VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Tommi Raudasoja ja Jouni Hamm

TIETOJÄRJESTELMÄPROJEKTIN HALLINTA JA TESTAUS

Jouskari

Liiketalous
2016

VAASAN AMMATTIKORKEAKOULU
Tietojenkäsittelyn koulutusohjelma

TIIVISTELMÄ

Tekijä	Tommi Raudasoja & Jouni Hamm
Opinnäytetyön nimi	Tietojärjestelmäprojektin Hallinta ja Testaus: Jouskari
Vuosi	2016
Kieli	suomi
Sivumäärä	51
Ohjaaja	Sirkka Hellman

Tämä opinnäytetyö käsittelee ohjelman kehitystä C#-ohjelmointikielellä, testivetoista ohjelmistokehitystä ja ketterää projektinhallintaa. Tarkoituksena oli kehittää toimeksiantajalle ajastintaulusovellus jousiammuntaan vanhan Java-sovelluksen tilalle. Ohjelman tarkoituksena on helpottaa jousiammuntakilpailun seuraamista ja hallintaa.

Tutkimuksen teoriaosuudessa käsitellään yleisesti C#-ohjelmointikieltä, SQL-kyselykieltä, tietokantaohjelmia, projektissa käytettyjä ohjelmia, olio-ohjelmointia, erilaisia ketterän projektinhallinnan tapoja kuten Scrum ja Extreme Programming ja ohjelman testausta testivetoisesti. Tavoitteena oli syventää osaamistamme ohjelmiston kehityksestä ja projektin loppuun viemisestä.

Työn tuloksena oli prototyyppi versio 1, jonka kehitystä mahdollisesti jatketaan raportin valmistumisen jälkeen. Työtä tehdessä opimme, kuinka ohjelmistoprojektissa pitäisi edetä, mitkä asiat mahdollisesti vaikuttavat projektin onnistumiseen ja miksi kommunikaatio on tärkeä osa ohjelmistoprojektia.

Avainsanat	Olio-ohjelmointi, testivetoinen, ketterä ohjelmistokehitys, Scrum

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Tietojenkäsittelyn koulutusohjelma

ABSTRACT

Author	Tommi Raudasoja & Jouni Hamm
Title	Project Management and Testing in Information Systems Project: Jouskari
Year	2016
Language	Finnish
Pages	51
Name of Supervisor	Sirkka Hellman

This thesis studied software development in C# programming language, test-driven software development and agile project management. A timer board application was developed for archery competition to replace the client's old Java application. The goals of the software are to ease the representation and the control of archery competition.

The theoretical background of this thesis consists of an overview of C# programming language, SQL query language, database programs, the software used in the project, object oriented programming, different kinds of methods in agile project development such as Scrum and Extreme Programming as well as test driven software testing. The goal was to deepen our expertise on software development and learn to complete project.

The result of the project was a prototype version 1, which development will probably be continued after the completion of this report. As we worked on this project we learned how we should proceed, what things may affect on the success of the project and why communication is a key component for a software project.

Keywords: Object-oriented programming, Test-driven, agile software development, Scrum

SISÄLLYS

KUVIO- JA TAULUKKOLUETTELO

1 JOHDANTO

Toteutimme opinnäytetyönä toimeksiantajalle jousiammuntaan ajastintaulusovelluksen, jonka tarkoituksena on auttaa kilpailun toimitsijoita esittämään kilpailijoille ja katsojille kilpailun kulkua helposti ymmärrettävällä tavalla. Työssä tullaan käyttämään jonkin verran kuvia ja ohjelmakoodia esimerkkinä havainnollistamaan projektin vaiheita. Projektin kuvaamisessa käytetään myös Scrumin termistöä, koska käytimme projektimme hallinnassa Scrumia. Ohjelmakoodiesimerkit kertovat, mitä haluamme ohjelman tekevän ja kuvilla näytämme, miten ohjelmakoodi toimii käytännössä. Raportin kirjoitushetkellä ohjelmasta tuli valmiiksi prototyyppi versio 1 eli ohjelmakoodi ja työn näyttökuvat on otettu tästä versiosta.

Työn tarkoituksena on syventää osaamistamme C#-ohjelmoinnista ja miten olio-ohjelmointia hyödynnetään ohjelmoinnissa. Halusimme myös kehittää osaamistamme projektinhallinnassa käyttäen ketterää ohjelmistokehitystä, jota monet ohjelmistotalot nykypäivänä käyttävät ohjelmistoprojekteissaan. Pidimme myös tärkeänä perehtyä ohjelmistokehityksen yhteen tärkeimmästä osa-alueesta eli ohjelmiston testaamisesta testivetoisen kehityksen pohjalta. Aiemmin mainittujen asioiden pohjalta kehitimme seuraavat tutkimuskysymykset, joihin pyrimme raportissa vastaamaan:

- Mikä on olio-ohjelmoinnin tarkoitus ja miten sitä hyödynnetään C#-ohjelmoinnissa?
- Mikä tekee ketterästä ohjelmistokehityksestä niin suositun ja miksi käytimme sitä omassa projektissa?
- Mitkä ovat testivetoisen kehityksen hyödyt ja miksi sitä kannattaa käyttää ohjelmistokehityksessä?

Ensimmäiseen tutkimuskysymykseen ”Mikä on olio-ohjelmoinnin tarkoitus ja miten sitä hyödynnetään C#-ohjelmoinnissa?” etsimme tietoa monista eri lähteistä, joista tärkeimpinä ovat kirja C#-ohjelmointi, Microsoft Developer Network ja Mika Kolarin materiaali sivustolla www.mikakolari.fi.

Toiseen tutkimuskysymykseen ”Mikä tekee ketterästä ohjelmistokehityksestä niin suositun ja miksi käytimme sitä omassa projektissa?” käytimme lähteinä monia erilaisia kirja- ja nettilähteitä, joista muutaman mainitaksemme olivat Agile Manifeston materiaali osoitteessa www.agilemanifesto.org ja Petri Heiramon materiaali. Tutkimuskysymykseen vastatessa tuli käytettyä myös omia kokemuksiamme, tuntemuksiamme ja havaintojamme projektin edetessä.

Kolmannen tutkimuskysymyksen ”Mitkä ovat testivetoisen kehityksen hyödyt ja miksi sitä kannattaa käyttää ohjelmistokehityksessä” ratkaisuun käytimme lähteinä nettilähteitä, kirjaa Ohjelmistotestauksen käsikirja, Vaasan ammattikorkeakoulussa käytyä kurssia Ohjelman testaus ja ylläpito ja oman ohjelmamme testaamisesta saatuja tuloksia.

Raportti koostuu seitsemästä luvusta, joista ensimmäinen on johdanto, jonka tarkoituksena on antaa yleiskuva lukijalle tehdystä työstä. Luvussa kaksi kerromme yleisesti käytetyistä työkaluista ja tekniikoista niiden ominaisuuksista ja miksi valitsimme ne. Kolmas ja neljäs luku on omistettu testaamiselle ja projektinhallinnalle. Viidennessä luvussa kerromme oman projektimme hallinnasta ja järjestelmän määrittelystä. Kuudennessa luvussa kerromme itse ohjelman teosta ja työmme kulusta kuvilla ja ohjelmakoodia selostaen. Seitsemännessä luvussa on yhteenveto saaduista tuloksista, oppimisprosessista ja mahdollisista kehitysehdotuksista.

2 TYÖKALUT JA TEKNIIKAT

Tässä kappaleessa käsitellään työkaluja ja tekniikoita, joita käytimme ohjelmaa tehdessä. Työssä käytettyjä ohjelmia olivat Microsoft Access 2010, Microsoft Visual Studio 2010 Ultimate ja Notepad++. Käytettyihin tekniikoihin kuuluivat relaatiotietokanta, C#-ohjelmointikieli ja tietokannanhallinta kieli SQL. Kerromme myös muista tietokantojen hallintajärjestelmistä, jotka ovat varteenotettavia vaihtoehtoja Microsoft Accessille.

2.1 Access

Microsoft Access on tietokannan hallintajärjestelmä, joka on ohjelmoitu C#-ohjelmointikielellä. Access sisältää ohjelmistokehitystyökaluja ja graafisen käyttöliittymän. Sillä on mahdollista tehdä esimerkiksi loppukäyttäjäikkunoita. Access on relaatiotietokanta, jossa on mahdollista käyttää tauluja, kyselyitä, lomakkeita, raportteja, makroja ja moduuleja. Makroja ja moduuleja voi luoda Accessiin Visual Basic for Applications ohjelmointikielellä. Taulut, kyselyt, lomakkeet ja raportit voidaan luoda graafisessa käyttöliittymässä, mutta jos haluaa niitä voi myös luoda SQL-kyselyillä. (Microsoft 2016a)

2.2 Visual Studio

Microsoftin Visual Studio on integroitu kehitysympäristö, jota käytetään tietokoneohjelmien, verkkosivujen, sovellusten ja verkkopalvelujen kehitykseen. Visual Studio käyttää Microsoftin ohjelmistokehitysympäristöjä. Ohjelmassa on koodieditori, joka tukee IntelliSenseä ja ohjelmakoodin refaktorointia. Visual Studion muita ominaisuuksia ovat Windows Forms Designer, Class Designer ja Data Designer. Editoriin saa myös asennettua lisäosia, jotka parantavat sen toiminnallisuutta. Visual Studio tukee melkein kaikkia mahdollisia ohjelmointikieliä ja sisäänrakennettuja ohjelmointikieliä ovat C, C++, VB. NET, C# and F#. Tukea muihin ohjelmointikieliin saa lataamalla kielipalveluja, jotka asennetaan ohjelmaan erikseen. (Microsoft 2016b)

2.3 Notepad++

Notepad++ on ilmainen avoimen lähdekoodin koodieditori, jonka on kehittänyt Don Ho. Ohjelma on tehty C++-ohjelmointikielellä ja se on GNU-lisenssin alainen. Ohjelman maininnan arvoisia ominaisuuksia, joita ei välttämättä kaikista koodieditoreista löydy ovat:

- kokonaan muokattava graafinen käyttöliittymä
- syntaksin korostaminen
- PCRE
- sanojen, funktioiden ja funktioiden parametrien automaattitäydennys
- WYSIWYG tulostus
- macrojen ja lisäosien luonti ja käyttö. (Don Ho 2016)

2.4 Ohjelmointikieli C#

C# on tyyppiturvallinen oliopohjainen ohjelmointikieli, jolla ohjelmoijat tekevät ohjelmia .NET ohjelmistokehykseen. C#-ohjelmointikielellä voi esimerkiksi tehdä Windows työpöytäsovelluksia, XML Web palveluja, hajautettuja komponentteja, asiakas-palvelinsovelluksia, tietokantasovelluksia ja monia muita ohjelmia. C#-ohjelmoinnissa käytetään yleensä Visual Studio koodieditoria, koska se on kehitetty erityisesti C#-ohjelmointia silmällä pitäen. (Microsoft 2016c)

C#-kielen syntaksi on hyvin kuvaava mutta se on suhteellisen yksinkertainen ja helppo oppia varsinkin, jos on tutustunut C, C++ tai Java -ohjelmointikieliin. C#-kielen syntaksi on kehitetty yksinkertaistamaan monia C++ monimutkaisuuksia. C# tarjoaa ominaisuuksia kuten nollattavia tyyppiarvoja, e-numerointia, delegointia, lambda ilmauksia ja oikosiirron, joita ei esimerkiksi Java-ohjelmointikielestä löydy. C# tukee geneerisiä metodeja ja tyyppejä, jotka tarjoavat turvallisuutta ja suorituskykyä. (Microsoft 2016c)

Ohjelman kääntämisprosessi on yksinkertaisempi verrattuna C++-kieleen ja paljon joustavampi kuin Javassa. Prosessissa ei ole erillisiä otsikkotiedostoja eikä metodeja ja tyyppejä tarvitse julistaa missään tietyssä järjestyksessä. C#-kielen

lähdetiedostossa voi määritellä niin monta luokkaa, konstruktoria, liitosta tai tapahtumaa tahansa kuin tarvitsee. (Microsoft 2016c)

2.4.1 Ohjelman rakenne

C#-ohjelma koostuu joukosta ennalta laadittuja käskyjä ja komentoja, joita ovat tunnisteet ja avainsanat, muuttujat, lauseet ja lausekkeet, kommentit ja operaattorit. Aiemmin mainitut käskyt ja komennot tietokone kääntää ja sen jälkeen suorittaa ne.

