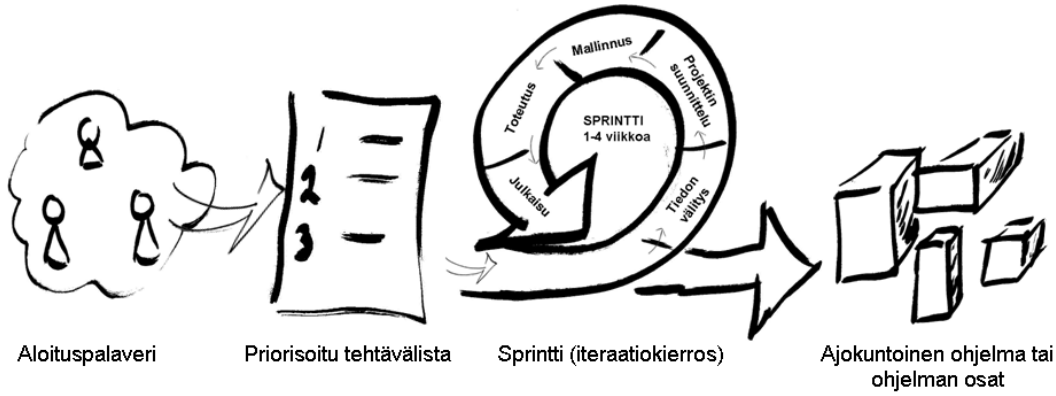
nopean muutoksen hallintaan. Spiraalimallissa iteraation kestoa ei ole millään lailla valmiiksi määritelty. Ketterä-menetelmissä pääpaino on itse ohjelman toimivuudessa ja suorassa viestinnässä dokumenttien sijaan. Tämä toiminta kuulostaa hyvin nopeasti ajateltuna samalta kuin Ohjelmoi ja korjaa -mallissa, jossa dokumentaation sijaan keskitytään ohjelman toteutukseen. Näin on ajatellut myös Dilbert-sarjakuvan tekijä (Kuvio 23). Ketterä-menetelmät kuitenkin määrittelevät tiukasti ohjelmiston elinkaaren eri vaiheiden lisäksi itse projektiorganisaation ja projektin hallinnan työkalut, kuten ohjelmointitiimin jäsenet, palaveriaikataulut ja palaverien suuntaa antavan sisältöehdotelman. Ketterä-menetelmiä on useita ja jokaisessa on pieniä vivahde-eroja projektin hallintaan liittyvissä käytännöissä. Kaikille on yhteistä se että menetelmät perustuvat vuonna 2001 laadittuun yhteiseen manifestiin, Manifesto for Agile Software Development. Manifestin sisältö on seuraavanlainen:

> ”Me etsimme parempia keinoja ohjelmistojen kehittämiseen tekemällä sitä itse ja auttamalla siinä muita. Tässä työssämme olemme päätyneet arvostamaan:
> – Yksilöitä ja vuorovaikutusta enemmän kuin prosesseja ja työkaluja
> – Toimivaa sovellusta enemmän kuin kokonaisvaltaista dokumentaatiota
> – Asiakasyhteistyötä enemmän kuin sopimusneuvotteluita
> – Muutokseen reagoimista enemmän kuin suunnitelman noudattamista.
>
> Vaikka oikeallakin puolella on arvoa, me arvostamme vasemmalla olevia asioita enemmän.” (Agilemanifesto 2010; Pressman 2010, 65.)

KUVIO 23. Ketterä ohjelmistokehitys Dilbert-sarjakuvassa (Scott 2007)

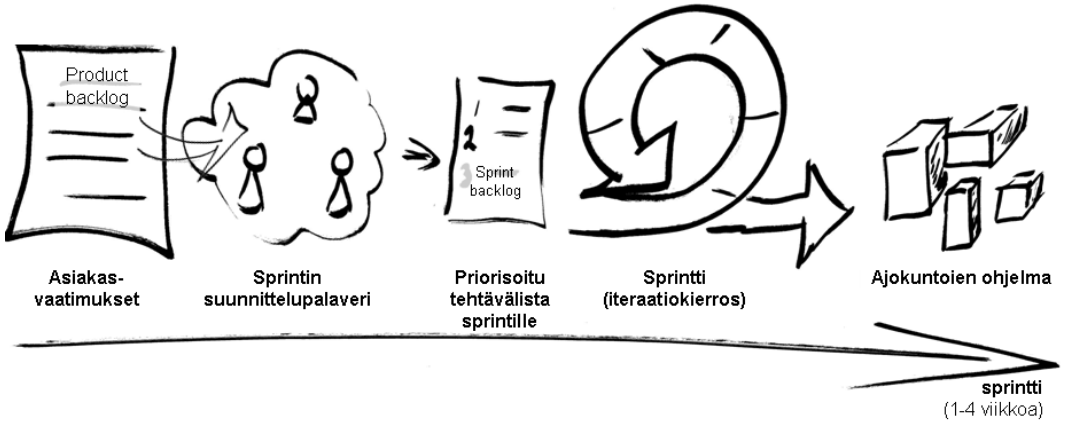
Yksi Ketterä-ohjelmistokehityksen menetelmistä on Scrum-menetelmä. Scrum-menetelmässä kehitteillä oleva ohjelmisto rakentuu pala palalta valmiimmaksi, useiden iteraatiokierrosten aikana. Iteraatiokierrosta kutsutaan pyrähdykseksi (engl. Sprint), eli ns. sprintiksi. Yhden sprintin kesto on 1–4 viikkoa, ja se riippuu toteutettavan ohjelman koosta ja muutoksiin varautumisen asteesta. Mitä lyhyempi sprintti on, sitä nopeammin voidaan reagoida muutoksiin. Jokaisen sprintin sisältö sovitaan ennen periodin aloitusta yhteisesti tiimiläisten kesken omassa aloituspalaverissa. Seuraavan pyrähdyksen tehtäviksi valitaan

