LAUREA

.Net ohjelmointiympäristön suunnittelu Handelsbanken Central Data Finlandille

Antti Saarikivi

2010 Leppävaara

Laurea-ammattikorkeakoulu
Laurea Leppävaara

.Net ohjelmointiympäristön suunnittelu Handelsbanken Central Data Finlandille

Saarikivi, Antti
Tietojenkäsittelyn koulutusohjelma
Opinnäytetyö
Elokuu 2010

Laurea-ammattikorkeakoulu **Tiivistelmä**
Laurea Leppävaara
Tietojenkäsittelyn koulutusohjelma

Antti Saarikivi

.Net-ohjelmointiympäristön suunnittelu Handelsbanken Central Data Finlandille

Vuosi **2010** **Sivumäärä** **46**

Tämän opinnäytetyön tarkoituksena on suunnitella Handelsbanken Central Data Finlandille (CD-FI) .Net ohjelmointia tukeva ympäristö. Tässä opinnäytetyössä tullaan käyttämään Handelsbanken Central Data Finlandista jatkossa lyhennettä CD-FI.

Ohjelmistojen kehittäjät eivät pysty työskentelemään tehokkaasti, mikäli ohjelmistokehityksen parissa työskentelevällä organisaatiolla ei ole keskitettyä järjestelmää, jolla hallinnoidaan kehityksestä syntyvää informaatiota. Tämä johtuu siitä, että kehittäjällä kuluu turhaa aikaa tarvittavan informaation kasaamiseen useasta eri järjestelmästä.

Ympäristön suunnittelussa on otettu huomioon CD-FI:n tarpeet niin ohjelmistokehityksen kuin projektiorganisaation vaatimien ylläpitotehtävien osalta. CD-FI:n vaatimukset selvitettiin haastattelemalla CD-FI:n osastopäällikköä, joka toimii myös tämän työn tilaajana. Opinnäytetyössä määritellään lyhyesti, mitä .Net tarkoittaa sekä esitellään ympäristön tärkeimmät toimijat.

Ohjelmointiympäristö rakentuu Application Lifecycle Management (ALM) –ohjelmiston ympärille. ALM – ohjelmiston tehtävänä on kerätä kaikki ympäristön informaatio yhden ohjelman alle. Ohjelmasta käsin voidaan hallita informaatiota sekä jakaa sitä ympäristön käyttäjille.

Markkinoilla on saatavilla useita ALM-ohjelmistoja, mutta tähän opinnäytetyöhön valittiin kolme potentiaalisinta ohjelmistoa, joista paras valittiin Kepner Tregoe-menetelmällä. Vertailtavia ohjelmistoja olivat VersionOne Enterprise, Microsoft Team Foundation Server sekä Borland StarTeam Enterprise.

Ympäristön ALM-ratkaisuksi valittiin VersionOne Enterprise, koska se osoittautui ylivoimaisesti parhaiten muokattavaksi ALM-järjestelmäksi. Myös ohjelman helppo käytettävyys sekä yksinkertainen yhdistäminen jo CD-FI:llä käytössä oleviin ohjelmiin vahvistivat valintaa.

Määrittelemällä jo olemassa olevan ympäristön toiminnot on organisaation mahdollista löytää heille parhaiten sopiva ALM-järjestelmä. Ympäristöön sopivalla ALM-järjestelmällä voi jopa alle kymmenen hengen organisaatio saada huomattavaa lisäarvoa ohjelmistokehitykseen.

Asiasanat **.Net, Ohjelmointiympäristö, Application Lifecycle Management**

Laurea University Of Applied Sciences **Abstract**
Laurea Leppävaara
Business Information Technology

Antti Saarikivi

Designing a .Net Programming Environment for Handelsbanken Central Data Finland

Year 2010 **Pages 46**

The goal of this thesis is to contemplate an environment that supports .Net programming for Handelsbanken Central Data Finland (CD-FI). I will be using a shortening CD-FI for Handelsbanken Central Data Finland.

If organization that develops software does not have centralized system to manage information that developing software generates, it is possible that software developers cannot work efficiently. This occurs because the software developer has to gather information from variable amount of systems.

While contemplating this environment the requirements for software development and for the administrating task for the project management of CD-FI, has been taken into account. The requirements were clarified by interviewing the department head of CD-FI whom also acts as the subscriber of this thesis. In this thesis the meaning of .Net and the main actors of the environment are shortly described.

The environment builds around Application Lifecycle Management (ALM) software. The purpose of this software is to collect all the information in this environment under one program, where one can easily manage it and distribute it among the users of this environment.

There are multiple ALM-software distributions in the market, but for this thesis I have chosen the three most promising software which were ranked with Kepner Tregoe method. The three programs that were compared include VersionOne Enterprise, Microsoft Team Foundation Server and Borland StarTeam Enterprise.

The chosen ALM-software package for this environment is VersionOne. VersionOne was chosen because it was found to be the most modifiable ALM software. Also the ease of use and the simple connectivity to already existing software at CD-FI confirmed the selection.

Defining the existing environment and the functionalities that are needed, it is possible for an organization to find the most suitable ALM solution for their demand. With an ALM that suits the environment it is possible for an organization even smaller than ten persons to intensify their software development excessive.

Key words **.Net, Programming environment, Application Lifecycle Management**

Sisällys

1 Johdanto

”Ohjelmoija joka ohjelmoi kaasu pohjassa, joutuu pitämään miljoonia asioita muistissaan samanaikaisesti - Joel Spolsky” (Erickson 2009, 1). Gloria Markin mukaan ”Ohjelmistokehittäjä työskentelee keskimäärin 11 minuuttia yhden tehtävän kanssa ennen kuin hänet keskeytetään. Keskeytyksen jälkeen aikaa kuluu keskimäärin 25 minuuttia siihen että, kehittäjä on päässyt jatkamaan alkuperäistä tehtäväänsä.” (Erickson 2009, 1).

Yleinen toimintamalli ohjelmistokehityksessä on ollut useat erilliset ohjelmat, joilla ohjelmistokehitys hoidetaan. Kun tärkeää informaatiota säilytetään useassa eri järjestelmässä, on työyhteisön hankalaa työskennellä tehokkaasti yhdessä. Tämä johtaa edelleen lisätyöhön ja mahdollisesti kommunikaatiovirheisiin. (Microsoft Corporation 2007,5.)

Tämän ongelman ratkaisuksi on kehitetty Application Lifecycle Management (ALM) -määritelmä, jonka tarkoituksena on yhdistää kaikki ohjelmistokehityksessä ja projektinhallinnassa tarvittavat ohjelmat yhdeksi kokonaisuudeksi. Ensimmäiset ALM-versiot, eivät kuitenkaan täyttäneet niiltä vaadittuja toimintoja. Tämän vuoksi ALM-järjestelmät eivät ole tulleet suuren yleisön tietoisuuteen. Tällä hetkellä markkinoilla on olemassa versio kahden (2) ALM-järjestelmiä, jotka täyttävät vähitellen alkuperäiset vaatimukset. Tämän vuoksi niiden tuottama lisäarvo on helposti todettavissa ympäristöissä, joissa useampi ihminen työskentelee saman projektin parissa.

Tämän opinnäytetyön tavoitteena on määritellä .Net-ohjelmointiympäristö Handelsbanken Central Data Finlandin (CD-FI) käyttöön. Vaikka opinnäytetyön tarkoitus on palvella CD-FI:n tavoitteita ja ratkaisu on tehty heidän ympäristönsä sekä tarpeidensa mukaisesti, on tästä opinnäytetyöstä apua kaikille, jotka työskentelevät vastaavanlaisen projektin kanssa sekä kaikille .Net- tai projektinhallintaympäristöistä kiinnostuneille.

2 Tutkimusongelma

Tarve tämän opinnäytetyön tekemiselle syntyi kun Handelsbanken Central Data Finlandissa todettiin, että .Net-ohjelmointi tulee yrityksessä lisääntymään. Yrityksellä ei kuitenkaan tällä hetkellä ole minkäänlaisia määrityksiä .Net-kehitykselle vaan kehittäjät tekevät työtänsä pelkästään omilla työasemillaan ilman yhtenäistettyjä tuotantoonottorutiineja sekä dokumentaatiota. Tätä ei ole koettu tarpeelliseksi muuttaa, sillä .Net-kehittäjiä yrityksessä

on ollut vain muutamia, kun nyt henkilöstöä ryhdytään kouluttamaan .Net:n käyttöön on tarpeellista saada yhtenäinen ympäristö .Net-kehitykselle.

Tämän työn tutkimusongelmana onkin selvittää, minkälainen .Net-ympäristö sopisi parhaiten Central Data Finlandin tarpeisiin. Lisäksi selvitetään, miten tulevasta ympäristöstä saadaan mahdollisimman helppokäyttöinen sekä toimintaperiaatteeltaan samanlainen kuin jo olemassa olevat ympäristöt.

3 Handelsbanken

Handelsbanken perustettiin vuonna 1871 Tukholmassa, ja se on levittäytynyt jo yli 20 maahan. Suomessa Handelsbanken on toiminut vuodesta 1985 saakka ja konttoreita sillä on 45 kappaletta. Handelsbankenin konttorit toimivat paikallisesti, mikä tarkoittaa sitä, että jokaisella konttorilla on valta ja vastuu alueensa liiketoiminnasta. Tämä mahdollistaa sen, että asiakasta koskevat päätökset voidaan tehdä nopeasti, joustavasti ja lähellä asiakasta.

Vakaus on Handelsbankenille tärkeä periaate. Handelsbanken perustaa konttoreita harkiten, mutta tulee jäädäkseen. Vakaus merkitsee myös varovaisuutta luottopolitiikassa. 1990-luvulla Handelsbanken oli ainoa pohjoismainen pankki, joka selvisi pankkikriisistä ilman valtion tukea. (Handelsbanken 2010.)

4 Kepner Tregoe

Kepner Tregoe-matriisi on analysointi- ja päätöksentekometodi. Kepner Tregoe on vaiheittainen lähestymistapa systemaattiseen ongelmanratkaisuun, päätöksentekoon sekä potentiaalisten riskien analysointiin. Kepner Tregoe-matriisin käyttö helpottaa päätöksentekoa tarjoamalla työkaluja tehostamaan päätöksenteossa vaadittavia toimintamalleja. Näitä toimintamalleja ovat kriittinen ajattelu, systemaattinen informaation organisointi ja priorisointi, tavoitteiden asettaminen, vaihtoehtojen arviointi sekä vaikutusten analysointi (Decide Guide-Internet sivut).

Kepner Tregoe-matriisi on suosittu työkalu, koska se karsii tietoisia sekä alitajuisia harhoja, joilla on tapana ohjata päätös pois pääsääntöisistä tavoitteista. Kepner Tregoe-matriisia voidaan soveltaa usean tyyppisiin päätöksiin, mutta sen monimutkaisen vaikutelman sekä

jatkuvan harjoittelun takia Kepner Tregoe mallia käytetään pääsääntöisesti liikkeenjohdon tai liiketoiminnan päätöstentekoon (Decide Guide-Internet sivut).

Kepner Tregoe määrittelee seuraavat vaiheet, joilla saavutetaan onnistunut päätöksenteko analyysi:

1. määrittele päätöslausunto, joka sisältää toiminnon sekä lopputuloksen
2. osoita strategiset vaatimukset (musts), operationaaliset tavoitteet (wants) sekä rajoitukset (limits)
3. Arvota tavoitteet ja määrittele relatiiviset painot
4. Listaa vaihtoehdot
5. Määrittele relatiivinen pisteytys jokaiselle vaihtoehdolle tavoitteiden mukaisesti
6. laske painotettu arvo jokaiselle vaihtoehdolle, sekä identifioi kaksi tai kolme korkeinta pistearvoa saanutta vaihtoehtoa
7. Listaa haitalliset seuraukset jokaisesta identifioidusta vaihtoehdosta ja määrittele todennäköisyys (korkea, keskitaso, matala) sekä vakavuusaste (korkea, keskitaso, matala)
8. Tee lopullinen päätös identifioiduista vaihtoehdoista

(Decide Guide-Internet sivut)

5 Ohjelmointiympäristön vaatimukset

Tätä työtä varten haastateltiin Petri Halosta, joka toimii CD-FI:n osastopäällikkönä. Petri Halonen on todennut tarpeen .Net-ohjelmointiympäristön luomiselle CD-FI:lle. Haastattelun tarkoituksena oli määritellä vaatimuksia ohjelmointiympäristölle. Nämä vaatimukset pisteytetään Kepner Tregoe – mallin mukaisesti, jolloin saamme parhaan mahdollisen ohjelmiston CD-FI:n tarpeisiin.

