HAAGA-HELIA
ammattikorkeakoulu

Ketterä hyväksyntätestauslähtöinen ohjelmistokehitys

Case Tilastointiohjelmisto

Tuukka Rintala

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma
8.11.2012

HAAGA-HELIA
ammattikorkeakoulu

Tietojenkäsittelyn koulutusohjelma

Tiivistelmä

8.11.2012

Tekijä tai tekijät Tuukka Johannes Rintala	**Ryhmätunnus tai aloitusvuosi** Y08TM
Raportin nimi **Ketterä hyväksyntätestauslähtöinen ohjelmistokehitys** **Case Tilastointiohjelmisto**	**Sivu- ja liitesivumäärä** 52+22
Opettajat tai ohjaajat Kai Kivimäki	

Tässä opinnäytetyössä tutkitaan kuinka ketterää hyväksyntätestauslähtöistä ohjelmistokehitystä voidaan soveltaa pienohjelmistotuotantoon. Opinnäytetyön tavoitteena on esitellä ketterä hyväksyntätestauslähtöinen ohjelmistoprosessimalli ja soveltaa esiteltyä mallia case-projektina suoritettuun ohjelmistoprojektiin.

Tietoperustassa käydään läpi hyväksyntätestauskeskeisen ohjelmistokehityksen eri vaiheet askel askeleelta. Case-osiossa esiteltävä projekti suoritettiin 10.7 – 10.9.2012 välisenä aikana Frisbeegolffarit-ryhmälle tuotettuna tilastointiohjelmistona. Ohjelmisto otettiin beta-käyttöön 11.9.2012 ja projekti jatkuu älypuhelin version kehityksellä.

Ohjelmistoprojektista saatujen kokemuksien perusteella voitiin päätellä, että hyväksyntätestauslähtöisen ohjelmistotuotannon menetelmät skaalautuvat hyvin ohjelmistojen pientuotantoon. Mallista puuttuva roolijako ja vertaistuki aiheuttivat ylimääräisiä suunnittelu- ja ohjelmointivirheitä sekä työmäärän ylikuormittumista. Ylikuormittumista voitiin helpottaa ohjelmistoprosessimallin säätelymekanismeja hyväksikäyttäen, mikä laski tehtyjen virheiden määrää.

Malli soveltuu hyvin helppokäyttöisten kevyiden ohjelmistojen tuottamiseen, koska malli perustuu asiakaskeskeiseen ja kokekemusperäiseen prototyyppikehitykseen. Malli ei soviellu kriittisten tai integroitujen järjestelmien kehittämiseen, koska se ei sisällä työkaluja raskaaseen formaaliin tarkasti dokumentoituun ohjelmistokehitykseen.

Asiasanat
Ketterät menetelmät, hyväksyntätestauslähtöisyys, ohjelmistotuotanto

HAAGA-HELIA
University of Applied Sciences

Abstract

8.11.2012

Degree programme in Information Technology

Authors Tuukka Johannes Rintala	**Group or year of entry** Y08TM
The title of thesis Lean-agile acceptance test-driven software development Case Statistic Software	**Number of pages and appendices** 52+22
Supervisor(s) Kai Kivimäki	

This study examines how lean-agile acceptance test-driven software development can be applied to small software projects. The goal of the study was to present a lean-agile acceptance test-driven software development process model and apply the presented model to a small software project developed as a part of this study.

The study includes theory and case section. The theory section presents a software process model using lean-agile acceptance test-driven software development. The model and the activities it has are presented with step-by-step top-down approach. The model is applied to a software project presented in the case section. The case project using the model was successfully completed during 10.7 – 10.9.2012 by developing statistics software for Frisbeegolffarit-group.

The software project clearly demonstrated that lean-agile acceptance test-driven software development scaled well to small software project. At the beginning of the project the stacking of project roles and the lack of peer support caused unnecessary design flaws and programming mistakes. Project management techniques were successfully applied to solve the issues and the quality of design and code increased. The model suits best to the development of light weight software emphasized by easy usability, since the model is based on customer experience centric development and prototypes. The model is not suitable for developing critical or integrated systems, since it does not contain tools for strict formal heavy weight software development.

Key words
Agile methods, acceptance test-driven, software development

Sisällys

1 Johdanto

Viimeisen kymmenen vuoden aikana ohjelmistoteollisuus on kokenut suuren muutoksen. Samalla kun tuotettujen ohjelmistojen volyymi on kasvanut, on tuotettujen ohjelmistojen keskimääräinen koko pienentynyt. Selkeimmin tämä trendi näkyy viimeisen viiden vuoden aikana yleistyneiden älypuhelimien ohjelmistomarkkinoilla. Älypuhelimien ohjelmistomarkkinoita dominoi lukumääräisesti pienet hyöty- tai viihdekäyttöön tarkoitetut palveluntarjoajan palvelimilta ladattavat ohjelmat, joiden tuotantokustannukset eivät ole AAA-luokituksen omaavien ohjelmistojen tasolla.

Alati kiihtyvä kilpailutilanne on asettanut pienohjelmistokehityksen samankaltaiseen tilanteeseen kuin minkä tahansa teollisuudenalan. Tuotantoon haetaan tehokkuutta ja kustannussäästöjä, jolloin vanhat menetelmät eivät enää välttämättä toimi uusien lainalaisuuksien vallitessa. Asetettuihin tehokkuus- ja kustannustavoitteisiin voidaan päästä monin eri tavoin, mutta yksi tehokkaimmista tavoista on tehostaa yrityksen ohjelmistoprosessia, jotta se soveltuisi paremmin yrityksen tuottamien ohjelmistojen tehokkaaseen kehittämiseen. Toinen tehokas tapa on vähentää vaatimus-, suunnittelu-, ja ohjelmointivirheitä, joiden korjaaminen saattaa muodostaa suuren osan ohjelmistoprojektin kustannuksista.

Raskaisiin ja formaaleihin prosesseihin perustuva perinteinen ohjelmistokehitys ei sovellu pienien ja keveiden ohjelmistojen tehokkaaseen tuotantoon. Vuosituhannen taitteessa yleistyneet ketterät menetelmät vastasivat pienohjelmistojen tuotannon asettamiin haasteisiin vain osittain. Vaikka ketteryys muutti ohjelmistoprosessia keveämmäksi ja mukautuvammaksi, silti ohjelmistokehitystä käsittelevä prosessikirjallisuus keskittyi edelleen pääsääntöisesti keskisuurten tai suurten ohjelmistokehitysryhmien hallinnoimiseen. Kirjallisuuden konservatiivisuus ja hidas painopisteen muutos on jättänyt pienryhmän ja yksinäisen ohjelmistokehittäjän mentävän aukon ohjelmistokehitystä käsittelevään kirjallisuuteen. Pienten ohjelmistokehitysryhmien ohjelmistokehitystä tutkivalle tutkimukselle on todellinen tarve.

Tämän opinnäytetyön tarkoituksena on tutkia kuinka ketterä hyväksyntätestauslähtöinen ohjelmistokehitys soveltuu ohjelmistojen pientuotantoon. Tavoitteena on esitellä ketterän hyväksyntätestauslähtöisen ohjelmistokehityksen teoria ja soveltaa sitä käytännön projektiin. Tavoitteeseen pyritään esittelemällä opinnäytetyön tietoperustassa yksi mahdollinen tapa käyttää hyväksyntätestauslähtöistä ohjelmistokehitystä ja soveltamalla tätä tapaa tuotettavaan case-projektiin. Opinnäytetyön case-projekti suoritetaan tuottamalla tilastointiohjelmisto vuonna 2011 perustetulle vapaamuotoisia pelitilaisuuksia ja pelimatkoja järjestävälle Frisbeegolffarit-ryhmälle.

Hyväksyntätestauslähtöisen ohjelmistokehityksen pohjaksi on valittu Ken Pugh:in ehdottama prosessi, jossa siirrytään askeleittain tarkempiin ja tarkempiin ohjelmiston toimintakuvauksiin ja niiden hyväksyntätesteihin. Prosesesi käyttää hyväkseen rekursiivisuutta (recursive) ja refaktorointia (refactoring), jolloin kehitettävän ohjelmiston ominaisuuksia, toiminnallisuutta ja hyväksymistestejä voidaan arvioida uudelleen milloin tahansa ohjelmistokehityksen aikana. Niiltä osin kuin Pughin prosessi ei tarjoa työkaluja tai toimintamalleja esiteltävään prosessiin, sitä täydennetään Dean Leffingwellin ja Lasse Koskelan työkaluilla ja toimintamalleilla.

Opinnäytetyön lähdemateriaalina käytetyssä kirjallisuudessa kirjailijoiden käyttämä termistö ei ollut yhdenmukainen. Tästä syystä opinnäytetyön laatija on parhaan kykynsä mukaan yhdenmukaistanut opinnäytetyössä käytettävän termistön, jolloin opinnäytetyön käyttämä termistö ei välttämättä vastaa täydellisesti kunkin lähdemateriilina käytetyn teoksen termistöä. Termien yhdenmukaistaminen oli välttämätön toimi, koska osa termeistä oli teosten välillä ristiriitaisia tai samat termit merkitsivät eri asioita (liite 1.).

2 Hyväksyntätestauslähtoinen prosessimalli

Hyväksyntätestauslähtöinen ohjelmistokehitys poikkeaa perinteisistä ohjelmistotuotannon prosessimalleista siten, että valmiin jäykän prosessin sijasta hyväksyntätestauslähtöinen ohjelmistokehitys tarjoaa kehysmallin ohjelmistokehitykseen. Ketterä hyväksyntätestauslähtöinen kehysmalli sisältää joukon toisiinsa löyhästi sidottuja muokattavia valinnaisia työtapoja. Kehittäjät voivat valita projekteihinsa sopivat hyväksyntätestauslähtöiset työtavat ja liittää ne omaan ketterään ohjelmistoprosessiinsa.

Hyväksyntätestauslähtöinen ohjelmistokehitys lainaa perinteisestä ylhäältä alaspäin tapahtuvasta modulaarisesta suunnittelumallista (top-down design) kerroksittain tarkentuvat kuvaukset. Tarkentuvien arkkitehtuuri- tai moduulikuvausten sijasta hyväksyntätestauslähtöinen ohjelmistokehitys käyttää alati tarkentuvia kuvauksia toiminnallisuuden vaatimuksista ja hyväksyntätesteistä, jotka toiminnallisuuden täytyy läpäistä, jotta se voitaisiin hyväksyä valmiiksi.

On tärkeää huomioida, että hyväksyntätestauslähtöisessä ohjelmistotuotannossa hyväksyntätestin merkitys on erilainen kuin perinteisessä ohjelmistotuotannon prosessissa. Perinteisessä ohjelmistotuotannon prosessissa hyväksyntätestillä mitataan jälkikäteen, että toteutettu toiminallisuus on sellainen kuin asiakas haluaa. Hyväksyntätestauslähtöisessä ohjelmistotuotannossa ennen toteutusta laadittavilla hyväksyntätesteillä varmistetaan etukäteen, että toiminallisuudesta tullaan toteuttamaan sellainen kuin asiakas haluaa. Käytäntö varmistaa ohjelmiston laadun siten, että toiminallisuudella on mitattavat vaatimukset, joiden perusteella toiminnallisuus voidaan toteuttaa ja hyväksyä.

Sen sijaan, että asiakas olisi aktiivinen vain projektin vaatimusmäärittelyssä ja perinteisessä hyväksyntätestausvaiheessa, hyväksyntätestauslähtöinen ohjelmistokehitys aktivoi asiakkaan jokaisessa projektin vaiheessa. Asiakkaalla on tärkeä osuus koko projektin laatimisessa. Asiakkaan työpanos näkyy eritoten käyttäjätarinoiksi (user stories) kutsuttujen käyttöesimerkkien ja niiden hyväksyntätestien laatimisessa.

2.1 Ketterät periaatteet ja asiakaslähtöisyys

Ketterien menetelmien pääperiaatteet ovat kirjattu Ketterään Julistukseen (Agile Manifesto). Ketterissä menetelmissä painotetaan henkilöitä ja kommunikaatiota raskaiden prosessien ja työkalujen sijasta. Toimivaa ohjelmistoa pidetään tärkeämpänä kuin ohjelmistoprojektin läpikotaista ja tarkkaa dokumentointia. Kyky vastata muutokseen on tärkeämpää kuin ennalta laaditun suunnitelman seuraaminen. Joustava asiakasyhteistyö on tärkeämpää kuin tiukka sopimusneuvotteluiden noudattaminen. (Agilemanifesto 2001)

Ketterän Julistuksen kohtien taustalla vaikuttavat seuraavat periaatteet. Ketterän prosessin päämäärä on tyydyttää asiakkaan tarpeet jatkuvilla ja aikaisilla lisäarvoa tarjoavilla julkaisuversioilla. Ohjelmiston vaatimuksia voidaan tarvittaessa muuttaa missä tahansa projektin vaiheessa, jos asiakkaan etu niin vaatii. Projetkiryhmä tulee rakentaa päivittäiseen yhteistyöhön kykenevistä luotettavista sekä motivoituneista henkilöistä, joita työympäristö tukee varauksetta. Prosessissa keskitytään yksinkertaisuuteen, jatkuvuuteen ja tekniseen toimivuuteen. Tällä tavoin saavutetaan jatkuva keskeytyksetön kehitys, jolloin toimivan ohjelmistoversion toimittamista voidaan pitää edistyksen mittarina. Projektiryhmä toimii oman kehityksensä peilinä ja mittarina. Parhaat tulokset saavutetaan itsensä organisoivilla projektiryhmillä. (Agilemanifesto 2001)

Ketterä asiakaslähtöisyys saavutetaan asiakaskommunikaatiolla. Ketterissä menetelmissä asiakkaan tarpeet pyritään tyydyttämään aikaisin ja usein. Projektin prioriteeteissa asiakkaan kokemus on etusijalla. Parhaimpaan tulokseen päästään aktivoimalla asiakas jokaisessa projektin vaiheessa ja luomalla tarvittavat kommunikaatiokanavat asiakkaan kanssa. Asiakas ottaa osaa projektin suunnittelun eri vaiheisiin vaatimusmäärittelystä testaussuunnitteluun, kokouksiin, testaukseen ja muihin projektin eri toimintoihin liiketoiminnasta hyväksyntätestaukseen. (Drego 2010, 1-4)

2.2 Projektiryhmä

Ketterissä menetelmissä tehtäväryhmien roolijako ei noudata perinteiseen hierarkiseen malliin perustuvaa roolijakoa. Leffingwell havainnollistaa jakoa tehtäväkehyksillä. Perinteisesti ryhmät ovat jakaantuneet työtehtävän määräämiin kehyksiin, kuten testaukseen, suunnitteluun tai prosessinhallintaan. Ketterissä menetelmissä vertikaalinen jako hajotetaan ja muutetaan horisontaaliseksi siten, että ryhmässä työskentelee tarvittaessa jokaista eri työtehtävää edustava henkilö. (Leffingwell 2011, 50-51)

Ken Pugh jakaa mallissaan projektiryhmän roolit kolmeen pääryhmään, asiakkaaseen, testaajaan ja kehittäjään. Asiakaspääryhmä määrittää vaatimukset, laatii hyväksyntätestit ja päättää kehityskohteiden prioriteetit. Kehittäjäpääryhmä toteuttaa asiakaspääryhmän asettamat vaatimukset ja varmistaa, että tuotettu ohjelmisto läpäisee sille asetetut hyväksyntätestit. Testauspääryhmä laatii hyväksyntätestit yhdessä asiakaspääryhmän kanssa ja tarkistaa testauksen avulla, että kehityspääryhmän toteuttama ohjelmisto toimii oikein. (Pugh 2010, 16.)

Läsnä oleva asiakas (on-site customer) ei nimestään huolimatta välttämättä ole suoranainen asiakkaan oma edustaja. Läsnä olevan asiakkaan tointa hoitamaan voidaan valita tuotepäällikkö, asiantuntija tai toimiala-asiantuntia. Läsnä olevan asiakkaan tehtävänä on määrittää ohjelmiston ominaisuudet määrittelyprosessin avulla ja ohjata ominaisuuksien kehitystä julkaisusuunnitelman kautta. Käytännössä tämä tarkoittaa läsnä olevan asiakkaan vastuuta projektin vision säilyttämisestä sekä tarinoiden että ominaisuuksien tunnistamisesta yhdessä muun projektiryhmän kanssa ja yhteyden pidosta todelliseen asiakkaaseen. (Shore & Warden 2007, 29-30)

Tuotepäällikön tehtävänä on pitää projektin tavoitteet selkeinä ja tarvittaessa ohjata projektiryhmän toimintaa priorisoimalla julkaisusuunnitelman tavoitteita. Tuotepäällikkö näyttelee tärkeää roolia asiakasyhteistyössä. Tuotepäällikön tehtäviin kuluu yhteyden pito sidosryhmiin, palautteen hallitseminen, työn arviointi, asiakassuhteet ja organisaatiopolitiikan hallinta. Hänen tehtäväalueeseensa kuluu myös ohjelmiston asiakaspromootio.Tuotepäällikkö on erityisen tärkeä henkilö projektin

alkuvaiheessa, jolloin projektin tavoitteet ja tavoitteiden prioriteetit eivät välttämättä ole vielä selviä asiakkaalle tai projektiryhmälle. Projektin edetessä tuotepäällikkö voi pienentää rooliaan, jos projektiryhmän työskentely sujuu kitkatta. Tällöin tuotepäällikkö voi siirtää osan vastuualueensa toimistaan läsnä olevalle asiakkaalle. (Shore & Warden 2007, 31-32)

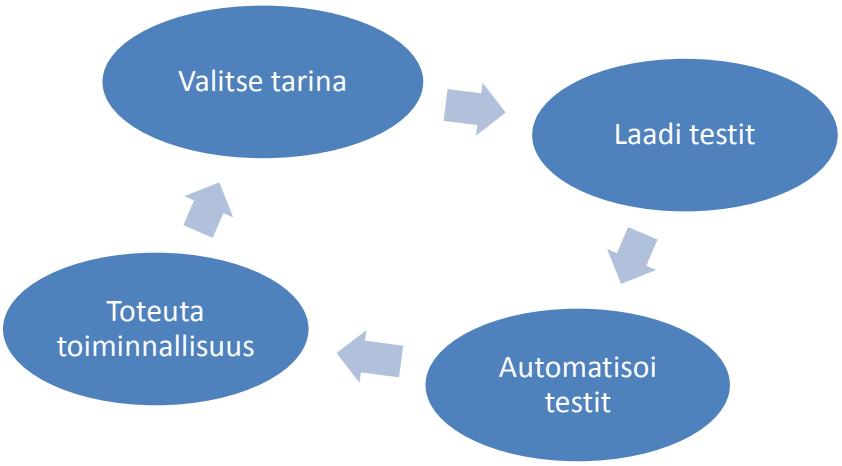
Ohjelmistokehittäjät laativat ohjelmakoodin tarinoiden pohjalta pariohjelmointia hyödyntäen. Ohjelmistokehittäjien vastuualueeseen kuuluvat myös asiakasyhteistyö, testien laatiminen ja suorittaminen yhdessä testaajien kanssa. Lisäksi ohjelmistokehittäjät kirjoittavat tarvittavat metodit automaattiseen testaukseen. Tarvittaessa ohjelmistokehittäjät muodostavat ryhmiä yhden tai useamman testaajan kanssa (Leffingwell 2011, 52.). Tavallisilla ohjelmistokehittäjillä on usein puutteita alakohtaisissa tiedoissa. Asiantuntijan tehtävänä on täydentää projektin alakohtaista tietämystä ja ohjata projektia siten, että alakohtaiset vaatimukset täyttyvät. Tällä tavoin voidaan varmistaa, että tuotettava ohjelmisto on tarkoituksen mukainen ja toimii alakohtaisen säännöstön mukaan. Vaikeissa ja epäselvissä tapauksissa asiantuntijat luovat asiakastestit yhteistyössä testaajien kanssa asiakkaalle. (Shore & Warden 2007, 32.)