vain ne asiat, joiden merkitys projektin onnistumiseen on sillä hetkellä suurin. Tavoitteena on, että jokaisen sprintin jälkeen ohjelma olisi ajokuntoinen, toisin sanoen yksikään sprintin aikana tehtäväksi sovittu asia ei olisi jäänyt kesken. (KUVIO 24.) Projektitiimi esittelee sprintin lopuksi sprintin saavutukset, esimerkiksi uusimman version kehitteillä olevasta ohjelmistosta. (Beedle & Schwaber 2002.)



KUVIO 24. Scrum-menetelmän iteraatiokierros (mukaillen Beedle & Schwaber 2002, 8.)

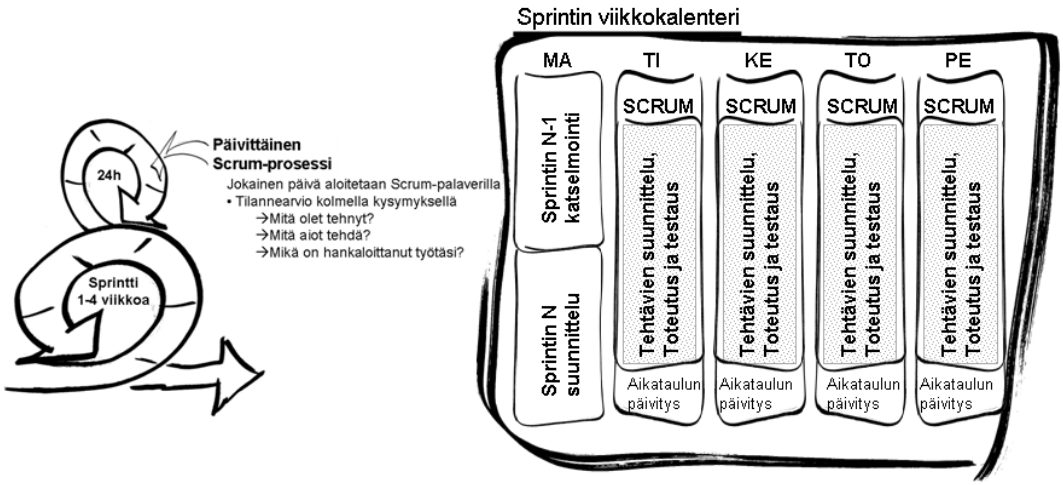
Scrum-menetelmässä yksi pääpaino on tiimiläisten keskinäisessä ja asiakkaan ja tiimiläisten välisessä vuorovaikutuksessa. Jotta vuorovaikutus olisi mahdollisimman tehokasta projektin edistymisen kannalta, ohjelmistojen kehittämiseen käytettyjen itsenäisten tiimien koko on rajattu 5–9 henkilöön. Tiimi päättää yhdessä kunkin sprintin tavoitteet ja tehtävät ja siten myös sitoutuu vastaamaan yhdessä siitä, että asetettuihin tavoitteisiin päästään. Jokaisen sprintin alussa käydään vaatimusmäärittely (product backlog) ja siihen mahdollisesti tulleet muutokset läpi yhdessä asiakkaan kanssa (KUVIO 25). Scrum-ryhmän yksi jäsen, tuotepäällikkö (engl. Product Owner), toimii asiakkaiden edustajana. Hänen tehtävänsä on priorisoida vaatimukset ja huolehtia projektin kannattavuudesta sekä pitää huolta, että hänen priorisoimat tehtävät tulevat hoidetuiksi jokaisessa sprintissa. Toinen tärkeä jäsen Scrum-tiimissä on Scrum-mestari (engl. Scrum Master). Scrum-mestari pitää huolen siitä, että prosessi etenee tasaisesti ja sovittuja pelisääntöjä noudatetaan. Hänen tehtävänä on myös poistaa tiimin työtä hankaloittavat esteet ja suojata tiimiä turhilta keskeytyksiltä. Muut tiimin jäsenet keskittyvät sprintissä määriteltyjen tehtävien suorittamiseen. (Beedle & Schwaber 2002.)



KUVIO 25. Asiakasvaatimusten huomioiminen Scrum-prosessissa (mukaillen Beedle & Schwaber 2002, 8.)