CD-FI:ssä on ollut jo pitkään käytössä kaksi muuta ohjelmointikieltä, MAPPER ja COBOL, joten näillä ohjelmointikielillä ovat ympäristöt, tuotantoon otto- sekä versionhallintarutiinit jo varsin selkeät. Tulevan .Net ohjelmointiympäristön olisi hyvä muistuttaa näitä, jo olemassa olevia ympäristöjä mahdollisimman hyvin.

CD-FI on tullut siihen tulokseen, että .Net-ohjelmoinnin osaamisen tarve tulee lisääntymään osastolla. Heiltä kuitenkin puuttuu yhtenäistetty ympäristö ja näin ollen kehitys on ollut hajanaista. CD-FI:n nykyinen henkilöstö tarvitsee myös lisäkoulutusta .Net-ohjelmointiin, koska monelle työntekijälle koko olio-ohjelmoinnin käsite on täysin vieras. Näiden syiden

takia ohjelmointiympäristön tulee olla mahdollisimman selkeä ja sen tulee sulautua CD-FI:n nykyisiin toimintamalleihin ja infrastruktuurin mahdollisimman hyvin. Ohjelmointiympäristö suunnitellaan vain .Net-kehitykseen ja CD-FI:n tarpeisiin. Tämä tarkoittaa sitä, että liiketoiminnan osuutta tai muita ohjelmointikieliä ei tarvitse ympäristön suunnittelussa ottaa huomioon.

CD-FI haluaa ohjelmointiympäristön pystyyn mahdollisimman nopeasti, jotta he pääsevät kouluttamaan henkilöstöä .Netin käyttöön. Ympäristön asennuksen helppoudella ei ole suurta painoarvoa, sillä se tehdään vain yhden kerran, mutta ympäristön ylläpidon tulisi olla mahdollisimman helppoa ja tehokasta. (Halonen 2010.)

Haastattelussa selvisi, että CD-FI:llä on lisenssit Microsoftin Visual Studioon, joka on luotu .Net-ohjelmointiin, näin ollen tulee ympäristön kaikkien osapuolten tukea Visual Studiota. Tuki Visual Studiolle kuuluukin ohjelmointiympäristön, Kepner Tregoe – mallin mukaisiin, strategisiin vaatimuksiin. Myös tuki TortoiseSVN versionhallinalle lukeutuu ohjelmointiympäristön strategisiin vaatimuksiin, koska CD-FI:llä on jo kattavasti TortoiseSVN käytössä. Taulukossa 1 on kuvattu ohjelmointiympäristön vaatimukset Kepner Tregoe – mallin mukaisesti kokonaisuudessaan.

Taulukko 1 Ohjelmointiympäristön vaatimukset Kepner Tregoe-mallin mukaisesti

Strategiset vaatimukset (”must haves”)	
	Microsoft Visual Studio tuki
	TortoiseSVN tuki
Operationaaliset tavoitteet (”want haves”)	
	Helppo käyttöönotto
	Räätälöinti mahdollisuus
	Kattavat tuki palvelut
	Ylläpidon helppous
	Scrum tuki
Rajoitukset (”limits”)	
	Korkea tietoturvataso

Näitä haastattelussa selvinneitä vaatimuksia tutkitaan tarkemmin kappaleessa kahdeksan, jossa vertaillaan eri ALM järjestelmiä sekä valitaan CD-FI:n vaatimuksiin nähden paras ALM-järjestelmä.

6 .Net-ympäristö

Tässä luvussa kerrotaan lyhyesti, mitä .Net tarkoittaa sekä hieman sen keskeisimmistä toiminnoista ja käsitteistä.

.Net on Microsoft Corporationin kehittämä tuote, ja he kuvaavat sitä seuraavasti: ".Net on Microsoftin Web Service –strategia, jonka tarkoituksena on yhdistää informaatio, ihmiset, järjestelmät ja laitteet ohjelmistojen avulla. Koska .Net-teknologia on integroitu Microsoft-pohjaisiin järjestelmiin, se mahdollistaa toisiinsa yhteydessä olevien, turvallisempien Web Service -ratkaisuiden nopean luomisen, levittämisen sekä hallinnan. .Net- ratkaisut mahdollistavat sen, että yritykset voivat integroida järjestelmänsä nopeammin ja joustavammin sekä auttavat yrityksiä ymmärtämään, kuinka arvokasta on kun informaatio on saatavilla milloin ja missä tahansa, millä tahansa laitteella."(Devotopics 2008.)

.Net tarjoaa työkaluja sekä ohjelmointikirjastoja, jotka nopeuttavat ja helpottavat Windows-ohjelmien kehittämistä. .Net hyödyttää loppukäyttäjää siten, että se mahdollistaa ohjelmien paremman suorituskyvyn, laadun sekä tietoturvan.

.Net koostuu neljästä pääkomponentista:

- Common Language Specification (CLS) - sininen
- Framework Class Library - punainen
- Common Language Runtime – vihreä
- .Net työkalut - keltainen

6.1 Common Language Specification (CLS)

CLS on ohjelmistoalusta, joka mahdollistaa ohjelmistokoodin sekä ohjelmistokomponenttien yhdistämisen, vaikka ne olisi kirjoitettu eri .Net-kielellä. Toisin sanottuna .Net-ohjelma voidaan kirjoittaa usealla eri .Net-ohjelmointikielellä eikä eri osien yhdistäminen vaadi kehittäjältä mitään lisäponnistuksia. Tosin ohjelmointikoodin siirtäminen kieleltä toiselle voi olla hankalaa. (Devotopics 2008.)

6.2 Framework Class Library (FCL)

FCL:llä tarkoitetaan luokkakirjastoa, joka sisältää yli 7000 luokkaa ja datatyyppiä, jotka puolestaan mahdollistavat sen, että .Net-ohjelmat voivat lukea tiedostoja ja kirjoittaa niihin, olla yhteydessä tietokantoihin, käsitellä XML:ää, piirtää grafiikkaa, käyttää Web Servicejä

sekä paljon muuta. FCL pakkaa valtavan Win32 API:n yksinkertaisempiin .Net-objekteihin, joita voi käyttää C#-,C++-,VB- sekä muut .Net -ohjelmointikielet. (Devotopics 2008.)

6.3 Common Languge Runtime (CLR)

CLR on ajonaikainen käyttöympäristö ja se toimii rajapintana .Net-ohjelmien sekä käyttöjärjestelmän välillä. CLR tarjoaa useita toimintoja kuten:

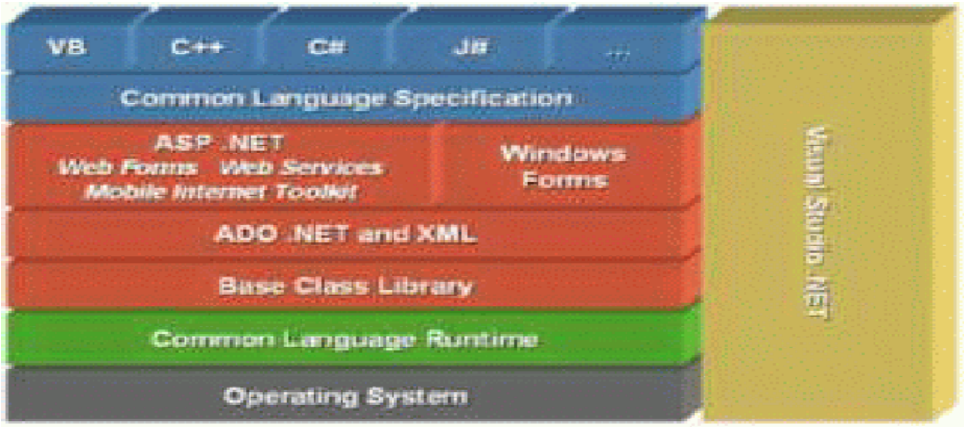
- ohjelmistokoodin lataaminen ja suorittaminen
- ohjelmien kääntäminen konekielelle
- muistin ja prosessien erottaminen
- objektien ja muistin hallinnointi
- pääsyoikeuksien ja ohjelmakoodin valvonta
- poikkeuksien käsittely
- rajapinnan hallitun koodin, COM-objektien ja DLL-tiedostojen välillä
- oikeinkirjoituksen tarkistus
- ohjelmistokoodin metadata
- profiloinnin sekä virheiden etsintä

(Devotopics 2008.)

6.4 .Net-työkalut

Visual Studio on Microsoftin lippulaiva Windows-ohjelmistojen kehittämiseen. Visual Studio tarjoaa integroidun ohjelmistoympäristön Windows-ohjelmistojen, interaktiivisten verkkosivujen, verkkosovellusten ja verkkopalveluiden rakentamisen ja ajamisen millä tahansa alustalla, joka tukee .Net-tekniikkaa. (Devotopics 2008.)

Visual Studio ei toki ole ainoa ohjelmointityökalu .Net-työskentelyyn, mutta koska CD-FI:llä on Visual Studio ollut jo aikaisemmassa käytössä, olisi vaihtaminen turhaa.



Kuva 1 .Net (Devotopics 2008)

7 Application Lifecycle management (ALM)

Tässä luvussa kerrotaan mitä ALM-järjestelmällä tarkoitetaan sekä mitä varten ALM-järjestelmät on luotu. Lisäksi käydään myös lyhyesti läpi Agile Development -nimisen yrityksen tekemän haastattelututkimuksen keskeisimmät tulokset.

Ohjelmistoprojekteissa esiintyy usein samat ongelmat. Kehittäjät tuskailevat sitä, että vaatimuksia tulee koko ajan lisää, olemassa olevat vaatimukset muuttuvat ja koko ajan olisi pakko tehdä ylitöitä. Projektijohtajien kannalta projektit ovat aina 99,99 % valmiita, julkaisupäivä ei ole tiedossa, ei tiedetä mitkä osat ovat riskialtteimmat ja tehtävien priorisointi on hankalaa.

ALM on järjestelmä, joka mahdollistaa liiketoiminnan ja ohjelmistokehityksen saumattoman yhteistyön. Tämä tapahtuu tarjoamalla kullekin henkilölle juuri se informaatio, mitä he tarvitsevat, ilman että heille syötetään kasoittain dataa, jota he eivät tarvitse. (deJong 2008,3.)

Kun kaikki tieto on yhdessä paikassa liitettynä toisiinsa, kaikki tapahtumat voidaan jäljittää ja verrata niitä alkuperäisiin vaatimuksiin, on projektin johdon helppo seurata ja hallinnoida projekteja. Näin myös vältetään tilanteet, joissa kaikki tieto olisi yhden kehittäjän takana. Tämä mahdollistaa myös sen, että kaikilla asianosaisilla on reaaliaikainen näkymä projektin tilaan ja laatuun.

7.1 Hyödyt

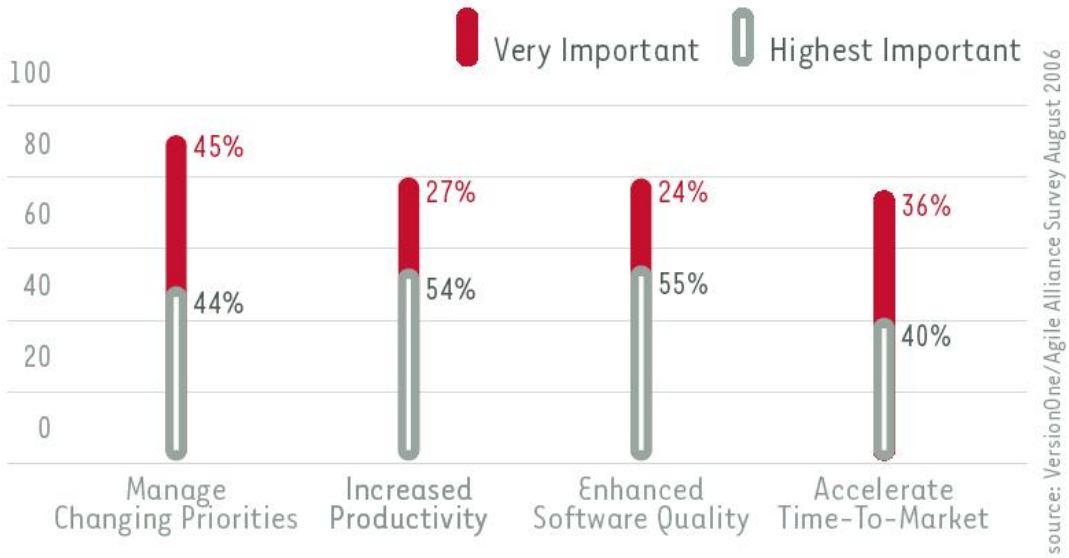
Tässä luvussa kerrotaan, mitä hyötyä yritykselle on ALM-järjestelmän käyttöönotosta ja avataan hieman myöhemmin esiteltävän, VersionOne-ohjelmiston valmistaja Agile Developmentin vuonna 2006 tekemän tutkimuksen tuloksia.