Testaajien tärkein tehtävä on taata laadittavan ohjelmiston laatu. Perinteisestä testaajan toimenkuvasta poiketen testaajan pääasiallinen tehtävä ei ole etsiä ohjelmointivirheitä kattavalla testauksella vaan etsiä ratkaisuja joilla ohjelmoijat voivat kirjoittaa mahdollisimman virheetöntä koodia. Testaajat käyttävät käytettävyystestauksesta tuttua vapaan läpikäynnin testausmenetelmää etsiäkseen ohjelmiston heikkoja kohtia. Siten testaajat toimivat teknisinä neuvonantajina ohjelmoijille. Lisäksi testaajien vastuulla on kerätä tietoa ohjelmiston laadullisista ominaisuuksista, kuten käytettävyydestä, vakaudesta tai skaalautuvuudesta. (Shore & Warden 2007, 35.)

2.3 Iteraatio

Hyväksyntäkeskeisen ohjelmistokehityksen perusta voidaan kaikessa yksinkertaisuudessaan esittää neljänä toistettavana askeleena. Malli käyttää hyväkseen

muista ohjelmistotuotannon evoluutiomalleista tuttua iteratiivista (iterative) ja kasvattavaa (incremental) mallia, jossa jokainen iteraatio on pieni askel kohti valmista ohjelmistoa. Iteraation sisäistä neljän askeleen mallia toistetaan niin pitkään kuin iteraation tehtävälistassa on jäljellä toteutettavia käyttäjätarinoita (kuvio 1.). (Koskela 2008, 334-335)



Kuvio 1. Iteraation kiertokulku. (Koskela 2008, 335.)

Jokainen iteraatio on ajallisesti rajattu (time-boxed), itsenäinen (autonomous) ja suljettu (closed). Itsenäisellä ja suljetulla tarkoitetaan sitä, että kulloinkin keskitytään parhaillaan olevaan iteraation tehtävälistassa oleviin tehtäviin ja ylivuotoa seuraavaan iteraatioon pyritään välttämään. Käytännössä tämä ei kuitenkaan usein ole mahdollista ja mallissa joudutaan turvautumaan jatkuvan suunnittelun (continuous planning) periaatteisiin, jolloin seuraavaa iteraatiota suunnitellaan edellisen iteraation aikana. (Koskela 2008, 346-347)

2.4 Projektin kehitysjono ja iteraation tehtävälista

Projektin kehitysjono (project backlog) toimii ohjelmistokehityksen priorisointityökaluna. Kehitysjono elää jokaisessa iteraatiossa ja siten kehittyy projektin mukana. Kehitysjonoon kirjataan tuotteen toiminnot, ominaisuudet, vaatimukset, parannukset ja vaaditut korjaukset, jotka mahdollisesti toteutetaan tulevissa iteraatiossa. Jokaiseen kehitysjonon kohtaan on liitetty kuvaus, joka sisältää vähintään kohdan kuvauksen, järjestyksen kehitysjonossa ja arvion työmäärästä (taulukko 1.). Iteraation

tehtävälista on yksinkertaisesti lista kehitysjonon kohdista, jotka toteutetaan kyseisessä iteraatiossa. (Schwaber & Sutherland 2012, 11-13)

Taulukko 1. Esimerkki projektin kehitysjonosta.

Sija	Nimi	Työ	Kuvaus
1.	Tulostietokannan suunnittelu	2	Suunnitellaan tulostietokannan rakenne
2.	Tulostietokannan toteutus	2	Toteutetaan tulostietokanta mdb-tietokantana.
3.	Käyttäjätietokannan suunnittelu	0.5	Suunnitellaan käyttäjätietokanta
4.	Käyttäjätietokannan toteutus	0.5	Toteutetaan käyttäjätietokanta mdb-tietokantana.
5.	Sisäänkirjautumisen ja sisäänkirjautumissivun suunnittelu	1	Suunnitellaan sisäänkirjautumissivu ja sen tekninen toteutus.
6	Sisäänkirjautumisen toteuttaminen	1	Toteutetaan sisäänkirjautuminen

Kehitysjonossa korkeimmalla olevat kohdat priorisoidaan mahdollisuuksien mukaan seuraavan iteraation tehtävälistalle, koska niiden kehittäminen on usein välttämätöntä projektin jatkumiselle. Tästä seuraa myös se, että korkeimmalla olevat kehityskohteet ovat parhaiten ja yksityiskohtaisimmin suunniteltuja, koska niiden suunnitteluun ja tutkimiseen on jo sijoitettu resursseja. Matalammalla olevat kehityskohteet ovat usein epämääräisiä, koska niiden toteuttaminen välittömästi ei ole välttämätöntä ja siten niihin ei ole vielä sijoitettu resursseja. (Schwaber & Sutherland 2012, 11-13)

Mike Cohn suosittelee pitäytymään yhdessä kehitysjonossa per projekti. Kiusaus useampaan eri kehitysjonoon on suuri massiivisissa projekteissa, joissa ohjelmistoa kehittää useampi eri kehittäjäryhmä. Tämä johtaa priorisointiongelmaan, koska ohjelmiston ja projektin kannalta vähemmän tärkeät kehityskohteet saattavat olla yksittäisellä listalla korkeammalla kuin toisella listalla oleva projektin jatkumiselle välttämätön kehityskohde. Ongelmasta päästää eroon luomalla yksi ainoa kehitysjono ja tarjoamalla siitä erilaisia näkymiä eri kehitysryhmille, jotka valitsevat toteutettavat kehityskohteet omasta näkymästään. Tällöin koko projektin tuotantojärjestystä pystytään priorisoimaan helpommin. (Cohn 2010, 330-331)

2.5 Prosessin nopeus ja nopeuden hallinta

Yhdessä iteraatiossa toteutettavien tehtävälistaan merkittyjen tarinoiden yhteenlaskettua painoarvoa kutsutaan ohjelmistoprosessin nopeudeksi (velocity). Yksittäisen tarinan painoarvo mitataan arvioimalla sen vaatima työmäärä optimaalisissa työpäivissä (story-points) (taulukko 2.). Hyvä arvio vaatii kokemusta ja hyvää kykyä arvioida tarinan toteutukseen vaadittava työmäärä. Usein paras tulos saavutetaan rinnastamalla toteutettavan tarinan painoarvo jo aikaisemmin toteutettuun samankaltaiseen tarinaan. (Shore & Warden 2007, 263-266)

Taulukko 2. Esimerkki iteraation tehtävälistasta ja prosessin nopeudesta.

Iteraatio *N:n tehtävälista*	
Tarina	Työpäiviä
Uuden käyttäjän rekisteröinti	1
Käyttäjätietojen muokkaaminen	2
Käyttäjän poistaminen rekisteristä	1
Käyttäjätietojen rekisteristä haku	3
Nopeus	***7***

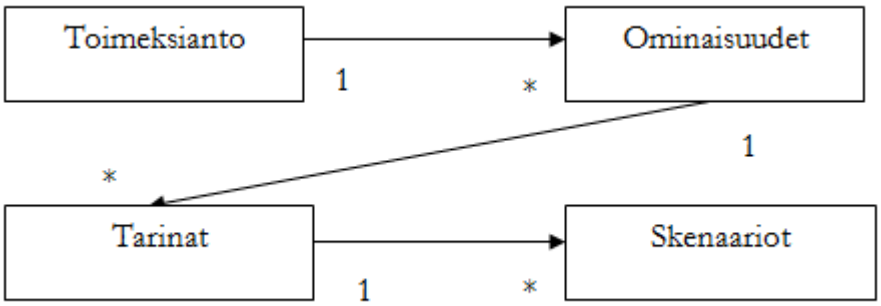
Jokainen iteraatio toimii myös palautteena projektiryhmän suorituskyvystä. Jos projektiryhmä joko ylittää tai alittaa iteraatiolle määritellyn nopeuden on syytä miettiä

syitä ali- tai ylisuorittamiseen ja asettaa iteraation nopeus sopivalle tasolle. Palaute toimii myös automaattisena säätelymekanismina, koska nopeuden kasvaessa tai vähetessä työtaakka asettuu automaattisesti projektiryhmän optimaaliselle suoritustasolle. (Shore & Warden 2007, 262-266)

Koskela esittää kolme yksinkertaista tapaa saavuttaa parhaan mahdollisen tuloksen uhraamatta laatua, jos iteraation nopeus on liian suuri. Yhteistä kaikille kolmelle tavalle on nopeuden lasku iteraatiossa toteutettavien tarinoiden avulla. Ensimmäinen ja yksinkertaisin vaihtoehto on yksinkertaisesti pudottaa tarina iteraation tehtävälistalta. Toinen vaihtoehto on vaihtaa työläs tarina toiseen vähemmän työtä vaativaan tarinaan ja kolmas vaihtoehto on jakaa tarina kahtia pienempiin tarinoihin, jotta iteraatio pysyy sille rajatussa aikataulussa. (Koskela 2008, 346-348)

3 Toimeksiannosta tarinoiden käyttötapauksiin

Hyväksyntätestauslähtöinen ohjelmistotuotanto voidaan kuvata sarjana toimintoja, joiden fokus tarkentuu jokaisella askeleella. Jokainen askel koostuu yhdestä tai useammasta seuraavan askeleen määrittelemästä toiminnosta. Esimerkiksi ominaisuus koostuu yhdestä tai useammasta käyttäjätarinasta ja käyttäjätarina koostuu yhdestä tai useammasta skenaariosta (käyttötapauksesta) (kuvio 2.).



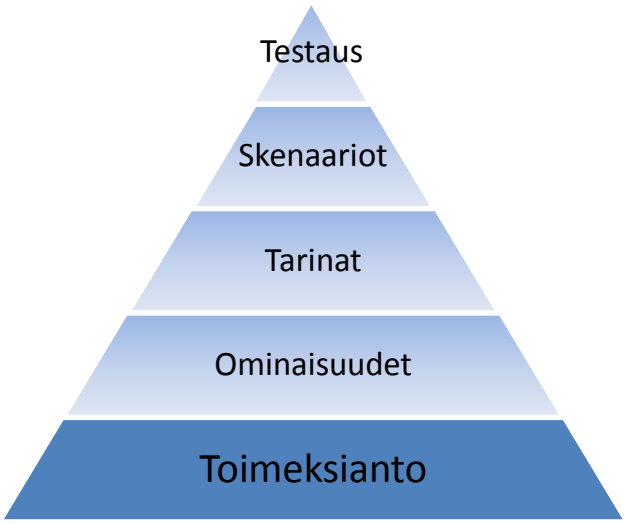
Kuvio 2. Toimeksiannon, ominaisuuksien, tarinoiden ja skenaarioiden suhde toisiinsa.

Kokonaisuutena projekti alkaa toimeksiannon (charter) laatimisesta. Toimeksiannossa määritellään projektin tavoitteet, jotka luovat pohjan koko projektille. Jotta tavoitteet voitaisiin saavuttaa, täytyy ohjelmistolle määritellä toteutettavat ominaisuudet (features). Ohjelmiston täytyy toteuttaa määritellyt ominaisuudet hyväksytysti, jotta ohjelmistoa voitaisiin pitää valmiina ja projektia hyväksytysti suoritettuna. (Pugh 2010, 39-68)

Ominaisuudet koostuvat yhdestä tai useammasta käyttäjätarinasta (user-stories). Käyttäjätarinat jakautuvat yhteen tai useampaan käyttötapaukseen (use-case), jotka voivat jakaantua useampaan skenaarioon (scenario). Kaikkien edellä mainittujen osasten on läpäistävä niille asetetut hyväksyntätestit, jotta projekti voidaan hyväksytysti suorittaa loppuun. (Pugh 2010, 39-68)

3.1 Toimeksianto

Toimeksiannossa määritellään projektin tavoitteet (objectives), visio (vision), toimintasuunnitelma (mission), laajuus (scope) ja periaatteet (principles). Visio kuvaa miksi projektia ollaan tekemässä. Visio realisoituu seuraamalla toimintasuunnitelmassa luonnosteltua polkua projektin tavoitteita kohti toimintaperiaatteita noudattaen. Toimeksianto luonnostellaan yleensä projektin ensimmäisessä kokouksessa kaikkien projektin osapuolten edustajien kesken (kuvio 3.). (Pugh 2010, 39-40)



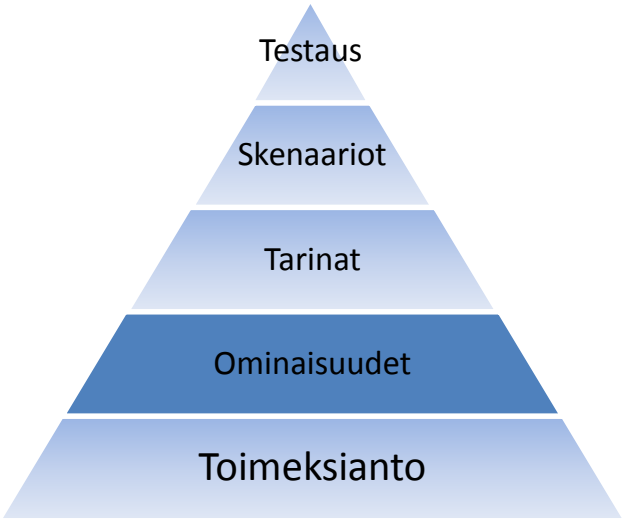
Kuvio 3. Toimeksianto projektin fokuspyramidin pohjana Ken Pughia mukaillen.

Tavoitteet näyttelevät tärkeää osaa toimeksiannossa, koska tavoitteet ovat mitattavissa olevia vaatimuksia, jotka voidaan hyväksyntätestata. Tavoitteet muodostavat projektin onnistumiskriteerien ja hyväksyntätestauksen korkeimman tason ja siten tavoitteiden toteutumisella voidaan mitata onko projekti suoritettu onnistuneesti. Hyvin laadituista tavoitteista käytetään akronyymiä SMART. Hyvät tavoitteet ovat hyvin rajattuja (specific), mitattavissa olevia (measurable), saavutettavissa olevia (achieveable), asiaan kuuluvia (relevant) ja ajoitettavia (time-boxed). (Pugh 2010, 40-41)

3.2 Ominaisuudet

Ohjelmiston ominaisuudet (features) määrittelevät ohjelmiston toiminnallisuuden korkealla tasolla (kuvio 4). Ominaisuudet eivät ota kantaa ohjelmiston sisäiseen toimintaan vaan ne kuvaavat ohjelmiston toimintaa abstraktisti. Leffingwell määrittelee

ominaisuudet ohjelmiston sidosryhmille tarjoamiksi palveluiksi. Ominaisuuksien avulla voidaan määritellä yleiskuva ohjelmiston toiminnasta asiakkaan ymmärtämällä kielellä. Projektin kehitysjonoon kirjatut ominaisuudet muodostavat projektiryhmälle koko ohjelmistoprojektin selkärangan ja toteutussuunnitelman. (Leffingwell 2011, 75-76)



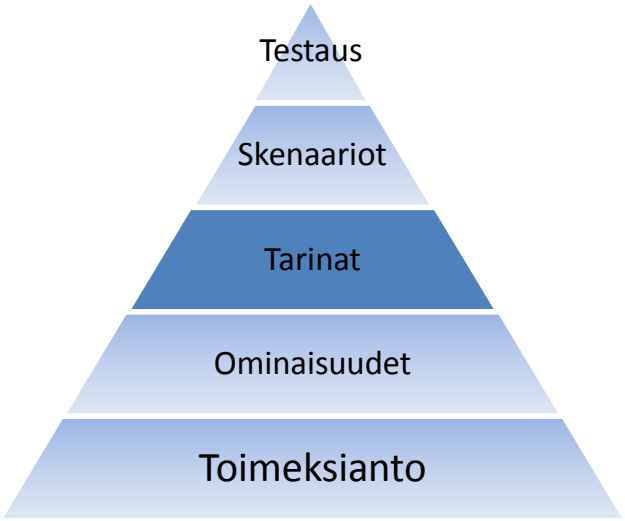
Kuvio 4. Ominaisuuksien sijoittuminen projektin fokuspyramidissa Ken Pughia mukaillen.

Ominaisuudet määritellään yhdessä asiakkaan ja koko projektiryhmän kanssa. Määrittelyn mallina voidaan käyttää aivoriiheä (brainstorming), jossa esitettyjä ominaisuuksia verrataan asiakkaan esittämiin vaatimuksiin ja tavoitteisiin. Tämän jälkeen toteutettavat ominaisuudet valitaan ja niille laaditaan hyväksyntäkriteerit, jotta projektin kuluessa voidaan todentaa milloin ominaisuudet ovat hyväksytysti toteutettu. (Pugh 2010, 43-45)

3.3 Tarinat

Vaatimusmäärittely voidaan suorittaa käyttäjätarinoiksi (user stories) kutsuttujen käyttöesimerkkien avulla. Tarinoiden lähteenä toimivat asiakkaan edustajan kanssa yhteistyössä määritellyt ohjelmiston ominaisuudet. Tarinat eroavat ominaisuuksista siten, että ne ovat abstrakteihin ominaisuuksiin verrattuna yhtä tasoa alempana sekä teknisesti että kuvaannollisesti (kuvio 5.). Tämän vuoksi tarinoiden pohjalta on helpompi luoda laadukkaita hyväksyntätestejä. (Pugh 2010, 47-48)

Asiakkaan edustajan läsnäolo tarinoiden laatimisessa takaa sen, että ohjelmiston kehittäjät saavat ensikädessä tietoa siitä, kuinka ohjelmiston loppukäyttäjä tulee ohjelmistoa käyttämään ja mitä asiakas itse asiassa ohjelmiston toiminnallisuudelta haluaa. Yhdessä asiakkaan kanssa laaditut tarinat antavat luonnollisen pohjan käyttötapauksien rakentamiselle. Käyttötapauksista luotujen testien laatu pyritään varmistamaan testaajan mukaan ottamisella vaatimusmäärittelyprosessiin. (Watkins 2009, 207.)



Kuvio 5. Tarinoiden sijoittuminen projektin fokuspyramidissa Ken Pughia mukaillen.