Scrum-menetelmässä 1–4 viikon sprintin eteneminen varmistetaan lyhyillä päivittäisillä Scrum-palavereilla. Scrum-prosessin toimintatapa päivittäisessä työskentelyssä käy hyvin ilmi kuviosta 26, jossa on kuvattu yhden viikon mittaisen sprintin viikkokalenteri. Sprintin ensimmäinen päivä on varattu edellisen sprintin (Sprintti N-1) katselmoinnille ja tulevan sprintin suunnittelulle (Sprintti N). Katselmoinnissa tarkistetaan, tuliko kaikki edellisen sprintin tehtävät hoidettua vai oliko jokin tehtävä laajuudeltaan sen verran suuri, että se tulee ottaa mukaan myös seuraavan sprintin tehtäviin. Suunnittelussa priorisoidaan tulevan sprintin tehtävät ja nimetään tehtäville tekijät. Suunnittelupalaverissa poimitaan asiakkaalta tulleesta vaatimusmäärittelystä (KUVIO 25) ja edellisestä sprintistä tekemättä jääneistä tehtävistä ne, jotka varmasti ehditään suorittaa tulevan sprintin aikana. Suunnittelupalaverin tuloksena syntyy priorisoitu tehtävälista tulevalle sprintille (KUVIO 25). Viikon muut päivät tiistaista perjantaihin on tehtävien toteutusta. Päivittäiset Scrum-palaverit ovat pituudeltaan noin 15 minuutin mittaisia, ja ne on merkitty jokaisen päivän alkuun (KUVIO 26). Palavereissa jokaiselta tiimiläiseltä kysytään vain kolme kysymystä: Mitä olet tehnyt edellisen palaverin jälkeen?, Mitä aiot tehdä seuraavaan palaveriin mennessä? ja Mikä on hankaloittanut työtäsi?. Kaikki muut kysymykset ovat kiellettyjä. Scrum-palaverin jälkeen on yleensä keskusteluhetki, jossa pyritään ratkomaan mahdolliset ongelmat sekä suunnitellaan yhdessä tulevaa työtä. Jos tiimi ei pysty esille tullutta ongelmaa itsenäisesti ratkomaan, jätetään se Scrum-mestarin harteille. Mahdollisimman suuri osa työpäivästä on varattu itse toteutusvaiheelle. Toteutusvaihe käy jokaisen sprintin tehtävän kohdalla läpi ohjelmiston elinkaaren eri vaiheet. Osa elinkaaren vaiheista tulee toteutettua jo sprintin aloituspalaverissa, mutta vähintään suunnittelu, toteutus ja testaus ovat jokaiselle tehtävälle

kuuluvia toimenpiteitä itse sprintin aikana. Jokaisen päivän päätteeksi päivitetään aikataulut. Aikataulun päivitys on tärkeä yksittäinen toimenpide, sillä vain sillä tavoin saadaan valmistumisaikataulu tarkennettua Scrum-prosessin mukaisesti. (Beedle & Schwaber 2002.)



KUVIO 26. Päivittäinen Scrum-prosessi palavereineen (mukaillen Beedle & Schwaber 2002, 8.)

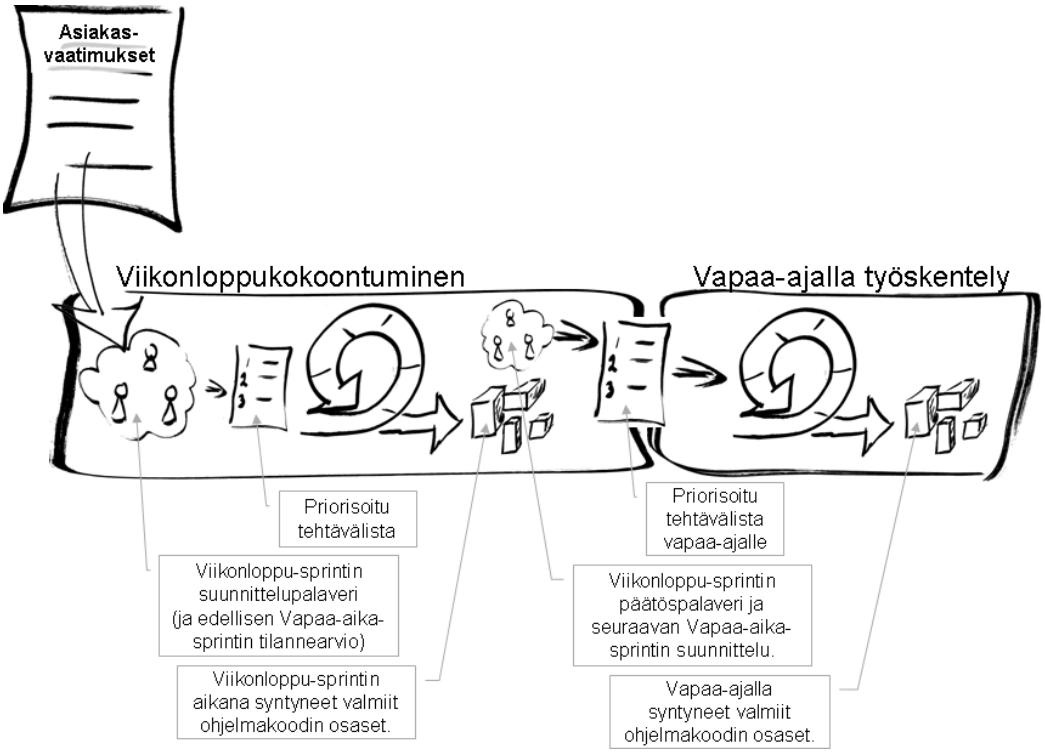
4.6 Talkootyöhön sopivan ohjelmistokehitysmallin luominen