Tutkimuksessa paljastui, että ohjelmistojen kehityksen kanssa työskenteleville organisaatioille neljä tärkeintä tarvetta ovat:

1. tarve hallita muuttuvia määrityksiä
2. tarve parantaa kannattavuutta
3. tarve parantaa laatua
4. tarve nopeuttaa tuotantoon ottoja.

(VersionOne, LLC 2007, 3.)

Kuten kuvassa 2 on todettu, jopa 89 % vastanneista pitää määritysten muuttumisen hallinnoimista tärkeänä tai tärkeimpänä osa-alueena. Tämä tekee siitä hieman yllättäen tärkeämmän osa-alueen kuin tuottavuuden parantaminen. Tuottavuuden parantamista pitää tärkeänä tai tärkeimpänä 81 % vastanneista. Samalle tasolle ylsi myös tarve ohjelmistojen laadun parantamiseen. Näitä kahta osa-aluetta yhdistää se, että ne saivat ylivoimaisesti eniten kannatusta kun katsotaan kaaviosta ainoastaan tärkeimpiä arvoja.

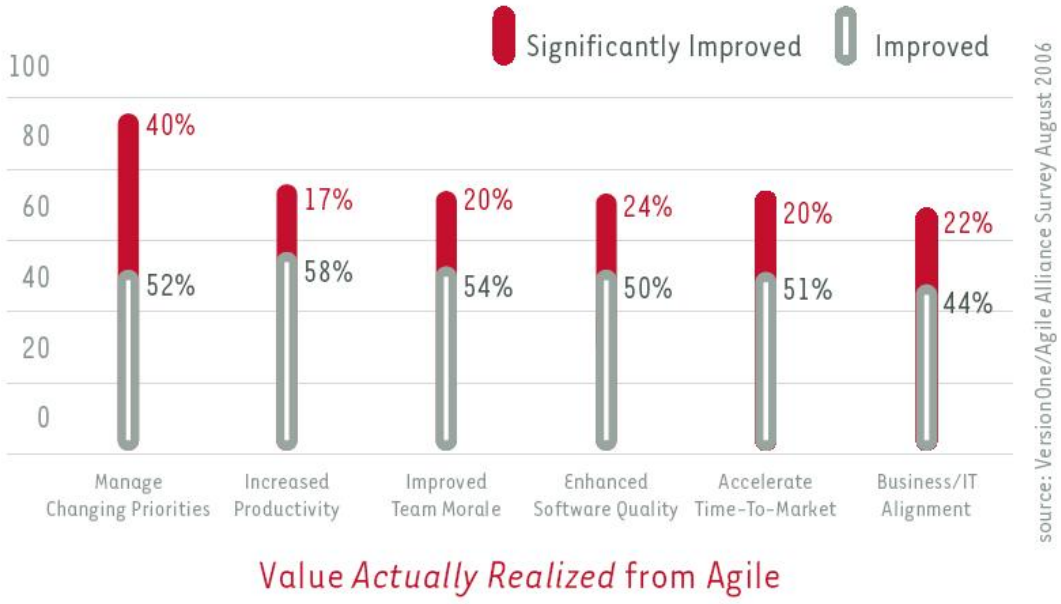


Kuva 2 VersionOne 1 (VersionOne, LLC 2007, 3)

Kuten kuvasta 3 voimme todeta, 92 % yrityksistä oli sitä mieltä, että tärkeimmäksi valittu osa-alue määritysten hallinnoiminen parani (52 %) tai parani huomattavasti (40 %). 75 %

vastanneista totesi, että heidän tuottavuutensa parani, joista 17 % ilmoitti sen parantuneen huomattavasti. Parantuneista osa-alueista kolmanneksi suurimman prosentin (74 %) sai parantunut työilmapiiri. Tämä kertoo siitä, että kun tarvittava tieto on helposti saatavilla ja hallittavissa, eivät työntekijät turhaudu hankalaan tiedon hankintaan. Vastaajista 74 % oli sitä mieltä, että ohjelmistojen laatu parani, 24 % oli sitä mieltä, että laatu parani huomattavasti. Tämä on toiseksi suurin huomattavasti parantunut osa-alue, määritysten hallinnoimisen jälkeen. 71 % vastanneista ilmoitti, että tuotantoonottoajat lyhenivät ja jopa 20 % oli sitä mieltä, että ne lyhenivät huomattavasti. Tämä viestittää siitä, että ALM-järjestelmää, tässä tapauksessa VersionOnea, käyttävillä yrityksillä on selkeä tuotantoketju, jonka jokainen osa-alue on hallinnoitu.

Usein puhutaan, kuinka hankalaa on saada liiketoiminnan edustajia sekä IT -puolen edustajia työskentelemään yhdessä. Tutkimuksen osallistuneista yrityksistä 66 % prosenttia ilmoitti tämän osa-alueen parantuneen. Tämä kertoo siitä, että yritysten prosessit ovat muuttuneet läpinäkyvämmiksi. ALM-järjestelmä mahdollistaa myös kattavan raportoinnin liiketoiminnalle. Tämän raportoinnin toimivuuden todistaa se, että 22 % vastanneista ilmoitti liiketoiminnan sekä IT-osaston välisen yhteistyön on parantuneen huomattavasti.



Kuva 3 VersionOne 2 (VersionOne, LLC 2007, 3)

Tutkimuksen perusteella voidaan todeta, että yritykset jotka ovat ottaneet ALM-järjestelmän käyttöön, ovat kokeneet seuraavia hyötyjä:

- muutosten hallinta on parantunut
- tuottavuus on parantunut
- työyhteisön ilmapiiri on parantunut
- ohjelmistojen laatu on parantunut
- tuotteet on saatu julkaistuksi nopeammin
- liiketoiminnan ja IT-osaston välinen kommunikaatio on parantunut

7.2 ALM 1.0

Ensimmäisten ALM-järjestelmien toiminta perustui siihen, että ne saivat useat erilliset ohjelmat keskustelemaan keskenään. Tällainen eri ohjelmien integraatio ei ole niin syvää ja joustavaa kun mainostettiin, varsinkaan jos ALM-järjestelmän yhdistämät ohjelmat ovat eri valmistajien. Toinen ongelma oli ajatus siitä, että muodostettaisiin yksi työkalu yhdelle roolille. Rooliperustaisten työkalujen ongelmana on se, että roolit vaihtelevat suuresti ympäristöstä riippuen. Kun yrityksen roolit eivät vastaa niitä rooleja, jolle ohjelmisto on kehitetty, täytyy yrityksen valita seuraavista vaihtoehdoista: vaihtaa yrityksen omia rooleja, ostaa enemmän työkaluja yhdelle roolille tai ostaa enemmän toimintoja kuin yksi rooli tulee koskaan tarvitsemaan. Ohjelmistonkehittäjät ovat vastaavasti alkaneet tekemään sellaisia ohjelmia, että ne sopivat kaikille käyttäjärooleille. Tämä tekee ohjelmista turhan laajoja ja kalliita, koska niissä on enemmän toimintoja kun kukaan yksittäinen henkilö tulee koskaan tarvitsemaan. (Orcanos:n Internet-sivut.)

ALM 1.0-järjestelmän haitoiksi tai piilokustannuksiksi voidaan laskea jo edellä mainittu perusajatus, yksi työkalu jokaiselle roolille. Tämä aiheuttaa huonon tuottavuuden monimutkaisten työkalujen takia. Kuten aikaisemmin mainittiin, tämä myös johtaa siihen, että roolien yhteensopimattomuus lisää yrityksen kustannuksia, koska heidän täytyy hankkia ylimääräisiä lisenssejä. ALM 1.0-järjestelmissä on yllättävän paljon tarpeetonta ja usein epäsäännöllistä toimintaa, ja pahinta on, että tieto, joka hyödyttäisi koko kehitystiimiä, on usein saatavana vain yhden työkalun kautta.

7.3 ALM 2.0

ALM 2.0 on enemmän alusta kehitystapahtumien hallinnointiin, kuin epämääräinen paketti työkaluja, joissa on lukittuja sekä rajoitettuja ALM-toimintoja. Kun käytetään yhtä työkalua kaiken ALM-toiminnan hallinnoimiseen, niin säästetään aikaa ja rahaa koko

kehittämisprosessissa. ALM 2.0:n tarkoitus ei ole olla työkalu pelkästään kehittäjille, vaan kaikille rooleille joita tarvitaan, että ohjelmisto vastaa liiketoiminnan tarpeita. (Gardner 2007.)

8 ALM-järjestelmien vertailu

Tässä kappaleessa verrataan muutamia johtavia ALM-ratkaisuja sekä CD-FI:n nykyistä tilannetta keskenään. Markkinoilla on useita ALM-järjestelmiä saatavilla, mutta tähän työhön on valittu kolme potentiaalisinta ratkaisua, joista valitaan yksi järjestelmä. Järjestelmät arvotetaan kahdella eri menetelmällä, saatavilla olevien ominaisuuksien perusteella sekä Kepner Tregoe päätöksentekomatriisin mukaisesti. Vertailujen tulokset yhdistetään ja eniten pisteitä saanut ALM-järjestelmä valitaan parhaaksi CD-FI:n tarpeisiin. Lisäksi selvennetään mitkä ovat suurimmat hyödyt mitä CD-FI voi ALM- järjestelmän käytöstä saada.

8.1 ALM järjestelmien vertailu saatavilla olevien toiminnallisuuksien perusteella

Alla olevassa taulukossa vertaillaan kolmen ALM-järjestelmän ominaisuuksia. Koska kaikista ALM-järjestelmistä on saatavilla erihintaisia ja -tasoisia paketteja niin kaikilta palveluntarjoajilta on valittu heidän keskikokoinen järjestelmänsä.

Taulukosta 1 nähdään, että VersionOne Enterprise ja Team Foundation Server Premium ovat valittujen toimintojensa osalta lähes identtiset, kun taas Boreland:n StarTeam eroaa ominaisuuksiltaan selkeästi. Tästä huolimatta Boreland on hinnoiteltu kilpailijoitaan huomattavasti kalliimmaksi. Boreland:n ALM-ratkaisun hinnat ovat tosin Moonsoft:n Internet-sivuilta, koska Boreland ei ole hinnastoa sivuillaan julkistanut. Microsoftin Team Foundation Server olisi muuten potentiaalinen vaihtoehto, mutta se on suunniteltu huomattavasti suuremmille käyttäjämäärille, kuin mitä CD-FI:llä on henkilöstöä. Microsoft Team Foundation Server kyllä suoriutuu pienemmistäkin käyttäjämääristä, mutta järjestelmä tuhlaa turhan paljon ympäristön resursseja käyttäjiin nähden. VersionOneen vahvuus onkin mahdollisuus liittää kolmannen osapuolen ohjelmistoja järjestelmään. VersionOne:n Internet-sivuilta löytyy kattava lista saatavilla olevista yhdistäjistä. Mikäli listasta puuttuu vaadittavan ohjelman yhdistämiskomponentti, niin VersionOne tarjoaa avoimen lähdekoodin järjestelmään yhdistymiseksi. Tämän johdosta yritys voi tarvittaessa ohjelmoida oman yhdistämiskomponenttinsa ja näin ollen yhdistää käytännössä minkä tahansa ohjelman VersionOne:n. Tämä tietenkin vaatii sen, että yhdistettävä ohjelman tarjoaa myös vastaavan rajapinnan ohjelmien yhdistämiselle.

Taulukko 2 ALM-järjestelmien vertailu (VersionOne, MS Team Foundation & Boreland StarTeam Internet-sivut)

	VersionOne Enterprise	**Microsoft Team Foundation Server Premium**	**Borland StarTeam Enterprise**
Tulevaisuusnäkymä	Ei	Ei	Ei
Projektikohtaiset näkymät	Kyllä	Ei mainintaa	Osittain
Julkaisu- ja iteraatio-suunnitelmat	Kyllä	Kyllä	Ei
Virheiden seuranta	Kyllä	Kyllä	Ei
Projektien ja tiimien väliset raportit	Kyllä	Kyllä	Kyllä
Web-Service API	Kyllä	Kyllä	Ei
Java & .Net SDK	Kyllä	Kyllä	Kyllä
Ilmaiset, avoimen lähdekoodin yhditämiskomponentit	Kyllä	Kyllä (käyttöä ei suositella)	Ei
Hinta	Per käyttäjä $29 kk/ $595 ikuinen	5 käyttäjää $499 (Useampi käyttäjä eri hinnaston mukaan)	Nimetty käyttäjä $2119* vapaa lisenssi $6357*

*Boreland StarTeam hinnasto Moonsoft:n Internet-sivuilta.