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5
6  namespace EsimerkkiOhjelma
7  {
8      class Program
9      {
10         static void Main(string[] args)
11         {
12             //Esitellään muuttuja ja annetaan sille alkuarvo
13             string viesti = "Hei!";
14             //Kirjoitetaan muuttujan arvo ruudulle
15             Console.WriteLine(viesti);
16         }
17     }
18 }
```

Kuva 1. C#-ohjelman perusrakenne.

Kuvassa 1 on C#-ohjelman perusrakenne ja seuraavassa listassa vielä selitettynä esimerkkiohjelman syntaksi rivi riviltä:

- usingilla määritellään mitä nimiavaruuksia ohjelma käyttää
- namespacella määritellään uusi nimiavaruus
- class Programilla määrittellään uusi luokka
- static void Main (string[] args) määritellään metodi Main

- string viesti määrittelee muuttujan viesti, joka on tyyppiä string ja sille annetaan arvo Hei
- console.WriteLine(viesti); kutsuu Console -luokan metodin (WriteLine), jolla on parametrina muuttuja viesti
- { } merkkejä käytetään koodilohkon aloituksessa ja lopetuksessa
- C#-ohjelmoinnissa jokainen lause päättyy puolipisteeseen. (Kolari 2014a)

2.4.2 Tietotyypit

Tietotyypit on mahdollista jakaa arvo- ja viittaustyyppeihin. Arvotyyppeihin kuuluvat bool, char, enum, struct ja numeeriset tyypit ja viittaustyyppeihin kuuluvat object, string, class, delegate ja interface. Arvotyypeissä arvo on suoraan sisällä ja niiden muisti varataan pinosta. Viittaustyypeissä pinossa on viittaus keon muisti-osoitteeseen, josta viittaustyypit varaavat muistin. (Kolari 2014b) Tietotyyppejä voi joskus olla tarve muuttaa ja silloin tarvitsee tehdä tyyppimuunnos. Tietotyyppien muuttamiseen käytettäviä metodeja ovat Parse-, TryParse- ja Convert-luokan metodit. (Kolari 2014c)

2.4.3 Suojausmääreet

Suojausmääreet määrittelevät luokkien tai niiden jäsenten saatavuuden ohjelmassa. C#-kielessä kyseisiä määreitä on neljä ja niitä pystyy määrittelemään vain yhden per luokka tai jäsen. Määreet ovat

- public = näkyy kaikille koko ohjelmassa
- protected internal = näkyy luokan sisällä, perineissä luokissa ja assemblyn sisällä
- internal = näkyy assemblyn sisällä
- protected = näkyy luokan sisällä ja perineissä luokissa
- private = näkyy vain luokan sisällä. (Microsoft 2016d)

2.4.4 Ohjausrakenteet

Ohjausrakenteisiin kuuluvat hyppylauseet break, continue, goto, return ja throw, ehtolause if, valintalause switch ja toistolauseet for, foreach, while ja do…while.

Hyppylauseita ja ehtolausetta if käytetään ohjelmassa välittömään siirtymiseen kohdasta toiseen. Ehtolause if:in ja hyppylauseiden erona on se, että if-ehtolauseen ehdon pitää olla ensin täyttynyt ennen kuin ohjelmassa voi siirtyä eteenpäin. Valintalausetta switch käytetään erilaisten toimintojen suorittamiseen erilaisten ehtojen perusteella. Toistolauseiden tarkoitus on toistaa tiettyä ehtoa niin kauan kuin kyseinen ehto pätee ja sen jälkeen ohjelmassa siirrytään eteenpäin. (Kolari 2014d; Moghadampour 2012, 109-115)

2.4.5 Olio-ohjelmointi ja luokat

Olio-ohjelmointi on ohjelmoinnissa lähestymistapa, jossa ohjelmointiongelmat ratkaistaan olioiden yhteistoiminnalla. Olio-ohjelmoinnin tarkoitus siis on auttaa kirjoittamaan ohjelmia, joilla voidaan suorittaa hyvinkin monimutkaisia tehtäviä. Olio-ohjelma on siis kokoelma toistensa kanssa yhteistyössä toimivia olioita. Olio-ohjelmoinnin perusosia ovat luokat, jotka kokoavat yhteen tietyn toiminnallisuuden, jota voidaan käyttää ohjelmassa moneen kertaan. Attribuutit ja metodit ovat luokan jäseniä ja luokan sisältö määräytyy sen jäsenten mukaan. (Moghadampour 2012, 134-135)

Olion luominen tapahtuu määrittelemällä ensin luokka, josta sen jälkeen luodaan instansseja eli olioita. Olion luonti tapahtuu new-operaattorilla, jolla varataan tietokoneen muistista tila oliota varten. (Moghadampour 2012, 139) Kuvassa 2 luodaan ensin luokka Henkilö, jolle määritellään attribuutti nimi. Luokan määrittelyn jälkeen luodaan luokan Henkilö olio nimeltä pentti, jolle annetaan nimi attribuutin arvoksi Pentti.