Yksikään ohjelmistokehitysmalli ei tahdo soveltua suoraan työhön, jossa tekijöiden vaihtuvuus on suuri ja käytetty työaika erittäin rajallinen. Perinteinen vesiputousmalli voisi toimia talkootyöympäristössä hyvin dokumenttiohjautuvuutensa vuoksi. Kun on selkeät määrittelydokumentit, koko järjestelmän ymmärtäminen onnistuu, vaikka tulisi mukaan kesken projektin. Ohjelmoija voi keskittyä niihin toiminnallisuuksiin, jotka ovat vielä toteuttamatta. Ongelmaksi saattaa tulla se, että määrittelyvaiheessa ei välttämättä ole voitu ennustaa jonkin toiminnallisuuden laajuutta, ja siten sen toteuttaminen pienellä viikkotuntimäärällä voi viedä kohtuuttoman paljon aikaa. Toinen suurempi ongelma on se, että vähäisellä viikkotuntimäärällä itse dokumentaation tekeminen vie niin paljon aikaa, että vaatimukset ehtivät vanhentua moneen kertaan, ennen kuin päästään itse ohjelmointivaiheeseen. V-malli, joka on muunneltu versio perinteisestä vesiputousmallista, tuo itse toteutukseen lisää selkeyttä kytkemällä ohjelmistokehityksen eri vaiheet ja vaiheiden hyväksynnät tiiviisti toisiinsa. V-mallissa on kuitenkin perusongelma sama kuin vesiputousmallissa: työ on dokumenttiohjautuvaa, ja rajallinen työaika venyttää projektia liikaa.

Yksi varteenotettavista vaihtoehdoista on spiraalimalli. Spiraalimallissa ohjelmiston kehitys lähtee liikkeelle pienistä palasista ja näiden pienten palasten määrittelystä. Määrittelyn jälkeen kyseinen ohjelmiston osa toteutetaan ja testataan ja siirrytään seuraavan osasen määrittelyyn. Vaihe vaiheelta tehtäessä saadaan helposti jotain valmistakin aikaan, vaikka työtuntimäärä olisikin erittäin vähäistä. Vaiheistus mahdollistaa myös ohjelmiston asiakkaan mukanaolon kehityksen eri vaiheissa. Määrittely tarkentuu vaihe vaiheelta samalla kun ohjelmiston koko kasvaa. Spiraalimalli olettaa, että ohjelmiston tekijöillä ja asiakkaalla on jokin tieto siitä, mikä on se alkuperäinen tavoite eli mitä mahdollisesti tulisi tehdä edellisen vaiheen jälkeen. Toki vaiheistus mahdollistaa tavoitteiden muuttamisen kehityksen aikana, mutta jokin idea pitää olla siitä, minkälaista ohjelmaa ollaan tekemässä. Jos ohjelmiston tavoite on hukassa, itse tekemisestä tulee valmista ohjelmistoa tärkeämpi. Ohjelmoijien vaihtuessa saattaa tuo lopullinen tavoite mennä hukkaan tai muuttua väärään suuntaan ja yhden vaiheen suorittamisen jälkeen ei kenelläkään ole selkeää kuvaa siitä, mikä pitäisi olla seuraava vaihe. Spiraalimallin heikkoutena on myös sen aikataulullinen seuranta: jonkin osasen toteutus voi työstövaiheessa tuoda eteen uusia ongelmia, joiden poistamiseen menee kohtuuttomasti aikaa. Osavaiheen valmistumista on hanakala aikatauluttaa.

Scrum-prosessissa, ohjelmiston kehitys on vaihe vaiheelta toteutusta kuten spiraali-mallissakin, mutta siinä otetaan osatehtävien toteutukseen mukaan aikataulu. Scrum-prosessissa jokainen tehtävä on pilkottu niin pieniin osiin, että ne voidaan toteuttaa esim. yhden työpäivän aikana. Ohjelmiston kehitys koostuu vaiheista, sprinteistä, joiden aikataulullinen kesto on määritelty, ja vaiheen aikana toteutettavien tehtävien määrä valitaan siten, että ne ehditään lähes varmasti toteuttaa kyseisen vaiheen aikana. Ideaalitapauksessa jokainen osatehtävä on aikataulullisesti kooltaan samansuuruinen, jolloin projektin etenemistä on helppo arvioida. Scrum-prosessin toteuttaminen onnistuu myös ympäristössä, jossa on mahdollisuus tehdä vain rajallinen määrä työtunteja, koska työtehtävien määrä tarkentuu käytettävissä olevan työajan mukaan. Tietenkin ohjelmiston kehitysaikataulu venyy talkoympäristössä, mutta se on tiedostettu asia jo työhön lähtiessä. Scrum-prosessi tuo osatehtävien seurantaan työkalut. Scrum-prosessi myös sitoo tekijöitä paremmin, sillä yhden iteraatiokierroksen eli sprintin tehtävät priorisoidaan ja tehtäville määritellään tekijät aina yhdessä.

Aikaisemmista ongelmista viisastuneena SRK:n ohjelmointitiimissä todettiin Scrum-prosessin aktivoivan juuri oikealla tavalla ohjelmointikehitystyön seurantaa. Koska tiimi on hajallaan ympäri Suomea, jouduttiin pohtimaan, kuinka voidaan toteuttaa käytännössä Scrum-prosessin edellyttämät sprintit ja niihin liittyvät palaverit. Päätettiin rakentaa oma muokattu Scrum-prosessi. Ohjelmointitiimin keskusteluissa nousi esiin se, että tehokkaimmalta olivat tuntuneet ne tilanteet joissa yhdessä koko tiimin tai osatiimin voimin oli kokoonnuttu kokonaiseksi viikonlopuksi viemään kehitystyötä eteenpäin. Kahden kuukauden välein tapahtuva viikonloppukokoontuminen päätettiin ottaa räätälöidyn Scrum-prosessin rungoksi. (KUVIO 27.) Pääsääntöisesti nämä viikonloppukokoontumiset ovat alkaneet perjantaina jonkinlaisella tehtävälistauksella. Tämä aloitus oli helppo muuttaa Scrum-prosessin mukaiseksi sprintin katselmointi- ja suunnittelukokoukseksi. Yleensä viikonlopun kokoontumiset päättyivät jonkinlaiseen tilannekatsaukseen. Tämä tilannekatsaus päätettiin vakiinnuttaa ja todettiin sen vastaavan hyvin Scrum-prosessin katselmointia. Todettiin, että viikonloppukurssin aikana, jolloin kaikki ovat lähellä, on tehtävien etenemistä helppo seurata jatkuvasti. Erillisiä Scrum-palavereita ei tarvita, sillä viikonloppu on sen verran lyhyt.