8.2 ALM järjestelmien pisteytys Kepner Tregoe-matriisin mukaisesti

Strategiset vaatimukset ovat pakollisia ympäristön toiminnalle. Mikäli jokin vaatimus ei toteudu, niin kyseinen järjestelmä voidaan hylätä suoraan.

Taulukko 3 Strategiset vaatimukset (”must haves”)

Vaatimus	VersionOne Enterprise	Microsoft Team Foundation Server Premium	Borland StarTeam Enterprise
Microsoft Visual Studio tuki	Kyllä	Kyllä	Kyllä
TortoiseSVN tuki	Kyllä	Kyllä	Kyllä

Seuraavaksi painoarvotamme haastattelun perusteella selvinneet operationaaliset objektiivit (”want haves”).

Taulukko 4 Operationaaliset tavoitteet (”want haves”)

Tavoite	Painoarvo
Helppo käyttöönotto	9
Räätälöintimahdollisuus	8
Kattavat tukipalvelut	6
Ylläpidon helppous	7
Scrum-tuki	3

Seuraava vaihe on pisteyttää kuinka hyvin kukin ALM-järjestelmä toteuttaa operationaaliset tavoitteet. Tämän jälkeen jokaisen vaihtoehdon pisteet kerrotaan tavoitteen painoarvolla. Näin saamme selville, mikä ALM-järjestelmä toteuttaa parhaiten kunkin tavoitteen.

Kuten taulukoista 5,6 ja 7 näemme, VersioOne Enterprise sai korkeimman yhteispistemäärän (262) vertailtavista ALM-järjestelmistä. VersionOne Enterprise sai myöskin korkeimmat pisteet kahdesta tärkeimmästä osa-alueesta, jotka ovat helppo käyttöönotto (7) sekä räätälöinti-mahdollisuus (9). Käyttöönoton helppoudesta VersionOne sai seitsemän pistettä, koska heiltä on saatavilla esittelyvideo, jossa perustoiminnallisuudet selitetään erittäin yksinkertaisesti. Team Foundation Serveriltä ja Borland StarTeamilta, ei löytynyt tällaista opastusta. VersionOne:n ALM-järjestelmissä on mahdollista muokata lähes kaikki otsikot sekä kenttien nimet itselleen sopiviksi. Mikäli yrityksellä on jo valmiita nimityksiä tietyistä prosesseista, niiden integroiminen VersionOnen ALM-järjestelmään on erittäin yksinkertaista. Kaikilla toimijoilla oli saatavilla kattavat tukipalvelut. Borland StarTeam sai yhden pisteen vähemmän kuin VersionOne sekä Team Foundation Server, koska yrityksestä ei koskaan vastattu lähettämääni sähköpostiviestiin.

Team Foundation Server on ensisijaisesti suunniteltu huomattavasti suuremmalle käyttäjäjoukolle, kuin mitä CD-FI:ssä on tarpeen. Tämä on johtanut siihen, että Team Foundation Server on ajoittain todella raskas ja siitä löytyy paljon enemmän toimintoja kuin mitä CD-FI:n kokoinen käyttäjäyhteisö tarvitsee. Tämän johdosta Team Foundation Server sai ylläpito osuudesta yhden pisteen vähemmän kuin VersioOne sekä Borland, joiden ominaisuudet sopivat hyvin pienemmän käyttäjäryhmän tarpeisiin.

Kaikki vertailtavat järjestelmät tukevat täydellisesti Scrum menetelmiä. Vaikka VersionOne on lähtökohtaisesti suunnattu tukemaan Scrumia, ei se mielestäni pysty tuottamaan niin suurta lisäarvoa kilpailijoihinsa nähden, että sen olisi voinut pisteyttää muita korkeammalle.

Taulukko 5 Operationaalisten tavoitteiden pisteet: VersionOne Enterprise

Tavoite	**Painoarvo**	**Toteutus**	**Painotettu tulos**
Helppo käyttöönotto	9	7	63
Räätälöinti-mahdollisuus	8	9	72
Kattavat tukipalvelut	6	8	48
Ylläpidon helppous	7	7	49
Scrum-tuki	3	10	30
Yhteensä:			262

Taulukko 6 Operationaalisten tavoitteiden pisteet: Microsoft Team Foundation Server Premium

Tavoite	**Painoarvo**	**Toteutus**	**Painotettu tulos**
Helppo käyttöönotto	9	6	54
Räätälöinti-mahdollisuus	8	4	32
Kattavat tukipalvelut	6	8	48
Ylläpidon helppous	7	5	35
Scrum-tuki	3	10	30
Yhteensä:			199

Taulukko 7 Operationaalisten tavoitteiden pisteet: Borland StarTeam Enterprise

Tavoite	Painoarvo	Toteutus	Painotettu tulos
Helppo käyttöönotto	9	6	54
Räätälöinti-mahdollisuus	8	7	56
Kattavat tukipalvelut	6	7	42
Ylläpidon helppous	7	7	49
Scrum-tuki	3	10	30
Yhteensä:			231

Kepner Tregoe matriisin mukaisesti seuraava vaihe on harkita mahdollisia ongelmia tai negatiivisia vaikutuksia mitä jokaisen ALM-järjestelmän käyttöönotolla on (decision-making-confidence Internet-sivut). Nämä ongelmat tulee pisteyttää todennäköisyyden sekä ongelman vakavuuden perusteella. Tätä pistemäärää verrataan operatiivisten tavoitteiden vertailussa saatuun pistemäärään ja korkeimman pistemäärän saavuttanut järjestelmä sopii parhaiten vaatimuksiin.

Taulukoissa 8,9 ja 10 on eritelty jokaisen vertailtavan järjestelmän saamat pisteet toiminnan kannalta negatiivisista asioista. Kun yrityksessä otetaan käyttöön uusia järjestelmiä, on selvää, että niiden toiminnallisuuksien opetteleminen vie työntekijän aikaa niin sanotusti varsinaisen työn tekemiseltä. Vaikka vertailtavien järjestelmien käyttöönoton helppoudessa on pieniä eroja, ne eivät kuitenkaan heijastu käyttöönotosta syntyviin haittoihin, vaan kaikki järjestelmät saivat samat pisteet tästä kohdasta.

Negatiivisten vaikutusten toinen kohta, mitä jos järjestelmää pyörittävä palvelin katuu, on mielestäni yhtä todennäköistä kaikissa tapauksissa, sillä tämä on enemmänkin käytössä olevasta palvelimesta kiinni kuin siinä pyörivästä ALM-järjestelmästä. Vaikka Team Foundation Server kuluttaakin enemmän palvelimen resursseja niin se ei ole riittävän paljon nostamaan todennäköisyyttä koko palvelimen kaatumiselle.

Taulukko 8 Haitallisten vaikutusten pisteet: VersionOne Enterprise

Haitallinen vaikutus	**Todennäkäisyys**	**Vakavuusaste**	**Painotettu tulos**
Uuden järjestelmän opetteleminen vie työntekijöiden aikaa varsinaisilta töiltä	10	2	20
Järjestelmää pyörittävä palvelin kaatuu	3	6	18
Järjestelmä kuluttaa tarpeettoman määrän palvelimen resursseja	4	6	26
Yhteensä:			64

Taulukko 9 Haitallisten vaikutusten pisteet: Microsoft Team Foundation Server Premium

Haitallinen vaikutus	**Todennäkäisyys**	**Vakavuusaste**	**Painotettu tulos**
Uuden järjestelmän opetteleminen vie työntekijöiden aikaa varsinaisilta töiltä	10	2	20
Järjestelmää pyörittävä palvelin kaatuu	3	6	18
Järjestelmä kuluttaa tarpeettoman määrän palvelimen resursseja	8	6	48
Yhteensä:			86

Taulukko 10 Haitallisten vaikutusten pisteet: Borland StarTeam Enterprise

Haitallinen vaikutus	**Todennäkäisyys**	**Vakavuusaste**	**Painotettu tulos**
Uuden järjestelmän opetteleminen vie työntekijöiden aikaa varsinaisilta töiltä	10	2	20
Järjestelmää pyörittävä palvelin kaatuu	3	6	18
Järjestelmä kuluttaa tarpeettoman määrän palvelimen resursseja	3	6	18
Yhteensä:			56

Taulukossa 11 pisteet on laskettu yhteen niin, että operatiivisten tavoitteiden pisteistä on vähennetty haitallisten vaikutusten pisteet. Vaikka Borland StarTeam Enterprisen haitallisten vaikutusten pisteet olivat pienemmät kuin VersionOne Enterprise järjestelmän, se jäi vertailussa kuitenkin sijalle kaksi. VersionOne Enterprise on vertailun mukaan paras vaihtoehto CD-FI:n vaatimuksiin, sillä sen yhteispisteet olivat 198, kun Borland StarTeam Enterprisen ylsi 175 pisteeseen. Microsoftin Team Foundation Server jäi tässä vertailussa sijalle kolme 113 pisteellä. Suurin syy Team Foundation Serverin alhaiseen pistemäärään on järjestelmän raskas käyttö pienillä käyttäjämäärillä. VersionOne Enterprisen vahvuus oli sen ylivertainen muokattavuus sekä käyttöönoton helppous.

Taulukko 11 Yhteenveto Kepner Tregoe-matriisin mukaisista pisteistä

ALM-järjestelmä	Operatiivisten tavoitteiden pisteet	Haitallisten vaikutusten pisteet	Yhteensä
VersionOne Enterprise	262	64	198
Microsoft Team Foundation Server Premium	199	86	113
Borland StarTeam Enterprise	231	56	175

9 VersionOne

Tässä luvussa esitellään kyseessä olevaan ympäristöön valittu ALM-ratkaisu. Lisäksi kerrotaan myös hieman järjestelmän asennuksesta sekä miten ALM-järjestelmään saadaan yhdistettyä Microsoftin Visual Studio ja versionhallintaohjelma TortoiseSVN. Lopuksi selvennetään myös hieman ALM-järjestelmän perustoiminnallisuutta.

VersionOne on The Agile Management Companyn vuonna 2002 julkaisema projektinhallinta-ohjelmisto. VersionOne:n avulla on helppoa ja yksinkertaista hallinnoida niin yksittäisiä projekteja kuin suurempiakin projektikokonaisuuksia, sillä se tarjoaa työkaluja niin projektin johdolle kuin yksittäiselle työntekijällekin. Järjestelmä mahdollistaa usean eri ohjelman tietojen keräämisen yhden käyttöliittymän alle. Tämä yksinkertaistaa projektien hallinnointia, sillä projektijohdon ei tarvitse itse kasata tietoa yhdeksi kokonaisuudeksi.

VersionOne:sta löytyy kolme eritasoista pakettia: Team, Enterprise sekä Ultimate. Paketit eroavat tarjottavien toiminnallisuuksien suhteen, joka heijastuu edelleen hintaan. VersionOne Team edition on siitä mainio versio, että se on ilmainen kymmenelle (10) käyttäjälle, joten pienelle organisaatiolle VersionOne:n käyttö on ilmaista. (VersionOne:n Internet-sivut.)

9.1 Asennus

VersionOne on saatavilla kahdenlaisena asennuspakettina: normaalina, omalle palvelimelle asennettavana palveluna sekä täysin ylläpidettynä palveluna, joka tarjotaan verkossa. Jälkimmäinen verkossa tarjottava palvelu on toimiva pienille yrityksille, koska heillä ei välttämättä ole resursseja hankkia omaa palvelinta, jolla VersionOne:a pyöritettäisiin.

Verkossa tarjottavassa palvelussa kaikki tieto säilytetään VersionOne:n omilla palvelimilla, jonne ollaan yhteydessä Internetin välityksellä. Riskinä tässä on kuitenkin tietoturva. Vaikka yhteys onkin SSL (Secure Sockets Layer)-protokollan mukaan suojattu, niin verkkoliikenne on aina riskialtista. Mikäli VersionOne:lla käsiteltävät projektit ja tieto on osa yrityksen keskeistä liiketoimintaa, ei tiedon tallentaminen oman verkon ulkopuolelle ole välttämättä kovin tietoturvallinen ratkaisu. (VersionOne Internet-sivut.)