Ron Jeffries määrittää tarinan konseptin jakamalla tarinan kolmeen erilliseen elementtiin, korttiin (card), keskusteluun (conversation) ja varmistukseen (confirmation). Kortin tehtävänä on toimia muistiinpanovälineenä, joka kuvaa muutamalla lauseella kyseessä olevan tarinan ja toimii lupauksena toteutettavasta toiminnallisuudesta. Keskustelu kuvaa neuvottelua sidosryhmien välillä, minkä perusteella tarinan tarkemmat yksityiskohdat muodostetaan. Varmistus kuvaa tarinan hyväksyntätestiä, toisin sanoen varmistus merkitsee niitä vaatimuksia, joiden toteutuessa tarinaa voidaan pitää valmiina. (Leffingwell 2011, 102-103)

Lasse Koskela kiteyttää tarinoitten konseptin yksinkertaiseen toteamukseen, ”kuka tekee, mitä tekee ja miksi tekee”. Koskela huomauttaa, että usein tarina vastaa vain kahteen ensimmäiseen kysymykseen ja vastaus kolmanteen kysymykseen on

pääteltävissä tarinan kontekstista. Tarinan kuvaus lyhykäisyydessään ja niukkasanaisuudessaan ei ole Koskelan mielestä riittävän tarkka varsinaiseksi vaatimukseksi vaan edustaa lupausta ohjelmistokehittäjän ja asiakkaan kanssa myöhemmin toteutettavasta toiminnallisuudesta. (Koskela 2008, 24.)

3.4 Tarinoiden standardimuoto

Tarinoiden kirjaamiselle on kehitetty standardisoitu muoto, jota kutsutaan käyttäjätarinakortiksi (user story voice). Standardimuodolle on tilaus, koska sotkuisesti useampaan kertaan korjatut tarinat lisäyksineen voivat muuttaa tarinan niin epäselväksi, että tarinan toteuttaja ei enää tiedä mitä hänen pitäisi toteuttaa. Standardisoitu kirjaamistapa noudattaa seuraavaa yksinkertaista kaavaa. (Leffingwell 2011, 103-104)

> <rooli> voi tehdä <toiminto>, jotta <hyöty>
>
> missä:
>
> <rooli> edustaa toimijaa tai hyödyn saajaa.
> <toiminto> edustaa järjestelmältä pyydettyä toimintoa.
> <hyöty> edustaa toiminnosta saavutettavaa hyötyä.

Vaihtoehtoisesti tarina voidaan esittään Leffingwellin sanallistamuotoa mukaillen. (Leffingwell 2011, 189.)

Käyttäjänä <rooli> haluan vertailla tuloksiani tältä vuodelta <toiminto jonka haluan toteutettavan>, jotta voin seurata omaa kehitystäni <hyöty>.

Esimerkistä nähdään kuinka jokainen elementti tarjoaa tärkeää tietoa toteutettavasta toiminnosta. Rooli (user role) antaa toiminnolle käyttäjäkontekstin ja kartoittaa eri käyttäjäryhmien erilaisia tarpeita. Toiminto kuvaa roolin käyttäjän vaatimusta järjestelmän toiminnasta, jonka lisäarvon hyöty määrittelee. Yksinkertaistetusti rooli ilmaisee ”kuka tekee”, toiminto ”mitä tekee” ja hyöty ”miksi tekee”. (Leffingwell 2011, 103-104)

Tarinoille täytyy laatia yhtenäinen sanasto (domain language), koska yhtenäistetty sanasto vähentää väärinkäsityksiä ja siten virheitä testauksessa. Testien tulee käyttää samaa sanastoa kuin tarinasta laadittu käyttötapaus. Jos sanasto ei kuitenkaan ole yhtenäinen, tulee sanasto tarvittaessa korjata yhdenmukaiseksi. Moniselitteisiä termejä on vältettävä ja ne on korvattava yksiselitteisillä termeillä. (Pugh 2010, 73.)

Lasse Koskela kritisoi standardimuotoista käyttäjätarinakorttia liian jäykäksi tavaksi kirjata tarinoita ja suosittelee tarinoiden esitystavan pitämistä mahdollisimman lähellä asiakkaan ymmärtämään kieltä ja kuvausta. Koskela ei kuitenkaan täysin tyrmää käyttäjätarinakortteja eikä pidä niitä täysin epäonnistuneena ideana vaan hakee joustavampaa tapaa ilmaista tarinat. (Koskela 2008, 24.)

3.5 Käyttäjäpersoonat

Roolia voidaan laajentaa antamalla sille ominaisuuksia. Esimerkiksi ylläpitäjän roolin ominaisuudet ja oikeudet ovat erilaisia kuin peruskäyttäjän vastaavat. Rooliin voidaan lisätä myös käyttötapoja ja tavoitteita kuvaavia ominaisuuksia tai tunnuslukuja kuten roolin alakohtainen tietämys tai kuinka monta kertaa päivässä rooli keskimäärin käyttää toiminnallisuutta (taulukko 3.). Tällä tavoin roolista ja sen tehtävistä saadaan selkeä kuva tarinoiden rakentamista varten. Roolista voidaan rakentaa myös muistisääntönä toimiva persoona (persona) , joka kuvaa puhekielellä roolin edustaman henkilön. (Pugh 2010, 49-51)

Taulukko 3. Esimerkki rooleja Ken Pughia ja Dean Leffingwelliä mukaillen

Persoona	**Kategoria**	**Odotukset**	**Toiminnot**
Käyttäjä	Ensisijainen käyttäjä (ihminen)	Tulosten tilastointi ja seuraaminen	Tulosten talletus, muokkaus ja poisto
Tietokantao Ohjelmisto	Toissijainen käyttäjä (laite/ohjelmisto)	Seuraa käyttäjän toimia ja suorittaa tietokantaoperaati ot käyttäjän toimien mukaisesti	Suorittaa tietokantaoperaati ot
Ylläpitäjä	Toissijainen käyttäjä (ihminen)	Asentaa ohjelmiston käyttäjän tietojärjestelmään	Asentaa ohjelmiston ja ylläpitää ohjelmistoa

Roolia voidaan laajentaa edelleen palvelemaan ohjelmiston suunnittelua. Roolit voidaan jakaa kahteen ryhmään, pääkäyttäjiin (primary user) ja toissijaisiin käyttäjiin (secondary user). Jako kahteen eri käyttäjäryhmään on tarpeellinen, koska käyttäjäryhmien perustarpeet poikkeavat toisistaan (taulukko 3.). Ohjelmiston käyttöliittymä ja käyttörajapinnat (interfaces) suunnitellaan pääkäyttäjien tarpeita varten sillä olettamuksella, että pääkäyttäjä ei muuten kykenisi suoriutumaan tehtävistään. Toissijainen käyttäjä kykenee suorittamaan itselleen tärkeät operaatiot hyväksikäyttämällä pääkäyttäjän ehdoilla suunniteltua ohjelmistoa. (Leffingwell 2011, 126-127)

Rooleja voidaan hyväksikäyttää järjestelmän kokonaissuunnittelussa. Pääkäyttäjä roolien joukosta voidaan valita pääkäyttäjien pääkäyttäjä, jonka tarpeiden pohjalta koko järjestelmän käytettävyys suunnitellaan. Tunnistamalla sekä pääkäyttäjien että toissijaisten käyttäjien tarpeet voidaan tunnistaa, mitä ominaisuuksia järjestelmän täytyy toteuttaa ja millainen painotus kullekin ominaisuudelle täytyy antaa. (Leffingwell 2011, 128.)

3.6 Tarinoiden laatukriteerit

Tarinoiden kirjoittamisessa yritetään säilyttää INVEST-kriteerit (independent, negotiable, valuable to users or customers, estimatable, small, testable). Riippumattomuudella (independent) tarkoitetaan sitä, että kunkin tarinan tulisi välttää mahdollisuuksien mukaan riippuvuussuhteita muihin tarinoihin. Riippuvuussuhteita yritetään välttää sen vuoksi, että riippuvuussuhteet luovat priorisointi, aika-arviointi ja työjärjestysongelmia. Riippumattomuus voidaan saavuttaa toisistaan riippuvien tarinoiden yhdistämisellä tai tarinan jakamisella useiksi pienemmiksi tarinoiksi. (Cohn 2004, 17-18)

Neuvoteltavuus (negotiable) merkitsee sitä, että tarinakuvauksiin kirjatut tarinat eivät ole kiveen hakattuja vaatimuksia, vaan asiakkaan kanssa neuvoteltuja ja uudelleen neuvoteltavissa olevia ohjeita ohjelmiston toteuttamiseen. Hyödyllisyys käyttäjille ja asiakkaille (valuable to users or customers) saavutetaan siten, että pyritään välttämään mahdollisuuksien mukaan tarinoita, jotka ovat hyödyllisiä vain ohjelmiston kehittäjille. Arvioitavuudella (estimatable) tarkoitetaan sitä, että tarinan koko ja toteutukseen kuluva aika pitäisi olla arvioitavissa. (Pugh 2010, 55-56)

Pienuus (small) kriteerin nimestä huolimatta tarkoittaa tarinoiden kirjoittamista sopivan kokoiseksi. Sopivan tarinakoon saavuttamiseksi voidaan käyttää samoja välineitä kuin tarinan riippumattomuuden saavuttamiseksi. Sopivalla tarinakoolla voidaan välttää liian suuresta tarinakoosta aiheutuvia suunnitteluongelmia. Tarinat ovat kirjoitettava siten, että ne ovat myös testattavia (testable). Testattavuudella varmistetaan, että tarinalla on selkeät kriteerit milloin voidaan todeta, että tarina on valmis ja toimii tarkoituksen mukaisesti. (Pugh 2010, 55-56)

Tarinoiden koossa pyritään siihen, että tarina on toteutettavissa yhdessä iteraatiossa. Tarinan sopiva koko riippuu useasta tekijästä, kuten iteraation nopeudesta ja montako tarinaa projektiryhmä haluaa julkaista julkaisukierroksessa. Tarinoita voidaan tarvittaessa jakaa ja yhdistellä, jotta sopiva koko saavutetaan. (Shore & Warden 2007, 258-259)

3.7 Tarinoiden jakaminen osatarinoiksi

Tarinat ovat useimmiten johdettu suuremmista kokonaisuuksista kuten ominaisuuksista tai useista toiminnoista koostuvista taruista (epics). Tarut ja ominaisuudet (features) ovat luonteeltaan tarinakokoelmia (compound stories) ja niistä johdetut osatarinatkin saattavat olla liian suuria toteutettavaksi yhdessä iteraatiossa. Tällöin liian suuria tarinoita täytyy pilkkoa pienemmiksi iteraatiossa toteutettavissa oleviksi tarinoiksi. Vaikka tarinoiden pilkkomiselle ei ole mitään virallista mallia, Leffingwell ehdottaa seuraavia kymmentä eri kriteeriä tarinoiden hajottamiseen pienemmiksi osatarinoiksi. (Leffingwell 2011, 111-114)

Leffingwellin ensimmäinen ehdotus on tarinan jakaminen osatarinoihin työjärjestyksen mukaan. Tällöin tarinasta yritetään tunnistaa tavoitteeseen vaadittavista osatoiminnoista selkeä työjärjestys ja toteutetaan osatoiminnot erillisinä tarinoina. Toinen vaihtoehto on jakaa tarina osatarinoihin liiketoimintasääntöjen mukaan. Kyseinen jako voi tulla kysymykseen esimerkiksi erilaisten hakuparametrien käytössä kuten haku postiosoitteen, siviilisäädyn tai ostokäyttäytymisen mukaan. Kolmannessa tapauksessa suuri tarina voidaan jakaa osatarinoihin osatarinoiden toteutusjärjestyksen ja tärkeyden mukaan. Tällainen jako voi tulla kysymykseen esimerkiksi silloin, kun osatarinaa tarvitaan järjestyksessä seuraavan tarinan toteuttamiseen. (Leffingwell 2011, 111-114)

Neljännessä tapauksessa tarina kasvaa kasvamistaan määrittelykeskusteluissa. Tällöin on syytä miettiä mikä on yksinkertaisin mahdollinen versio tarinasta, jolla vaadittu toiminnallisuus voidaan toteuttaa. Tällöin monimutkaiset variaatiot voidaan rikkoa omiksi helpommin toteutettavissa oleviksi osatarinoiksi. Viidennessä tapauksessa aineiston erilainen esitystapa tai toisistaan poikkeavat tietolähteet ovat merkittäviä ohjelmiston monimutkaisuutta kasvattavia tekijöitä. Tällöin voidaan harkita viime hetken (just-in-time) lisäyksiä tarinoihin. Kielilokalisaatio on hyvä esimerkki tästä lähestymistavasta. (Leffingwell 2011, 111-114)

Kuudennessa tapauksessa varsinainen totetutettava toiminnallisuus ei ole monimutkainen, mutta käyttöliittymän esillepano tekee tarinasta monimutkaisen. Tällöin tarinasta voidaan erottaa yksinkertaisin esitystapa ja rakentaa

monimutkaisemmat esillepanot myöhemmin erillisinä tarinoina. Seitsemännessä tapauksessa järjestelmän perusvaatimusten toteuttaminen ei ole vaikeaa, mutta niiden hiominen nopeammaksi, sulavammaksi, tarkemmaksi ja skaalautuvammaksi on. Tällaisessa tapauksessa tarina voidaan jakaa perustarinaan, joka toteuttaa perustoiminnallisuuden ja lisätarinoihin, jotka toteuttavat perustarinasta saatujen kokemusten perusteella monimutkaisemmat toiminnot. (Leffingwell 2011, 111- 114)

Kahdeksannessa tapauksessa avainsanat kuten muokkaa, järjestä tai ylläpidä sisältävät ajatuksen useammista eri operaatiosta, joiden perusteella tarina on luonnollista jakaa eri operaatiot toteuttaviin osatarinoihin. Yhdeksännessä tapauksessa monimutkaiset tarinat joista on kehitetty käyttötapauksia, voidaan usein jakaa osatarinoiksi käyttötapausten mukaan. Kymmenentenä ja viimeisenä tapauksena Leffingwell tarjoaa Extreme Programming kehysmallia noudattavan XP-piikin. Monimutkaisen tai haastavan tarinan ratkaisemiseen voidaan käyttää joko teknistä tai toiminnallista piikkiä (technical/functional spike). Tarina voidaan jakaa osatarinoihin piikistä saatujen ratkaisumallien avulla. (Leffingwell 2011, 111-114)

Shore ja Warden ehdottavat, että keskimääräisen projektiryhmän kannattaa ottaa tavoitteeksi sellainen tarinakoko, että projektiryhmä voi toteuttaa 4 – 10 tarinaa julkaisukierroksessa. He ehdottavat käytettäväksi seuraavia tarinatyyppejä käyttäjätarinoiden lisäksi; virhetarina (bug story), laatutarina (non-functional story) ja dokumentaatiotarina (documentation story). Virhetarinoiden tarkoitus on aikatauluttaa olemassa olevan tarinan virheenkorjaus projektin kehitysjonoon. Laatutarinoilla pyritään varmistamaan ohjelmiston laadullisten kriteerien täyttyminen. Dokumentaatiotarinoilla varmistetaan riittävä dokumentaatio kuten käyttäjäohjeet. (Shore & Warden 2007, 258-259)

3.8 Käyttötapaukset

Jotta tarina voisi läpäistä sille asetetut hyväksyntäkriteerit, täytyy tarinalla olla käyttötapauksia, joista hyväksyntätestit laaditaan. Käyttötapaus kuvaa tarinan kulkua askel askeleelta käyttäjän ja järjestelmän välisenä toiminto ja vaste keskusteluna. Käyttötapaus on usein osa työnkulkua (workflow). Työnkulku sisältää järjestelmän

ulkopuolisia toimintoja, joihin itse kehitettävä järjestelmä ja käyttötapaus eivät ota kantaan (taulukko 4). Tällöin käyttötapauksen laatijoiden täytyy eristää varsinainen käyttötapaus työnkulusta. (Pugh 2010, 57-59)

Taulukko 4. Tuloksen kirjaamisen työnkulku Ken Pughia mukaillen.

Tuloksen kirjaaminen tietokantaan
1. Käyttäjä saa uutta tietokantaan talletettavaa tietoa.
2. Käyttäjä käynnistää tietokantasovellutuksen.
3. Käyttäjä kirjautuu sisään.
4. Käyttäjä syöttää ja pyytää järjestelmää tallettamaan tiedot haluamalleen paikalle.
5. Järjestelmä tallettaa tiedot.
6. Järjestelmä ilmoittaa tietojen onnistuneesta tallentamisesta.

Peter Haumer määrittelee käyttötapauksen (use case) yleisesti sarjana toimintoja, jotka tuottavat toimijalle (actor) todennettavissa olevan hyödyn. Haumerin kuvauksessa käyttötapaus määrittelee toimijan näkökulmasta toiminnalliset ja laadulliset vaatimukset toimijan asettaman tavoitteen saavuttamiseksi. Tästä syystä läyttötapauksen laatimista varten on selvitettävä kuka käyttötapauksen toimija (Haumer 2004, 5.). Leffingwell määrittelee toimijan henkilöksi tai muuksi osapuoleksi, joka käyttää järjestelmää. Leffingwellin jaottelussa toimijat jaetaan kolmeen yleisryhmään, käyttäjiin, vieraisiin järjestelmiin tai sovellutuksiin sekä laitteisiin (Leffingwell 2011, 370.).

Käyttötapauksella on yksi pääkulku (main course, happy path) joka kuvaa ongelmitta onnistuneen käyttötapauksen kulun. Käyttötapaus voi kuitenkin sisältää poikkeuksia tai vaihtoehtoisia kulkuja, jolloin käyttötapaus haaroittuu useammaksi eri kuluksi, joihin kuhunkin pätee omat suoritussääntönsä. Tästä syystä on syytä aloittaa käyttötapauksen rakentaminen pääkulusta ja lisätä mahdolliset poikkeukset ja vaihtoehtoiset kulut pääkulun suunnittelun jälkeen. Pääkulkua, vaihtoehtoisia kulkuja ja poikkeuksia kutsutaan yleisesti skenaarioiksi. Pääkulkua voidaan erottelun helpottamiseksi kutsua pääskenaarioksi. (Pugh 2010, 59-60)

Käyttötapauksille määritellään myös järjestelmältä vaadittu alkutila (pre-conditions), jolloin käyttötapauksen suoritusta voidaan yrittää ja onnistuneen suorituksen jälkeinen lopputila (post-conditions). Jokaisella käyttötapauksella on myös toimija (actor), joka lähes aina on järjestelmän käyttäjä. Käyttötapauksesta voidaan laatia yksinkertainen kuvaus, johon kaikki yllämainitut tiedot voidaan merkitä. Kuvaukseen voidaan liittää myös muuta tietoa kuten esimerkiksi aihealueen omia sääntöjä. (Pugh 2010, 59-63)

4 Hyväksyntätestaus

Toisin kuin perinteisissä ohjelmistoprosesseissa ketterässä hyväksyntätestauslähtöisessä ohjelmistokehityksessä hyväksyntätestit eivät merkitse pelkästään jälkikäteen asiakkaan kanssa tehtäviä testejä, joiden tarkoitus on tarkistaa täyttääkö tuotettu ohjelmisto sille asetetut vaatimukset. Ketterä hyväksyntätestauslähtöinen ohjelmistokehitys kääntää koko prosessin päälaelleen ja muuttaa hyväksyntätestien merkityksen jälkijättöisestä tarkastustoimenpiteestä aktiiviseksi osaksi koko ohjelmistoprosessia.