KUVIO 27. Talkootyöhön räätälöity Scrum-prosessi

Ohjelmointiprojektit ovat lähes aina sen verran suuria, että niitä ei yhden viikonlopun eli sprintin aikana saada valmiiksi. Koska viikonloppukokoontumisia tulisi kahden kuukauden välein, tulisi ohjelmointityöhön aivan liian pitkiä taukoja. Todettiin yhdessä, että tulisi keksiä keino, kuinka ohjelmointitehtäviä voitaisiin suorittaa halitusti myös kokoontumisien välillä yksikseen omalla vapaa-ajallaan. Päätettiin rakentaa kahden toisistaan erilaisen sprintin Scrum-prosessi. Viikonloppukokoontumisilla olisi oma sprintti, Viikonloppu-sprintti ja kokoontumisien väliin jäävälle ajalle oma Vapaa-aika-sprintti. (KUVIO 27.) Viikonloppu-sprintti olisi tiivis työvaihe, jossa keskityttäisiin sellaisiin tehtäviin, joiden prioriteetti on korkea tai jotka vaativat yhteistä pohdintaa ja päätöksiä. Viikonloppu-sprintin tavoitteet ja tehtävät määriteltäisiin aina kokoontumisen alussa, ja tilannearvio olisi kokoontumisen lopussa. Viikonloppu-sprintin tehtävälista pyrittäisiin rakentamaan siten, että tehtävät varmasti ehdittäisiin toteuttamaan viikonlopun aikana. Vapaa-aika-sprintin tehtävät listattaisiin ja priorisoitaisiin aina Viikonloppu-sprintin lopussa ja tilannearvio suoritettaisiin aina seuraavan Viikonloppu-sprintin alussa, ennen Viikonloppu-sprintin tehtävien listausta ja priorisointia. Tehtävien suorituksien seuranta olisi organisaatiomallin mukaisesti sektorivastaavilla. Myös tehtävien alustava listaus ja priorisointi olisivat sektorivastaavien tehtävänä yhdessä taustaorganisaation ja sidosryhmien kanssa.

Tehtävien hallintaan ja samalla kaikkien projektien seuraamiseen päätettiin ottaa jokin valmis työkalu käyttöön. Ohjelmointitiimin jäsenet tiesivät monia hyviä työkaluja, ja monella oli henkilökohtaisia kokemuksia ja mieltymyksiä työnsä kautta tiettyihin työkaluihin. Keskustelussa nousivat esille projektinhallinnassa ja tehtävien seurannassa käytetty Trac (Edgewall 2010) ja sille läheinen sukulainen Redmine (Redmine 2010). Osa ohjelmointitiimin projekteista oli jo toteutettu kyseisiä työkaluja käyttämällä, joten lähinnä päätettäväksi tuli, kumpaa ohjelmistoa tullaan käyttämään jatkossa. Koska Redminen yksi ominaisuus oli useamman projektin yhtäaikainen hallinta, oli valinta loppujen lopuksi helppo. Kaikki tehtävät listattaisiin jatkossa Redmine-työkaluun sektorivastaavien toimesta. Täysin internetpohjaisena työkaluna Redmine mahdollistaa tehtävien hallinnan omalta kotikoneelta.

Ohjelmistokehitysmalliksi rakennettu räätälöity Scrum-prosessi ja siihen selkeästi kuuluva tehtävien priorisointi yhdessä tehtävienhallintaohjelmiston kanssa näyttää ohjaavan jokaista ohjelmoijaa automaattisesti toimimaan tietyn prosessin mukaisesti. Toimintamalliksi

näyttäisi muodostuvan kuvion 28 kaltainen prosessi, jossa asiakkaiden vaatimusmäärittelyt kirjataan Redmineen ensin sanallisessa muodossa ja sitten myöhemmin yksittäisiksi tehtäviksi. Tehtävät priorisoidaan alustavasti asiakkaan tarpeiden mukaan ja tarkennetaan Viikonloppu-sprintin alussa pidettävässä suunnittelupalaverissa. Samassa yhteydessä niille nimetään tekijä. Koska työkalut ohjaavat ohjelmoijia toimimaan saman mallin mukaan, voimme olettaa ohjelmistojen laadun paranevan jatkossa.

KUVIO 28. SRK:n ohjelmistokehitystiimin toimintamalli.