Koska asennuspaketteja on kahdenlaisia, niin niillä on varsin erilaiset laitteistovaatimukset. Verkossa tarjottavan palvelun laitteistovaatimukset ovat:

Taulukko 12 VersionOne verkossa tarjottavan palvelun laitteistovaatimukset (VersionOne Internet-sivut.)

Selain	Internet Explorer 7,8;Mozilla Firefox 3.0–3.6;Chrome 4; Safari 4
Käyttöjärjestelmä	Windows (XP, 2003, 2008, 7);Mac OS 10.3+;Linux
Prosessori	1 Ghz
Muisti	512 Mb RAM

Palvelimelle asennettavan version laitteistovaatimukset ovat:

Taulukko 13 VersionOne-palvelimelle asennettavan palvelun laitteistovaatimukset (VersionOne Internet-sivut.)

Selain	Internet Explorer 7,8;Mozilla Firefox 3.0–3.6;Chrome 4; Safari 4
Verkko-/Ohjelmistopalvelin	Internet Information Server (IIS) 6, 7 ASP.Net tulee olla asennettuna
Framework	Microsoft .NET 3.5
Tietokanta	Microsoft SQLServer 2005,2008* kokonaisen tekstin haku komponentti tulee olla asennettuna (full text search) * SQLServer 2008 tulee olla vähintään versio 10.0.1767
Prosessori	2 Ghz
Muisti	2 Gb RAM

Handelsbankenin tapauksessa tietoturva on kriittinen asia, jonka vuoksi tässä työssä suositellaankin, että VersionOne tullaan asentamaan Handelsbankenin omalle palvelimelle fyysisenä asennuksena. Kun asennus tehdään fyysisesti omalle palvelimelle omaan verkkoon, tulee ottaa huomioon, että järjestelmä vaatii muitakin resursseja kuin pelkän palvelimen. Palvelin tarvitsee tietenkin normaalin ylläpidon sekä järjestelmän ympärille rakennetut tietoliikenneyhteydet. Palvelimelle asennettavan VersionOne:n asennus on erittäin yksinkertaista. VersionOne:n Internet-sivuilta ladataan haluttu versio palvelimelle, sitten ajetaan ladattu exe-tiedosto ja noudatetaan ohjeita. Asennukseen kuluu aikaa noin 5 minuuttia ja mahdollisiin päivityksiin on hyvä varata saman verran aikaa (VersionOne:n Internet-sivut).

9.2 Perustoiminnallisuus

Kun VersionOne on asennettu palvelimelle ja sen käyttö aloitetaan, niin itse ohjelma antaa hyvät ohjeet alkuun pääsemisestä. Järjestelmään luodaan projekteja sekä käyttäjiä, joille voidaan antaa erilaisia määrityksiä sekä rooleja tarpeen mukaan. Käyttäjiä ja projekteja on helppo jakaa ryhmiin, millä estetään turhan tiedon näyttäminen käyttäjälle. Yksi VersionOne:n parhaista toiminnallisuuksista onkin sen tarjoamat ilmaiset yhteyspalikat eri kolmannen osapuolten ohjelmien yhdistämiseen. Tämä mahdollistaa sen, että vaikka yrityksellä olisi käytössään toiminnallisuuksiltaan riisutuin Team Edition, niin näitä vapaan lähdekoodin yhteyspalikoita käyttämällä voi yritys rakentaa itselleen varsin kattavan ympäristön. Tämän opinnäytetyön aiheena olevassa ympäristössä tärkeimmän yhteyspalikat ovat lisäosat, jotka yhdistävät TortoiseSVNn sekä Microsoft Visual Studion VersionOne:n kanssa.

9.3 VersionOne:n, Visual Studio:n sekä TortoiseSVN:n yhdistäminen

VersionOne:n yhdistäminen Visual Studio:n sekä TortoiseSVN:n kanssa on äärimmäisen yksinkertaista. Ohjelmien yhdistämiseen tarvitaan palvelimelle asennettu VersionOne, johon käyttäjät saavat yhteyden. Jokaisen käyttäjän koneella tulee olla asennettuna Visual Studio, TortoiseSVN sekä molempien ohjelmien yhdistämisohjelmat. Käyttäjän koneille vaadittavat ohjelmat voidaan ajaa etäasennuksena tai mikäli käyttäjiä on vähän, voidaan asennukset tehdä suoraan käyttäjien koneilta. Vaikka VersionOne:a ei olisi vielä ympäristöön asennettu, voidaan yhdistysohjelmat ajaa valmiiksi. Yhdistysohjelma lisää yhdistettäviin ohjelmiin valikon VersionOne:lle, jonne lisämääritykset syötetään. Lisämäärityksillä tarkoitan VersionOne:n verkko-osoitetta sekä käyttäjätunnusta, jolla VersionOne:een otetaan yhteys.

Vaikka VersionOne:a, Visual Studio:ta sekä TortoiseSVN:ia voidaan käyttää ympäristön hyväksi myös ilman niiden yhdistämistä toisiinsa, tuo ohjelmien yhdistäminen suurta hyötyä käyttäjälle. Visual Studion tapauksessa kehittäjä näkee suoraan Visual Studiosta projektikohtaista tietoa, kuten vikailmoitukset tai työtilaukset. Kehittäjä voi myös päivittää kaikkia projektiin liittyviä tietoja Visual Studion kautta. Näin kehittäjän ei tarvitse liikkua usean eri ohjelman välillä päivittämässä tekemisiään.

10 Ympäristön muut toimijat

Tässä luvussa esitellään eri ohjelmia, jotka yhdistämällä saadaan aikaan kattava ohjelmointiympäristö. Lukuunottamatta ympäristön ALM-järjestelmää, jota käsitellään seuraavassa luvussa. Ohjelmien valinnassa on otettu huomioon CD-FI:n-ohjelmointitiimin koko, sekä ohjelmien käytön opettelun ja ylläpidon helppous.

Ohjelmointiympäristön perustana on Microsoftin Visual Studio, joka on rakennettu .Net-ohjelmointia varten. Ympäristön toiminnoissa ja toimijoiden valinnassa on ollutkin tärkeimpänä kriteerinä se, että ne osaavat kommunikoida Visual Studion kanssa.

10.1 Versionhallinta

Versionhallinta on pitkään ollut kriittinen työkalu ohjelmoijille, jotka tyypillisesti viettävät aikansa tehden pieniä muutoksia ohjelmiin ja sitten peruen tai tarkastaen joitakin noista muutoksista seuraavana päivänä. Näin tapahtuu kun ohjelmoijia on useita ja he saattavat päivittäin vaihdella ohjelmaa jota kehittävät (Küng, Onken, Large 2010, 1). Versionhallinnan tehtävänä on varmistaa, että kaikilla on uusin ohjelmakoodi käytössään sekä helpottaa vertaamista edellisiin versioihin ja tarvittaessa palauttaa niitä. Tämän vuoksi versionhallintaa voidaan kutsua ohjelmoijan aikakoneeksi (Collins-Sussmann, Fitzpatrick, Pilato 2008, xix).

Toimivan versionhallinnan toteuttamiseen käytämme Apache Subversion- nimistä versionhallintajärjestelmää sekä TortoiseSVN-ohjelmaa, joka toimii käyttöliittymänä versionhallintaan. Molemmat ohjelmat ovat ilmaisia avoimen lähdekoodin ohjelmia. Apache Subversion:n käyttämä lisenssi on Apache License ja TortoiseSVN:n käyttämä The GNU General Public License. Nämä lisenssit mahdollistavat molempien ohjelmien täyden muokkaamisen käyttäjän omiin tarpeisiin aina lähdekoodin muokkaamisesta lähtien. Molemmat ohjelmat on tosin rakennettu niin joustaviksi, että niiden muokkaaminen tuskin tulee olemaan tarpeellista.

10.1.1 Apache Subversion

Jotkin versionhallintajärjestelmät ovat myös Software Configuration Management (SCM)-järjestelmiä. SCM-järjestelmät on rakennettu niin, että ne ymmärtävät ohjelmointikieltä ja osaavat rakentaa tiedostopuurakenteita lähdekoodista sekä jopa tarjoavat työkaluja ohjelmien rakentamiselle. Vaarana tässä on kuitenkin se, että SCM-järjestelmä ei tee mitään asiaa täydellisesti. Apache Subversion ei edes yritä olla mitään muuta kuin versionhallintajärjestelmä. Apache Subversion ei myöskään ota kantaa siihen minkä tyyppistä dataa se hallinnoi. (Collins-Sussmann ym. 2008, xix.)

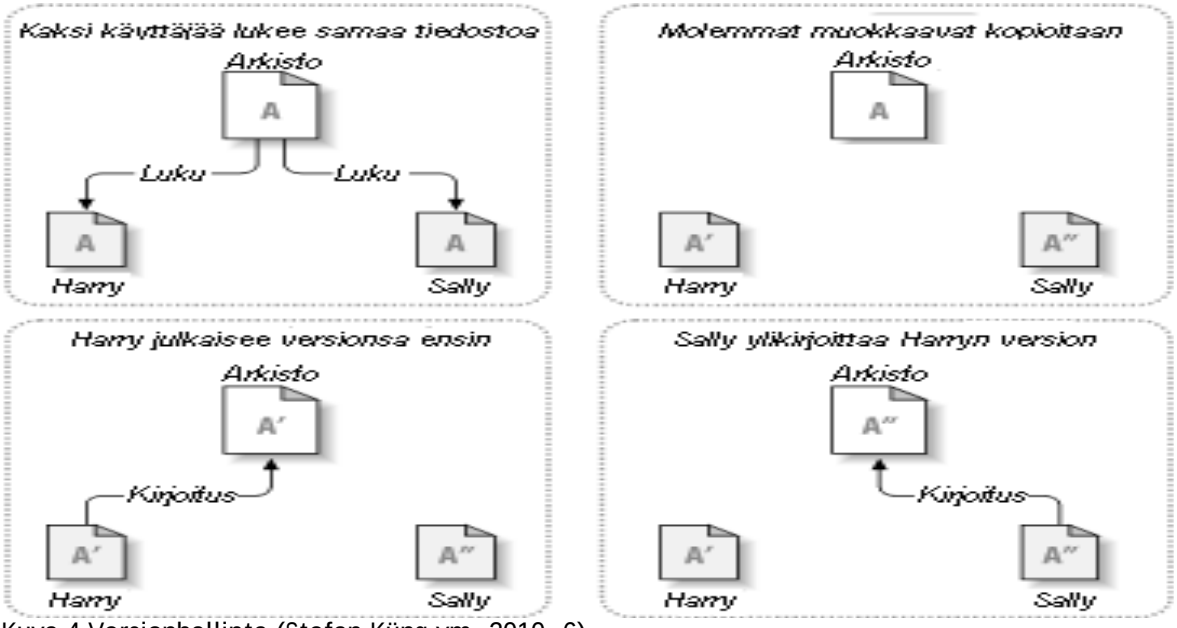
Tämä yksinkertaistettu toimintamalli tekee Apache Subversion:sta ehdottomasti parhaan valinnan, sillä sen käyttöönotto on todella yksinkertaista ja helppoa.