Hyväksyntätestien tarkoitus ei ole toimia pelkästään toiminnallisuuden testaajina vaan niiden tehtävä on toimia mitattavina vaatimuksina, jotka laaditaan ennen testauksen kohteen laatimista. Tällä tavoin tiedetään etukäteen mitä toiminnallisuudelta vaaditaan ja vaatimus voidaan testata. Lähestymistapa jossa hyväksyntätestit määrittelevät tuotettavan toiminnallisuuden ominaisuudet ja rajat vähentävät virheitä sekä takaavat tuotetun ohjelmiston laadun ja asiakkaan tarpeiden tyydyttämisen.

Hyväksyntätestien laatu ja testaustapa riippuvat testauksen kohteen laadusta. Käyttötapauksien hyväksyntätestaus eroaa projektin hyväksyntätestauksesta. Yleisenä mittarina voidaan pitää sitä, että mitä korkeamman tason vaatimuksesta on kysymys, sitä abstraktimpaa hyväksyntestaus on. On kuitenkin tärkeää huomioida, että projektia ei voida sanoa onnistuneesti suoritetuksi, ellei se läpäise kaikkia hyväksyntätestauksen tasoja. . Lisäksi on pantava merkille, että hyväksyntätestaus ei syrjäytä millään tavoin perinteistä yksikkö- tai integegraatiotestausta vaan kaikkia testaustoimia suoritetaan lomittain.

4.1 Hyväksyntätestin ominaisuudet

Lasse Koskela määrittelee hyväksyntätestit lyhyellä viiden kohdan listalla. Asiakas omistaa hyväksyntätestit. Hyväksyntätestit kirjoitetaan yhdessä asiakkaan, kehittäjän ja testaajan kanssa. Hyväksyntätestit käsittelevät kysymystä ”mitä” ei kysymystä ”kuinka”. Hyväksyntätestit kirjoitetaan aihealueen (problem domain) kielellä. Hyväksyntätestit ovat suppeita, tarkkoja ja yksiselitteisiä. (Koskela 2007, 28-30)

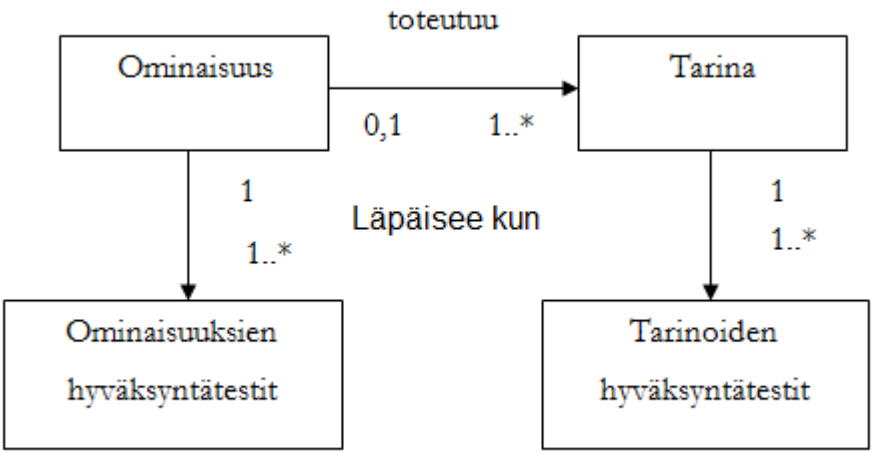
Asiakas omistaa hyväksyntätestit, koska asiakkaalla on paras käsitys ongelma-alueesta ja selkein ymmärtämys siitä, mitä toiminnallisuudelta vaaditaan. Kun asiakas osallistuu määräävässä roolissa hyväksyntätestien laatimiseen kehittäjien ja testaajien kanssa, voidaan estää ohjelmistokehittäjille ominainen keskittyminen teknisiin yksityiskohtiin itse ominaisuuden toimivuuden sijasta. (Koskela 2007, 28-30)

Keskittymällä kysymykseen ”mitä” ilmaistuna asiakkaan ymmärtämällä kielellä voidaan teknisen toteutuksen sijasta keskittyä siihen, mitä asiakas ohjelmiston toiminnallisuudelta haluaa. Tällä tavoin taataan asiakkaalle mahdollisimman suuri lisäarvo. Samoista syistä hyväksyntätestit yritetään myös pitää suppeina, tarkkoina ja yksiselitteisinä, jolloin yksittäinen testi keskittyy tarkasti rajattuun mitattavissa olevaan asiaan. Testit täytyy kirjoittaa myös aihealueen käyttämällä kielellä, jotta asiakas ymmärtää testin. (Koskela 2008, 29-31)

4.2 Projektin ja ominaisuuksien hyväksyntätestaus

Projektin toimeksiannossa mainitut projektin tavoitteet muodostavat projektin korkeimman hyväksyntätestauksen tason. Ellei asetettuja tavoitteita ole saavutettu, ei projektia voida pitää valmiina. Projektin tavoitteet ovat siis korkeimman tason mitattavissa olevia vaatimuksia. On myös mahdollista, että tavoitteet ovat ominaisuuksista tai tarinoista riippumattomia, jolloin ominaisuudet ja tarinat toteuttavat tavoitteet välillisesti. Esimerkki tällaisesta tavoitteesta voisi olla tilastointiohjelman tulosten käsittelyajan puolittaminen. (Pugh 2010, 39-44)

Ominaisuuksien hyväksyntätestaus on läheisessä suhteessa tarinoiden hyväksyntätestaukseen, koska ominaisuudet koostuvat yhdestä tai useammasta tarinasta. Hyväksyntätestauskeskeisen ohjelmistokehityksen hyväksyntätestaus pyörii tarinoiden hyväksyntätestauksen (story acceptance testing) ympärillä. Ominaisuutta ei voida hyväksyä valmiiksi ennen kuin se läpäisee kaikki ominaisuudelle asetetut hyväksyntätestit ja kaikki ominaisuuteen liitetyt tarinat ovat läpäisseet hyväksytysti omat hyväksyntätestinsä (kuvio 6.). Luonteeltaan tarinoiden testit ovat toiminnallisuustestejä (functional test). (Leffingwell 2011, 187-188)



Kuvio 6. Ominaisuuksien suhde tarinoihin (Leffingwell 2011, 187.)

4.3 Tarinoiden hyväksyntäkriteerit

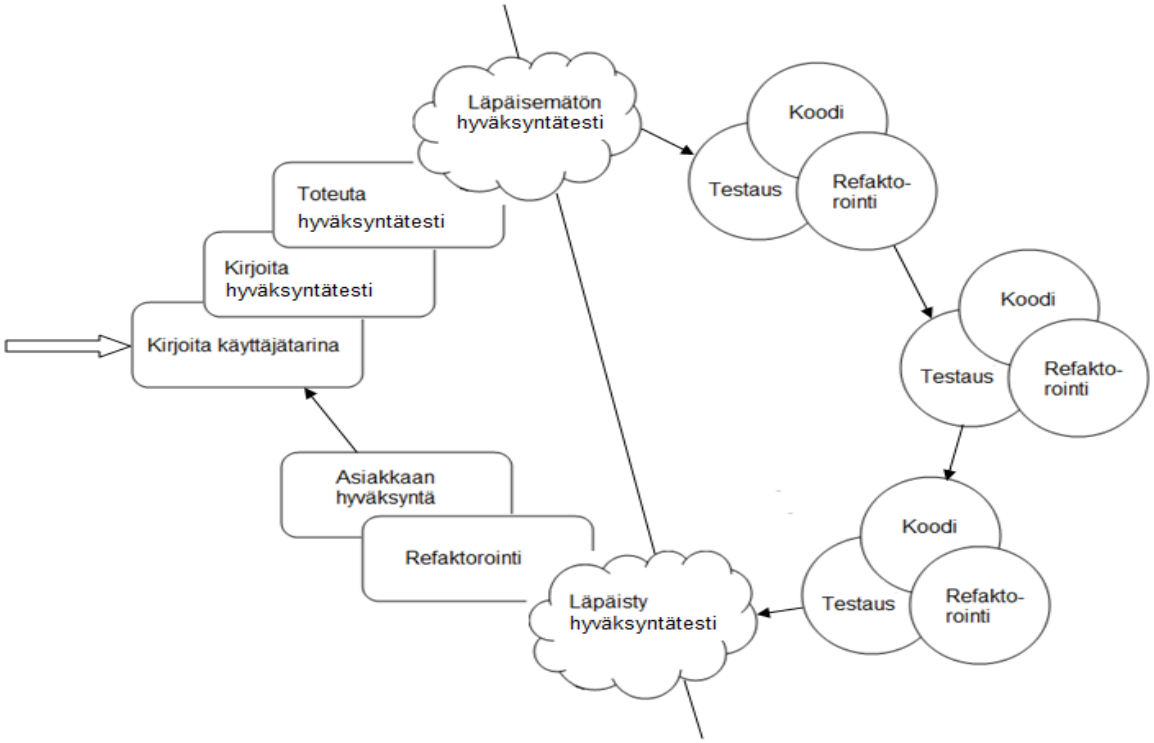
Jotta tarinoille voidaan kirjoittaa kattavat hyväksyntätestit, tarinalle täytyy laatia hyväksyntäkriteerit. Hyväksyntäkriteerit kuvaavat puhekielellä tarinalta vaaditut toiminnallisuudet ja rajoitteet. Hyväksyntäkriteerit eivät ota kantaa tarinan tekniseen toteutukseen ja niiden tehtävä on antaa ohjelmiston kehittäjille ja asiakkaalle kokonaiskuva tarinan hyväksymiseen tarvittavista ominaisuuksista (taulukko 5). Hyväksyntäkriteerit laajennetaan varsinaisiksi hyväksyntätesteiksi juuri ennen kuin tarina toteutetaan konkreettisesti. (Pugh 2010, 52.)

Taulukko 5. CD-vuokraamon vuokrausprosesin hyväksyntäkriteerit (Pugh 2010, 52.)

Hyväksyntäkriteerit eivät siis ole varsinaisia testejä vaan hyväksyntäkriteerit asettavat tarinalle ne kriteerit jotka tarinan täytyy toteuttaa, jotta tarinan toiminnallisuuden toteutusta voidaan pitää hyväksyttynä. Kriteerit muodostuvat usein tarinan toteutusta koskevista asiakaskeskusteluista ja niihin saattaa tulla lisäyksiä tarinan toteutuksen yhteysdessä. (Leffingwell 2011, 104-105,192)

4.4 Tarinoiden hyväksyntätestit

Tarinoiden hyväksyntätestaus on monivaiheinen toimi, jolla pyritään selvittämään milloin tarinaa voidaan kutsua hyväksytysti toteutetuksi ja julkaisukelpoiseksi. Toisin sanoen tarinoiden hyväksyntätestit luovat tarinalle mitattavat vaatimukset, jotka tarinan täytyy täyttää jotta se voidaan hyväksyä valmiiksi (done done) (kuvio 7). Hyväksyntätestit laaditaan yhteistyössä asiakkaan kanssa ja laatimisessa käytetään apuna aikaisemmin asetettuja tarinan hyväksyntäkriteerejä. (Pugh 2010, 52,63.)



Kuvio 7. Kuvaus tarinan hyväksymisestä julkaisukelpoiseksi. (Koskela 2008, 342.)

Hyväksyntätestit kirjoitetaan tarinan käyttötapauksille. Testausprosessin helpottamiseksi käyttötapauksesta pyritään eristämään käyttötapauksen pääkulku (main course), sivukulut (alternative course) ja poikkeukset (exceptions), joille kirjoitetaan omat hyväksyntätestinsä. Hyväksyntätesti voidaan onnistuneesti laatia ja läpäistä vain jos hyväksyntätestille on määritetelty testin alkutila ja tavoiteltava lopputila. Testin alkutilan ja tavoiteltavan lopputilan määräävät testattavan käyttötapauksen alkutila ja odotettu lopputila. (Pugh 2010, 60-63)

Tarinoiden hyväksyntätestit voidaan jakaa karkeasti kahteen eri luokkaan. Sääntötesteissä (business rule test) testataan, että tarinat noudattavat aihealueen sääntöjä ja skenaariotestit (test scenario) testaavat käyttötapauksista laaditut skenaariot. Sääntötestit eroavat skenaariotesteistä teknisesti siten, että sääntötesteillä ei välttämättä ole testin alkutilaa ja lopputilaa. Sääntötesti testaa yksinkertaisesti järjestelmän toimintaa vertailemalla syötettä ja vastetta oletettuun oikeaan vasteeseen. Tästä johtuen sääntötestit ovat yleensä yksinkertaisempia kuin käyttötapauksista kirjoitetut skenaariotestit. Sääntötestejä voidaan myös kutsua laskennallisiksi testeiksi (calculation test). Joissain tapauksissa on mahdollista, että käyttötapaukseen sekoittuu piirteitä, jotka

vaativat molempia testityyppejä, tällöin on syytä erottaa sääntötestit ja skenaariotestit erillisiksi kokonaisuuksiksi, jotta ylimääräistä monimutkaisuutta ja päällekkäistä testausta voitaisiin välttää. (Pugh 2010, 71-73)

4.5 Hyväksyntätestin ja testimateriaalin kuvaus

Jotta testitapauksia voitaisiin luoda, tarvitaan testimateriaalia. Testimateriaali voidaan esittää taulukoidussa muodossa ja taulukko upottaa hyväksyntätestin kuvaukseen, jotta koko hyväksyntätesti olisi helpompi hahmottaa. Testimateriaalitaulukot voidaan jakaa kolmeen eri ryhmään, laskennalliseen taulukkoon (calculation table) (taulukko 6.), tietotaulukkoon (data table) (taulukko 7.) ja toimintotaulukkoon (action table) (taulukko 9.). (Pugh 2010, 73-75)

Laskennallista taulukkoa käytetään ilmaisemaan syöte-vasteparin oletettua lopputulosta. Tietotaulukkoa käytetään ilmaisemaan järjestelmässä olemassa olevaa tietoa, kuten muuttujien arvoja tai tietokannan riviarvoja. Ehtotaulu on variaatio tietotaulusta, jossa näytetään vain tietyn ehdon täyttävät rivit tietotaulusta (taulukko 8.). Toimintotaululla voidaan kuvata testin kulkua käyttöliittymän näkökulmasta. (Pugh 2010, 73-76)

Taulukko 6. Esimerkki laskennallisesta taulukosta.

Asiakasryhmien alennuksien suuruus 100e ylittävästä ostosta		
Osto	Asiakasryhmä	Alennusprosentti
200e	tavallinen	0%
50e	vakio	0%
200e	vakio	2%
150e	premium	5%

Taulukko 7. Esimerkki tietotaulukosta

Asiakastiedot	
Nimi	Tunnus
Heikki Heikkilä	12041956
Maija Maijala	14121948

Taulukko 8. Esimerkki ehtotaulukosta.

H-kirjaimella alkavat asiakkaiden sukunimet	
Nimi	Tunnus
Heikki Heikkilä	12041956

Taulukko 9. Esimerkki toimintataulukosta.

Sähköpostin lähettäminen		
Toiminto	Tapahtuma	Arvo
Syötä	Sähköpostiosoite	heikki.heikkila@jee.mail
Paina	Lähetä-nappula	
Tarkista	Lähetetty?	true

Lasse Koskela esittää äärimmilleen yksinkertaistetun tavan kuvata hyväksyntätestin testimateriaali ja testin kulku. Koskelan mallissa hyväksyntätestin kuvaus on äärimmilleen yksinkertaistettu ja testimateriaalin, testin tarkoituksen ja testin kulun ymmärtämiseen ei tarvita teknistä osaamista. Kuvaustavassa on erotettu kaksi pääkohtaa, toiminto (action) ja parametrit (parameters), jotka kuvaavat testin kulkua askel askeleelta (taulukko 10.). (Koskela 2008, 338.)

Taulukko 10. Lasse Koskelan laatima hyväksyntätesti asiakkaan nimen poimimisesta puhelinnumeron perusteella ja nimen tulostaminen näytölle. (Koskela 2008, 338.)

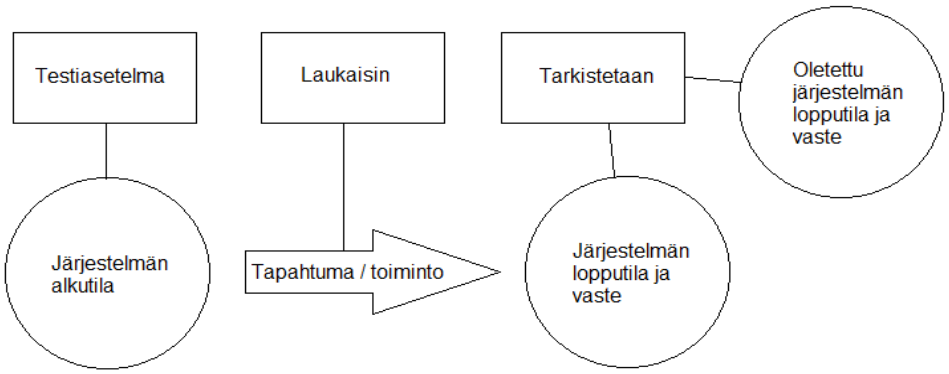
Toiminto	Parametrit
Aloita puhelu	numero 5555-1234, tili 123456
Hyväksy puhelu	numero 5555-1234
Tarkista teksti	tili 123456
Hyväksyt teksti	Antti Asiakas

Ken Pugh:in versio hyväksyntätestin kirjallisesta esittämisestä on Lasse Koskelan versiota yhtä astetta monimutkaisempi. Pugh:in versio käyttää hyväkseen alkutila, laukaisin, lopputila – ketjua testin etenemisen kuvaamiseen. Koskelan tyylistä poiketen Pugh:in tyylillä testin kuvaukseen voidaan upottaa suurempi määrä testimateriaalia käyttämällä aikaisemmin luvussa kuvattuja laskenta-, tieto-, ja toimintataulukkoja (Pugh 2010, 78-79)

4.6 Hyväksyntätestin rakenne ja konteksti

Yksinkertaisimmillaan testin rakenne noudattaa kolmea askeletta. Testi alkaa ennalta määrätystä järjestelmän alkutilasta (initial system state), jonka muutoksen laukaisijana (trigger) toimii tapahtuma (event) tai toiminto (action), josta seuraa järjestelmän tilan muutos. Lopullista järjestelmän tilaa (final system state) tai saatua vastetta (output) verrataan oletettuun oikean tilaan tai vasteeseen, jotta voidaan varmistaa toimiiko järjestelmä testin puitteissa oikein (kuvio 8.). (Pugh 2010, 71.)