5 YHTEENVETO

Tämän opinnäytetyön lähtökohtana oli tarve selvittää vapaaehtoisen ohjelmistokehitystyön ongelmat sekä etsiä ongelmiin ratkaisut. Selvitystyö tuli ajankohtaiseksi, kun Suomen Rauhanyhdistysten Keskusyhdistyksen (SRK) tarpeisiin räätälöityjen ohjelmistojen kehitystyössä havaittiin ongelmia, joihin ei osattu etukäteen riittävästi varautua. Ongelmien havaittiin olevan yhtenevät niiden ohjelmistokehityksen ongelmien kanssa, joita on käsitelty ohjelmistoalan kirjallisuudessa jo vuosien ajan. Pieniä eroja nousi esille lähinnä ongelmien painotuksessa ja esiintymistiheydessä. Tavoitteeksi tuli löytää sellainen organisaatio ja ohjelmiston kehittämisen malli, jotka toimisivat yhdessä hyvin myös hajautetussa talkootyössä.

Vapaaehtoisten ohjelmointiprojektien aikana ohjelmoijat olivat törmänneet usein ohjelmointiprojektin hallintaan liittyviin ongelmiin. Ohjelmiston määrittely oli puutteellinen, ja usein jälkeenpäin tulleet yksittäiset pyynnöt ja muutostarpeet ohittivat suvereenisti aikaisemmat määrittelyt. Myös ohjelmistoprojektin hallittu päättäminen ja ohjelmiston ylläpito ontuivat. Ongelmia oli yritetty ratkaista ohjelmointitiimin sisällä erilaisilla suullisilla päätöksillä. Päätökset havaittiin pian turhiksi, sillä niille ei löytynyt vastuunkantajaa. Ohjelmointiin suuntautuneen vahvan ja yhtenäisen organisaation puute aiheutti sen, ettei kukaan ollut selkeästi vastuussa tuloksista. Jokainen ohjelmointitiimin jäsen oli omassa ansiotyössä tottunut selkeään organisaatioon. Alkuun organisaation puute oli tuntunut vapauttavalta, mutta myöhemmin kurinalaisen toiminnan puute aiheutti täysin turhia ongelmia ja heikensi näin ollen myös motivaatiota. Selvitystyön aikana havaittiin suurimmaksi ongelmaksi ohjelmointiin suuntautuneen organisaation puute. Oli mahdoton määritellä ohjelmointiprosessia ja varsinkin noudattaa sitä, jos kurinalaista taustaorganisaatiota ei ollut olemassa.

Organisaatio rakennettiin talkoilla toteutettavaksi yhdessä SRK:n olemassa olevan organisaation ja ohjelmistotiimin jäsenten kanssa. Ohjelmistojen kohderyhmiä oli selkeästi viisi: suuret seuratapahtumat, leirityö, mobiilisovellukset, paikallisyhdistykset ja arkistointi. Näistä ryhmistä luotiin organisaatioon omat sektorit, ja ohjelmoijat jakautuivat näiden sektoreiden alle. Organisaatioon lisättiin myös yksi sektori, jonka tehtävänä oli ylläpitää

palvelinlaitteistoja. Tämän laitteistosektorin tehtävänä oli palvella kaikkia muita sektoreita tarjoten työkalut, kehitysympäristön ja julkaisualustan kehitettäville ohjelmistoille. Ohjelmistokehitystyön kontrollointiin perustettiin SRK:n olemassa olevan organisaation ja edellä mainittujen sektoreiden väliin oma tietotekniikkatoimikunta. Tietotekniikkatoimikunnan viidestä jäsenestä kolme oli SRK:n perusorganisaation jäseniä ja kaksi oli talkoo-ohjelmoijia. Ohjelmointitiimin mukaan ottaminen organisaatiomallin suunnitteluun oli menestys. Organisaatio saatiin rakennettua juuri sellaiseksi, että se palveli sekä ohjelmistojen käyttäjiä että ohjelmoijia parhaalla mahdollisella tavalla. Yhteistyössä rakennettu organisaatiomalli sitoutti ohjelmoijia paremmin työhönsä. Ohjelmointityö eteni joutuisammin, koska jokainen tiesi oman tehtävänsä. Kaikki vähäinen talkootyöhön annetusta ajasta saatiin käytettyä tehokkaasti.

Organisaatio toi selkeyttä työnkuvaan ja vastuiden jakamiseen. Selkeä työnjako poisti paljon päällekkäisiä töitä, ja jokaiselle tehtävälle löytyi tekijä. Selvittelytyön alkuvaiheessa oli tullut esille, ettei organisaatio yksin riitä ohjaamaan itse ohjelmointityötä. Ilman toimivaa ohjelmistokehitysmallia ohjelmoijilla oli, varsinkin työn vapaaehtoisen luonteen vuoksi, taipumusta keskittyä niihin asioihin, jotka tuntuivat mukavilta toteuttaa. Välttämättä nämä mukavat asiat eivät olleet tärkeimpien asioiden listalla ohjelmiston lopputulosta ajatellen. Erilaisia prosessimalleja ohjelmiston kehitykseen oli ohjelmointitiimin jäsenillä jo tiedossa oman ansiotyönsä kautta, mutta yksikään malli ei tuntunut oikein sopivan talkootyöllä toteutettavaan kehitystyöhön. Erilaisista ohjelmistokehitysmalleista parhaimmalta tuntui ketterä ohjelmistokehitys, jossa pääpaino on itse ohjelman toimivuudessa ja suorassa viestinnässä dokumenttien sijaan. Talkootyöhön varattava aika haluttiin käyttää mahdollisimman tehokkaasti itse ohjelmistojen eteenpäin viemiseen. Talkoo-ohjelmointityötä oli tehty lähinnä viikonloppuisin yhdessä ja jonkin verran itsenäisesti kotona. Ohjelmointiprosessi haluttiin rakentaa sellaiseksi, että lyhyessäkin ajassa saataisiin jotain valmista aikaiseksi. Scrum-ohjelmointiprosessissa, yhdessä Ketterä-ohjelmistokehityksen malleista, ohjelmointityö pilkotaan sen kokoisiin osakokonaisuuksiin, jotka on mahdollista toteuttaa annetussa ajassa. Nämä osakokonaisuudet priorisoidaan yhdessä ohjelmoijien kanssa sen mukaan, mikä on senhetkinen asiakasvaatimus. Toimivaksi malliksi tarkentui räätälöity Scrum-prosessi, jossa yhteisille viikonloppukokoontumiselle rakennettiin oma priorisoitu tehtävälista ja itsenäiselle kotityöskentelylle varattiin omat tehtävät. Tehtävien priorisointiin ja seurantaan laitteistosektorin jäsenet asensivat internetpohjaisen Redmine-