Alla lista Apache Subversion:in tärkeimmistä toiminnoista.
Taulukko 14 (Küng ym. 2010, 2)

Toiminto	Selitys
Versioituvat hakemistot	Apache Subversion toteuttaa ”virtuaalisen” versioidun tiedostojärjestelmän, joka tallettaa koko kansiopuun muutokset ajan kuluessa.
Atomiset toimitukset	Toimitus arkistoon onnistuu joko täydellisesti tai ei ollenkaan. Tämän ansiosta muutokset voidaan tehdä ja toimittaa loogisina kokonaisuuksina
Versioituvat metatiedot	Jokaisella tiedostolla ja hakemistolla on näkymätön joukko ”ominaisuuksia”. Voit laajentaa joukkoa mielivaltaisilla nimi/arvo -pareilla. Ominaisuudet versioituvat aivan kuin tiedostojen sisällöt.
Useita verkkokerroksia arkistoon kytkeytymiseksi	Apache Subversion-järjestelmässä arkistoon kytkeytyminen on irrotettu omaksi rajapinnakseen, mikä helpottaa uusien kytkeytymistapojen kehittämistä.Apache Subversionin monipuolisin palvelinvaihtoehto on toteutettu Apache-verkkopalvelimen modulina, joka ymmärtää HTTP-protokollan WebDAV/ DeltaV -nimistä muunnelmaa. Tämä tekee Apache Subversionista vakaamman ja parantaa sen yhteiskäytettävyyttä, sekä tarjoaa myös useita keskeisiä ominaisuuksia ilmaiseksi (esim. tunnistaminen, valtuuttaminen, pakkaus ja arkiston selaus). Apache Subversion-palvelimesta on tarjolla myös kevyempi, itsenäisesti toimiva vaihtoehto. Tämä vaihtoehto noudattaa omaa

	yhteyskäytäntöään, joka voidaan helposti tunneloida ssh-yhteyden yli.
Johdonmukainen tiedonkäsittely	Apache Subversion esittää tiedostojen eroavuudet käyttäen binääristä erottelualgoritmia, joka toimii identtisesti sekä tekstiä (ihmisen luettavissa) että binääristä dataa (ei ihmisen luettavissa) sisältäville tiedostoille. Molemmat tiedostotyypit tallennetaan samaan tapaan pakattuina arkistossa, ja vain erot siirretään kumpaankin suuntaan verkkoyhteyksien ylitse.
Tehokas haarautuminen ja merkintöjen teko	Haarojen luonnin (engl. branching) ja versioiden merkitsemisen (engl. tagging) kustannusten ei tarvitse olla suhteessa projektin kokoon. Apache Subversion luo haarat ja merkit yksinkertaisesti kopioimalla projektin linkittämällä. Tästä johtuen nämä toiminnot vievät vain vähän aikaa (vakioajan) ja varaavatvain vähän tilaa arkistosta.
Muunneltavuus	Apache Subversionilla ei ole historian painolastia, se on toteutettu joukkona jaettuja C-kirjastoja, joilla on selkeät ohjelmointirajapinnat. Tämä tekee Apache Subversionista erittäin ylläpidettävän ja käyttökelpoisen muille sovelluksille ja kielille.
Tiedon eheys	Lukitse/muuta/vapauta sekä kopio/muokkaa/yhdistä – ratkaisut ovat tuettuja.

Versionhallinnasta puhuttaessa esiin tulee aina sama ongelma. Miten estetään saman tiedoston muokkaaminen samaan aikaan eri käyttäjien osalta. Kuvassa 3 kuvataan kuinka kaksi käyttäjää, Harry ja Sally, muokkaavat samaa tiedostoa samaan aikaan. Harry tallentaa tiedostonsa ensin ja Sally tallentaa oman tiedostonsa Harryn jälkeen. Vaikka Harryn tiedosto on tallessa, niin Harryn tekemät muokkaukset eivät ole mukana Sallyn tallentamassa tiedostossa.



Kuva 4 Versionhallinta (Stefan Küng ym. 2010, 6)

Edellä kuvatun ongelman välttämiseksi monet versionhallintajärjestelmät käyttävät lukitse/muuta/vapauta – ratkaisua. Käytännössä tämä tarkoittaa sitä, että ensin käyttäjä lukitsee tiedoston, jota rupeaa muokkaamaan. Kun tiedosto on lukittu, niin muut käyttäjät eivät pääse kyseessä olevaa tiedostoa muokkaamaan. Tämä malli on kuitenkin hyvin rajoittava ja osoittautuu usein pullonkaulaksi käyttäjille. Lukitse/muuta/vapauta – ratkaisussa on mahdollista, että käyttäjä A lukitsee tiedoston, mutta unohtaa vapauttaa sen lomalle lähtiessään. Mikäli käyttäjä B tarvitsee kyseessä olevaa tiedostoa, tulee hänen pyytää ylläpitäjää vapauttamaan tiedosto. Tämä aiheuttaa tarpeetonta viivettä ja ajanhukkaa. Ajoittain tämä malli myös rajoittaa työntekoa turhaan. Esimerkiksi työntekijä A:lla tulee tarve muokata teksti tiedoston alkua ja työntekijä B:n saman tiedoston loppua. Muutokset eivät ole päällekkäisiä, joten he voisivat muokata tiedostoa yhtä aikaa. On myös mahdollista että työntekijät A ja B muokkaavat eri tiedostoja, jotka kuitenkin ovat toisistaan riippuvaisia. Kun molemmat työntekijät ovat tehneet muutokset, huomataan että tiedostot eivät enää toimikaan yhdessä.

Apache Subversionin ratkaisu kuvassa 1 olevaan ongelmaan on niin sanottu kopio/muokkaa/yhdistä – ratkaisu. Tämä malli perustuu siihen, että käyttäjällä on omalla työasemallaan työkopio tiedostosta, jota hän muokkaa. Tällöin käyttäjät voivat muokata omilla työasemillaan samaa tiedostoa. Kun muutokset ovat valmiita, niin kopiot yhdistetään lopulliseksi versioksi.

Esimerkki 1: Työntekijä A on tallentamassa muuttamaansa tiedostoa versionhallintaan, mutta työntekijä B on tallentanut samasta tiedostosta uuden version versionhallintaan sen jälkeen kun työntekijä A on ladannut oman työkopionsa. Tällöin versionhallinta ilmoittaa työntekijä

A:lle, että hänen työkopionsa on vanhentunut. Tässä tapauksessa työntekijä A valitsee seuraavassa kappaleessa esiteltävästä TortoiseSVN-asiakasohjelmasta toiminnon, joka yhdistää arkistosta löytyvän kopion omaan työkopioonsa. Mikäli työntekijät A ja B ovat muokanneet eri kohtia tiedostoista, niin yhdistäminen onnistuu automaattisesti. Mikäli muutoksia on kuitenkin tehty päällekkäin, niin TortoiseSVN ilmoittaa tästä ristiriidasta. Nämä ristiriidat täytyy ihmisen ratkoa, sillä asiakasohjelma ei voi automaattisesti ratkoa ristiriitoja.

Tiivistettynä kopio/muokkaa/yhdistä – ratkaisu mahdollistaa sen, että työntekijöiden ei tarvitse koskaan odotella toisiaan. Ristiriitojen ratkomiseen tarvittava aika on usein huomattavasti lyhyempi kuin lukitusjärjestelmän käytössä hukattu aika.
Lukitse/muuta/vapauta – ratkaisu on parempi silloin kun kyseessä olevat tiedostot ovat sellaisia joita ei voi yhdistää, kuten kuvatiedostot. Kuten Taulukossa 1 todettiin, niin Apache Subversion tukee molempia malleja, joten tarvittaessa myös tiedoston lukitusta voidaan käyttää. (Küng ym. 2010, 7-9.)

10.1.2 TortoiseSVN

TortoiseSVN on asiakassovellus Apache Subversion-versionhallintajärjestelmälle, joka integroituu täysin Windowsin resurssienhallintaan. Tämä tekee TortoiseSVN:n käyttöönottokynnyksen matalaksi, koska käyttäjän ei tarvitse vaihtaa eri sovellukseen versionhallintaa käyttääkseen. TortoiseSVN-kontekstivalikot toimivat monissa muissa tiedostotyökaluissa, kuten myös avaa tiedosto-ikkunassa, jota käytetään useimmissa Windows-sovelluksissa. TortoiseSVN on kuitenkin ensisijaisesti rakennettu Windowsin resurssienhallinnan laajennukseksi, joten on mahdollista, että muissa sovelluksissa integrointi ei ole täydellinen.

TortoiseSVN lisää resurssienhallinnan kontekstivalikkoon oman alavalikkonsa, josta kaikki tarvittavat Apache Subversion komennot ovat käytettävissä. TortoiseSVN käyttää myös kuvakepäällyksiä, joilla ohjelma ilmoittaa versionhallinnassa olevan tiedoston tai kansion tilan, kun verrataan kehittäjän koneella olevaa sekä versionhallinnan säilytyspaikassa olevaa. Näin kehittäjä näkee yhdellä vilkaisulla, onko jostain tiedostosta saatavilla uudempi versio tai onko hänen muokkaamansa tiedosto vielä lisäämättä versionhallintaan. (Küng ym. 2010, 1.)

10.2 Atlassian Confulence Document wiki & Atlassian JIRA

Haastattelussa ilmeni myös tarve sille, että ympäristön tulisi tukea dokumentaatiota sekä virheiden hallintaa. Seuraavaksi esitellään lyhyesti kaksi ohjelmaa, jotka hallitsevat nämä tarpeet ja ovat täysin yhteensopivia VersionOne:n kanssa.

Molemmat ohjelmat ovat Atlassian- nimisen yrityksen rakentamia. Näistä Confulence on keskittynyt dokumentaation hallintaan sekä jakamiseen ja JIRA on virheidenhallinnan työkalu. Koska toisessa CD-FI-projektissa ollaan jo kyseisiä ohjelmia määrittelemässä, niin tässä opinnäytetyössä niitä ei tarkemmin kuvata. VersionOne-järjestelmä tukee täysin niin Confulencea kuin JIRA:a, mikä tarkoittaa sitä, että heidän Internet-sivuiltaan on ladattavissa valmis yhdistäjä ohjelmien yhdistämiseksi VersionOne:n kanssa.

11 Kehitys-, testi- ja tuotantoympäristö

Tässä luvussa käydään läpi kehitys-, testi- ja tuotantoympäristöjen vaatimuksia sekä määrityksiä joiden avulla ympäristöistä saadaan mahdollisimman tehokkaita sekä yksinkertaisia. Lisäksi kerrotaan myös miten versionhallinnan tiedostopuu tulisi rakentaa kehitysympäristöön, joka noudattaa Apache Subversion- määrityksiä.

On tärkeää, että ohjelmiston kehittäjillä sekä testaajilla on hyvät, selkeät ja toisistaan erotellut alueet ohjelmointiympäristön sisällä. Nämä alueet ovat jo itsessään ympäristöjä, koska ne sijaitsevat omilla palvelimillaan ja niistä jokaisesta löytyvät omat ohjelmistot, jotka pyörittävät ympäristön toimintoja. Kun ympäristöt yhdistetään, saadaan aikaan toimiva, kattava ja turvallinen toimintaympäristö ohjelmistokehitykselle. Tärkeää onkin luoda mahdollisimman yksinkertainen, joustava ja samalla turvallinen tapa siirtää ohjelmia kehitysympäristöstä testiympäristöön ja sieltä aina tuotantoon asti.

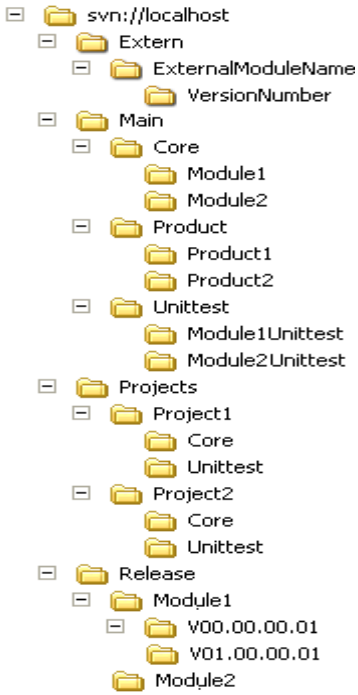
Seuraavaksi kuvataan lyhyesti jokaisen ympäristön toimintaperiaatteet sekä ratkaisu miten ympäristöt saadaan yhdistettyä yhdeksi isoksi toimijaksi.

11.1 Kehitysympäristö

Ohjelmistokehityksessä kehitysympäristön muodostavat tarvittavat prosessit ja ohjelmointityökalut, joita ohjelmiston tekemiseen tarvitaan (TechTargetin Internet-sivut). Kehitysympäristön tarkoituksena on luoda ohjelmistonkehittäjille hiekkalaatikkoympäristö. On tärkeää, että kehitysympäristö on pystytetty omalle palvelimelleen, sillä milloin tahansa voi olla tarpeellista käynnistää koko ympäristö uudestaan ja tästä johtuvan työskentelykatkon tulisi häiritä mahdollisimman vähän kehittäjien, testaajien sekä käyttäjien työaikaa.

Kehitysympäristön tehokas toiminta ja tiedostojen jakaminen toteutetaan jo aikaisemmin mainitulla Apache Subversion:lla ja tämän TortoiseSVN-asiakasohjelmalla.

Kuvassa 5 on esimerkki kehitysympäristön tiedostopuun rakenteesta, joka tukee Apache Subversionin dokumentaatiota.



Kuva 5 Versionhallintatiedostopuu (Desmedt 2006)

Tiedostopuun malli perustuu Apache Subversion:n dokumentaatioon. Avaan tämän rakennetta seuraavaksi.

Versionhallinnan talletuspaikka (repository) jaetaan neljään kansiotasoon:
1. Pää (Main): Apache Subversionin dokumentaatio kutsuu tätä nimellä ”Trunk”
2. Projekti (Project): Kansioon tehdään alikansioita projekteittain ja säilytetään kunkin projektin lähdekoodeja.
3. Julkaisu (Release): Kansiossa säilytetään julkaistujen ohjelmien lähdekoodeja. Subversion käyttää kansiosta nimeä ”Tags” .
4. Ulkoinen (Extern): Täällä säilytetään ulkopuolisia kirjastoja, jotka ovat käytössä.
(Desmedt, 2006.)