Testille tulee määritellä konteksti. Kontekstin määritteleminen on tärkeää, koska testauksen tulos riippuu järjestelmän tilasta testaushetkellä. Joissain tapauksissa järjestelmän syötteet ja tulosteet ovat riippuvaisia edeltävistä syötteistä, koska syötteestä muodostettava tuloste voi muuttaa sisäisesti persistentin järjestelmän tilaa tai ulkoisesti persistentin järjestelmän ulkoista tietovarastoa. (Pugh 2010, 70-71)



Kuvio 8. Testin kulku Ken Pughin mallia mukaillen. (Pugh 2010, 71.)

Esimerkiksi sisäisesti persistentissä järjestelmässä CD:n vuokrausprosessissa CD:n vuokraus epäonnistuu, jos CD on jo vuokrattu edellisellä syötteellä ja järjestelmän sisäinen tila on muuttunut. Ulkoisesti persistentti järjestelmä käsittelisi tilannetta vain yhtenä syöte- ja tulosteketjuna, jossa lopputulemana voisi olla vuokrasopimuksen tulostaminen ja tulosteen tallettaminen ulkoiseen tietokantaan tai toisen yrityksen aiheuttama poikkeustilanne tietokannassa. (Pugh 2010, 70-71)

4.7 Hyväksyntätestien totetuttaminen

Hyväksyntätestien toteuttaminen on tapauskohtaista. Toteuttamistapaan vaikuttaa järjestelmän toteuttamiseen käytetyt tekniikat ja järjestelmän rakenne. Usein ei ole mahdollista käyttää hyväksyntätestien laatimiseen samaa ohjelmointikieltä kuin järjestelmän toteutukseen on käytetty, koska kaikki ohjelmointikielet ja tekniikat eivät sovellu hyvin hyväksyntätestien vaatimaan toiminnallisuuden testaamiseen. On helppoa testata WWW-sovellus HTML-protokollan avulla käytännössä millä tahansa ohjelmointikielellä, mutta integroidun järjestelmän testaaminen millään muulla kielellä kuin järjestelmän toteutukseen käytetyllä kielellä on usein käytännöllisesti mahdotonta. (Koskela 2008, 333.)

Mahdollisuuksien mukaan laaditut hyväksyntätestit voidaan automatisoida. Tämä säästää aikaa ja resursseja. Testitien automatisoinnissa tulee pyrkiä tilanteeseen, jossa

automatisoitu testi palauttaa testistä yksinkertaisyn hyväksytty tai hylätty arvon. Testi voidaan automatisoida testitapauksen kulkua hyväksi käyttämällä, jolloin testitapauksen kulun toiminnot ja parametrit toimivat testin automatisoituina askeleina. (Koskela 2008, 339)

Hyväksyntätestien toteuttamiseen voidaan antaa neljä perustekniikkaa manuaalisen testaamisen lisäksi. Jos toteutettava järjestelmä ja sen käyttöliittymä tukee skriptikieltä, hyväksyntätestit voidaan toteuttaa skriptikielellä. Tällöin testi voidaan toteuttaa simuloimalla käyttäjän toimintaa askel askeleelta jolloin testin lopputulosta voidaan verrata oletettuun lopputulokseen. Toinen mahdollisuus on luoda erillinen graafinen (GUI)- tai komentokehote (CLI) käyttöliittymä hyväksyntätestien suorittamiseen. Kolmas tapa on käyttää yksikkötestaukseen käytettävää kehystä ja neljännessä tavassa voidaan käyttää eritoten hyväksyntätestaukseen suunniteltua testauskehystä. (Pugh 2010, 31-34)

5 Ohjelmistoprojekti - case Frisbeegolf-tilastointityökalu

Projektin tavoitteena oli luoda vuonna 2011 perustetulle Frisbeegolffaajat-ryhmälle räätälöity tilastointityökalu. Frisbeegolffarit ovat joukko frisbeegolffin harrastajia, jotka järjestävät pelimatkoja, peli-iltoja ja ajanvietettä työn, perhe-elämän ja opiskelupaineiden välillä tasapainottelevalle miesjoukolle. Opinnäytetyön puitteissa ohjelmisto toteutettiin vain välttämättömillä toiminnoilla. Ryhmän jäsenistön toivomuksesta ohjelmiston kehitystyötä jatketaan opinnäytetyön valmistumisen jälkeen älypuhelin integraatiolla. Projekti toimi myös opinnäytetyön laatijan työnharjoittelun osana. Kokonaisuudessaan projekti testaa opinnäytetyönlaatijan osaamista ohjelmistotuotannon kaikilla osa-alueilla, kuten asiakasyhtyössä, arkkitehtuuri- ja käyttöliittymäsuunnittelussa, riskianalyysissa, vaatimusmäärittelyssä, tietokantojen hallinnassa ja ohjelmoinnissa.

Usein ohjelmistotuotantoa käsittelevät teokset määrittelevät prosessinsa keskisuurille tai suurille tuotantoryhmille palvellen ohjelmistoteollisuuden tarpeita. Yksin ohjelmoivat tarveohjelmoijat tai pienet kahden kolmen hengen nyrkkipajat eivät välttämättä hyödy suuremmille ryhmille suunnitelluista työmenetelmistä millään tavoin. Tämän vuoksi projektin toisena tavoitteena oli antaa näkökulma ohjelmistojen pientuotantoon ja muodostaa kuva siitä kuinka tutkitut menetelmät soveltuivat pientuotannon tarpeisiin. Case-osion yhteenvedossa peilataan valittujen työmenetelmien sopivuutta pientuotantoon ja arvioidaan miten työmenetelmiä tulisi muuttaa, jotta ne olisivat sopivampia pienemmän mittakaavan ohjelmistotuotantoon.

5.1 Prosessimalli ja aikataulu

Projektissa tarinapohjainen lyhyisiin iteraatioihin perustuva ohjelmistokehitys liitettiin prototyypitykseen ja nopeaan julkaisukiertoon. Tällä tavoin projektista saatiin nopeita ja konkreettisia tuloksia. Tällöin kommunikaatio ja suunnittelu asiakasryhmän kanssa voitiin käydä asiakkaan ymmärtämillä termeillä ja konkreettisilla käytännön esimerkeillä ilman teknistä kieltä tai abstrakteja esimerkkejä. Ketterän hyväksyntätestauslähtöisen mallin ulkopuoliset menetelmät valittiin siten, että ne tukivat mahdollisimman hyvin tuotettavan ohjelmiston tuotantoa ja lopputuleman laatua. Yhtenä valintakriteerinä

kutakin menetelmää valittaessa käytettiin opinnäytetyön laatijan aikaisempaa kokemusta. Tällöin opinnäytetyön laatija pystyi keskittymään opinnäytetyölle keskeisten uusien asioiden oppimiseen.

Projektin ensimmäisen vaiheen takarajaksi valittiin 10.9.2012, jonka jälkeen ohjelmistokehitystä ei enää suoritettu opinnäytetyön puitteissa. Case-osio peilaa projektin saavutuksia ja ohjelmiston tilaa tuona ajankohtana. Johtopäätökset ja pohdinta muodostavat tästä poikkeuksen, koska 10.9.2012 jälkeinen tutkimus tuotti uusia näkökulmia projektin arvioimiseen.

5.2 Roolitus ja käyttäjäpersoonat

Ensimmäinen suuri ongelmakohta oli roolitus. Projektin kaltaisessa pientuotannossa ohjelmiston toteuttajat joutuvat omaksumaan enemmän kuin yhden roolin. Opinnäytetyön tapauksessa projektin toteutti tasan yksi henkilö, joten roolien rajauksia oli syytä miettiä tarkasti. Projektin tapauksessa projektin toteuttaja oli myös ongelma-alueen (problem domain) paras asiantuntija sekä teknisesti että harrastusluontoisesti. Sen vuoksi päädyttiin seuraavaan ratkaisuun. Opinnäytetyön laatija toimi sekä ohjelmistokehittäjänä, testaajana ja läsnä olevana asiakkaana, jota varsinainen asiakas eli Frisbeegolffarit-ryhmän jäsenet avustivat vaatimusmäärittelyssä ja hyväksyntätestien laatimisessa koko projektin ajan jatkuvana toimena. Frisbeegolffarit-ryhmän kirjuri toimi tarvittaessa avustavana läsnäolevana asiakkaana.

Käyttäjäpersooniksi valittiin peruskäyttäjä, joka toimii samalla pääkäyttäjänä. Toissijaiseksi käyttäjäksi valikoitui tietokantanta (liite 4.). Ratkaisuun päädyttiin, koska ohjelmisto on tarkoitettu kunkin käyttäjän henkilökohtaiseen käyttöön. Tästä syystä projektin toteuttaja ja sen asiakasryhmä ei nähnyt syitä rajoittaa peruskäyttäjien käyttöoikeuksia. Viite-eheyksien säilyttämiseksi peruskäyttäjien oikeuksia muokata perustietoa kuten rata- tai käyttäjäprofiileja ei rajoitettu, jolloin jokainen peruskäyttäjä on vastuussa omista toimistaan. Paikallisena ohjelmistona tietokannan eheys ei aiheuttanut suunnittelutoimenpiteitä rinnakkaisuuden vuoksi.

Tietokannan käyttäjäroolitukseen valittiin Elmasrin ja Navathen määrittelemä kolmijako tietokannan hallinnoitsijaan, suunnittelijaan ja loppukäyttäjiin (Elmasri & Navatha 2000, 12-13). Projektin tapauksessa projektin toteuttaja toimi kaikissa kolmessa roolissa ja asiakasryhmän jäsenille annettiin hallinnoitsijan ja loppukäyttäjän roolit sillä poikkeuksella, että loppukäyttäjällä ei ole oikeutta tehdä uusia ohjelmallisia hakuja, koska tuotettava ohjelmisto ei tue tätä ominaisuutta.

5.3 Työkalut

Ohjelmointityökaluna käytettiin Microsoftin Visual Basic 2010 Expressiä ja Access 2010:iä. Visual Basic Expressin ja Access 2010 välisen kommunikoinnin moottorina käytettiin Microsoft Access Database Engine 2010 - tietokantamoottoria. Dokumentoinnin kirjaamiseen käytettiin Microsoft Officen Professional editionin tarjoamia työkaluja. Ohjelmointilaitteistona käytettiin vuonna 2010 hankittua PC-laitteistoa ja rinnakkaislaitteistona testausta varten vuonna 2008 hankittua kannettavaa.

Presentaatiolaitteistona käytettiin vanhahkoa 17’ näytöllä varustettua kannettavaa tietokonetta, johon oli asennettu Windows Vista. Lisäksi ohjelmiston uusinta versiota testattiin Frisbeegolffarit-ryhmän kirjurin omalla kotikokoonpanolla, joka koostuu uudehkosta PC-laitteistosta. Kuvamateriaali tuotettiin digikameralla ja tiedostonsiirto toteutettiin kahdella 2Gb:n muistitikulla.

5.4 Iteraation pituus ja nopeus

Projektin iteraation pituudeksi valittiin kolme päivää ja iteraation nopeudeksi kaksi. Nopeaan ja kevyeen iteraatioon päädyttiin siksi, että toteutettava ohjelmisto on suhteelisen yksinkertainen ja sen toteutuksesta vastasi vain yksi henkilö, jolloin päällekkäistä tarinan kehitystä oli vaikea suorittaa. Lukuun kaksi päädyttiin myös siksi, että sillä tavoin voitiin testata projektin nopeuden itsesäätelymekanismeja.

Käytännössä iteraation nopeus vaihteli nollasta viiteen riippuen mahdollisista ongelmista. Yksin ohjelmoitaessa riski omille ajatusvirheille ja sokeus tietotaulujen indeksointivirheille aiheutti ylimääräisiä ongelmia. Tässä suhteessa ryhmän antamaa vertaistukea jäätiin kaipaamaan. Lisäksi iteraation nopeuden säätelyyn ja sen tasaisena

säilyttämiseen vaikutti suuresti ohjelmiston toteuttajan projektin ulkopuoliset aktiviteetit. Aina ei ollut mahdollista käyttää riittävää määrää aikaresursseja, jotta iteraation tavoite olisi saavutettu.

Lopulta päädyttiin suunnittelutapaan, jossa nopeuden sijasta valittiin toteutettava tarina, joka pyrittiin toteuttamaan työpäivän aikana. Tämä paransi työmotivaatioita ja vähensi aikataulutuksen aiheuttamaa kiirettä. Kiireen väheneminen ja työmotivaation kasvu vaikuttivat positiivisesti koodin laatuun ja virheiden määrään. Projektin loppupuolella ei ilmennyt yhtään vakavaa ohjelmointivirheittä.

5.5 Vaatimusmäärittely

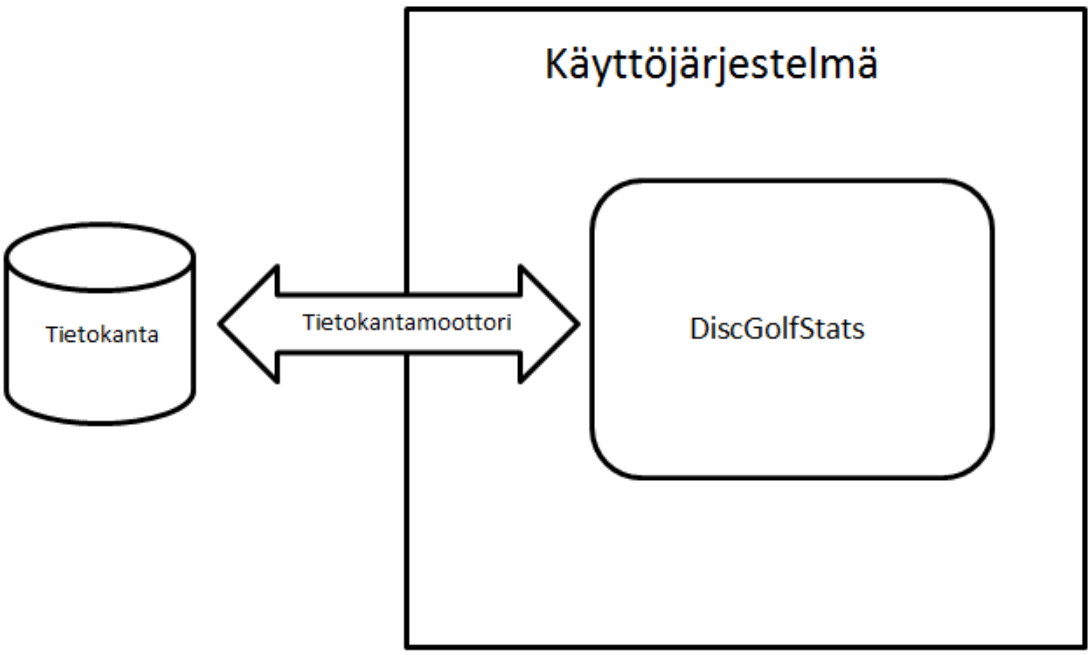
Vaatimusmäärittely suoritettiin opinnäytetyön otsikon mukaisesti ketterän hyväksyntätestauskeskeisen ohjelmistotuotannon periaatteiden mukaan. Vaatimusmäärittelyn suorittamiseen käytettiin seuraavia työtapoja. Vaatimusmäärittely aloitettiin aivoriihellä, jossa sovittiin ohjelmiston perustoiminnallisuudesta. Tämän jälkeen vaatimusmäärittelyä suoritettiin jatkuvasti projektin kuluessa, joko epävirallisissa keskusteluissa, ryhmäkeskusteluissa, välikatsauksissa ja uuden prototyypin julkaisun yhteydessä. Tekniset vaatimukset määräytyivät oletettujen käyttäjien omistamien laitteistojen mukaan.

Käytännössä uusien ominaisuuksien lisäämis-, pudotus-, ja poistamispäätöksiin vaikutti ryhmän jäsenten prioriteetit ja lajikohtaiset säännöt. Esimerkiksi toteutettavaksi valittu heittojen pituuksien kirjaaminen toteutettiin osittain julkaisuversioon, mutta myöhemmin kävi ilmi, että frisbeegolfin virallisten sääntöjen vuoksi heittojen tarkka mittaaminen on turnausolosuhteissa kiellettyä. Tästä syystä kyseinen ominaisuus löytyy ohjelmistosta käyttäjälle näkymättömänä.

5.6 Tekniset vaatimukset ja arkkitehtuuri

Teknisesti ohjelmiston täytyi toimia Windows 7 käyttöjärjestelmällä varustetussa PC-järjestelmässä, johon on asennettu Microsoft Access Database Engine 2010-tietokantamoottori. Windows rajoitukseen päädyttiin, koska tietokanta ja tietokantamoottori pyöri Microsoftin tarjoamilla ratkaisuilla. Lisäksi ohjelmiston

käyttäjäryhmästä vain yksi henkilö käytti jotain muuta kuin Windows-pohjaista käyttöjärjestelmä kokoonpanoa.



Kuvio 9. Ohjelmiston arkkitehtuurikuvaus

Ohjelmiston arkkitehtuuri pyrittiin pitämään mahdollisimman yksinkertaisena. Arkkitehtuuri koostuu kahdesta Windows-käyttöjärjestelmän alla pyörivästä osasta ja Kiintolevylle talletettuna olevasta tietokannasta. Ensimmäisen käyttöjärjestelmän alaisen osan muodostaa varsinainen DiscGolfStats-ohjelmisto. Toisen osan muodostaa tietokannan ja ohjelmiston kommunikointiin käytettävä 32 bittinen Microsoft Access DataBase Engine (kuvio 9.).

Ohjelmisto itse on sisäisesti persistenssi. Käynnistettäessä ohjelmisto lataa tiedot käyttäjän saataville ja tallettaa tarvittaessa tehdyt muutokset tietokantaan. Ohjelmiston lähtötila riippuu siis tietokannan sisällöstä käynnistyshetkellä ja lopputila käyttäjän toimista ohjelmiston käynnissä ollessa. Sisäisesti persistenssiin ratkaisuun päädyttiin, koska projektissa haluttiin kokeilla Access-tietokannan käsittelyä Visual Basic 2010 Expressin DataGridView-komponenteilla.

5.7 Projektin visio, tehtävä, tavoitteet ja periaatteet

Projektin visiona oli toteuttaa kevyt ja helppokäyttöinen tilastointityökalu Frisbeegolffaajat ryhmälle. Visio toteutui toteuttamalla seuraavat tehtävät ja tavoitteet. Ohjelmiston täytyi kyetä tallettamaan, muokkaamaan ja vertailemaan sekä tuloksia että käyttäjäprofiileja Visual Basic 2010 Expressillä toteutetulla ohjelmistolla (taulukko 11.). Tuotannon periaatteena käytettiin minimalismia ja yksinkertaisuutta, koska opinnäytetyön puitteissa toteutettavassa projektin ensimmäisessä vaiheessa suurimpana ongelmana ovat uudet ja vieraat työtavat. Tästä syystä oli syytä välttää ylimääräistä monimutkaisuutta, joka saattaisi hukuttaa alleen opinnäytetyön toisen tavoitteen toimia testinä ketterän hyväksyntätestauslähtöisen ohjelmistokehityksen skaalautuvuudelle.