projektinhallintasovelluksen. Redmine-työkalulla tehtävien jakaminen ja priorisointi oli helppoa ja samalla antoi näkyvyyden projektien etenemiselle.

Tämän opinnäytetyön pohjalta rakennettu organisaatio ja ohjelmistokehitysmalli otettiin käyttöön syksyn 2009 aikana. Scrum-ohjelmistokehitysprosessi koeponnistettiin käytännössä ohjelmointileirillä keväällä 2010. Yhdessä ohjelmoijien kanssa rakennetun organisaation havaittiin motivoivan ohjelmoijia entistä paremmin. Selkeä organisaatio toi esille ohjelmistojen loppukäyttäjät ja heidän todellisen tarpeen. Jokainen ohjelmoija koki vahvasti, että juuri hänen työpanoksellaan on merkitystä. Myös organisaation johdon osallistuminen ohjelmointileireille kannustusmielessä vahvisti työn merkittävyyttä. Räätälöity ohjelmointiprosessi toimi hyvin viikonloppukokoontumisilla, mutta kotona tehtävän työn seuranta on edelleen hieman hankalaa. Projektinhallintaan valitut työkalut mahdollistivat seurannan, mutta koska kyseessä on vapaaehtoistyö, ei ketään voi oikein pakottaa pysymään tietyssä aikataulussa. Ohjelmoijat ovatkin sitä mieltä, että vapaa-aikana paras komentaja tulee olemaan keskinäinen sosiaalinen paine. Samalla kun jokainen tiimin jäsen oppii tuntemaan omat ja toistensa taidot ja sen, kuinka paljon kenelläkin on mahdollisuus tehdä ohjelmointia omalla vapaa-ajallaan, opitaan pilkkomaan ohjelmistoprojekti oikeansuuruisiksi osakokonaisuuksiksi. Tärkeimpänä pidettiin sitä, että vaikka ohjelmointityön eteneminen olisikin hidasta, seuranta estää liian optimaalisen aikataulun tavoittelun. Aikaisemmin ohjelman saatettiin olettaa olevan melkein valmis, vaikka sen ohjelmointia ei ollut välttämättä edes vielä konkreettisesti aloitettu. Nyt kehitystyön etenemistä pystyivät seuraamaan jopa ohjelmiston tulevat käyttäjät. Seuranta estää myös yksittäisten ohjelmoijien liiallisen kuormittamisen saaden talkootyön näyttämään edelleen mielekkäältä ja mukavalta.

Aikaisemmin täysin unohduksiin jätetty ohjelmistojen ylläpito koki myös muutoksen. Ohjelmointitiimin jäsenten aika ei riittänyt hoitamaan ohjelmistojen ylläpitoa, ja uusien tiimiläisten rekrytointi vain ohjelmistojen ylläpitotehtäviin kuulosti jo alun alkaen haasteelliselta. Puhdas ylläpitotyö ei välttämättä motivoisi uutta tiimiläistä pysymään mukana tiimin toiminnassa kovin pitkään. Ylläpito päätettiin jalkauttaa ohjelmien käyttäjille. Jokaisen ohjelman käyttäjäorganisaatioista valittiin yksi henkilö, jolle annettiin vastuulle ohjelmien ongelmien ja muutosehdotusten kirjaaminen. Näille henkilöille annettiin myös hieman syvällisempää tietoa ohjelmien toiminnasta ja heidät valtuutettiin hoitamaan ohjelmien

käyttöopastus oman organisaationsa sisällä. Ohjelmien ongelmien ja muutostarpeiden kirjaaminen ja lähettäminen aina ohjelmointitiimin sektorivastaavalle sähköpostilla koettiin myös hyväksi ratkaisuksi. Ongelmien kirjaaminen suoraan projektinhallintajärjestelmään, Redmineen, vaatisi käyttäjältä tietoteknisen kielen hallintaa, jotta myös ohjelmoija ymmärtäisi yksiselitteisesti ongelman tai muutostarpeen kuvauksen. Sektorivastaava, joka on lähimpänä käyttäjäorganisaatioita ja tuntee hyvin ohjelmiston rakenteen, sen ominaisuudet ja rajoitteet, pystyy parhaiten muuttamaan käyttäjien ongelmat ja ideat ohjelmoijalle ymmärrettävään muotoon. Sektorivastaava kirjaa ongelmat ja ideat tehtäviksi projektinhallintajärjestelmään. Nämä ongelmat priorisoidaan ja otetaan tehtäväksi aina seuraavassa ohjelmointikokoontumisessa räätälöidyn Scrum-prosessin mukaisesti.