```
class Henkilö
{
    public string nimi;
}

static void Main(string[] args)
{
    Henkilö pentti;
    pentti = new Henkilö();
    pentti.nimi = "Pentti";
    Console.WriteLine(pentti.nimi);
}
```

Kuva 2. Olion luonti.

2.4.6 Käytännöt

C#-ohjelmoinnissa ei ole määriteltyjä koodausstandardeja mutta tiettyjä suositeltuja käytäntöjä on, joita ohjelmoijan kannattaa noudattaa. Käytäntöjen tarkoitus on tehdä ohjelmakoodista johdonmukaista ja helposti ymmärrettävää. Käytännöt myös helpottavat ohjelmakoodin kopiointia, hallintaa ja muokkaamista. Hyviä käytäntöjä muistaa ovat

- nimeämiskäytännöt
- ulkoasu käytännöt
- kommentointi käytännöt
- kieli suuntaviivat. (Microsoft 2016e)

2.5 Tietokanta

Tietokanta on toisiinsa liittyvistä tiedoista koottu kokoelma, jolla on jotakin merkitystä ja jonka tietoja halutaan käyttää johonkin tarkoitukseen. Yleensä tietokanta suunnitellaan niin, että kukin tieto tallennetaan vain kerran eli pyritään välttämään turhaa toistoa. Yleisimmin tietokannat ovat relaatiotietokantoja.

2.6 Kyselykieli SQL

SQL on standardisoitu kyselykieli, jolla tehdään relaatiotietokantoihin erilaisia hakuja, muutoksia ja lisäyksiä. Tunnettuja relaatiotietokantojen hallintaohjelmia ovat muun muassa MySQL, Access, MariaDB ja Microsoft SQL Server. SQL-kyselykielessä on tietyt standardikomennot, jotka ovat samat kaikissa tietokannan hallintaohjelmissa. Standardikomentoja ovat ”Select”, ”Insert”, ”Update”, ”Delete”, ”Create” ja ”Drop”. (Koulutus- ja konsultointipalvelu KK Mediat 2016)

MySQL on avoimen lähdekoodin relaatiotietokannan hallintaohjelma ja se on suosittu erityisesti web-palveluiden tarjoajien keskuudessa. MySQL:ää käytetään selaimen kautta vaikkakin tietokantajärjestelmä toimii palvelimella. MySQL:ssä tiedot tallennetaan tauluihin, jotka sisältävät rivejä ja sarakkeita. Alun perin MySQL oli oma yrityksensä mutta nykyään MySQL:n omistaa Oracle Corporation. (W3Schools 2016)

MariaDB on erittäin suosittu tietokantapalvelin. Se on MySQL:n alkuperäisiltä tekijöiltä ja sekin kuten MySQL on avoimen lähdekoodin ohjelma. MariaDB:n sanotaan olevan korvaaja MySQL:lle ja sen sanotaan olevan nopeampi ja skaalautuvampi. Siinä on myös uusia ominaisuuksia ja vähemmän ohjelmointivirheitä verrattuna MySQL:ään. (MariaDB Foundation 2016)

Microsoft Sql Server on Microsoftin kehittämä relaatiotietokantojen hallintajärjestelmä ja tietokantapalvelin. Se on kehitetty C ja C++ ohjelmointikielillä ja se toimii Microsoftin omalla Windows käyttöjärjestelmällä mutta siinä on myös tuki Linux käyttöjärjestelmiin. (Microsoft 2016f)

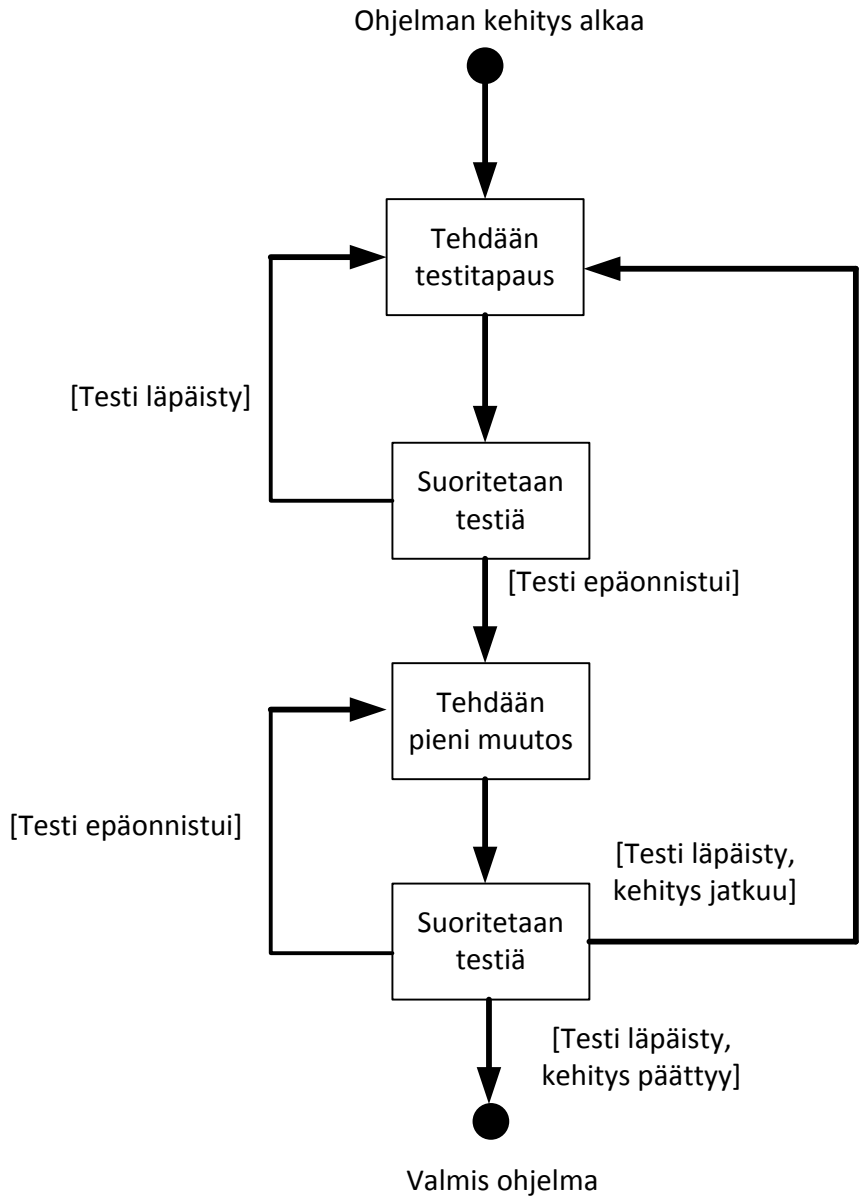
3 TESTAUS

Ohjelmistonkehitykseen kuului myös muutakin kuin koodaamista, kuten testaus. Testaus tarkoittaa sellaista työtä, jolla varmistetaan että ohjelmisto on sellainen kuin suunniteltiin ja ominaisuudet toimivat niin kuin on tarkoitus. Testaus on erittäin tärkeä osa ohjelmistokehitystä, sillä ilman testausta emme saisi tietää, toimiiko ohjelmisto, ennen kuin se luovutetaan asiakkaalle tai käyttöön. Itse käytimme niin sanottua manuaalista testausta työssämme emmekä automatisoineet sitä. Manuaalinen testaus vie tietysti aikaa enemmän, mutta automaattinen testaus ei välttämättä voi korvata ihmistä täysin. Manuaalinen testaus tarkoittaa, että klikkailimme ja täytimme kentät omin käsin, jotta näemme miten ohjelmisto reagoi ja tuleeko virheilmoituksia. (Barber 2012)

Omaksuimme siis kehitysmenetelmäksemme testivetoisen ohjelmistokehityksen. Tärkeää on huomata, että testivetoinen ohjelmistokehitys ei ole pelkästään testausmalli vaan myös ohjelmistonkehitysmalli. Testivetoinen ohjelmistokehitys tarkoittaa yleisesti, että projektia viedään eteenpäin testitapausten avulla. Ensin pohditaan testitapaus, jonka tekeillä olevan ohjelman tulee toteuttaa iteraation aikana. Eli ensin suunnitellaan testitapaus, jonka jälkeen kehitetään ohjelmaa niin kauan, että kyseinen testitapaus toteutuu. Testitapauksen suorittamisen jälkeen voidaan tämä tapaus niin sanotusti uudelleenkirjoittaa. Uudelleenkirjoittaminen tarkoittaa karkeasti sanottuna, että ohjelmankoodi kirjoitetaan paremmin eli teknillisesti kehittyneemmin. Tästä edetäänkin seuraavaan vaiheeseen eli uuden testitapauksen suunnitteluun, kun edellinen toteutus on todettu onnistuneeksi. Testivetoinen ohjelmistokehitys eroaa perinteisistä testausmalleista siinä, että tavallisissa malleissa testaus jätetään loppupäähän, kun taas testivetoisessa testaus on pääosassa ja on se asia, joka johtaa projektin etenemistä. (Kasurinen 2013, 148-150)

Niin kuin monessa muussakin asiassa, on testivetoisessa kehityksessä hyötyjä ja haittoja verrattuna perinteisiin malleihin. Yksi hyödyistä on se, että projektin etenee testivetoisesti eli testaus on pääasia. Tietty testitapaus täytyy saada iteraation aikana valmiiksi. Perinteisissä menetelmissä virheet korjattaisiin jälkeenpäin, mikä vie tuhottomasti aikaa, varsinkin jos virheitä alkaa löytyä useampia. Jälkeen-

päin virheitä korjattaessa saattaa myös käydä, niin että aikaisemmassa vaiheessa iteraatiota on keksitty niin sanottu purkkaratkaisu eli pikakorjaus johonkin toimimattomaan ominaisuuteen yms. Antaahan testivetoinen kehitys myös jatkuvaa palautetta siitä toimivatko ominaisuudet, ohjelmakoodi tai jokin muu asia projektissa, johon pystytään sitten tekemään nopeasti muutoksia. Ei tarvitse tehdä muuta kuin testata kehittäjän ohjelmakoodia. (Sininen meteoriitti 2013a) Kuvassa 3 näkyy, kuinka testivetoinen kehitys etenee testitapausten mukaan.



Kuva 3. Testivetoisen kehityksen toiminta.

4 PROJEKTINHALLINTA

Projektinhallinta on käytettävissä olevien niin sanottujen resurssien hallintaa sekä resurssien organisointia, jonka avulla päästään projektitehtävän loppuun sille suunnitellun laadun ja sisällön sekä budjetin mukaisesti. Projektinhallinnan avulla siis valvotaan, että asiat menevät niin kuin on suunniteltu. Yleensä projektit jaetaan karkeasti vaiheisiin, jotka ovat aloitus, suunnittelu, toteutus ja lopetus. (Jansson & Juselius 2004)

Projektinhallinnassa kaikista tärkein asia on projektin ositus eli koko kokonaisuus jaetaan pienempiin osiin, jotta asiat saadaan sujumaan ja valmiiksi. Projektinhallintaan on olemassa tekniikoita ja tehtäviä, kuten projektikokonaisuuden jäsentäminen, osakokonaisuuksien toteutus aikataulutus ja osakokonaisuuksien toteutuksien rytmitys, työprosessien suunnittelu, osatehtävien työnjaollinen vastuutus ja dokumentointi. (Viirkorpi 2000a, 32-34)

Olisi hyvä hallita työtehtävien toteutusta projektissa, jotta työt tulisi tehtyä aikataulun mukaisesti ja niin kuin ne pitäisi. On yleensä vaikea laskea paljonko tiettyihin tehtäviin tarvitaan aikaa, koska vastaan voi tulla yllättäviä vastoinkäymisiä, joihin ei ollut varauduttu. Toisaalta on pakko tehdä edes karkeita aikataulutus arvioita, että saadaan jonkunlainen realistinen aikataulu projektille. (Viirkorpi 2000b, 34)

Projektinhallintaan liitetään erilaisia tekniikoita, joista esimerkkeinä ovat ketterät menetelmät. Ketteriin menetelmiin liittyy Scrum ja Extreme Programming projektinhallinta tekniikat. Ketterät menetelmät itsessään tarkoittavat karkeasti sitä, että projekti paloitellaan pieniin iteraatioihin ja jokaisen iteraation päätteeksi ainakin ohjelmistokehityksessä tulisi versiojulkaisu. Ketterät menetelmät sopivat yleensä ohjelmistokehitykseen, jossa saattaa olla sellaisia muuttujia, joita ei pysty ennakoimaan. Ketterät menetelmät antavat tähän helpotusta sillä tavoin, että projektin aikana voidaan tarkentaa tiettyjä tarpeita yms. kun aletaan tuntea niitä niin sanottuja muuttujia, joita ei pysty aluksi ennakoimaan. (Heiramo 2010)

Ketterässä ohjelmistokehityksessä on Agile Manifesto eli ketterän ohjelmistokehityksen julistus, joka pitää sisällään mitä kyseisessä julistuksessa arvostetaan sekä millaisia periaatteita ketterässä ohjelmistokehityksessä on. Agile Manifesto sivustolla mainitaan julistuksesta näin: ”Löydämme parempia tapoja tehdä ohjelmistokehitystä, kun teemme sitä itse ja autamme muita siinä. Kokemuksemme perusteella arvostamme: Yksilöitä ja kanssakäymistä enemmän kuin menetelmiä ja työkaluja. Toimivaa ohjelmistoa enemmän kuin kattavaa dokumentaatiota. Asiakasyhteistyötä enemmän kuin sopimusneuvotteluja. Vastaamista muutokseen enemmän kuin pitäytymistä suunnitelmassa. Jälkimmäisilläkin asioilla on arvoa, mutta arvostamme ensiksi mainittuja enemmän." (Agile Manifesto 2001a.)

Ketterässä ohjelmistokehityksessä on periaatteita, jotka ohjaavat ketterän ohjelmistokehityksen menetelmiä. Kyseisiä periaatteita on 12 ja periaatteet on koottu merkittävien ketterän kehityksen puolestapuhujien toimesta.

- ”Asiakas täytyy tyydyttää toimittamalla säännöllisesti ja aikaisessa vaiheessa asiakkaan tarpeet täyttäviä versioita ohjelmistosta. Tämä on tärkein tavoite.”
- ”Otamme vastaan muuttuvat vaatimukset myös kehityksen myöhäisessä vaiheessa. Ketterät menetelmät hyödyntävät muutosta asiakkaan kilpailukyvyn edistämiseksi.”
- ”Toimitamme versioita toimivasta ohjelmistosta säännöllisesti, parin viikon tai kuukauden välein, ja suosimme lyhyempää aikaväliä.”
- ”Liiketoiminnan edustajien ja ohjelmistokehittäjien tulee työskennellä yhdessä päivittäin koko projektin ajan.”
- ”Rakennamme projektit motivoituneiden yksilöiden ympärille. Annamme heille puitteet ja tuen, jonka he tarvitsevat ja luotamme siihen, että he saavat työn tehtyä.”
- ”Tehokkain ja toimivin tapa tiedon välittämiseksi kehitystiimille ja tiimin jäsenten kesken on kasvokkain käytävä keskustelu.”
- ”Toimiva ohjelmisto on edistymisen ensisijainen mittari.”

- ”Ketterät menetelmät kannustavat kestävään toimintatapaan. Hankkeen omistajien, kehittäjien ja ohjelmiston käyttäjien tulisi pystyä ylläpitämään työtahtinsa hamaan tulevaisuuteen.”
- ”Teknisen laadun ja ohjelmiston hyvän rakenteen jatkuva huomiointi edesauttaa ketteryyttä.”
- ”Yksinkertaisuus - tekemättä jätettävän työn maksimointi - on oleellista.”
- ”Parhaat arkkitehtuurit, vaatimukset ja suunnitelmat syntyvät itse organisoituvissa tiimeissä.”
- ”Tiimi tarkastelee säännöllisesti, kuinka parantaa tehokkuuttaan, ja mukauttaa toimintaansa sen mukaisesti.” (Agile manifesto 2001b.)

Scrum on ketteriin projektinhallinta metodeihin kuuluva tekniikka. Se keskittyy suurimmilta osin siihen, kuinka hallita tehtäviä tiimipohjaisessa kehitysympäristössä. Yleensä puhutaan Scrum-tiimistä ja niitä voi olla useampiakin kuin yksi. Niin kuin aikaisemmin mainitsimme yleisesti ketteristä menetelmistä, kuuluu Scrumiin erilaiset iteraatiot tai syklit. Scrumissa tällaisia ovat esimerkiksi sprintti. Sprintti on yksi kehitysjakso, jonka jälkeen ainakin periaatetasolla pitäisi olla julkaisukelpoinen tuote. Scrumissa on vain kolme roolia: tuotteen omistaja, Scrummestari, joka hoitaa asiat niin, että tiimi pystyy toimimaan tehokkaasti sekä tiimi, joka tekee projektia. Tiimin kasataan yleensä henkilöistä, jotka omaavat tarvittavan osaamisen. (Sininen meteoriitti 2013b)