- Pää (Main)

Apache Subversionin dokumentaatio kutsuu kansiota nimellä ”Trunk”. Main-kansio sisältää kolme alikansiota, jotka ovat ”Unittest”(Yksikkötesti), ”Core” (Ydin) ja ”Product”(Tuote).

”Core” kansio sisältää kehitettävät moduulit. Moduulilla tarkoitetaan lähdekoodien joukkoa, joka yhdistyy yhdeksi tiedostoksi tyyppiä exe tai dll. ”Product”-kansio sisältää alikansion jokaiselle olemassa olevalle tuotteelle. Se sisältää Visual Studion solution-tiedostoja, joista

selviää mitä mistä komponenteista tuote koostuu, sekä projektin (buildscript) rakennusskriptiä. ”Unittest”-kansio sisältää samat moduulikansiot kuin ”Core”-kansio. Eroavaisuutena on se, että moduuleilla on lisänimenä ”unittest”, joka selventää, että moduulit ovat testimoduuleita. (Desmedt, 2006.)

- Projekti (Projects)

”Projects”-kansio sisältää alikansion jokaiselle projektille. Jokaisen projektin alikansiot sisältävät samanlaisen rakenteen kuin ”Main”-kansio. On kolme syytä miksi rakennamme kaksi kansiolinjaa, joiden rakenne on melkein identtinen. Kokonaisuuden hallinta on huomattavasti helpompaa, kun jokaisella projektilla on oma kansio. Mikäli projektin tarkoituksena on luoda uusi ominaisuus jo olemassa olevaan tuotteeseen, niin projektin kansiosta löytyy kaikki moduulit, jotka tekevät kyseessä olevan tuotteen. Mikäli projektin lopputuloksena on täysin uusi tuote, sisältää kansio jo olemassa olevat moduulit, jota uusi tuote käyttää. Samalla kansio sisältää myös mahdolliset uudet moduulit, joita ei ole vielä lisätty ”Main”-kansioon. (Desmedt, 2006.)

- Julkaisu (Release)

Kansio sisältää alikansiot jokaiselle moduulille. Näiden moduulikansioiden alikansioina on julkaisukansiot, jotka on nimetty julkaisutunnuksella nämä sisältävät julkaisuun liittyvät lähdekoodit. (Desmedt, 2006.)

- Ulkoinen (Extern)

”Extern”-kansio on rakenteeltaan ja tarkoitukseltaan samanlainen kuin ”Release”-kansio, mutta sen sisältämät moduulit ovat ulkopuolisia moduuleita, jotka ovat käytössä projekteissamme. (Desmedt, 2006.)

Yllä kuvatulla tiedostopuun rakenteella on helppo hallita projekteja. Tällöin myös mahdollisten virhetilanteiden sattuessa on kaikki projektin moduulit sekä lähdekoodit helposti löydettävissä.

11.2 Testiympäristö

Testausympäristö sisältää kaikki tarvittavat ohjelmistot sekä laitteistot, joita kehitettyjen ohjelmistojen testaamiseen tarvitaan. Testiympäristön tulee olla erillään kehitys- ja tuotantoympäristöistä, koska sen on tarkoitus luoda ohjelmien testaajille mahdollisuus tehdä toiminnallisuustestejä rauhassa, ilman ohjelmistonkehityksestä aiheutuvia katkoksia. Testattavana olevat ohjelmat eivät myöskään saa vaikuttaa tuotantoympäristöön. Vaikka on todennäköistä, että CD-FI tulee pärjäämään yhdellä testiympäristöllä, voidaan ympäristöjen eriyttämistä puolustaa toisellakin faktalla. Mikäli jostain syystä CD-FI:n tulisi rakentaa toinen testiympäristö, niin verkkoliikennemallit ovat jo valmiina, koska kahta kehitysympäristöä on turha rakentaa. On myös varsin todennäköistä, että kehittäjät tai testaajat eivät pääse tuotantoympäristössä käyttämään tekemiään ohjelmia. Tämän takia testiympäristön tulisi vastata mahdollisimman hyvin tuotantoympäristöä.

Testausympäristöä on hyvä käyttää myös demotilaisuuksissa sekä muissa ohjelmiston esittelytilaisuuksissa. Tällä vältetään niin sanotun ”demo-efektin” tapahtumista. Demo-efektillä tarkoitetaan toimintoa, joka on saatu valmiiksi juuri ennen demoa, muttei sitä olla täysin päästy testaamaan. Täysin tämä ei tietenkään poista demo-efektiä, sillä aina uusi toiminto voidaan siirtää kehitysympäristöstä testausympäristöön demoa varten. Tämän vahvuus onkin siinä, että se pakottaa siirrosta vastaavan henkilön ottamaan kantaa siirron tärkeyteen ja riskiin jota se tuottaa demoa ajatellen. Liiketoiminnan edustajat, jotka usein toimivat tilaajina IT – projekteissa, katsovat paljon mieluummin demoa, joka etenee katkoitta ja jossa on hieman vähemmän toimintoja, kuin demoa, joka katkeaa ajoittain sen takia että jokin uusi hieno toiminto ei vielä täysin toimi. Eteen tulee varmasti tilanteita, joissa jokin toiminto täytyy ottaa demoon mukaan viime hetkellä, koska näin on määritelty tai tilaajalle luvattu. Tämän tyylisissä tapauksissa vika on kuitenkin muualla kuin ohjelmointiympäristössä.

Kun yrityksellä on käytössään erillinen testiympäristö, on järjestelmän huomattavasti yksinkertaisempaa tuottaa raportteja tehdyistä testeistä tai ainakin sen tuottamat raportit ovat paikkaansa pitävämpiä. Tämä perustuu erillisiin ympäristöihin, koska yleisoletuksena voidaan pitää sitä, että testiympäristössä tapahtuvat asia liittyvät testaukseen.
(Jaideep 2008.)

11.3 Tuotantoympäristö

Tuotantoympäristö on ympäristö, jossa säilytetään ja käsitellään todellista ja oikeaa informaatiota tai siihen on pääsy julkisesta verkosta tai palvelimelta (Michigan Department of Technology, Management and Budget:n Internet-sivut). CD-FI:n tapauksessa julkisella verkolla tarkoitetaan verkkoa, jonne on pääsy muillakin työntekijöillä kuin kehittäjillä tai testaajilla.

Tuotantoympäristön tarkoituksena on luoda niin stabiili ja toimintavarma ympäristö kuin mahdollista, sillä tuotantoympäristössä olevat toiminnat ovat kriittisiä liiketoiminnalle sekä näkyvillä asiakkaille. Tuotantoympäristössä tapahtuvat toiminnalliset virheet tulisi minimoida ja parhaiten se tapahtuu toimivan ja kattavan testausympäristön avulla. Tuotantoympäristöön ei tulisi olla mahdollista siirtää tiedostoja muualta kuin testiympäristöstä. Tällä minimoidaan tilanteet, joissa kehittäjä on siirtämässä ohjelmaa kehitysympäristöstä testiympäristöön, mutta valitseekin epähuomiossa väärän kohdeympäristön.

Kehitys- ja testausympäristöt voivat olla virtuaalisia ja mikäli resursseista on pulaa, voivat ne toimia myös samalla palvelimella. Tuotantoympäristön tulisi sen sijaan sijaita täysin omalla palvelimellaan. Ympäristöjä tulisi myös olla kaksi, jotka ovat käytännössä identtiset, mutta vain toinen olisi käytössä. Kahdennettu tuotantoympäristö on huomattavasti toimintavarmempi, kuin yhden palvelimen varassa oleva ympäristö. Mikäli käytössä oleva palvelin sattuisi kaatumaan, voidaan liikenne ohjata suoraan toiselle palvelimelle. Tällöin loppukäyttäjä ei välttämättä edes huomaa katkosta ja kaatunut palvelin voidaan korjata rauhassa. Suositeltavaa on myös se, että molemmat palvelimet sijaitsevat eri tiloissa ja molemmilla on omat varavirtajärjestelmänsä.

11.4 Yksittäisen ohjelmistokehittäjän ympäristö

Edellä mainittujen ympäristöjen lisäksi on jokaisella kehittäjällä oma, henkilökohtainen ympäristönsä omalla työkoneellaan. Versionhallinnassa on alkuperäiset kopiot jokaisesta kehitettävästä ohjelmistosta, josta jokainen kehittäjä lataa kopiot omalle koneelleen. Näitä ohjelmia kehittäjä kehittää omalla koneellaan ja kun hän on tyytyväinen työhönsä, niin kehittäjä päivittää versionhallinnassa olevan alkuperäiskappaleen omalla kappaleellaan jolloin se tulee kaikkien käytettäväksi. Tämä ei vielä tuo ohjelmistoa kehitysympäristön käyttöön. .Net komponenttien julkaisuun tulemme käyttämään ohjelmaa nimeltä DotNetSiirto.

DotNetSiirto-ohjelmaa ei tulla tässä työssä tarkemmin kuvaamaan, mutta seuraavaksi kuvataan pääpiirteittäin DotNetSiirto-ohjelman toiminnan. DotNetSiirto on C#:lla ohjelmoitu ohjelma, jonka tehtävänä on siirtää Windows-komponentteja ympäristöstä toiseen. Itse

ohjelma on asennettuna käyttäjän tietokoneelle, josta se komponentteja siirrettäessä ajetaan. Ennen kuin DotNetSiirto–ohjelma osaa komponentteja siirtää, tulee sille olla määriteltyinä tiettyjä parametreja. Yksittäisen käyttäjän ei tarvitse näistä määrityksistä kuitenkaan välittää, koska käyttäjä saa kaikki tarvittavat asetukset ladatessaan ohjelman versionhallinnasta TortoiseSVN-ohjelmaa käyttäen.

Kun käyttäjä on asentanut DotNetSiirto-ohjelman koneelleen, käyttäjän ei tarvitse kuin valita siirrettävä komponentti, siirrettävän komponentin ympäristö, josta siirretään sekä kohdeympäristö minne komponentti asennetaan. On tärkeää muistaa, että kun komponentti viedään ensimmäistä kertaa valittuun ympäristöön, tulee käyttäjän käydä kyseisessä ympäristössä määrittelemässä komponentin ympäristökohtaiset muuttujat kuntoon. DotNetSiirto–ohjelma poistaa siis vanhan version kyseessä olevasta komponentista ja asentaa uuden version tilalle, sekä käynnistää komponentin uudestaan.

DotNetSiirto-ohjelman hyviä puolia on se, että ohjelmalla voi valita ympäristöt, joita siirrossa käytetään. Tällä tarkoitan sitä, että esimerkiksi työntekijä saa työtehtävän, että komponentti X version ympäristössä Y ei toimi. Hänen ei tarvitse kuin käynnistää ohjelma, olettaen että käyttöoikeusasiat ovat kunnossa, valita siirrettävän komponentin ympäristö Y ja komponentiksi Y sekä kohde komponentiksi oman koneensa. Tällöin käyttäjä saa omaan työkansioon ongelmallisen version ja hänen on helppoa verrata sitä versionhallinnassa olevaan versioon ja korjata virheet.

11.5 Monta ympäristöä, yksi VersionOne

Kuten edellä mainittiin, niin eri ympäristöjen (kehitys, testi ja tuotanto) on tärkeää olla yhteydessä toisiinsa, mutta fyysisesti erillään. Tämä tarkoittaa sitä, että esimerkiksi kehitysympäristön alasajo ei vaikuta testiympäristöön. Tämä ei kuitenkaan tarkoita, että jokainen ympäristö tarvitsisi oman VersionOne-sovelluksen.

Kuten aikaisemmin mainittiin, niin VersionOne:ssa on mahdollista nimetä käytännössä kaikki haluamallaan tavalla. Helpoin tapa rakentaa yhden VersionOne:n sisälle kolme ympäristöä on aloittaa ohjelmiston käyttö niin, että luo kolme pääprojektia: kehitys, testi ja tuotanto. Näiden päätasojen alle voimme lisäillä myöhemmin oikeita projekteja, esimerkiksi: asiakastietojen päivitys (kehitys), asiakastietojen päivitys (testi) ja asiakastietojen päivitys (tuotanto). Näin jokaiselle vaiheelle voidaan luoda omat virhe-, kehitys-, tehtävä- ja muut tarvittavat listat sekä raportit. Näin kaikkien ympäristöjen pääsyoikeudet on helppo pitää ajan tasalla, koska pääsyoikeuksia voidaan antaa projektikohtaisesti.