Taulukko 11. Projektin toimeksianto

Toimeksiannon visio, tehtävä, tavoite ja periaatteet
Visio Luoda frisbeegolf-tilastointiohjelmisto, jolla tuloksia voi tallentaa, ladata ja vertailla.
Tehtävä Luodaan tilastointiohjelmisto käyttämällä Visual Studio Express Edition 2010 työkaluja.
Tavoitteet Käyttäjän ei tarvitse enää turvautua paperiseen kirjanpitoon. Ohjelmisto tarjoaa vertailupalveluja, joita ei ole voitu toteuttaa paperikirjanpidolla. Projekti toimii opinnäytetyön produktina.
Periaatteet Ohjelmiston kohderyhmän tarpeet ovat etusijalla. Ohjelmiston tuottaminen palvelee opinnäytetyön suorittamista. Monimutkaisuutta yritetään välttää ja ratkaisut pyritään pitämään yksinkertaisina.

5.8 Ominaisuudet

Aivoriihi järjestettiin lajille ominaisesti frisbeegolf-radalla vapaamuotoisina keskusteluina ja myöhemmin lyhyehkönä sähköpostikeskusteluna, jossa ominaisuudet

kerrattiin lyhykäisyydessään. Keskustelujen perusteella ohjelmistolta vaadittiin seuraavat ominaisuudet. Käyttäjän tulee voida muokata, tallettaa ja lisätä tuloksia. Käyttäjän tulee voida tarkastella omaa tuloskehitystään ja verrata sitä muiden käyttäjien tuloksiin.

Ohjelmiston tuli toimia PC-laitteistossa, jonka käyttöjärjestelmäksi on asennettu Windows7-käyttöjärjestelmä. Projektin kuluessa ominaisuuksia lisättiin asiakasryhmän toiveitten mukaan, joko vapaamuotoisten keskustelujen tai prototyyppiesittelyjen jälkeen. Projektin toisessa vaiheesssa ohjelmisto tullaan siirtämään älypuhelin ympäristöön.

5.9 Riskianalyysi

Riskianalyysin suoritustavaksi valittiin perinteinen riskianalyysi. Sommervillen ehdottamaa riskien jakamista kolmeen perusryhmään, projektiriskeihin, teknisiin riskeihin ja bisnesriskeihin harkittiin (Sommerville 2004, 104-105), mutta suoritettavan projektin luonteesta johtuen jälkimmäinen riskiryhmä ei sopinut projektin kuvaan. Tästä syystä päädyttiin jakamaan projektin riskit seuraaviin pääryhmiin. Riskianalyysissä käytetyt pääriskiryhmät olivat tekniset riskit, aikatauluriskit, henkilöstöriskit ja prosessiriskit. Nimikkeeseen prosessiriski päädyttiin, koska ohjelmiston tuotannossa käytetty ohjelmistoprosessimalli oli toteuttajalle uusi. Riskianalyysissä ja riskien hallinnassa pyrittiin noudattamaan Pressmanin ja Incen esittämiä (Pressman & Ince 2000, 145-163) periaatteita.

Tämän jälkeen riskit jaettiin geneerisiin ja ohjelmistokohtaisiin riskeihin. Geneeriset riskit pätevät kaikkiin ohjelmistotuotantoihin ja ohjelmistokohtaiset riskit liittyvät vain toteutettavaan ohjelmistoon. Esimerkkinä geneerisestä riskistä voidaan antaa pääohjelmoijan sairastuminen ja siitä johtuva aikataulun myöhästyminen. Ohjelmistokohtaisesta riskistä voidaan antaa esimerkkinä frisbeegolfin sääntömuutokset, jotka saattavat vaikuttaa ohjelmiston validiuteen.

Riskin kokonaisvaikutusta arvioitiin perinteisellä kaksitekijäisellä tunnuksella.

$$K_r = T_r \times V_r$$

K_r = riskin kokonaisvaikutus, T_r = riskin todennäköisyys, V_r = riskin vakavuus

Jossa riskin kokonaisvaikutus K_r määräytyy liitteessä 3. olevan vakavuusmatriisin mukaan.

Erityistä huomiota kiinnitettiin riskeihin joiden kokonaisvaikutus on keskinkertainen tai suurempi tai todennäköisyys todennäköinen tai vakavuus katastrofaalinen (liite 3.). Tällä tavoin pystyttiin kiertämään pelkän riskimatriisin käytön aiheuttamat ongelmat ääritapausten käsittelyssä. Riskit koottiin omiin riskikortteihinsa, joihin merkittiin riskin tunnusten lisäksi sen toteutumisen seuraukset, toimenpiteet riskin toteutuessa sekä ehkäisy- ja seurantamenetelmät (liite 5.). Projektissa pyrittiin proaktiiviseen riskihallintaan, koska oli huomattavasti vähemmän työlästä estää riskejä toteutumasta kuin reaktiivisesti korjata riskien toteutumisesta aiheutuneita vahinkoja.

5.10 Tarinat

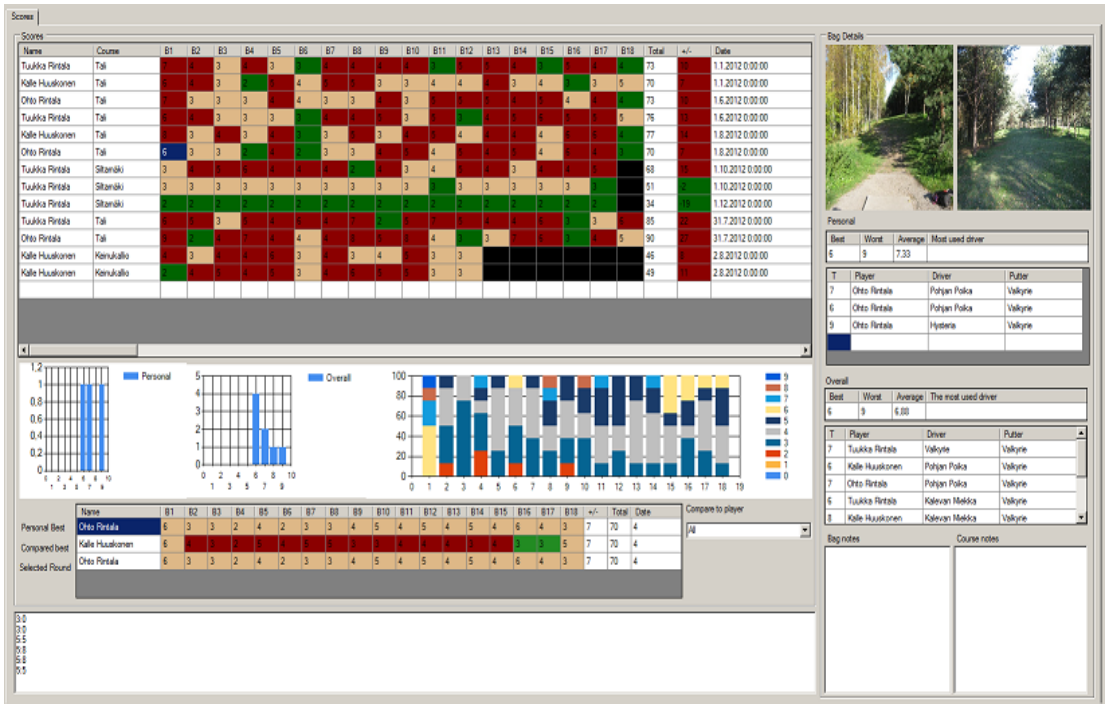
Tarinoiden kirjaamiseen projektin kehitysjonona toimivaan projektipäiväkirjaan käytettiin aluksi kevyttä Leffingwellin sanallista muotoa. Hyväksyntätestien kirjaamisen peruskortiksi valittiin karsittu version Leffingwellin kortista, joka oli alkujaan selkeästi suunniteltu suuremmille projekteille. Kortista karsittiin turha informaatio ja projektille olennaiset osat jätettiin kortille (taulukko 12.). Tarinoiden laatimisessa keskityttiin olennaisiin ja ohjelmistolle kriittisiin tarinoihin, jotta vältyttäisiin ylisuunnittelulta ja sen aiheuttamalta ylidokumentoinnilta sekä turhalta rutiinien toistolta (liite 6.).

Taulukko 12. Tarinan hyväksyntätestauskortti

Tarina	
ID : Tuloksen syöttö	Kuvaus : Käyttäjä syöttää valittuun rataprofiiliin kierroksen tulokset kori korilta kunnes kaikki korit on merkitty. Tämän jälkeen käyttäjä tallentaa tuloksen tietokantaan.
Hyväksymiskriteerit	
1. Käyttäjä voi syöttää tuloksen tietokantaan. 2. Varmistetaan, että tulos noudattaa rataprofiilia.	
Moduulit joihin vaikutetaan	
Syottosivu.html	Luodaan tarvittavat syöttökontrollit.
Tulokset.mdb	Tietokantaan lisätään yksi tulos lisää.
Dokumentit	
Käyttöohje	Lisätään käyttöohjeeseen ohjeet tuloksien tallentamiseen.
Testitapaukset	
Manuaalinen	Tarkista sopivatko tulokset rataprofiilin raja-arvoihin
Automaattinen	Lisätään syöttökontrolleihin validaattorit tietotyyppien tarkastusta varten .

5.11 Graafinen käyttöliittymä

Graafisen käyttöliittymän toteuttamiseen käytettiin Visual Basic Express Edition 2010:n perustyökaluja. Käyttöliittymän suunnittelussa käytettiin overview beside detail – mallia (OBD), jolla pyrittiin vähentämään sukeltavaa navigointia ja siten virtaviivaistaa käyttökokemusta. OBD-malli valittiin siksi, että se sopi hyvin tilastointityökalun käyttöliittymän toteuttamiseen, koska tilastotietoa ei haluttu piilottaa valikkojen tai sukeltavan navigaation taakse. Kaikki oleellinen tieto haluttiin suoraan näkyviin näyttöruudulle (kuvio 10.).



Kuvio 10. Toteutettavan ohjelmiston käyttöliittymä prototyyppivaiheessa

Käyttöliittymän keskeinen elementti on vasemmassa yläkulmassa sijaitseva tulostaulukko. Tulosten syöttö, vertailu ja muokkaus tapahtuu tulostaulukon avulla käyttöliittymän muita komponentteja hyväksikäyttäen. Suoraan tulostaulukon alapuolella sijaitsevat valitun radan ja korin graafiset heittojakaumakuvaajat. Rata- ja korikohtaiset vertailuelementit sijaitsevat heittojakaumakuvaajien alapuolella. Oikealla sijaitsevat korikohtainen tilastotieto sekä tulostaulukosta valitun korin esittelytiedot.

Käyttöliittymässä käytettiin hyväksi värejä punainen, puunruskea ja vihreä siten, että kori- ja ratatuloksia ilmaistaessa punainen merkitsee tulosta yli ihannetuloksen, puunruskea ihannetulosta ja vihreä ihannetuloksen alittavaa tulosta. Musta ilmaisee nollatulosta, joka tarkoittaa pelaamatonta koria. Vertailuelementti käyttää samoja värejä vertailtavien tulosten ilmaisemiseen siten, että ihannetuloksen asettaa vertailtava tulos. Ratakohtaisen heittojakauman ilmaisemiseen värejä käytettiin siten, että jokainen tulosjakaumakuvaajan väri ilmaisi eri heittomäärää. Opinnäytetyön puitteissa julkaistava versio ei käsittele keskenjääneitä kierroksia (DNF) erillisenä poikkeuksena.

5.12 Testaus

Testimateriaali valittiin siten, että osa testimateriaalista oli aitoja radalla heitettyjä tuloksia. Osa testimateriaalista luotiin satunnaisotannalla painottamalla tulosta kyseisen korin ihannetuloksella ja osa luotiin puhtaasti satunnaisesti. Testimateriaaliin lisättiin myös lähes mahdottomia, mutta frisbeegolfin sääntöjä noudattavia tuloksia. Virheellistä tulostietoa käytettiin testauksessa siten, että sitä ei syötetty varsinaiseen testimateriaaliin tulosvalidaattorien testauksen jälkeen, koska tulosvalidaattorit eivät hyväksyneet virheellistä tulostietoa tallennettavaksi. Tulosvalidaattorit toimivat projektin automatisoituina testeinä, jotka antoivat suoran hyväksytty-hylätty vasteen syötetylle tulokselle.

Ensimmäiseksi pohjamateriaali syötettiin käsin Access-tietokantaan, jonka avulla luotiin testeissä käytettävät käyttäjä-, rata-, ja koriprofiilit. Tämän jälkeen tietokantaan syötettiin testauksessa käytettävä perustulostieto. Perustulostieto toimi pohjana kaikelle tuloksiin liittyville testeille projektin jokaisen iteraation yhteydessä. Testausta helpotti Visual Basic Express Editionin ominaisuus, jossa uuden käännösversion tietokannasta luodaan väliaikainen kopio, jolloin testaus ei voinut sotkea alkuperäistä eheää tietokantaa.

Tarinoiden testaamiseen käytettiin tarinoiden hyväksyntäkriteereistä laajannettuja hyväksyntätestejä hyväksyntätestauslähtöisyyden perusteiden mukaisesti. Resurssien puutteen vuoksi automatisoiduista testeistä jouduttiin tulossvalidaattoreita lukuunottamatta luopua. Manuaaliset tarkistukset tehtiin tarinan toteuttamisen yhteydessä. Liitteissä mainitut tarinat testattiin siten, että niitä voidaan pitää valmiina (done), mutta yhtään tarinaa ei leimattu valmiiksi valmiiksi (done done), koska projekti jatkuu vielä opinnäytetyön valmistumisen jälkeen.

Ominaisuuksien hyväksyntätestaus toteutettiin niitä koskevien tarinoiden hyväksyntätestauksella. Ominaisuuksien hyväksyntätestaus toimi myös ohjelmiston integraatiotestauksena, koska ominaisuuksiin tiivistyi koko ohjelmiston toiminnallisuus ja teknisten ratkaisujen toimivuus ohjelmiston kaikilta osilta tietokannasta käyttöliittymäkomponentteihin. Integraatiotestaus suoritettiin osissa sitä mukaa kun

tarinoita ja ominaisuuksia valmistui. Ominaisuuksien ulkopuolisesta integraatiotestauksesta voidaan mainita ohjelmiston, tietokantamoottorin ja tietokannan välisen kommunikoinnin testaaminen.

Testauksessa löydetyt ongelmat pyrittiin korjaamaan välittömästi tai kirjaamaan korjaukset projektin kehitysjonoon. Testauksessa löydetyt virheet olivat pääsääntöisesti pieniä muutamien minuuttien tai korkeintaan parin tunnin korjausta vaativia virheitä. Yleisin virhetyyppi oli yksinkertainen väärän solutiedon käyttäminen funktiossa virheellisen indeksin vuoksi. Jälkikäteen ajateltuna näitten virheitten määrää olisi vähentänyt sarakenimien käyttö indeksien sijasta.

Projektissa ilmeni vain yksi suurempi ongelma, joka vaati viikon kestävän refaktoroinnin ja ohjelmistoarkkitehtuurin uudelleen suunnittelun, mutta se ei ilmennyt varsinaisen testauksen aikana. Ongelma esti tietueen lisäämisen tietokantaan Datagridview-komponentin kautta, koska tietokanta oli yhdistetty tietokanta-adapterilla suoraan Datagridview-komponenttiin. Tällöin rivin lisääminen Datagirdview-komponenttiin olisi tehnyt muutoksen myös suoraan tietokantaan, mikä ei ollut sallittua.

Ongelman ratkaisemiseksi päädyttiin käyttämään käyttäjälle näkymättömiä välitystauluja, joista tieto kopioitiin visualisointitaulukkoon ja mahdolliset muutokset kopioitiin tietokantaan poimimalla ne visualisointitaulukosta erillisten tallennuskäskyjen avulla. Arkkitehtuuria ratkaisu muutti siten, että tietokannan ja ohjelmiston suora yhteys katkaistiin ja kumpikin toimii itsenäisenä osana järjestelmää. Jälkikäteen arvioiden tämä oli turvallisempi ratkaisu ja noudatti paremmin hyvää ohjelmointitapaa, jolloin järjestelmän eri osat eivät ole toisistaan suoraan riippuvaisia. Alkuperäistä ratkaisua voidaan pitää suunnitteluvirheenä.

Suuria ohjelmointivirheitä oli kaksi kappaletta, joista toinen liittyi vaikeasti havaittavaan indeksointivirheeseen ja toinen periytyvään virheelliseen palautusarvoon. Indeksointivirhe aiheutti tilastotiedon graafisen esityksen ja todellisen tilan poikkeamisen toisistaan. Virheestä huolimatta tallennus toimi oikein, jolloin virheen alkusyytä oli vaikeampi löytää. Virheellinen palautusarvo koski tietotaulun tietueen

uniikin tunnuksen palauttamista virheellisesti taulukon rivinumerona. Muut virheet eivät olleet ohjelmoinnillisesti kiinnostavia.

6 Yhteenveto

Opinnäytetyön tietoperustalle asetettu tavoite esitellä yksi tapa käyttää hyväksyntätestauslähtöistä ohjelmistokehitystä saavutettiin. Esitelty malli käyttää hyväkseen ketterää ajattelua tarjoamalla ohjelmistokehittäjille joukon muokattavia, valinnaisia ja skaalattavia työtapoja, joista ohjelmistokehittäjät voivat valita kulloiseenkin projektiinsa sopivimmat työkalut. Malli pyrkii toimivan ohjelmiston tuottamiseen sitomalla asiakkaan ohjelmiston tuotantoprosessiin. Sen sijaan että asiakas olisi vain projektin alku ja loppuvaiheessa vaikuttava toimija, asiakas toimii aktiivisesti ohjelmistokehittäjien kanssa koko projektin ajan.

Mallin hyväksyntätestaus ei tarkoita vain asiakkaan kanssa tehtäviä perinteisiä hyväksyntätestejä, vaan on prosessi, jossa asiakkaan johdolla laaditaan ennen varsinaista toteutusta testattavat vaatimukset toteutettavalle toiminnallisuudelle. Hyväksyntätestejä laadittaessa yhteistyö asiakkaan kanssa on elintärkeää, koska asiakas on oman osaamisalueensa paras asiantuntija. Tällä tavoin voidaan varmistaa, että ohjelmisto toteuttaa toiminallisuuden asiakkaan haluamalla tavalla.

Teknisenä prosessina malli jakaa ohjelmiston toteutuksen askeleittain tarkentuviin testattaviin vaatimuksiin. Ylimmällä tasolla ovat projektin hyväksyntätestit, joiden alapuolella sijaitsevat ominaisuuksien, tarinoiden ja käyttötapausten hyväksyntätestit. On tärkeää huomioida, ettei hyväksyntälähtöinen testaus syrjäytä perinteistä yksikkö- tai integraatiotestausta vaan kaikkia toimia suoritetaan lomittain. Hyväksyntätestit pyritään automatisoimaan mahdollisuuksien mukaan työmäärän vähentämiseksi, jolloin projektissa voidaan keskittyä uuden tuottamiseen työlään regressiotestauksen sijasta.