Ennen Scrum-projektin aloitusta mietitään, mitä projektilta halutaan, jonka jälkeen muodostetaan lista siitä, mitä ominaisuuksia halutaan tuotteeseen. Lista muodostetaan tuotteen omistajan kanssa, joka myös mainitsee, mitkä asiat ovat tärkeyslistalla ensimmäisenä. Tämän jälkeen suunnitellaan sprintti eli niitä asioita, jotka yhden kehitysjakson aikana tulee toteuttaa. Tärkeä huomautus on, ettei sprintin aikana muuteta sprinttiin kuuluvia vaatimuksia. Scrumissa joka päivä projektin aikana tiimi kokoontuu, jolloin tehdään tilannekatsaus. Tiimin jäsenten täytyy vastata kolmeen kysymykseen tilannekatsauksen aikana, mitkä ovat: mitä teit edellisen päivän aikana?, mitä aiot tehdä seuraavana päivänä?, mitkä tekijät estävät sinua saavuttamasta sprintin tavotteita? Sprintin päätteeksi tuotetta esitetään

sen omistajalle sekä tarkastellaan sprinttiä Scrum-mestarin, tiimin ja sen omistajan kanssa yhdessä. (Schwaber & Sutherland 2013)

”Scrum ei rakennu valikoidusta joukosta käytäntöjä vaan pohjautuu muutamaan arvoon.”. (Sininen meteoriitti 2013b) Scrumin ensimmäinen arvo on sitoutuminen. Se tarkoittaa sitä, että tiimi sitoutuu yhteiseen päämäärään täysillä ja tiimin täytyy saada kaikki tuki, jotta kyseinen päämäärä saadaan toteutettua. Toinen Scrumin arvoista on keskittyminen, jolla tarkoitetaan sitä että, kun keskitytään vain muutamaan asiaan kerralla, saadaan paremmin tehtyä asioita yhdessä ja luotua hyvä tuote. Kolmantena arvona on kunnioitus, joka lyhykäisyydessään pitää sisällään sen faktan, että jokainen ihminen on erilainen ja ihmistä pitää kunnioittaa juuri sellaisenaan. Neljäntenä arvona pidetään avoimuutta eli Scrumissa asiat ovat avoimia. Asiat esitetään julkisesti, jotta niihin pystytään puuttumaan. Viimeisinpänä, mutta ei vähäisimpänä on rohkeus. Rohkeutta ei välttämättä tarvitse tarkemmin selittää, mutta esimerkiksi omien virheiden myöntäminen ja rohkeus sitoutua kaikkiin edellä mainittuihin arvoihin sekä joskus tehdä asiat eri tavoin kuin on suositeltavaa. (Sininen meteoriitti 2013b; Scrum Alliance 2016)

5 JÄRJESTELMÄN MÄÄRITTELY

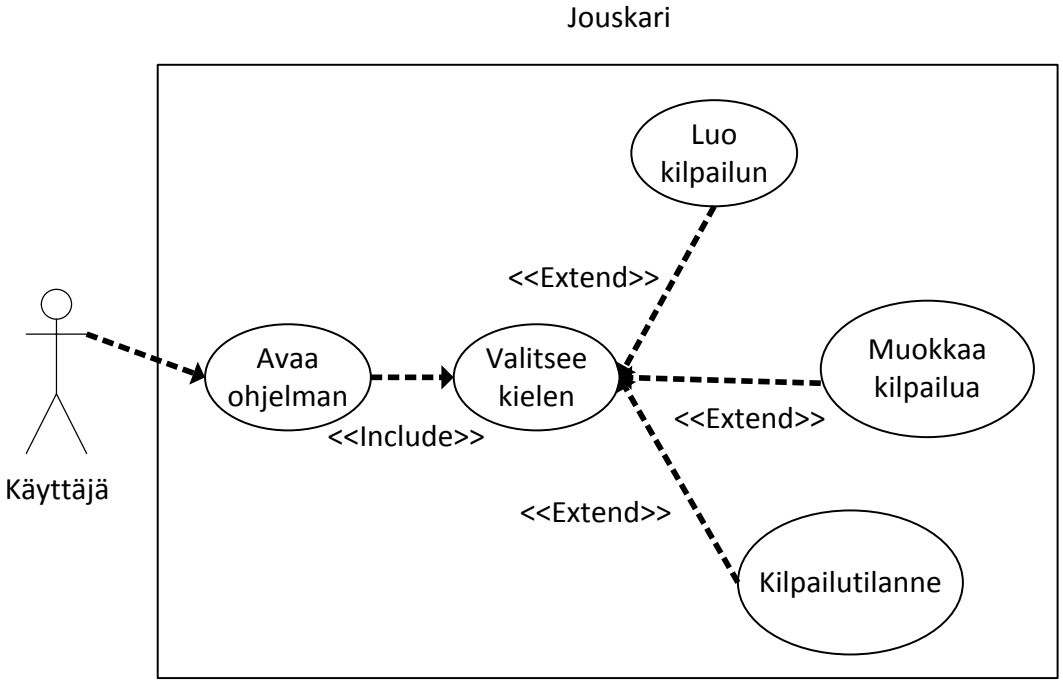
Aloitimme työn toimeksiantajan kanssa palaverilla, jossa yritimme saada ohjelmasta vaatimusmäärittelyn, jotta pystyisimme aloittamaan ohjelman kehityksen. Täydensimme toimeksiantajalta saamaamme vaatimusmäärittelyä tutkimalla toimeksiantajan vanhan ohjelman toimintaa, mistä saimme myös pohjaa käyttötapausten määrittelyyn. Vaatimuslistalle oli tarkoitus saada päivitystä ensimmäisen ohjelman prototyypin esittelyn jälkeen.

5.1 Vaatimusmäärittely

Ohjelman aloitusruudussa pitäisi olla kielivalikko, jossa valitaan ohjelman kieleksi joko suomi, ruotsi tai englanti. Ohjelman ohjelmavalikko on kilpailuvalikko eli tässä käyttäjä valitsee, minkä kilpailun ohjelma suorittaa. Ohjelmaan pitäisi tulla myös kilpailun luontivalikko, jossa käyttäjä voi luoda kilpailuja. Luontivalikossa pitäisi voida määritellä ampuma-aika, valmistautumisaika, nuolten määrä ja kilpailun nimi. Luontivalikkoon pitäisi saada myös helpotuksia esimerkiksi, kun käyttäjä tekee yhden ampumavuoron, niin pitäisi sitä ampumavuoroa voida kopioida jne. Kilpailuja pitäisi myös olla mahdollista muokata ja muokkauksessa pitäisi voida muokata kilpailun nimeä, ampuma-aikaa, nuolten määrää ja kilpailun valmistautumisaikaa. Kilpailua aloittaessa pitäisi olla mahdollista määritellä ampumavuorot ja onko kilpailu finaali vai ei. Meneillään olevan kilpailun aikana ohjelmassa pitäisi näkyä yleisölle, kenen ampumavuoro on, paljonko kilpailijalla on valmistautumisaikaa ja ampuma-aikaa. Valmistautumisaikaa määritellessä, jos arvoksi laittaa nollan tarkoittaa se, että valmistautumisaikaa ei ole. Finaalissa ruutu jaetaan kahtia kahden ampujan kesken niin, että molemmilta kilpailijoilta näkyy ampujan nimi, ammuntaa käytettävä aika ja ammutut pisteet.

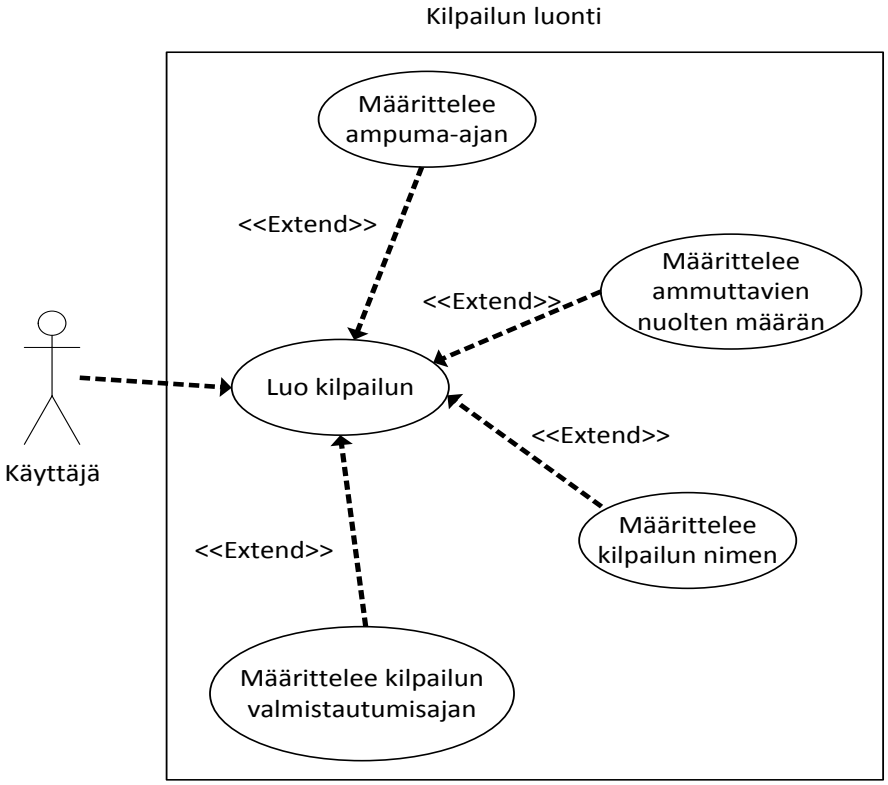
5.2 Käyttötapaukset

Aloimme pohtia toteutettavan ohjelman käyttötapauksia toimeksiantajalta saamamme vaatimusmäärittelyn pohjalta. Kuvassa 4 on ohjelman perustoiminnot eli käyttäjä avaa ohjelman, jonka jälkeen käyttäjä valitsee kielen. Seuraavaksi käyttäjä voi valita, luodaanko uusi kilpailu, poistetaanko vai muokataanko kilpailua tai aloitetaanko uusi kilpailu.



Kuva 4. Jouskarin käyttötapauskaavio.

Ensimmäisen käyttötapauskaavion jälkeen teimme jokaisesta käyttötapauksesta oman tarkemman kaavion. Kuvassa 5 näkyy ”Kilpailun luonti” -käyttötapaus.



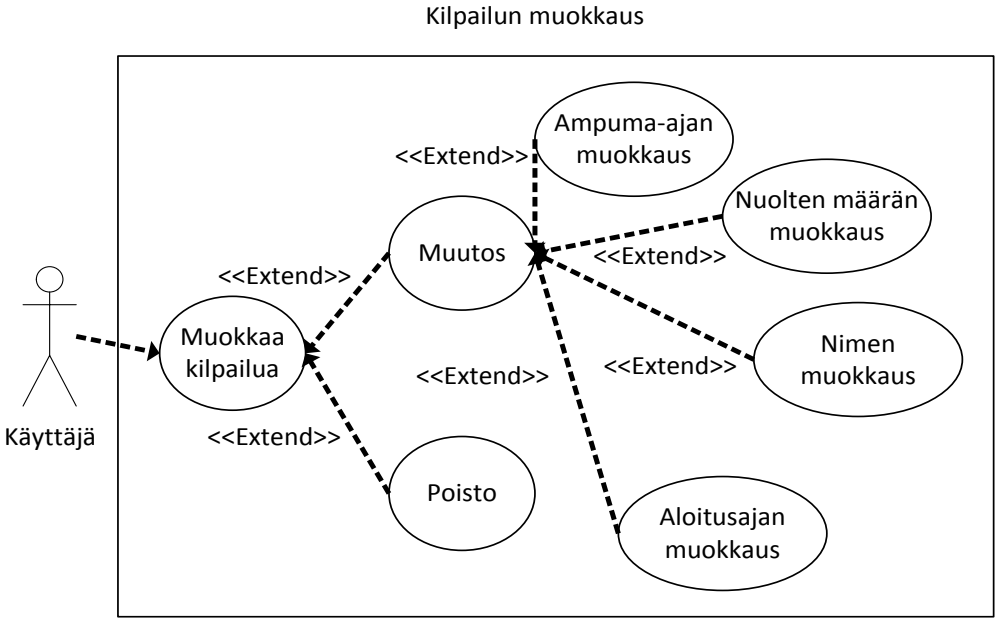
Kuva 5. Käyttötapauskaavio kilpailun luonnista.

Taulukossa 1 on ”Kilpailun luonti” -käyttötapauksen sanallinen muoto.

Taulukko 1. ”Kilpailun luonti” -käyttötapauksen sanallinen muoto.

Käyttötapaus	Kilpailun luonti.
Aktorit	Käyttäjä.
Esiehdot	Käyttäjä haluaa luoda uuden kilpailun.
Kuvaus	Käyttäjä syöttää kilpailun tietoja kenttiin, kun tiedot on täytetty, painaa käyttäjä tallenna ja järjestelmä kirjaa tiedot tietokantaan.
Poikkeukset	Joku kentistä on tyhjä tai sisältää virheellistä tietoa.
Jälkiehdot	Tiedot on syötetty kenttiin ja käyttäjä on painanut tallenna.
Lopputulos	Kilpailu on lisätty tietokantaan.

Kuvassa 6 on käyttötapaus ”Kilpailun muokkaus”.



Kuva 6. Käyttötapauskaavio kilpailun muokkaamisesta.

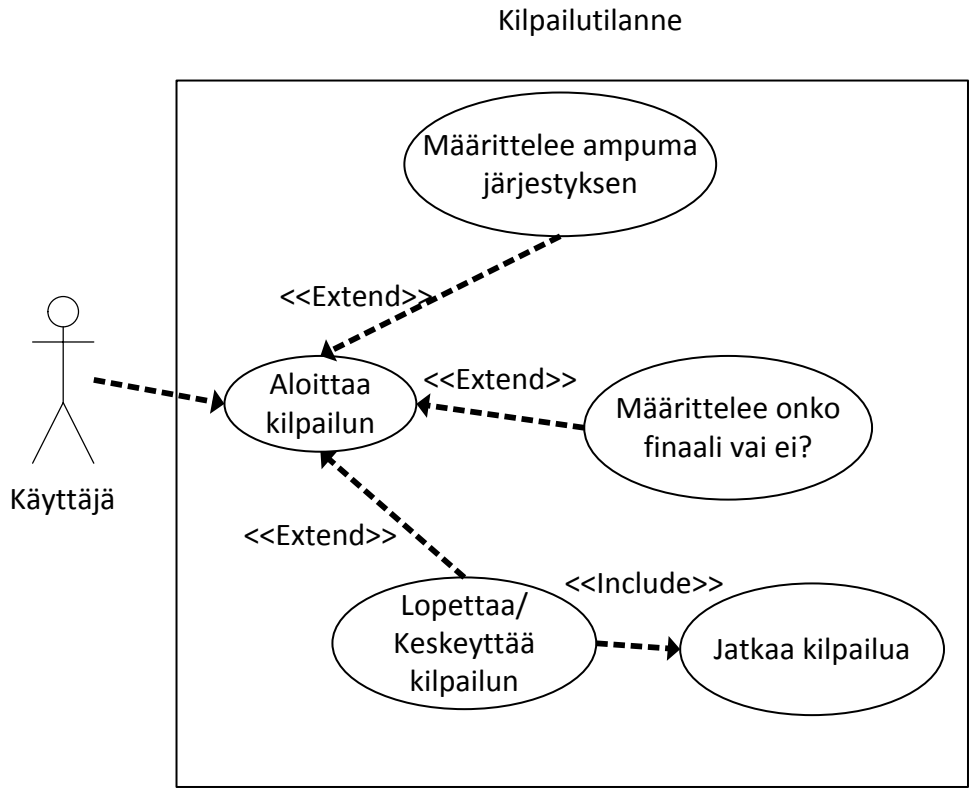
Taulukossa 2 on ”Kilpailun muokkaus” -käyttötapauksen sanallinen muoto.

Taulukko 2. ”Kilpailun muokkaus” -käyttötapauksen sanallinen muoto.

Käyttötapaus	Kilpailun muokkaus.
Aktorit	Käyttäjä.
Esiehdot	Käyttäjä haluaa muokata tai poistaa kilpailun.
Kuvaus	Käyttäjä muokkaa kilpailun tietoja kenttiin, kun tiedot on täytetty painaa käyttäjä tallenna ja järjestelmä muuttaa tiedot tietokantaan. Käyttäjä poistaa kilpailun valitsemalla kilpailun ja sen jälkeen painaa poista. Järjestelmä poistaa tiedon tietokannasta.
Poikkeukset	Joku kentistä on tyhjä, sisältää virheellistä tietoa tai tietokannassa ei ole yhtään poistettavaa kilpailua.

Jälkiehdot	Tiedot on syötetty kenttiin ja käyttäjä on painanut tallenna tai käyttäjä on valinnut kilpailun ja painanut poista.
Lopputulos	Kilpailua on muokattu tai se on poistettu.

Kuvassa 7 on käyttötapaus ”Kilpailutilanne”.



Kuva 7. Käyttötapauskaavio kilpailutilanteesta.

Taulukossa 3 on ”Kilpailutilanne” -käyttötapauksen sanallinen muoto.

Taulukko 3. ”Kilpailutilanne” -käyttötapauksen sanallinen muoto.

Käyttötapaus	Kilpailutilanne.
Aktorit	Käyttäjä.

Esiehdot	Käyttäjä haluaa aloittaa kilpailun.
Kuvaus	Käyttäjä syöttää ampumavuorot kenttiin ja valitsee onko kilpailu finaali vai ei. Kilpailun alettua käyttäjä voi lopettaa/keskeyttää kilpailun tai jatkaa keskeytettyä kilpailua.
Poikkeukset	Ampumavuoroja ei ole syötetty tai käyttäjä ei ole valinnut onko kilpailu finaali vai ei.
Jälkiehdot	Ampumavuorot on syötetty ja käyttäjä on valinnut onko kilpailu finaali vai ei. Käyttäjä lopettaa tai keskeyttää kilpailun.
Lopputulos	Kilpailu alkaa. Kilpailu lopetetaan tai keskeytetään.

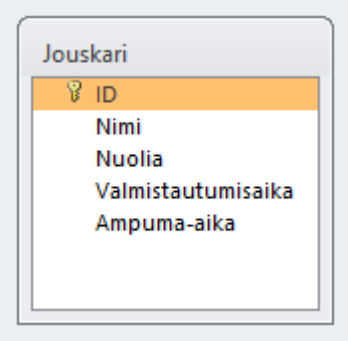
6 SUUNNITTELU JA TOTEUTUS

6.1 Jouskarin projektinhallinta

Jouskarin projektinhallinta oli pääasiassa ketterää ohjelmistokehitystä, jossa hyödynsimme Scrumia ja Extreme Programmia. Hyödynsimme Extreme Programmista pariohjelmointimenetelmää. Pariohjelmoinnin tarkoitus työtä tehdessämme oli saada mahdollisimman hyvää ja selvää ohjelmakoodia. Käytännössä siis toinen ohjelmoi ja toinen oli vieressä refaktoroimassa ohjelmakoodia koko ajan, samalla miettien olisiko kyseiselle ratkaisulle toista parempaa vaihtoehtoa. Pariohjelmoinnista oli hyötyä myös kirjoitusvirheiden löytämisessä ja se myös antoi toiselle ohjelmoijalle hyvän tauon ohjelmakoodin kirjoittamisesta. Scrumista otimme projektimme hallintaan Sprintit ja tiimipalaverit, joita oli aina kun kokoonnuimme ohjelmoimaan. Aikataulumme oli projektissa hyvin joustava ja sprintit kestivät yleensä 1-3 ohjelmointisession ajan. Aina kun sprintti päättyi, pidimme yleensä tiimipalaverin ja mietimme, mitä teemme ensi kerralla eli tässäkin käytimme itsellemme räätälöityä versiota Scrumista. Pidimme projektissamme myös kirjaa tehdyistä tunneista mutta loppua kohden huomasimme, että olisi ehkä pitänyt panostaa ajankäytön maksimointiin ja valvontaan vähän enemmän. Sprinttejä projektissa oli viisi mutta, ihan jokaisella kerralla emme saaneet versiojulkaisua Sprintin päätteeksi. Versiojulkaisuja oli kolme, koska kolmannessa ja neljännessä Sprintissä emme saaneet versiojulkaisuja tehtyä.

6.2 Tietokannan luonti

Päätimme heti ensimmäisenä luoda ohjelmaan tietokannan saatujen tietojen perusteella ja tulimme siihen tulokseen, että yhden taulun tietokannalla pystymme täyttämään tämän hetkisen vaatimuslistan tarpeet. Koska tietokantaamme tuli vain yksi taulu, päätimme heti, että helpoin tapa on tehdä paikallinen Access-tietokanta. Kuvassa 8 on tietokantamme taulu Jouskari.



Kuva 8. JouskariKanta tietokannan taulu Jouskari.

Annoimme tietokannallemme nimen JouskariKanta ja loimme siihen taulun Jouskari, jossa on kentät ID, Nimi, Nuolia, Valmistautumisaika, Ampuma-aika. ”ID” -kenttä on taulun perusavain, jonka tietotyyppi on laskuri ja sen koko on pitkä kokonaisluku. Aina kun käyttäjä luo uuden kilpailun tarvitaan sille oma ID, koska samannimisiä kilpailuja voi olla monta. ”Nimi” -kenttä on kilpailun nimi ja sen tietotyyppi on teksti ja kentän koko on 255 merkkiä. Jätimme ”Nimi” -kentälle tarkoituksella sen oletuskoon 255 merkkiä, koska emme tiedä, kuinka pitkiä kilpailun nimet tulevat olemaan. ”Nuolia” -kenttään tulee ammuttavien nuolien määrä kilpailussa per kilpailija ja sen tietotyyppi on luku ja kentän koko kokonaisluku. ”Valmistautumisaika” -kentässä on kuinka kauan kilpailijalla on aikaa valmistautua omalla vuorollaan ja ”Ampuma-aika” -kenttään tulee kuinka kauan kilpailijalla on aikaa ampua kaikki nuolensa. Molempien kenttien tyyppi on luku ja koko kokonaisluku, koska molemmat ajat määritellään sekunteina. Kuvassa 9 näkyvät tauluun Jouskari syötetyt tiedot.

Jouskari

ID	Nimi	Nuolia	Valmistautumisaika	Ampuma-aika
1	Tesitkilpailu	6	30	120
2	Testikilpailu2	10	20	300
3	Testikilpailu3	15	30	400
4	Finaali	10	25	300

Kuva 9. ”Jouskari” -taulun tiedot.

6.3 Ohjelman toteutus

6.3.1 Yhteisalue

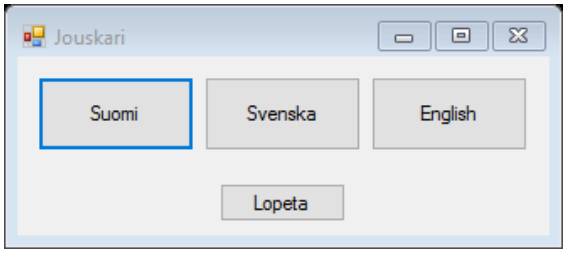
Ensimmäisenä loimme ohjelmaan yhteisen alueen YhteisetTiedot, jossa määrittelemme julkiset muuttujat, data-adapterit, alustamme lomakkeet, tietokantayhteyden ja datasetit. Yhteisalueen tarkoituksena on luoda ohjelmaan tietoja, jotka näkyvät jokaiselle ohjelman osalle. Esimerkiksi, jos halutaan välittää tietoja toiselta lomakkeelta toiselle tai käyttää jotakin tiettyä muuttujaa moneen kertaan, jolloin sitä voidaan nollata mistä tahansa osasta ohjelmaa. Yhteisalueen kaikille tiedoille annettiin suojausmääre ”Public”, koska ilman sitä tiedot eivät olisi näkyvissä ohjelman muissa osissa. Kuvassa 10 näkyy yhteisalueen ohjelmakoodi.