SRK:n ohjelmointitiimin ohjelmistot palvelevat tyypillisesti erilaisia kesäisin pidettävien seurojen seuravieraita ja talkoilla toimivia organisaatioita. Yleensä nämä seuraorganisaatiot perustetaan kertakäyttöisiksi ja ne vaihtuvat vuosittain. Ylläpidon jalkauttaminen näihin organisaatioihin oli onnistunut valinta, sillä se helpotti ohjelmoijien työtaakkaa. Organisaation vaihtuminen aiheuttaa kuitenkin sen, että myös kyseisessä organisaatiossa valittu ylläpitohenkilö vaihtuu. Ylläpitohenkilöiden tarkoitus on kouluttaa aina seuraavan organisaation ylläpitohenkilö. Koska heiltä kuitenkin puuttuu syvällinen ohjelmoinnin ja ohjelman tuntemus, saattaa osa tietotaidosta hävitä matkalla. Jossain vaiheessa saattaa siis vielä olla tarvetta päivittää tässä opinnäytetyössä esitettyä organisaatiota sellaiseksi, että siinä olisi pysyvämpi ylläpitotiimi. Jos tällaisen ylläpitotyön hoitaminen talkoilla ei onnistu, siten että tietotaito pikemminkin kasvaisi kuin vähenisi, tulee mahdollisesti harkita ylläpitohenkilön palkkaamista SRK:n perusorganisaatioon.

Täysin vapaaehtoisvoimin toteutetussa ohjelmistokehitystyössä korostuu ennen kaikkea motivaatio. Eri ihmisillä on erilaiset tarpeet, ja heitä voidaan motivoida monella eri tavalla. Jos motivaatiokeinona toimii itse työ, onnistuu sen toteuttaminen myös talkoilla. Uusilla organisaatio- ja ohjelmistokehitysmalleilla työ saatiin näyttämään mielekkäältä ja innostavalta. ”Jos ihmiset ovat kunnolla motivoituneita, pätevyyden kehittämisestä tulee osa heidän omaa tavoitettaan” (Drucker 2008, 162). Pätevyyden kehittäminen auttaa parantamaan jatkuvasti jo kehitettyjä prosesseja, ja sitä kautta ohjelmistojen laatu paranee entisestään.

LÄHTEET

Agilemanifesto. 2010. Manifesto for Agile Software Development. Www-dokumentti. Saatavissa: http://www.agilemanifesto.org. Luettu 28.1.2010.

Beedle, Mike & Schwaber, Ken. 2002. Agile Software Development with Scrum. New Jersey, USA: Pearson Education International.

Drucker, Peter F. 2008. Voittoa tavoittelemattoman organisaation johtaminen. Helsinki: Talentum Media Oy.

Edgewall. 2010. Welcome to the Trac Open Source Project. Www-dokumentti. Saatavissa: http://trac.edgewall.org. Luettu 6.5.2010.

Harisalo, Risto. 2008. Organisaatioteoriat. Tampere: Tampere University Press.

Hokkanen, Simo & Strömberg, Oiva. 2003. Ihmisten johtaminen. Jyväskylä: Sho Business Development Oy.

Kookas. 2010. Organisaatio on sopimus työnjaosta. Www-dokumentti. Saatavissa: http://www.kookas.fi/articles/read/7156. Luettu 19.1.2010.

Kookas2. 2010. Kehittyvä toiminta. Www-dokumentti. Saatavissa: http://www.kookas.fi/articles/read/6343. Luettu 19.1.2010.

Kopperoinen, Juho. 2009. SRK:n suviseurat Oripäässä 2009. Käykää kuulemaan, mitä Herra puhuu. Oulu: Suomen Rauhanyhdistysten Keskusyhdistys ry.

Laestadian Lutheran Church. 2009. Who we are. Www-dokumentti. Saatavissa: http://www.laestadianlutheran.org. Luettu 28.9.2009.

Lipiäinen, Toivo. 2000. Liiketoiminnan menestystekijät. Liiketoiminta uudella vuositu-hannella. Jyväskylä: Kaupunkitohtorit Oy.

McConnell, Steve. 2002. Ohjelmistotuotannon hallinta. Helsinki: Edita Publishing Oy.

Pressman, Roger S. 2010. Software Engineering. A Practitioner's Approach. 7. painos. New York, USA: Mc Graw Hill.

Redmine. 2010. Redmine. Www-dokumentti. Saatavissa: http://www.redmine.org. Luettu 6.5.2010.

Rissanen, Riitta & Sääski, Kaija & Vornanen, Jouni. 1996. Uudistuvat organisaatiot. Kuo-pio: Pohjois-Savon Ammattikorkeakoulu.

Scott Adams. 2007. Dilbert strips. Www-dokumentti. Saatavissa: http://www.dilbert.com/strips/comic/2007-11-26. Luettu 11.5.2010.

Suomen Rauhanyhdistysten keskusyhdistys ry. 2010. Näin me uskomme. Www-dokumentti. Saatavissa: http://www.srk.fi. Luettu 19.1.2010.

Suviseurat. 2010. SRK:n suviseurat. Www-dokumentti. Saatavissa: http://www.suviseurat.fi. Luettu 17.1.2010.

Suvivuorot. 2010. Työvuorojen hallintajärjestelmä. Www-dokumentti. Saatavissa: http://www.suvivuorot.net. Luettu 15.1.2010