Esitelty malli käyttää hyväkseen monikerroksista rekursiivista lähestymistapaa. Prosessin rekursiivisuus tulee ilmi jokaisella prosessin tasolla. Kaikkien tasojen hyväksyntäkriteerejä, hyväksyntätestejä, toiminnallisuutta ja koodia voidaan arvioida, refaktoroida ja uudelleen toteuttaa missä tahansa projektin vaiheessa niin monta kertaa kuin on tarpeellista suotuisan lopputuloksen saavuttamiseksi.

6.1 Projektin yhteenveto

Projekti sujui pääosin suunnitellulla tavalla ja sovitut ominaisuudet valmistuivat beta-testaukseen ennen annettua päivämäärää 10.9.2012. Suurin poikkeama projektin suunnitellusta etenemisestä johtui virheellisestä ohjelmistoarkkitehtuurista. Suunniteltu arkkitehtuuri ei toiminut yhteen Visual Basic Express 2010 Datagridview-taulujen kanssa ja tämä aiheutti viikon kestäneen refaktorointi- ja korjauskierroksen, jossa koko ohjelmisto kirjoitettiin käytännössä uudestaan niiltä osin kuin entistä koodia ei voitu uudelleenkäyttää. Lisäksi beta-testaus paljasti teknisiä ongelmia, jotka korjataan myöhempänä ajankohtana. Projektin kulku vaihe vaiheelta on kuvattu projektipäiväkirjassa (liite 2.).

Asiakasyhteistyö ja kommunikointi sujui hyvin. Kommunikointia helpotti asianomaisten hyvä ymmärtämys aihealueesta ja samansuuntaiset tavoitteet ja tarpeet. Ainoat hankaluudet olivat eri henkilöiden eriävät mielipiteet projektin ominaisuuksien priorisoinnissa. Priorisointiongelma tuli esille käsiteltäessä heittojen pituuksien mittaamista ja niiden merkitsemistä tietokantaan. Ominaisuuden priorisoinnista päästiin kuitenkin yksimielisyyteen selkeiden perusteluiden esittämisen jälkeen ja ominaisuus pudotettiin ensimmäisen vaiheen kehitysjonosta. Helpoiten kehityskeskustelut sujuivat kutsumalla asianosaiset henkilöt golf-kierrokselle.

Aikataulut ja projektisuunnitelman noudattaminen sujui tyydyttävästi. Projektiryhmän vertaispaineen puuttuessa pienissä asioissa kuten työjärjestyksessä tai asioiden ylöskirjoittamisessa oli välillä lipsuntaa. Lisäksi työtaakka jakautui epätasaisesti työpäivien kesken, jolloin työpäivät vaihtelivat 2 tunnin ja 14 tunnin välillä. Äärimmäisen pitkiin työpäiviin syynä oli pääsääntöisesti työhön uppoutuminen, mikä mahdollisti tarvittaessa rauhallisemman työtahdin myöhemmin. Iteraation nopeuden säätelyssä päädyttiin lopulta ratkaisuun, jossa valittiin yksi tarina, joka toteutettiin työpäivän aikana. Ratkaisu tasoitti työpäiville jakautuvaa työtaakkaa ja siten paransi työmotivaatiota ja rytmiä.

Ohjelmistosta tehtiin kolme eri prototyyppiversioita. Ensimmäinen prototyyppi oli käyttöliittymä-, tekniikka- ja perustoiminnallisuusdemo, jonka tarkoituksena oli

tutustuttaa projektin toteuttaja uusiin työvälineisiin ja toimia nopeana näyttönä projektin asiakasryhmälle projektin alussa. Aikatauluvaikeuksien vuoksi ensimmäinen esitelty prototyyppi oli toisen prototyypin aikainen versio, jonka pohjalta ohjelmistoa alettiin kehittää eteenpäin. Arkkitehtuurisuunnitteluvirheen vuoksi rakennettiin kolmas prototyyppi, jonka viimeisimmästä versiosta tuli projektin ensimmäisen vaiheen päättävä ja beta-testattava julkaisuversio.

Projektin ensimmäisen vaiheen lopputulemaan voi olla tyytyväinen, koska opinnäytetyölle ja projektille asetut tavoitteet saavutettiin. Ohjelmistoa voi käyttää niillä ominaisuuksilla, jotka luvattiin toteuttaa ensimmäisessä vaiheessa. Ohjelmistossa on vielä teknisiä ja kosmeettisia puutteita, jotka tullaan korjaamaan projektin toisessa vaiheessa. Opinnäytetyön osalta projekti oli antoisa kokemus uudesta tavasta tuottaa ohjelmisto. Käytännössä projektin työtavat olivat todella lähellä sitä tapaa tuottaa ohjelmistoja mitä opinnäytetyön laatija on intuitiivisesti aikaisemmin käyttänyt.

6.2 Pohdinta ja jatkotutkimuskohteet

Ketterä hyväksyntätestauslähtöinen lähestymistapa tuotetun ohjelmiston kaltaiseen projektiin oli onnistunut valinta. Menetelmät skaalautuivat hyvin alaspäin ja prosessiteknisiä ongelmia ei ilmennyt. Suurimmaksi ongelmaksi koettiin ohjelmistokehittäjän vertaistuen ja roolijaon puute. Vertaistuen puute näkyi eritoten turhina ohjelmointivirheinä, jotka eivät johtuneet virheellisestä suunnittelusta tai huonosti laadituista hyväksyntätesteista vaan sokeudesta omalle koodille. Tämä aiheutti täysin turhaa työtä ja hukattua aikaa, joka olisi voitu käyttää uusien ominaisuuksien kehittämiseen.

Perusvirheiden välttämiseksi arkkitehtuuria ja koodia olisi voinut tarkastella kokonaan projektin ulkopuolinen henkilö, joka olisi ollut helppo integroida projektimalliin ja henkilöstöön ulkopuolisena konsulttina. Roolijaon puute näkyi ohjelmistokehittäjän työmäärän kuormittumisena, jota voitiin helpottaa mallin tarjoamin projektihallinnan keinoin. Kokonaisuudessaan malli tuotti ominaisuuksiltaan ja toiminnoiltaan vaatimukset täyttävän ohjelmiston, jonka kehitys oli altis arkkitehtuuri- ja

ohjelmointitason virheille. Lopputulema oli odotettu, koska malli keskittyi vahvasti toiminnallisuuden määrittelyyn ja toteuttamiseen teknisen suunnittelun sijasta.

Suurien tai kriittisten järjestelmien tuottamiseen ketterää hyväksyntätestauslähtöistä ohjelmistokehitystä ei voida suositella. Malli ei kiinnitä huomiota tarpeeksi monimutkaisten järjestelmien suunnittelulle ominaisiin tarkkuutta ja tiukkaa dokumentaatiota vaativiin toimiin. Mallia ei voida myöskään suositella monimutkaisten integroitujen järjestelmien tuottamiseen, koska mallin mukainen jatkuva refaktoroituva hyväksyntätestaus voi olla vaikeaa tai kallista integroidun järjestelmän perusluonteen ja testausympäristörajoitteiden vuoksi.

Mallin esittämä iteraation nopeuden säätely ei myöskään sovellu sellaisenaan opinnäytetyön kaltaisen projektin tuotantoon, koska yksikin ratkaisematon ongelma tai väärin arvioitu työtaakka heitteli nopeutta puolelta toiselle, eikä nopeuden soveltuvuudesta voitu tehdä järkisyin pitäviä päätelmiä. Paremmaksi tavaksi ylläpitää projektin etenemistä koettiin toimintatapa, jossa jokaiselle päivälle valittiin yksi toteutettava tarina ja yli jäävä aika käytettiin projektin muiden toimintojen suorittamiseen. Käytännössä tämä tarkoitti sitä, että tarinoista pyrittiin pilkkomalla muokkaamaan yhden työpäivän mittaisia kokonaisuuksia, jolloin projektin nopeudeksi muodostui yksi optimaalinen työpäivä.

Perusluenteeltaan ketterä hyväksyntätestauslähtöinen ohjelmistokehitys oli intuitiivinen tapa tuottaa ohjelmistoja. Ohjelmiston toimintojen jaottelu ominaisuuksiin ja niistä johdettuihin tarinoihin ja käyttötapauksiin oli luonnollinen tapa pilkkoa kehitettevä toiminnallisuus pienempiin helpommin toteutettaviin palasiin. Opinnäytetyön laatijan aikaisemmin käyttämä tapa tuottaa ohjelmistoja muistutti suuresti ketterää hyväksyntätestauslähtöistä ohjelmistokehitystä, sillä poikkeuksella että asiakasyhteistyötä ei oltu painotettu yhtä paljon ja jaottelua ominaisuuksiin, tarinoihin ja käyttötapauksiin ei oltu formalisoitu.

Testauksen automatisointi vähin henkilöstö- ja ohjelmistoresurssein kaipaisi jatkotutkimusta. Opinnäytetyön puitteissa tulosvalidaattoreita lukuunottamatta automatisoitua testausta ei päästy suorittamaan. Automatisoitua regressiotestausta ei

suoritettu lainkaan, koska validaattorit olivat itsenäisiä toimijoita, joiden toimintaan muun ohjelmiston muutokset eivät vaikuttaneet. Mahdollisia kohteita syventävälle tutkimukselle voisi olla FIT tai FITnesse viitekehyksien käyttö pienprojekteissa tai muiden automatisoitua testausta pienprojekteissa tukevien menetelmien tutkiminen.

Lähdeluettelo

Leffingwell, D. 2011. Agile Software Requirements – Lean Requirements Practices for Teams, Programs, and the Enterprise. Pearson Education Inc. Boston.

Pressman, R S, Ince, D. 2000. Software Engineering A Practitioner's Adaptation 5th Edition (European Adaptation). McGraw-Hill Publishing Company. Berkshire.

Sommerville, I. 2004. Software Engineering. 7th edition. Pearson Education Limited. Harlow.

Shore, J. , Warden, S. 2007. The Art of Agile Development. Theory in Practice. O'Reilly. Sebastobol.

Cohn, M. 2010. Succeeding with Agile, Software Development Using Scrum. 5.painos. The Addison-Wesley Signature Series. Pearson Education, Inc. Boston.

Cohn, M. 2004. User Stories Applied: For Agile Software Development
http://www.google.fi/books?hl=fi&lr=&id=SvIwuX4SVigC&oi=fnd&pg=PR15&dq=test+driv-
en+development+epics&ots=VoVg77vVMS&sig=aeFbZztVuorxlGIlJKiGuseoW40&redir_esc=y#v=onepage&q&f=false
Luettu 1.9.2012

Pugh, K. 2010. Lean-Agile Acceptance Test-Driven Development : Better Software Through Collaboration. Pearson Education Inc. Boston.

Koskela, L. 2007. Methods and Tools, volume 2008 issue 2, Acceptance TDD explained.
http://www.methodsandtools.com/PDF/mt200802.pdf
Luettu : 5.5.2012

Koskela, L. 2008. Test Driven – Practical TDD and Acceptance TDD for Java Developers. Manning Publications Co. Greenwich.

Schwaber, K., Sutherland, J., 2012. The Scrum Guide.
http://scrumwell.files.wordpress.com/2012/01/scrum-guide-2011-fi-v1.pdf
Luettu : 5.5.2012

Haumer, P. 2004. UC-Based Software Development.
http://www.haumer.net/paper/UC-Based_Software_Development.pdf
Luettu : 15.9.2012

Agilemanifesto 2001. www.agilemanifesto.org
Luettu : 1.5.2012

Drego, V. 2010. Making Agile Customer-Centric.
http://www.siteworx.com/~/media/White-papers/making_agile_customer-centric
Luettu : 10.9.2012

Elmasri, R_Navathe, B. 2000. Fundamentals of Database Systems. 3rd Edition, International Edition. Addison-Wesley. New York.

Watkins, J. 2009. Agile Testing – How to Succeed in an Extreme Testing Environment. Cambridge University Press. New York.

Liitteet

Liite 1. käsiteluettelo

EHTOTAULUKKO on variaatio tietotaulusta, jossa kuvatataan hakuehdot täyttävät tietotaulun rivit.
HYVÄKSYNTÄKRITEERI (acceptance criteria) Kuvaavat puhekielellä projektilta, ominaisuudelta tai tarinalta vaaditut toiminnallisuudet ja rajoitteet.
HYVÄKSYNTÄTESTI (acceptance test) Hyväksyntätestauslähtöisessä merkityksessä testin muotoon laadittu mitattava vaatimus, jota voidaan soveltaa projektin tavoitteisiin, ominaisuuksiin ja tarinoihin.
INVEST-kriteerit (independent, negotiable, valuable to users or customers, estimatable, small, testable) Hyvien tarinoiden laatukriteerit.
ITERAATIO Hyväksyntätestauslähtöisen ohjelmistokehityksen ohjelmistoprosessin peruskierto.
KEHITYSJONO (project backlog) Priorisoitu lista ohjelmistoprojektin tehtävistä.
KÄYTTÄJÄROOLI (user role) Edustaa tarinan kuvauksessa toimijaa tai hyödyn saajaa.
KÄYTTÄJÄTARINA (user story) Katso kohtaa *tarina*.
KÄYTTÖTAPAUS (use-case) Kuvaus järjestelmän ja käyttäjän toiminto-vastekeskustelusta.
LASKENNALLINEN TAULUKKO (calculation table) Kuvaa testimateriaalina käytettävien syöte-vasteparien oletettuja lopputuloksia.
LÄSNÄ OLEVA ASIAKAS (on-site customer) Projektiryhmän jäsen joka edustaa asiakasta ja asiakkaan intressejä.
OMINAISUUS (feature) Ohjelmiston toiminnallisuus, joka koostuu yhdestä tai useammasta tarinasta.
OPTIMAALINEN TYÖPÄIVÄ (story-point) Yhden päivän täydellinen ja virheetön työpanos.
PERSOONA Muistisääntönä toimiva puhekielinen kuvaus käyttäjäroolista.
PROSESSIN NOPEUS (velocity) Iteraation tehtävälistaan merkittyjen tarinoiden yhteenlaskettu painoarvo.

PÄÄKULKU (main course, happy path) Käyttötapauksen ongelmaton kulku.
PÄÄKÄYTTÄJÄ Käyttäjäryhmä jonka tarpeiden pohjalta ohjelmiston käyttörajapinnat suunnitellaan.
REGRESSIOTESTAUS Testaustoiminto jolla pyritään etsimään ohjelmistovirheitä järjestelmään tehtyjen muutosten jälkeen.
ROOLI Katso kohtaa *käyttäjärooli*.
SKENAARIO Käyttötapauksen yksi mahdollinen kulku.
SKENAARIOTESTI (test scenario) Testaa käyttötapauksista laaditut skenaariot.
SMART-kriteerit (specific, measurable, achievable, relevant, time-boxed) Hyvin laadittujen tavoitteiden laatukriteerit.
SÄÄNTÖTESTI (business rule test) Testi jolla varmistetaan, että testauksen kohde noudattaa aihealueen sääntöjä.
TARINA (user story) Käyttöesimerkki jolla kuvataan ohjelmiston toimintaa käyttäjän näkökulmasta.
TARINAN PAINOARVO Arvio tarinan vaatimasta työmäärästä optimaalisissa työpäivissä.
TEHTÄVÄLISTA Lista iteraatiossa toteutettavista kehitysjonon kohdista.
TESTIN KONTEKSTI Järjestelmän tila testin alkamishetkellä.
TIETOTAULUKKO (data table) Kuvaa testimateriaalin tiedot.
TOIMINTOTAULUKKO (action table) Kuvaa testimateriaalin toiminnot askel askeleelta.
TYÖNKULKU (workflow) Tavoitteeseen tarvittavien toimintojen askeleittainen kuvaus, joka saattaa sisältää järjestelmän ulkopuolisia toimintoja.

Liite 2. projektipäiväkirja ja kehitysjono

Pvm.	Tapahtuma
10.7 – 14.7.2012	Alustava tutkimus välineistä, tavoitteista ja projektin tavoitteiden asettaminen.
15.07.2012	Ohjelmistotyökalujen hankinta ja asennus (Office Professional, Access Database Engine x86, Visual Studio 2010 Express)
16.07.2012	Peruslayoutin laatiminen
16.07.2012	Tuloskorttinäkymä
	Tarina : Käyttäjä järjestää tulokset päänäkymässä
	Ominaisuus : Tulosten väritys Tuloskortissa
17.07.2012	Tarina : Käyttäjä merkitsee tuloksen poistetuksi* Toteutetaan raakapoistona: Käyttäjä tuhoaa tulosrivin.
	Tarina : Käyttäjä lisää tuloksen päänäkymään
	Tarina : Käyttäjä muokkaa tulosta päänäkymässä
18.07.2012	Käyttöliittymän refaktorointi
19.07.2012	Tutustuminen Lappeenrannan rataan ja yleiskeskustelua
20.07.2012	Aivoriihi uusista ominaisuuksista
	Uusi ominaisuus : Käytetyn drive-kiekon lisäys tulokseen.
	Uusi ominaisuus : Korikohtaisen tuloksen tarkastelu.
	Uusi tarina : Käyttäjä lisää drive-kiekon tulokseen
	Uusi tarina : Käyttäjä tarkastelee korin tuloksia
21.07.2012	Suullinen sopimus kehitysversioiden presentaatiosta.
25.07.2012	Tarvittavien muutosten tekeminen tietokannan rakenteeseen.
	Refaktoroidun käyttöliittymän toteutus.