```
public static class YhteisetTiedot
{
    public static frmAlku alku = new frmAlku();
    public static frmJouskariFin JouskariFin = new frmJouskariFin();
    public static frmLuoKisa LuoKisa = new frmLuoKisa();
    public static frmMuoKisa MuoKisa = new frmMuoKisa();
    public static frmKilpailu Kilpailu = new frmKilpailu();
    public static frmSaaAmpua SaaAmpua = new frmSaaAmpua();

    // Tietokanta yhteys
    public static OleDbConnection yhteys = new OleDbConnection
        ("Provider=Microsoft.ACE.OLEDB.12.0;Data Source=JouskariKanta.accdb;Persist Security Info=False;");

    // Datasetin luonti
    public static DataSet dsJouskariKanta = new DataSet();

    // DataAdapterit
    public static OleDbDataAdapter daJouskari;

    // Julkiset muuttujat
    public static string JInd, JampumaAika, JvalmistautumisAika, Jnuolia, Jnimi, JAjärj;

    public static int yhI, yhE, yhAmp, yKierros, yAmpujNro, yKopioi;

}
```

Kuva 10. Jouskari ohjelman yhteisalue YhteisetTiedot.

6.3.2 Aloitusnäkymä

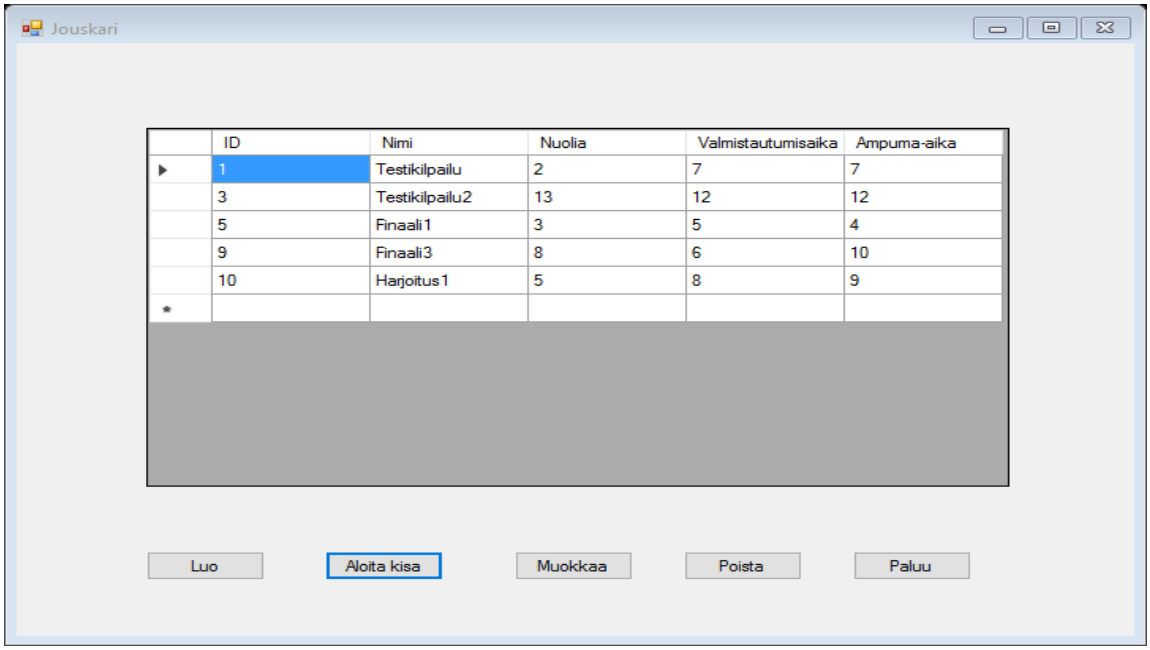
Yhteisen alueen luonnin jälkeen teimme ensimmäisen lomakkeen nimeltä frmAlku, josta ohjelma käyttö aloitetaan. Alku on yksinkertainen näyttö, jossa on mahdollisuus valita kieli, jolla haluaa ohjelmaa käyttää tai lopettaa ohjelma. Kuvassa 11 näkyy aloitusnäkymä.



Kuva 11. Aloitusnäyttö.

6.3.3 Perusnäkymä

Aloitusnäytön jälkeen siirrytään ohjelman perusnäkymään, jossa näkyvät kaikki kilpailut, jotka tietokantaan on lisätty. Kyseisessä näytössä on mahdollista luoda uusia kilpailuja, poistaa tarpeettomia kilpailuja, muokata muokkauksen tarpeessa olevia kilpailuja, aloittaa valittu kilpailu ja palata takaisin aloitusnäyttöön. Kuvassa 12 on ohjelman perusnäkymä.



Kuva 12. Jouskari ohjelman perusnäkymä.

Perusnäkymää tehdessämme päätimme, että tietojen esittämiseen tässä ohjelmassa käytämme ”DataGridView” -kontrollia. Kyseinen kontrolli esittää tiedot helposti ymmärrettävällä tavalla ja käyttäjän on helppo valita, mitä kilpailua halutaan käsitellä. ”dgvKilpailut” -kontrollin täyttö ja päivitys tapahtuvat, kun perusnäkymän näkyvyys vaihtuu aliohjelmien Kilpailut() ja Tyhjennä() avulla.

```
87          private void frmJouskariFin_VisibleChanged(object sender, EventArgs e)
88          {
89              // Kutsutaan aliohjelmaa tyhjennä
90              Tyhjennä();
91              YhteisetTiedot.yhteys.Open();
92              YhteisetTiedot.daJouskari = new OleDbDataAdapter("SELECT * from Jouskari", YhteisetTiedot.yhteys);
93              YhteisetTiedot.daJouskari.Fill(YhteisetTiedot.dsJouskariKanta);
94
95              // Kutsutaan aliohjelmaa kilpailut
96              Kilpailut();
97
98              YhteisetTiedot.yhteys.Close();
99          }
100
101         private void Tyhjennä()
102         {
103             if (YhteisetTiedot.dsJouskariKanta.Tables.Contains("Jouskari"))
104             {
105                 YhteisetTiedot.dsJouskariKanta.Tables["Jouskari"].Clear();
106             }
107             YhteisetTiedot.daJouskari = new OleDbDataAdapter("SELECT * from Jouskari", YhteisetTiedot.yhteys);
108             YhteisetTiedot.daJouskari.Fill(YhteisetTiedot.dsJouskariKanta, "Jouskari");
109
110         }
111
112         private void Kilpailut()
113         {
114             if (YhteisetTiedot.dsJouskariKanta.Tables.Contains("Jouskari"))
115             {
116                 YhteisetTiedot.dsJouskariKanta.Tables["Jouskari"].Clear();
117             }
118
119             YhteisetTiedot.daJouskari.Fill(YhteisetTiedot.dsJouskariKanta, "Jouskari");
120             dgvKilpailut.DataSource = YhteisetTiedot.dsJouskariKanta.Tables["Jouskari"];
121         }
```

Kuva 13. dgvKilpailut -kontrollin täyttö ja tyhjennys.

Kuvassa 13 lomakkeen frmJouskariFin näkyvyys Propertyn muuttuessa kutsutaan ensin aliohjelmaa Tyhjennä(), minkä jälkeen alustetaan uusi data-adapteri daJouskari, jolla täytetään datasetti dsJouskariKanta. Datasetin täytön jälkeen kutsutaan aliohjelmaa Kilpailut(), joka täyttää ”dgvKilpailut” -kontrollin tietokannasta haetuilla tiedoilla.

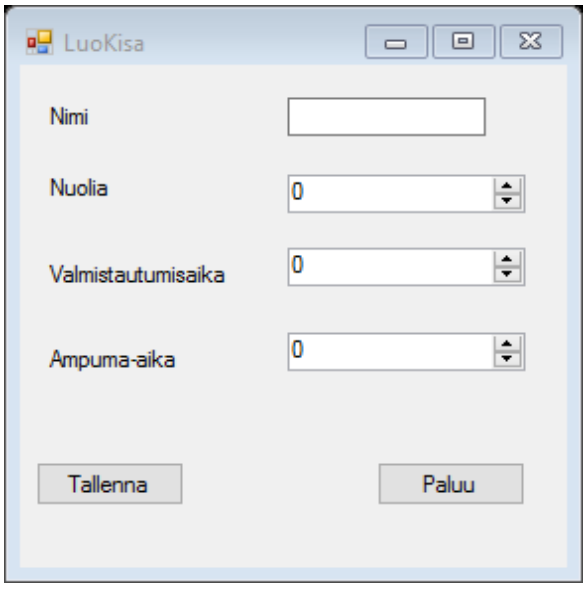
6.3.4 Kilpailun luonti

Uusien kilpailujen luontia varten teimme ”Luo” -napin, jolla käyttäjä voi luoda uusia kilpailuja. Painaessa nappia Luo siirrytään lomakkeelle frmLuoKisa. Kuvassa 14 näkyy ”btnLuoKisa” -napin ohjelmakoodi.

```
private void btnLuoKisa_Click(object sender, EventArgs e)
{
    YhteisetTiedot.LuoKisa.Show();
    this.Hide();
}
```

Kuva 14. ”Luo” -napin ohjelmakoodi.

Uuden kilpailun luontinäkymässä käyttäjälle näytetään neljä kenttää, joihin syötetään kilpailun tiedot ja napit Tallenna ja Paluu. Kuvassa 15 on kilpailun luonti näkymä.



Kuva 15. Kilpailun luontinäkymä.

Kaikki tarpeellinen ohjelmakoodi kilpailun luontia varten on laitettu ”Tallenna” -napin taakse. Napin Tallenna ohjelmakoodi näkyy kuvassa 16.

```
private void btnTallenna_Click(object sender, EventArgs e)
{
            OleDbCommand cmd = new OleDbCommand();
            cmd.CommandType = CommandType.Text;
            cmd.CommandText = "insert into Jouskari ([Nimi],[Nuolia],[Valmistautumisaika],[Ampuma-aika]) values (?,?,?,?)";
            cmd.Parameters.AddWithValue("@nimi", txtNimi.Text);
            cmd.Parameters.AddWithValue("@nuolia", nudNuoli.Value);
            cmd.Parameters.AddWithValue("@valmistautumisaika", nudValmistautumisAika.Value);
            cmd.Parameters.AddWithValue("@ampuma-aika", nudAmpumaAika.Value);
            cmd.Connection = YhteisetTiedot.yhteys;
            YhteisetTiedot.yhteys.Open();
            cmd.ExecuteNonQuery();
            System.Windows.Forms.MessageBox.Show
                ("Tiedot on lisätty onnistuneesti", "Ilmoitus", MessageBoxButtons.OK, MessageBoxIcon.Information);
            YhteisetTiedot.yhteys.Close();
}
```

Kuva 16. Kilpailun luonti lomakkeen ”Tallenna” -napin ohjelmakoodi.

”Tallenna” -nappia painaessa luodaan tietokanta komento cmd ja määritellään sen tyyppi ja itse komennon sisältö. Komennon cmd parametreiksi tulee käyttäjän syöttämät tiedot lomakkeen neljään kenttään, minkä jälkeen avataan yhteys tietokantaan, suoritetaan komento cmd ja ilmoitetaan käyttäjälle, että tiedot on lisätty onnistuneesti ja lopuksi vielä suljetaan tietokantayhteys.

6.3.5 Kilpailun muokkaus

Kilpailun muokkaus tapahtuu käyttäjän valitsemalle muokattavalle kilpailulle ”dgvKilpailut” -kontrollista ja painamalla ”Muokkaa” -nappia. Napin painalluksen jälkeen suoritetaan napin Muokkaa ohjelmakoodi ja siirrytään lomakkeelle frmMuoKisa. Arvot kontrollista dgvKilpailut, muokkauslomakkeelle saadaan ”Muokkaa” -napin ohjelmakoodilla, jossa ”dgvKilpailut” -kontrollin valitun rivin arvot siirretään YhteisAlueen muuttujiin. ”Muokkaa” -napin koodi näkyy kuvassa 17.

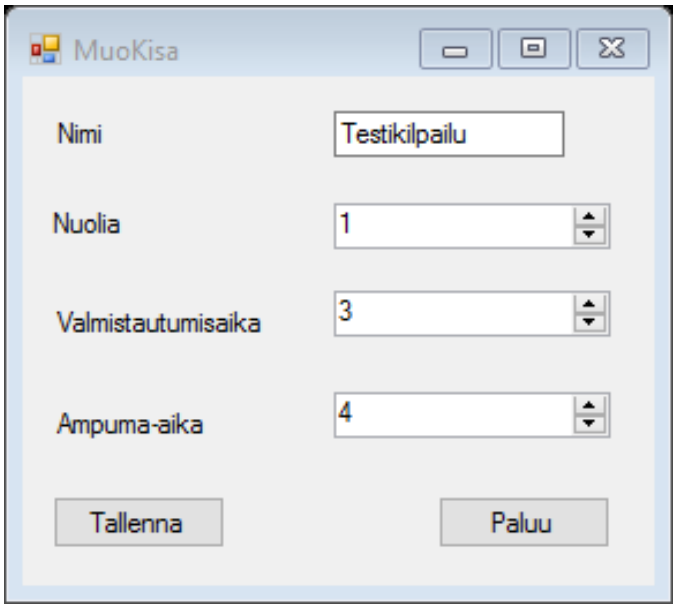
```
private void btnMuokkaa_Click(object sender, EventArgs e)
{
    YhteisetTiedot.JInd = this.dgvKilpailut.CurrentRow.Cells[0].Value.ToString();
    YhteisetTiedot.Jnimi = this.dgvKilpailut.CurrentRow.Cells[1].Value.ToString();
    YhteisetTiedot.Jnuolia = this.dgvKilpailut.CurrentRow.Cells[2].Value.ToString();
    YhteisetTiedot.JvalmistautumisAika = this.dgvKilpailut.CurrentRow.Cells[3].Value.ToString();
    YhteisetTiedot.JampumaAika = this.dgvKilpailut.CurrentRow.Cells[4].Value.ToString();

    YhteisetTiedot.MuoKisa.Show();
    this.Hide();
}
```

Kuva 17. ”Muokkaa” -napin ohjelmakoodi.

frmMuoKisa lomakkeen avautuessa haetaan aiemmin valitun kilpailun tiedot alueen YhteisetTiedot muuttujista ja syötetään ne oikeisiin kenttiin odottamaan muokkausta. Kyseisin tapahtuman voi nähdä kuvista 18 ja 19.

```
private void frmMuoKisa_Load(object sender, EventArgs e)
{
    txtNimi.Text = YhteisetTiedot.Jnimi;
    nudNuoli.Value = decimal.Parse(YhteisetTiedot.Jnuolia);
    nudValmistautumisAika.Value = decimal.Parse(YhteisetTiedot.JvalmistautumisAika);
    nudAmpumaAika.Value = decimal.Parse(YhteisetTiedot.JampumaAika);

}
```

Kuva 18. Lomakkeen lataus ja tietojen haku.



Kuva 19. Muokkaus lomake.

Muokkauksen jälkeen joko tallennetaan muutokset tai painetaan ”Paluu” -nappia jolloin muutoksia ei tallenneta. Nappia Tallenna painaessa avataan yhteys tietokantaan, jonka jälkeen luodaan komento cmd ja määritellään sen sisältö muokatuilla arvoilla. Komennon cmd määrittelyn jälkeen suoritetaan kyseinen komento ja sen jälkeen komento tallentaa tehdyt muutokset tietokantaan ja lopuksi suljetaan yhteys. Tämän jälkeen ohjelma ilmoittaa käyttäjälle, että kilpailun tietoja on muokattu onnistuneesti. Kuvassa 20 näkyy ”Tallenna” -napin ohjelmakoodi.

```
private void btnTallenna_Click(object sender, EventArgs e)
{
    YhteisetTiedot.yhteys.Open();

    OleDbCommand cmd;

    cmd = new OleDbCommand("UPDATE Jouskari SET [Nimi] = '"+txtNimi.Text+"', [Nuolia] = "+nudNuoli.Value+
        ", [Valmistautumisaika] = "+nudValmistautumisAika.Value+", [Ampuma-aika] = "+nudAmpumaAika.Value+
        " WHERE [ID] ="+YhteisetTiedot.JInd+"", YhteisetTiedot.yhteys);
    cmd.ExecuteNonQuery();
    YhteisetTiedot.yhteys.Close();

    MessageBox.Show("Tiedot muutettu");

    YhteisetTiedot.JouskariFin.Show();
    this.Hide();


}
```

Kuva 20. Kilpailun muokkaus lomakkeen ”Tallenna” -napin ohjelmakoodi.

6.3.6 Kilpailun poisto

Kilpailujen poistoa varten teimme ”Poista” -napin. Kilpailuja poistaessa valitaan poistettava rivi ja painetaan nappia Poista. ”Poista” -napin painamisen jälkeen ohjelma tarkistaa ensin, onko kontrollin dgvKilpailut valitut solut suurempi kuin nolla. Ehdon toteutuessa poistetaan kyseinen rivi tietokannasta ID tiedon avulla, jos taas ”dgvKilpailu” -kontrollissa ei ole yhtään riviä eli ehto ei toteudu, ilmoitetaan käyttäjälle, että poistettavia rivejä ei ole. Tämän jälkeen kyseinen rivi poistetaan myös ”dgvKilpailut” -kontrollista. Kuvassa 21 näkyy ”Poista” -napin ohjelmakoodi.