	Tarina: Käyttäjä tarkastelee korikohtaisia tuloksia.
	Kehitysversion toimivuuden tarkastaminen kannettavassa tietokoneessa vanhemmalla käyttäjärjestelmällä.
26.7.2012	Refaktoroidun käyttöliittymän toteutus
	Ominaisuus : Tuloskortin validointi
27.7.2012	Tarina : Avauskiekon lisäys koritulokseen -> refaktoroitu 6.8
	Refaktoroidun käyttöliittymän toteutus
29.7.2012	Ominaisuus : Tuloskortin tooltip korille -> siirretty detalji-lehdelle
	Ominaisuus : Tuloskortin tooltip radalle -> siirretty detalji-lehdelle
	Ominaisuus : Tuloskortin tooltip pelaajalle -> siirretty detalji-lehdelle
30.7.2012	Tarina : Käyttäjä tarkastaa korituloksen statistiikat tooltipillä -> siirretty detalji-lehdelle
	Tarina : Käyttäjä tarkastaa radan statistiikat tooltipillä - > siirretty detalji-lehdelle
	Tarina: Käyttäjä tarkastaa pelaajan statistiikat tooltipillä -> siirretty detalji-lehdelle
31.7.2012	Ominaisuus : Korikohtainen tuloskehitysgraafi
	Ominaisuus : Ratakohtainen tuloskehitysgraafi
1.8.2012	Prototyypin demo ja kehityskeskustelu
	Uusi ominaisuus : Draivin pituus
	Uusi ominaisuus : Putin pituus
	Uusi ominaisuus : Käytetty putteri

2.8.2012	Prosessin arviointi
	Resurssien hallinta
	Tietokannan testaus oikealla tulosdatalla
3.8.2012	Tutustuminen Nokian SM-rataan
	Tutustuminen Tampereen Vihijärven rataan
4.8.2012	Tutustuminen Tampereen Epilän rataan
	Pikainen käynti 10Tons-Entertainmentin tiloissa
	Keskustelua ohjelmistojen monetisoinnista jakopalvelujen kautta 10Tons-Entertainmentin toimitusjohtajan kanssa.
5.8.2012	Tutustuminen Julkujärven rataan
	Kehityskeskustelu ja yleistä kokemuksien vaihtoa ohjelmistoprojekteista
6.8.2012	Käyttöliittymän refaktorointi
	Tietokannan laajennus etäisyyksillä ja käytetyllä putterilla
	Ominaisuus : Yksittäiset koritulokset
	Tarina : Yksittäisen korituloksen tarkastelu
	Tarina : Käyttäjä lisää putterin
	Tarina : Käyttäjä lisää draiverin
7.8.2012	Keskustelua ominaisuuksista Tali-kierroksen lomassa
	Uusi ominaisuus : Korikohtainen graafinen heittojakauma
	Uusi ominaisuus : Korikohtainen henkilökohtainen graafinen heittojakauma
	Uusi ominaisuus : Oma verrokkisarja
	Uusi ominaisuus : Paras verrokkisarja
8.8.2012	Tarina : Käyttäjä vertaa tulosta omaan verrokkisarjansa
	Tarina : Käyttäjä vertaa tulosta parhaimpaan verrokkisarjan
9.8.2012	Graafisen jakaumakomponentin suunnittelu
10.8.2012	Tarina : Käyttäjä tarkastaa korikohtaisen heittojakauman
	Tarina : Käyttäjä tarkastaa ratakohtaisen heittojakauman

	Uusi ominaisuus : Ratakohtainen graafinen heittojakauma
	Tarina : Käyttäjä tarkastaa ratakohtaisen graafisen heittojakauman
11.8.2012	Heittokierros Siltamäessä ja projektin etenemisen raportointi sekä uusista ominaisuuksista keskustelu.
	Hole-in-one Siltamäen 9. Väylällä!
	Kehitetyt ominaisuudet katsottiin riittäviksi ja projektin ensimmäinen vaihe loppuu. Seuraava viikko leipäkoodin laatimista.
13.8.2012	Tallennusfunktioden suunnittelu
15.8.2012	Käyttöliittymän loppusilaus.
16.8.2012	Viikon tauko ja projektin reflektointi.
23.8.2012	Validaattorien testaus
	Validaattorien testaus paljasti kriittisen odottamattoman ohjelmointiteknisen ongelman ohjausobjektien hallinnassa (DataGridViewin DataBinder suorassa yhteydessä tietokantaan, jolloin rivejä ei voi lisätä ohjelmallisesti Datagridviewiin). Virheen korjaamiseen menee arviolta viikko.
24.8.2012	Korjaustyöt jatkuvat. Kehityskeskustelu ratakierroksen ohessa. Refaktorointia kehityskeskutelun päätösten mukaisesti. Heittojen pituutta käsittelevä osuus siirretään mahdollisesti projektin toiseen tai kolmanteen vaiheeseen.
25.8.2012	Korjaustyöt jatkuvat
27.8.2012	Dokumentaation laatiminen.
	Korjaustöiden jatko.
	Validaattorien uudelleen määrittely.
	Ominaisuus : Pelaajavalinta verrokkisarjoihin.
	Tarina : Käyttäjä valitsee verrattavan pelaajan verrokkisarjan tarkistamiseen

28.8.2012	Korjaustyöt jatkuvat.
29.8.2012	Korjaustyöt saatu loppuun muutamaa päivää etuajassa.
	Ominaisuus : Kiekkovalintojen sijoittaminen kontekstimenuihin.
	Tarina : Käyttäjä valitsee korilla käytetyn driverin.
	Tarina : Käyttäjä valitsee korilla käytetyn putterin.
30.8.2012	Ominaisuus : Verrokkipelaajan valitseminen.
	Tarina : Käyttäjä valitsee verrokkipelaajan.
5.9.2012	Prosessin arviointi.
6.9.2012	Dokumentaatio.
7.9.2012	Ominaisuus :Tallennusfunktioden laatiminen ja testaus.
9.9.2012	Kehitysversion esittely ja tulevaisuuden suunnitelmat.
10.9.2012	Käyttöliittymän viilaaminen julkaisukuntoon.
	Bugien korjausta.
	Dokumentaatio.
11.9.2012	Dokumentaatio.
12.9.2012	Dokumentaatio.
13.9.2012	Prosessin arviointi.
14.9.2012	Prosessin arviointi.
16.9.2012	Dokumentaatio.
17.9.2012	Prosessin refaktorointi.
19.9.2012	Prosessin refaktoirointi.
20.9.2012	Dokumentaatio.
21.9.2012	Dokumentaatio.
22.9.2012	Dokumentaatio.
23.9.2012	Prosessin reflektointi
24.9.2012	Prosessin refaktorointi
25.9.2012	Kokemuksien kirjaaminen.
26.9.2012	Projektin toisen vaiheen valmistelu alkaa. Tästä eteenpäin

	projektia ei käsitellä opinnäytetyön puitteissa.

Liite 3. vakavuusmatriisi

	T_r = olematon	T_r = pieni	T_r = keksinkertainen	T_r = Suuri	T_r = todennäköinen
V_r= olematon	Olematon	olematon	Olematon	Pieni	keskinkertainen
V_r= Pieni	Olematon	pieni	Pieni	keskinkertainen	keskinkertainen
V_r= Keskinkertainen	Pieni	pieni	keskinkertainen	keskinkertainen	suuri
V_r=suuri	Pieni	keskinkertainen	keskinkertainen	Suuri	katastrofaalinen
V_r= Katastrofaalinen	keskinkertainen	keskinkertainen	Suuri	katastrofaalinen	katastrofaalinen

Liite 4. käyttäjäpersoonat

Persoona	Kategoria	Odotukset	Toiminnot
Pääkäyttäjä	Henkilö, ensisijainen käyttäjä	Tulosten syöttäminen tietokantaan ja tulosten vertailu.	Tulosten talletus, muokkaus ja poisto sekä tulosten vertailu.
Tietokanta	Laitteisto	Tietokantaoperaatioiden toteutus käyttäjien toimien perusteella.	Suorittaa halutut tietokantaoperaatiot.

Liite 5. riskit

Riski	Verkkoyhteyden katkeaminen	**Kuvaus**	Internet- tai lähiverkkoyhteys katoaa.		
Seuraus	Verkkoresurssit eivät ole saatavilla ja työprosessi joudutaan keskeyttämään niiltä osin kuin se on tarpeellista. Tästä seurauksena on projektin aikataulun venyminen.				
T_r	keskinkertainen	**V_r**	keskinkertainen	**K_r**	keskinkertainen
Ehkäisy	Lähiverkon säännöllinen ylläpito. Kriittisen informaation ja tiedostojen paikallinen varastointi.				
Toimenpiteet	Tarkista oma laitteisto ja sen asetukset. Yritä yhteyttä 5 minuutin välein. Jos yhteys ei palaa kohtuullisessa ajassa ota yhteys palveluntarjoajaan.				
Seuranta	Jos pitkät tai useat lyhyet katkokset ovat yleisiä, on syytä harkita palveluntarjoajan vaihtoa.				
Tyyppi	Geneerinen / tekninen riski				
Tehdyt toimenpiteet	Ei toimenpiteitä				

Riski	Suunnitteluvirhe			**Kuvaus**	Virhe ominaisuuksien, käyttöliittymän tai arkkitehtuurin suunnittelussa.
Seuraus	Projekti keskeytyy ja tarvittavat korjaus- ja refaktorointityöt ovat suoritettava ennen kuin projekti voi jatkua.				
T_r	keskinkertainen	**V_r**	katastrofaalinen	**K_r**	Suuri
Ehkäisy	Huolellinen suunnittelu.				
Toimenpiteet	Tarvittavien korjausten tekeminen.				
Seuranta	Hyväksyntätestaus.				
Tyyppi	Ohjelmistokohtainen / tekninen riski.				
Tehdyt toimenpiteet	Arkkitehtuurivirhe toteutui 23.8.2012. Korjaustyöt kestivät noin viikon.				

Riski	Sairastuminen		**Kuvaus**	Ohjelmoija tai projektin sidosryhmiin kuuluva henkilö sairastuu.
Seuraus	Projekti viivästyy.			
T_r	suuri	**V_r**	suuri	**K_r**	suuri
Ehkäisy	Peruskunnon ylläpito.			
Toimenpiteet	Aikataulun uudelleen arviointi.			
Seuranta	Ei erityisiä toimenpiteitä.			
Tyyppi	Geneerinen / henkilöstöriski.			
Tehdyt toimenpiteet	-			

Riski	PC-laitteiston rikkoutuminen.	**Kuvaus**	PC-laitteisto tai sen osa rikkoutuu siten, että laitteistoa ei voida käyttää.
Seuraus	PC-laitteistoa ei voida käyttää projektissa.		
T_r	pieni	**V_r**	pieni	**K_r**	Pieni
Ehkäisy	Normaali PC-laitteiston ylläpito ja huolto sekä varmuuskopioiden pitäminen laitteiston ulkopuolisessa tallennusmediassa.		
Toimenpiteet	Varmuuskopioiden siirtäminen varalaitteistoon ja projektin jatkaminen varalaitteistolla.		
Seuranta	-		
Tyyppi	Geneerinen / tekninen riski		
Tehdyt toimenpiteet	-		

Riski	Projektitiedostojen häviäminen tai vioittuminen.	**Kuvaus**	Projektiin kuuluva tiedosto häviää tai vioittuu syystä tai toisesta.
Seuraus	Projektin aikataulu viivästyy, koska vioittuneet tai hajonneet tiedostot joudutaan palauttamaan edellisestä varmuuskopioista.		
T_r	keskinkertainen	**V_r**	keskinkertainen	**K_r**	Keskinkertainen
Ehkäisy	Varmuuskopioiden säännöllinen otto.		
Toimenpiteet	Palautetaan tiedostot varmuuskopioista.		
Seuranta	-		
Tyyppi	Geneerinen / tekninen riski		
Tehdyt toimenpiteet	Palautettiin versioita useita kertoja lähinnä ohjelmointivirheiden takia, koska joissain tapauksissa versiopalautus oli nopeampi tapa korjata ongelma.		

Liite 6. tarinat

Tarina	
ID : Käyttäjä lisää tuloksen MainDataGridViewiin	Kuvaus : Käyttäjä syöttää korituloskenttään pelatun korin tuloksen.
Hyväksymiskriteerit	
1. Käyttäjä voi syöttää tuloskenttään nollan tai positiivisen kokonaisluvun.	
Moduulit joihin vaikutetaan	
Discgolfstats.exe	Tarvittavat syöttökontrollit.
MainDataGridView	Lisätään koritulos MainDataGridviewiin.
Dokumentit	
Käyttöohje	Perusohjeistus korituloksen lisäämiseen.
Testitapaukset	
Manuaalinen	Tarkista sopivatko tulokset raja- ja par-arvoihin.
Automaattinen	Lisätään syöttökontrolleihin validaattorit ja värikoodaus tietotyyppien ja raja-arvojen tarkastusta varten .

Tarina	
ID : Käyttäjä muokkaa tulosta MainDataGridViewissä	Kuvaus : Käyttäjä valitsee muokattavan solun MainDataGridViewistä ja muuttaa olemassa olevan arvon toiseksi.
Hyväksymiskriteerit	
1. Käyttäjä voi valita olemassa olevan solun. 2. Käyttäjä voi muuttaa solussa olevan arvon. 3. Uusi lukuvoi olla nolla tai positiivinen kokonaisluku.	
Moduulit joihin vaikutetaan	
Discgolfstats.exe	Tarvittavat syöttökontrollit.
MainDataGridView	Muutetaan olemassaolevaan koritulosta valitussa solussa.
Dokumentit	
Käyttöohje	Perusohjeistus korituloksen muuttamiseen.
Testitapaukset	
Manuaalinen	Tarkista sopivatko tulokset raja- ja par-arvoihin.
Automaattinen	Lisätään syöttökontrolleihin validaattorit ja värikoodaus tietotyyppien ja raja-arvojen tarkistusta varten.

Tarina	
ID : Käyttäjä tuhoaa tulosrivin.	Kuvaus : Käyttäjä valitsee poistettavan rivin valitsemalla solun kyseiseltä riviltä. Käyttäjä avaa kontekstivalikon hiiren oikealla näppäimellä ja valitsee kontekstivalikosta rivin tuhoamisen.
Hyväksymiskriteerit	
1. Käyttäjä voi valita solun. 2. Käyttäjä voi avata kontekstivalikon. 3. Käyttäjä voi valita rivin tuhoamisen kontekstivalikosta. 4. Rivi tuhoutuu MainDataGridViewistä.	
Moduulit joihin vaikutetaan	
MainDataGridView	Rivi poistuu MainDataGridViewistä.
FDDB.mdb	Rivi poistetaan myös FDDB.mdb:stä piilodatagridviewien eheyden säilyttämiseksi.
DiscGolfStats.exe	Tarvittavat kontrollit.
Dokumentit	
Käyttöohje	Perusohjeet rivin poistoon.
Testitapaukset	
Manuaalinen	Rivin poisto kontekstimenun kautta ja tuloksen tarkistaminen sekä tietokannasta että datagridvieweistä.
Automaattinen	-

Tarina	
ID : Käyttäjä tarkastelee korin tuloksia	Kuvaus : Käyttäjä valitsee korituloksen MainDataGridviewistä kohdistamalla solun joko hiirellä tai näppäimistöllä. Korin tulokset tulostuvat koripaneeliin.
Hyväksymiskriteerit	
1. Käyttäjä voi valita solun/korin. 2. Korin tiedot tulostuvat koripaneeliin.	
Moduulit joihin vaikutetaan	
DiscGolfStats.exe	Tarvittavat komponentit korituloksen visualisoimiseen.
Dokumentit	
Käyttöohje	Peruskäyttöohje tulosten tarkasteluun.
Testitapaukset	
Manuaalinen	Valitaan solu ja tarkastetaan tulostuuko koritulokset oikein testimateriaalilla.
Automaattinen	-

Tarina	
ID : Käyttäjä valitsee verrokkipelaajan	Kuvaus : Käyttäjä valitsee tulosvertailuun käytettävän pelaajan tai kaikki pelaajat verrokkipulldownmenusta ja vertailutulos kohdistuu valittuun pelaajaan.
Hyväksymiskriteerit	
1. Käyttäjä voi valita verrokkipelaajan. 2. Vertailu kohdistuu valittuun pelaajaan. 3. Verrokkisarjat tulostuvat oikein.	
Moduulit joihin vaikutetaan	
DiscGolfStats.exe	Perus kontrollit
Dokumentit	
Käyttöohje	Perusohjeet tulosten vertailuun.
Testitapaukset	
Manuaalinen	Valitaan alasvetovalikosta verrokkipelaaja ja tarkastetaan tulostuuko tulokset oikein.
Automaattinen	-

Tarina	
ID : Käyttäjä vertaa tulosta verrokkisarjaan	Kuvaus : Käyttäjä valitsee tulosrivin ja vertaa sitä vertailuvalikosta valitun pelaajan parhaaseen sarjaan ja kaikkein parhaaseen sarjaan.
Hyväksymiskriteerit	
1. Käyttäjä voi valita rivin. 2. Vertailutieto tulostuu oikein.	
Moduulit joihin vaikutetaan	
DiscGolfStats.exe	Tarvittavat kontrollit ja visualisointikomponentit.
Dokumentit	
Käyttöohje	Peruskäyttöohje verrokkisarjojen tarkistukseen.
Testitapaukset	
Manuaalinen	Verrokkisarjojen tarkastaminen testimateriaalilla.
Automaattinen	Verrokkisarjojen värikoodaus.

Tarina	
ID : Käyttäjä lisää rivin tulosnäkymään.	Kuvaus : Käyttäjä painaa tulosnäkymän päällä hiiren oikeaa painiketta ja valitsee tulosnäkymän kontekstimenusta rivin lisäyksen.
Hyväksymiskriteerit	
1. Käyttäjä voi avata MainDataGridViewin kontekstivalikon ja valita siitä rivin lisäyksen. 2. Rivi lisäytyy MainDataGridViewiin.	
Moduulit joihin vaikutetaan	
DiscGolfStats.exe	Tarvittavat kontrollit.
Dokumentit	
Käyttöohje	Perusohjeet rivin lisäykseen.
Testitapaukset	
Manuaalinen	Tarkistetaan lisäytyykö rivin MaindataGridViewiin.
Automaattinen	-

Tarina	
ID : Käyttäjä tarkastelee ratakohtaista tulosjakaumaa	Kuvaus : Käyttäjä valitsee tulosnäkymästä yhden solun riviltä Jolla olevan radan ratakohtainen tulosjakauma tulostuu tulosjakaumakuvaajaan.
Hyväksymiskriteerit	
1. Käyttäjä voi valita solun. 2. Ratakohtainen tulosjakauma tulostuu tulosjakaumakuvaajaan.	
Moduulit joihin vaikutetaan	
DiscGolfStats.exe	Tarvittavat kontrollit.
Dokumentit	
Käyttöohje	Perusohjeet tulosten vertailuun.
Testitapaukset	
Manuaalinen	Valitaan tulossolu ja tarkastetaan tulostuuko jakauma oikein.
Automaattinen	-

Tarina	
ID : Käyttäjä tarkastelee korikohtaista tulosjakaumaa	Kuvaus : Käyttäjä valitsee yhden tulossolun tulosnäkymästä ja radan korikohtainen tulosjakauma tulostuu korikohtaisen tulosjakauman kuvaajaan.
Hyväksymiskriteerit	
4. Käyttäjä voi valita tulossolun. 5. Korikohtainen tulosjakauma tulostuu oikein.	
Moduulit joihin vaikutetaan	
DiscGolfStats.exe	Tarvittavat kontrollit.
Dokumentit	
Käyttöohje	Perusohjeet tulosten vertailuun.
Testitapaukset	
Manuaalinen	Valitaan tulossolu ja tarkistetaan tulostuuko jakauma oikein.
Automaattinen	-

Tarina	
ID : Käyttäjä tarkastelee henkilökohtaista korikohtaista tulosjakaumaa-	Kuvaus : Käyttäjä valitsee yhden tulossolun tulosnäkymästä ja henkilökohtainen korikohtainen tulosjakauma tulossolun edustamalla korilla ja radalla tulostuu henkilökohtaiseen tulosjakaumakuvaajaan.
Hyväksymiskriteerit	
1. Käyttäjä voi valita tulossolun. 2. Henkilökohtainen korikohtainen tulosjakauma tulostuu oikein.	
Moduulit joihin vaikutetaan	
DiscGolfStat.exe	Tarvittavat kontrollit.
Dokumentit	
Käyttöohje	Perusohjeet tulosvertailuun.
Testitapaukset	
Manuaalinen	Valitaan tulossolu ja tarkistetaan tulostuuko jakauma oikein.
Automaattinen	-