```
private void btnPoista_Click(object sender, EventArgs e)
{
    if (this.dgvKilpailut.SelectedCells.Count > 0)
    {

        int valittuRiviInd = this.dgvKilpailut.SelectedCells[0].RowIndex;

        DataGridViewRow valittuRivi = this.dgvKilpailut.Rows[valittuRiviInd];

        string a = Convert.ToString(valittuRivi.Cells["ID"].Value);

        string sql = null;

        sql = "DELETE FROM Jouskari WHERE ID=" + a;

        YhteisetTiedot.yhteys.Open();
        YhteisetTiedot.daJouskari.DeleteCommand = YhteisetTiedot.yhteys.CreateCommand();
        YhteisetTiedot.daJouskari.DeleteCommand.CommandText = sql;

        YhteisetTiedot.daJouskari.DeleteCommand.ExecuteNonQuery();


        dgvKilpailut.Rows.RemoveAt(this.dgvKilpailut.CurrentRow.Index);
        YhteisetTiedot.yhteys.Close();
    }
    else
    {
        MessageBox.Show("Poistettavia tietoja ei ole.");
    }
}
```

Kuva 21. ”Poista” -napin ohjelmakoodi.

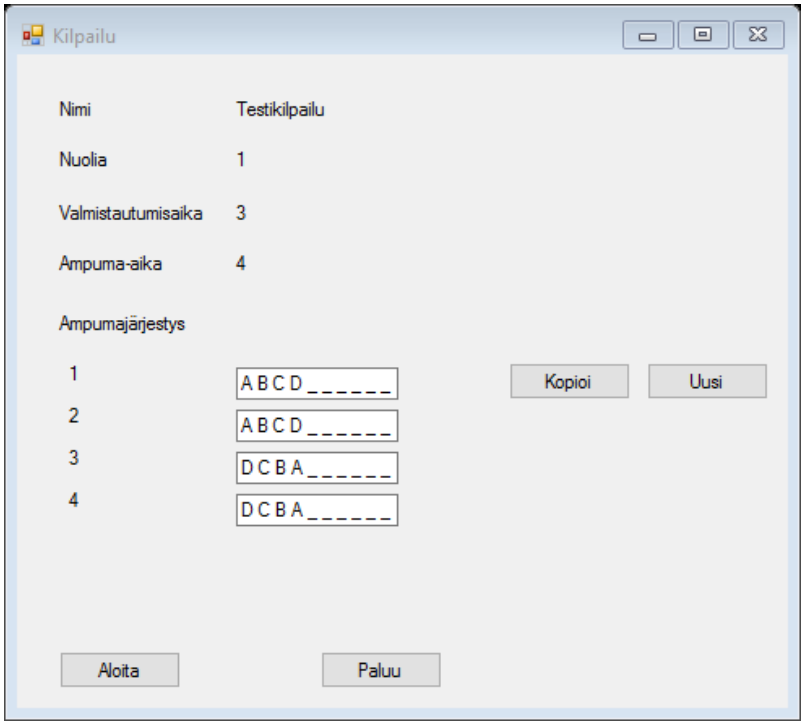
6.3.7 Kilpailun aloitus

Tässä vaiheessa ohjelman kehitys jäi kesken, mutta kerromme mitä saimme aikaan. Kilpailu aloitetaan valitsemalla haluttu kilpailu ja painamalla ”Aloita kisa” -nappia, minkä jälkeen kilpailun tiedot siirretään ohjelman yleisiin muuttujiin ja siirrytään kilpailu näkymään. Kuvassa 22 näkyy ”Aloita kisa” -napin ohjelmakoodi.

```
private void btnAloita_Click(object sender, EventArgs e)
{
    YhteisetTiedot.JInd = dgvKilpailut.CurrentRow.Cells[0].Value.ToString();
    YhteisetTiedot.Jnimi = dgvKilpailut.CurrentRow.Cells[1].Value.ToString();
    YhteisetTiedot.Jnuolia = dgvKilpailut.CurrentRow.Cells[2].Value.ToString();
    YhteisetTiedot.JvalmistautumisAika = dgvKilpailut.CurrentRow.Cells[3].Value.ToString();
    YhteisetTiedot.JampumaAika = dgvKilpailut.CurrentRow.Cells[4].Value.ToString();

    YhteisetTiedot.Kilpailu.Show();
    this.Hide();
}
```

Kuva 22. ”Aloita kisa” -napin ohjelmakoodi.

Kilpailunäkymän avautuessa haetaan kilpailun tiedot yleisistä muuttujista ja tulostetaan lomakkeen kenttiin. Kuvassa 23 näkyy ohjelman kilpailunäkymä.



Kuva 23. Kilpailunäkymä.

Kilpailunäkymässä käyttäjällä on mahdollista tarkastella kilpailun tietoja ja syöttää ampumajärjestykset. ”Uusi” -nappi lisää uuden kierroksen, johon käyttäjä voi syöttää haluamansa ampumajärjestyksen. Kun käyttäjä painaa nappia Uusi, luodaan otsikolle ja tekstikentälle oliot, jotka sen jälkeen tulostetaan lomakkeella olevaan ”FlowLayoutPanel” -kontrolliin. Ampumajärjestysten syöttöön valitsimme ”MaskedTextBox” -kontrollin, koska sillä voidaan rajoittaa, mitä käyttäjän on mahdollista syöttää kyseiseen kenttään. Kuvassa 24 on ”Uusi” -napin ohjelmakoodi.

```
public void btnUusi_Click(object sender, EventArgs e)
{
    Label otsikko = new Label();
    MaskedTextBox mtxAmpuJ = new MaskedTextBox("L L L L L L L L L");

    YhteisetTiedot.yAmpujNro = YhteisetTiedot.yAmpujNro + 1;
    flpAmpuj.Controls.Add(otsikko);
    otsikko.Text = YhteisetTiedot.yAmpujNro.ToString();
    flpAmpuj.Controls.Add(mtxAmpuJ);

}
```

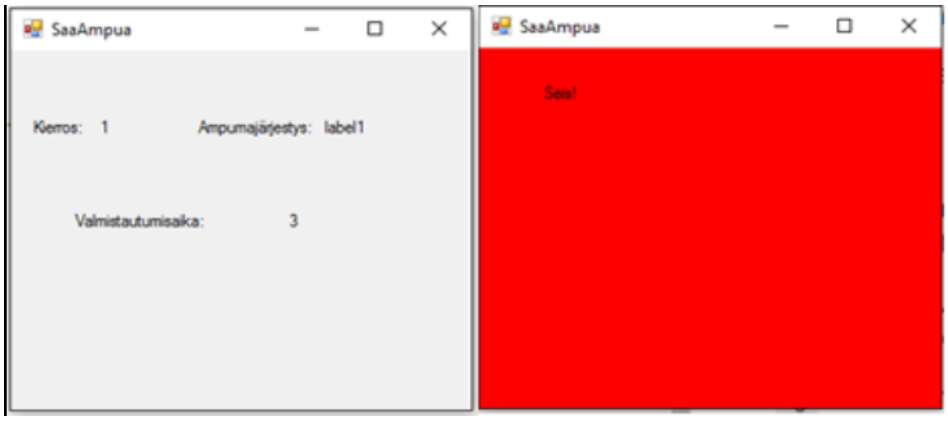
Kuva 24. ”Uusi” -napin ohjelmakoodi.

”Kopioi” -nappi kopio edellisen kierroksen tiedot ja luo uuden kentän, johon kopioidut tiedot sijoitetaan. Kuvassa 25 on napin Kopioi ohjelmakoodi.

```
private void btnKopioi_Click(object sender, EventArgs e)
{
    if (flpAmpuj.Controls.Count == 0)
    {
        MessageBox.Show("Lisää ensin uusi kenttä.");
    }

    else
    {
        Label otsikko = new Label();
        MaskedTextBox mtxAmpuJ = new MaskedTextBox("L L L L L L L L L L");
        YhteisetTiedot.yAmpujNro = YhteisetTiedot.yAmpujNro + 1;
        flpAmpuj.Controls.Add(otsikko);
        otsikko.Text = YhteisetTiedot.yAmpujNro.ToString();
        flpAmpuj.Controls.Add(mtxAmpuJ);
        mtxAmpuJ.Text = flpAmpuj.GetNextControl(otsikko, false).Text;
    }
}
```

Kuva 25. Kopioi -napin ohjelmakoodi.

Lopuksi kun käyttäjä on varmistanut, että kilpailun tiedot ovat oikein ja ampuma-järjestykset on syötetty, painetaan ”Aloita” -nappia. Kilpailun alkaessa siirrytään uudelle lomakkeelle, jossa olisi tarkoitus näyttää yleisölle meneillään oleva kierros ja kierroksen valmistautumisaika, ampuma-aika ja ampumajärjestys. Toimintoja kyseisellä ruudulla olisi ainakin automaattisen ajastimen käynnistäminen, keskeyttäminen ja itse kilpailun lopettaminen. Jos käyttäjä keskeyttää ohjelman, olisi myös tarkoitus näyttää punaisella ruudulla varoitus ”SEIS!”, jolloin kilpailijat lopettavat ampumisen, jos esimerkiksi kilpailualueella on sinne kuulumatonta liikettä. Ampumajärjestyksen olisi myös tarkoitus vaihtua aiemmin syötettyjen ampumajärjestysten mukaan. Kuvassa 26 on keskeneräinen kilpailun kulku näkymä ja seis näkymä.



Kuva 26. Keskeneräiset meneillään oleva kilpailunäkymä ja seis ruutu.

Meneillään olevan kilpailun ohjelmakoodissa ensimmäisenä haetaan tiedot lomakkeelle yhteisistä tiedoista, jonka jälkeen odotetaan että käyttäjä painaa ”Enter” -nappia näppäimistöltä. Kuvassa 27 näkyy tietojen haku yhteisalueelta lomakkeelle.

```
private void frmSaaAmpua_Load(object sender, EventArgs e)
{
    YhteisetTiedot.yhI = int.Parse(YhteisetTiedot.JvalmistautumisAika);
    YhteisetTiedot.yhI = YhteisetTiedot.yhI + 1;
    YhteisetTiedot.yhE = int.Parse(YhteisetTiedot.JampumaAika);
    YhteisetTiedot.yhE = YhteisetTiedot.yhE + 1;
    YhteisetTiedot.yKierros = 1;
    lblAikaOts.Text = "Valmistautumisaika:";
    lblKierros.Text = YhteisetTiedot.yKierros.ToString();
    lblTimeri.Text = YhteisetTiedot.JvalmistautumisAika;
    lblSeis.Hide();
}
```

Kuva 27. Tietojen haku lomakkeelle.

”Enter” painalluksen jälkeen timer1 alkaa laskea alaspäin valmistautumisaika muuttujan arvosta, kunnes arvo on nolla ja sen jälkeen ajastimelle annetaan uusi arvo, joka on ampuma-aika muuttujasta saatu arvo. Aiemmin mainittu tapahtuma pyörii niin kauan, kunnes käyttäjä päättää pysäyttää sen painamalla ”Enter”, jol-

loin ajastin pysähtyy tai painamalla välilyöntiä jolloin seis ruutu tulee näkyviin. Kuvissa 28 ja 29 näkyy keskeneräiset timer1 ohjelmakoodi ja timer1 vaikuttavat ”KeyPress” -tapahtumat.

```
private void timer1_Tick(object sender, EventArgs e)
{
    YhteisetTiedot.yhI--;
    lblTimeri.Text = YhteisetTiedot.yhI.ToString();

    if (YhteisetTiedot.yhI < 0)
    {
        timer1.Stop();
        lblAikaOts.Text = "Ampuma-aika:";
        YhteisetTiedot.yhE--;
        lblTimeri.Text = YhteisetTiedot.yhE.ToString();

        timer1.Start();

        if (YhteisetTiedot.yhE == 0)
        {
            lblAikaOts.Text = "Valmistautumisaika:";
            timer1.Stop();
            YhteisetTiedot.yhI = int.Parse(YhteisetTiedot.JvalmistautumisAika);
            YhteisetTiedot.yhI = YhteisetTiedot.yhI + 1;
            YhteisetTiedot.yhE = int.Parse(YhteisetTiedot.JampumaAika);
            YhteisetTiedot.yhE = YhteisetTiedot.yhE + 1;
            timer1.Start();
            YhteisetTiedot.yKierros++;
            lblKierros.Text = YhteisetTiedot.yKierros.ToString();
        }
    }
```

Kuva 28. Keskeneräinen timer1 ”timer_tick” -tapahtuman ohjelmakoodi.

```
private void frmSaaAmpua_KeyPress(object sender, KeyPressEventArgs e)
{
    if (e.KeyChar == (char)13)
    {
        YhteisetTiedot.SaaAmpua.BackColor = DefaultBackColor;
        lblAjärj.Show();
        lblKierros.Show();
        lblTimeri.Show();
        lblAikaOts.Show();
        lblAmpJOts.Show();
        lblKierrosOts.Show();
        lblSeis.Hide();

        if (timer1.Enabled)
        {
            timer1.Stop();
        }
        else
        {
            timer1.Start();
        }
    }
    if (e.KeyChar == (char)32)
    {
        if (timer1.Enabled)
        {
            YhteisetTiedot.yhI = int.Parse(YhteisetTiedot.JvalmistautumisAika);
            YhteisetTiedot.yhI = YhteisetTiedot.yhI + 1;
            YhteisetTiedot.yhE = int.Parse(YhteisetTiedot.JampumaAika);
            YhteisetTiedot.yhE = YhteisetTiedot.yhE + 1;
            lblTimeri.Text = YhteisetTiedot.JvalmistautumisAika;
            timer1.Enabled = false;
            lblAjärj.Hide();
            lblKierros.Hide();
            lblTimeri.Hide();
            lblAikaOts.Hide();
            lblAmpJOts.Hide();
            lblKierrosOts.Hide();
            lblSeis.Show();
            YhteisetTiedot.SaaAmpua.BackColor = Color.Red;
        }
    }
}
```

Kuva 29. Keskeneräinen timer1 ”KeyPress” -tapahtumien ohjelmakoodit.

6.4 Ohjelman testaus

Ohjelman testaus tapahtui yhtä aikaa ohjelmoinnin kanssa. Räätälöimme testivetoisen ohjelmistokehityksen omiin tarpeisiimme sopivaksi. Koska emme oikeastaan suunnitelleet testitapauksia ensimmäiseksi, kuten testivetoisessa ohjelmistokehityksessä on tapana vaan testitapauksia tuli samalla, kun lisäsimme jonkun ominaisuuden tai toiminnallisuuden. Huonona puolena meidän tavassamme oli se,

että itse testausta ei oikeastaan tullut juuri mitenkään dokumentoitua mutta ainakin kaikki työn alla olevat osat tuli testattua kunnolla ennen kuin ne tehtiin valmiiksi.

7 YHTEENVETO

7.1 Oppimisprosessi

Aloittaessamme työn ei meillä ollut kokemusta ohjelmistoprojektin hallinnasta ja loppuun viemisestä muuta kuin mitä ammattikorkeakoulussa olimme oppineet. C#-ohjelmoinnista meillä oli kuitenkin suhteellisen hyvä pohja, koska koulussa on tullut rutiinia ohjelmointiin suhteellisen hyvin. Pyrimme työtä tehdessä hyödyntämään kaikkea mahdollista koulussa oppimaamme mutta ei koulussa opitut tiedot ja taidot tietenkään voi täysin valmistaa työelämän tilanteisiin. Saimme siis projektia tehdessä hyvää kokemusta tuleviin mahdollisiin ohjelmistoprojekteihin, joihin tulemme työelämässä osallistumaan. Opimme esimerkiksi kuinka tärkeää kommunikaatio ja hyvä projektinhallinta ovat ohjelmistoprojektissa. Kommunikaatio puute aiheuttaa epätietoisuutta ja hidastaa projektin etenemistä ja jos projektinhallinta ei ole toivotulla tasolla, voi koko projekti kaatua. Huomasimme myös työtä tehdessämme, miksi ketterät menetelmät ja testivetoinen ohjelmistokehitys sopii niin hyvin ohjelmistokehitykseen. Koska ohjelmaa tehdessä on niin paljon muuttujia, on niitä vaikea hallita ilman ketteriä menetelmiä, sillä ketteriä menetelmiä hyödyntämällä projektin saa paloiteltua helposti osiin ja silloin vastuun jakaminen helpottuu. Testivetoista kehitystä kannattaa mielestämme hyödyntää kaikissa ohjelmistoprojekteissa, sillä sen avulla vähennetään huomattavasti ohjelman lopussa tulevaa testausjaksoa, koska suurin osa ohjelmasta on jo testattu osissa ja ohjelmaa ei tarvitse enää testata muuta kuin kokonaisuutena. Tulimme siihen tulokseen, että ketterää ohjelmistokehitystä ja testivetoista ohjelmistokehitystä kannattaa ja pitääkin hyödyntää ohjelmistoprojekteissa. Ainakin itse tulemme hyödyntämää molempia menetelmiä. Olio-ohjelmoinnista projektin aikana syvensimme jo aiemmin oppimaamme ja mielestämme olio-ohjelmointi on hyvä tapa ratkaista ohjelmointiongelmia, jotka muuten olisivat mahdottomia ratkaista. Ymmärrämme nyt myös paremmin, miten olio-ohjelmointi toimii C#-ohjelmoinnissa.

7.2 Tulokset

Tutkimuksestamme on mahdollisesti hyötyä kaikille, jotka eivät ole ennen osallistuneet ohjelmistoprojektiin, sillä tutkimuksesta saa hyvän kuvan millaista ohjelmistokehitys ja ohjelmistoprojektiin osallistuminen voi olla. Yksi tutkimuskysymyksemme käsitteli olio-ohjelmointia eli ”Mikä on sen tarkoitus ja miten sitä hyödynnetään C#-ohjelmoinnissa”. Olio-ohjelmointi kysymykseen tuli mielestämme tekstissä vastattua suhteellisen hyvin ja materiaali, jota käytimme vastaamaan kyseiseen kysymykseen, voidaan pitää luotettavana ja kattavana. Tärkeimpänä asiana olio-ohjelmoinnista oli se, miten hyvin sillä ratkaistaan monimutkaisimmatkin ongelmat. Tutkimuskysymykseen ”Mikä tekee ketterästä ohjelmistokehityksestä niin suositun ja miksi käytimme sitä omassa projektissa?” tuli mielestämme myös vastattua. Materiaalit, joita käytimme, olivat hyvin tehty ja niistä saimme hyvää lähdemateriaalia omaan tutkimukseemme. Tärkeimpinä asioina ketteristä menetelmistä olivat Scrum ja projektin jakaminen osiin. Kolmas tutkimuskysymys ”Mitkä ovat testivetoisen kehityksen hyödyt ja miksi sitä kannattaa käyttää ohjelmistokehityksessä?” jäi itse toteutuksen osalta vähäiseksi mutta raportissa olevassa materiaalissa tulee vastaus myös tähän kysymykseen. Ohjelman kehitys jäi kesken, koska kommunikaatio meidän ja toimeksiantajan kanssa ei toiminut, projektin suunnittelu oli toteutettu huonosti ja aikakin loppui kesken.

7.3 Ohjelman jatkokehitys

Mahdollisia jatkokehitys ideoita voisi olla laajempi ohjelman käytännön testaus ja projektin ajankäyttöön ja resurssien hallintaa olisi pitänyt keskittyä enemmän. Kehitysideoita itse ohjelmaan voisivat olla toteutukseen tehdyn tietokannan parempi suunnittelu ja eri tietokantajärjestelmän valinta. Ohjelmasta jäi myös puuttumaan vaatimusmäärittelyssä olleita toimintoja, jotka olivat mahdollisuus määritellä onko kilpailu finaali vai ei ja finaalinäyttö. Ohjelmaan olisi myös voinut lisätä enemmän varmistuksia esimerkiksi virheellisen tietojen syötön varalta. Ampumavuorojen syöttö jäi ohjelmasta myös keskeneräiseksi eli sen voisi tehdä valmiiksi ja kilpailunäyttöä pitäisi tehdä paremmaksi. Ohjelman ulkoasukin jäi keskeneräiseksi, joten sitä voisi kehittää käyttäjäystävällisemmäksi.

LÄHTEET

Agilemanifesto. 2001a. Ketterän ohjelmistokehityksen julistus. Viitattu 24.4.2016. http://agilemanifesto.org/iso/fi/

Agilemanifesto. 2001b. Julistuksen takana olevat periaatteet. Viitattu 24.4.2016. http://agilemanifesto.org/iso/fi/principles.html

Barber, D. 2012. Why Test-driven Development? Viitattu 14.4.2016 http://derekbarber.ca/blog/2012/03/27/why-test-driven-development/

Heiramo, P. 2010. Ketterä projektinhallinta. Viitattu 4.4.2016. https://agilecraft.files.wordpress.com/2010/03/kettera-projektinhallinta-datariina-16-3-20101.pdf

Ho, D. 2016. Notepad++. Viitattu 30.3.2016. https://notepad-plus-plus.org/

Jansson, R., Juselius, P. 2004. Projektiopas Ideasta liiketoimintaan. Helsinki. Tekes Viitattu 1.4.2016. http://www.tekes.fi/globalassets/julkaisut/projektiopas2004.pdf

Kasurinen, J. 2013. Ohjelmistotestauksen käsikirja. 1. Painos 2013. Jyväskylä. Docendo.

Kolari, M. 2014a. Ohjelman rakenne. Viitattu 5.4.2016. http://mikakolari.fi/csharp-dotnet/perusteet/ohjelman-rakenne/

Kolari, M. 2014b. Arvo- ja viittaustyypit. Viitattu 6.4.2016. http://mikakolari.fi/csharp-dotnet/perusteet/tietotyypit/arvo-ja-viittaustyypit/

Kolari, M. 2014c. Tyyppimuunnokset. Viitattu 6.4.2016. http://mikakolari.fi/csharp-dotnet/perusteet/tietotyypit/tyyppimuunnokset/

Kolari, M. 2014d. Ohjausrakenteet. Viitattu 6.4.2016. http://mikakolari.fi/csharp-dotnet/perusteet/ohjausrakenteet/

Koulutus- ja konsultointipalvelu KK Mediat. 2016. Johdatus SQL:n maailmaan. Viitattu 4.4.2016. http://www.2kmediat.com/sql/alkeet.asp

MariaDB Foundation. 2016. About MariaDB. Viitattu 30.3.2016. https://mariadb.org/about/

Microsoft. 2016a. Microsoft Developer Network. Viitattu 30.3.2016. https://msdn.microsoft.com/en-us/library/office/fp179695.aspx

Microsoft. 2016b. Microsoft Developer Network. Viitattu 30.3.2016. https://msdn.microsoft.com/en-us/library/dd831853.aspx

Microsoft. 2016c. Introduction to the C# Language and the .NET Framework. Viitattu 4.4.2016. https://msdn.microsoft.com/en-us/library/z1zx9t92.aspx

Microsoft. 2016d. Accessibility Levels (C# Reference). Viitattu 6.4.2016. https://msdn.microsoft.com/en-us/library/ba0a1yw2.aspx

Microsoft. 2016e. C# Coding Conventions (C# Programming Guide). Viitattu 4.4.2016. https://msdn.microsoft.com/en-us/library/ff926074.aspx

Microsoft. 2016f. Microsoft SQL Server. Viitattu 5.4.2016. https://msdn.microsoft.com/en-us/library/mt590198(v=sql.1).aspx

Moghadampour, G. 2012. C#-ohjelmointi. 4. Painos helmikuu 2012. Jyväskylä. Docendo.

Schwaber, K & Sutherland, J. 2013, The Definitive Guide to Scrum: The Rules of the Game. Viitattu 26.4.2016. http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-US.pdf#zoom=100

Scrumalliance. 2016. Scrum Values. Viitattu 26.4.2016. https://www.scrumalliance.org/why-scrum/core-scrum-values-roles

Sininen meteoriitti. 2013a. Ketteryys haltuun: Yleisimmät ketterät käytännöt. Viitattu 14.4.2016. http://www.meteoriitti.com/2013/06/06/ketteryys-haltuun-yleisimmat-ketterat-kaytannot/

Sininen meteoriitti. 2013b. Ketteryys haltuun: Scrum pähkinänkuoressa. Viitattu 19.4.2016. http://www.meteoriitti.com/2013/06/06/ketteryys-haltuun-scrum-pahkinankuoressa/

Viirkorpi, P. 2000a, 32-34. Onnistunut Projekti – opas kunta-alan projektityöskentelyyn. Helsinki. Suomen Kuntaliitto. Viitattu 1.4.2016. http://shop.kunnat.net/download.php?filename=uploads/p071005095633P.pdf

Viirkorpi, P. 2000b, 34. Onnistunut Projekti – opas kunta-alan projektityöskentelyyn. Helsinki. Suomen Kuntaliitto. Viitattu 1.4.2016. http://shop.kunnat.net/download.php?filename=uploads/p071005095633P.pdf

W3Schools. 2016. PHP MySQL Database. Viitattu 5.4.2016. http://www.w3schools.com/php/php_mysql_intro.asp