Yksi haittapuoli tässä kuitenkin on: kun käytössä on yksi VersionOne, joka sijaitsee yhdellä palvelimella, niin mikäli kyseinen palvelin kaatuu, ei minkään ympäristön VersionOne-projekteihin pääse käsiksi. Tämä ei kuitenkaan haittaa liiketoimintaa, sillä VersionOne ei sisällä suoranaisesti mitään toiminnallista dataa, vaan pääsääntöisesti informaatiota projekteista. Vaikka VersionOne-palvelin olisi alhaalla, se ei vaikuta muiden ohjelmien toimintaan. Muilla ohjelmilla tarkoitan tässä tapauksessa Visual Studiota sekä TortoiseSVN:ää. Mikäli käyttäjä on ladannut projektikohtaista tietoa VersionOne:lta, se näkyy hänen Visual Studiossaan. Käyttäjä voi suoraan päivittää Visual Studioon tekemisensä ja ne viedään VersionOne:n tietokantaan sitten kun yhteys seuraavan kerran saadaan muodostettua.

12 Käyttötapaus

Tässä kappaleessa esitellään käyttötapaus, jossa työntekijä tekee korjauksia jo olemassa olevaan ohjelmaan. Käyttötapauksessa kuvaan miten korjattu versio siirtyy ympäristöstä toiseen ja työvastuu siirtyy kehitysketjun mukaisesti ihmiseltä toiselle. Tällä käyttötapauskuvauksella saa kattavan kuvan ympäristön toiminnasta.

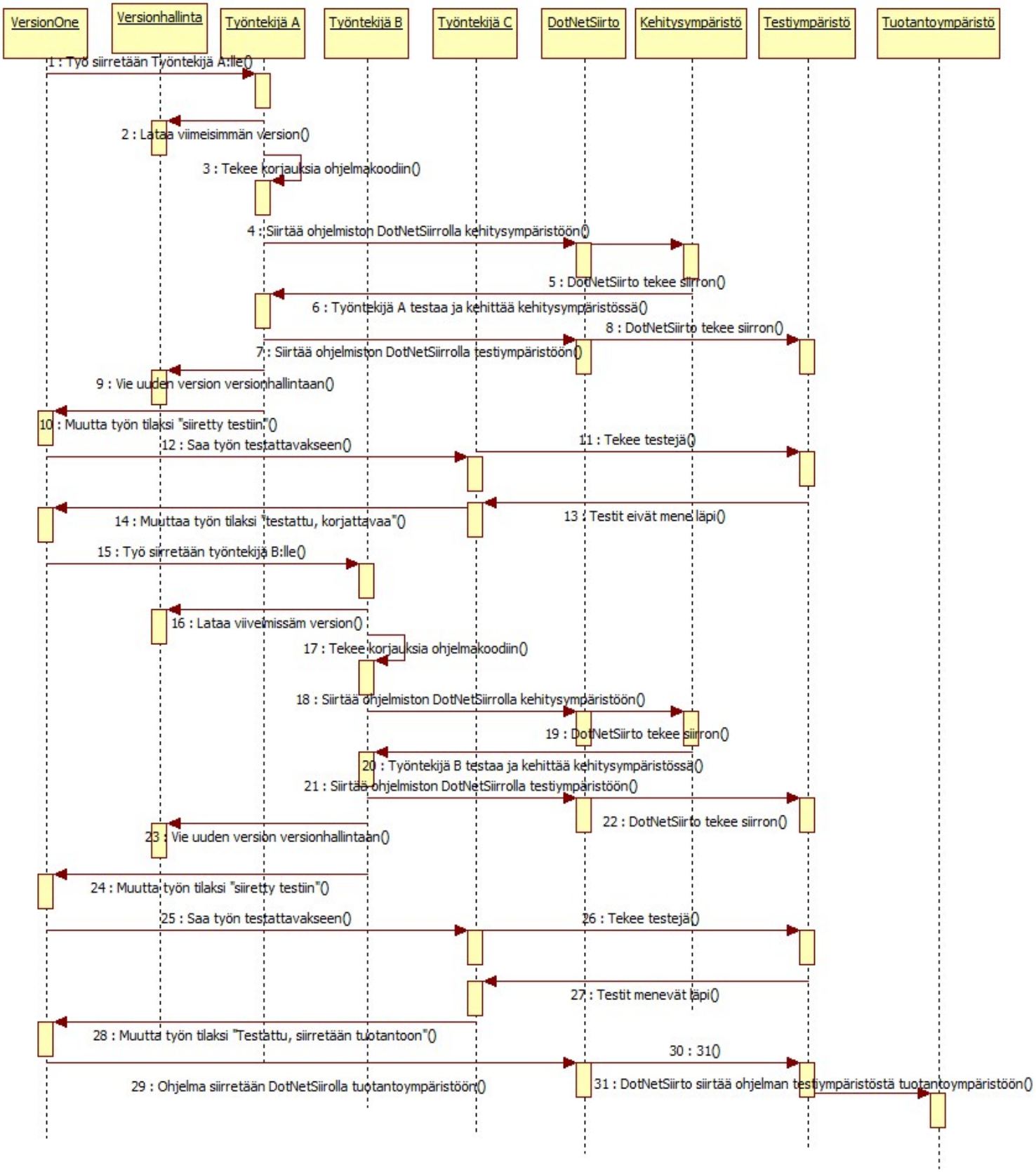
Työntekijä A avaa VersionOne-yhteyden ja huomaa, että hänen työlistalleen on siirretty uusi tehtävä. Tehtävän kuvauksessa ilmoitetaan, että työntekijä B:n tekemästä ohjelmasta on löytynyt jokin vika, joka täytyy korjata. Työntekijä hakee versionhallinnasta omalle koneelleen uusimman version ja avaa Microsoft Visual Studiolla kyseisen ohjelmakoodin. Työntekijä A aloittaa työn korjaukset ja muuttaa työn tilan ”työn alla” olevaksi suoraan Visual Studiosta. Kun tarvittavat korjaukset on saatu valmiiksi, niin työntekijä siirtää tämän uuden version versionhallintaan. Tässä tapauksessa kyseessä on palvelimella ajettava Windows-palvelu, joten työntekijä käyttää DotNetSiirto-nimistä ohjelmaa siirtääkseen kaikki palvelun komponentit oikeaan ympäristöön. DotNetSiirto-ohjelma huolehtii siitä, että palvelimella jo ennestään oleva palvelu sammutetaan, uudet tiedostot kopioidaan ja päivitetty palvelu käynnistetään. Kun työntekijä A on siirtänyt palvelun testiympäristöön hän päivittää työ tilaksi ”siirretty testiin”. Työn ilmestyy Työntekijä C:n työlistalle, koska hänet on määritelty testaajaksi tähän projektiin.

Työntekijä C tekee vaadittavat testit ja toteaa, että yksi tai useampi toiminto ei mene testeistä läpi. Työntekijä C muuttaa työn tilaksi ”Testattu, korjattavaa” ja kirjoittaa lyhyen kuvauksen testeistä, jotka eivät läpäisseet vaatimuksia. Tämän jälkeen työntekijä C voisi siirtää työn takaisin työntekijä A:n työlistalle, mutta koska on sovittu että projektin johtaja jakaa työt hän siirtää työn avoimelle listalle.

Projektin johtaja tekee tässä tapauksessa päätöksen kenelle työ määrätään. Koska työntekijä A on testien aikana aloittanut toisen tehtävän parissa ja työntekijä B:llä on paremmin aikaa työn tekemiseen, siirretään työ hänen listalleen. Työntekijä B saa sähköpostilla ilmoituksen, että hänen listalleen on siirretty uusi työ. Työntekijä B huomaa, että TortoiseSVN ilmoittaa hänelle, että hänen versionsa kyseessä olevasta ohjelman koodista on vanhentunut. Näin työntekijä B osaa ladata uuden version koneelleen ja voi tarkastella mitä muutoksia työntekijä A on ohjelmistokoodiin tehnyt. Näitä muutoksia ja testissä havaittuja puutteita vertailemalla työntekijä B:n on helppo saada kokonaiskuva ohjelman tilanteesta ja muutoksista. Kun työntekijä B on tehnyt korjaukset hän noudattaa samaa kaavaa ohjelman siirrossa testiympäristöön kuin työntekijä A aikaisemmin. Työntekijä C saa taas tiedon, että työ on siirretty testiin ja tällä kertaa testit onnistuvat, joten työntekijä C muuttaa työntilaksi ”Testattu, siirretään tuotantoon”.

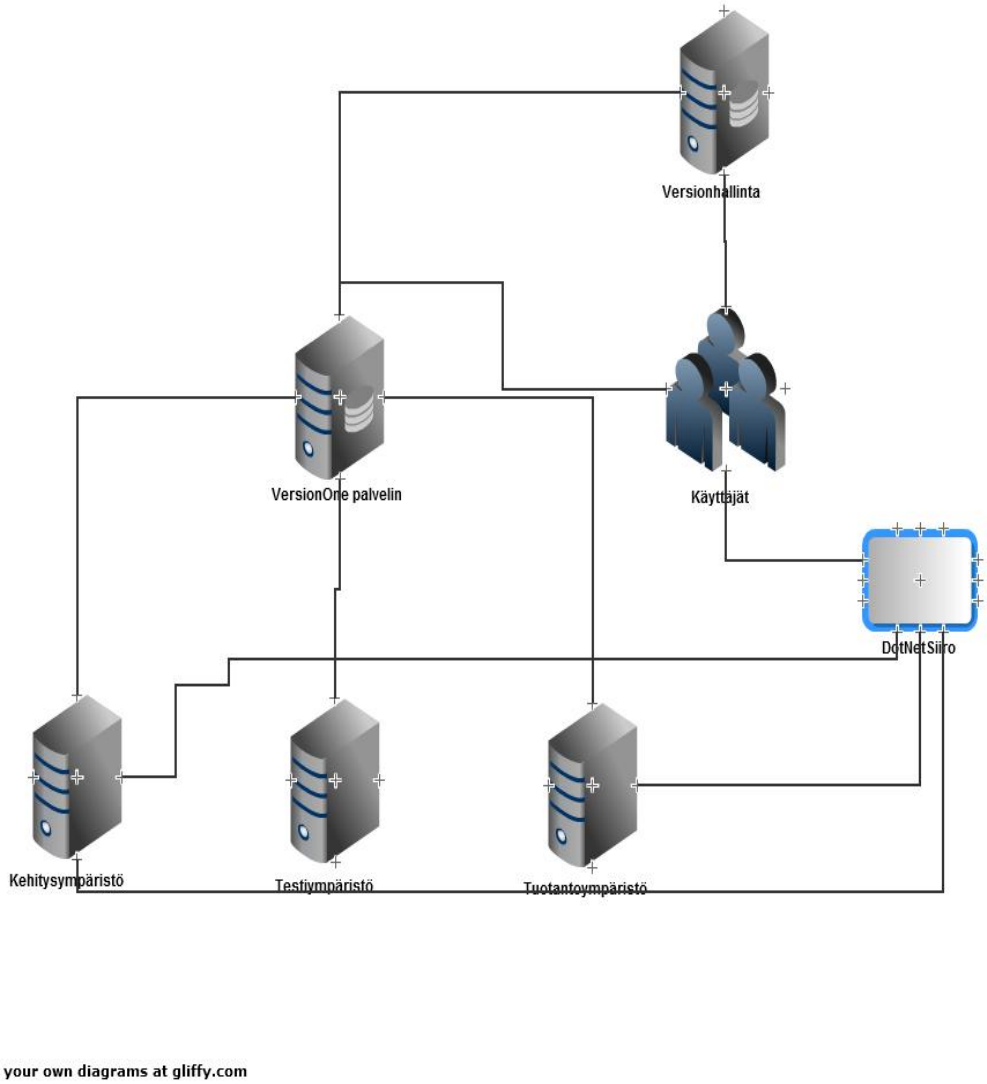
Projektin johtaja saa ilmoituksen, että ohjelma on valmis siirrettäväksi tuotantoon. Vian kriittisyyden ja päivityksen kiireellisyyden perusteella tehdään päätös milloin tuotantoon siirto tapahtuu. Itse tuotantoon siirto tehdään DotNetSiirto-ohjelmalla, tällä kertaa niin, että ohjelmisto siirretään testiympäristöstä tuotantoympäristöön.

Seuraavassa kuvassa on kuvattu sama käyttötapaus sekvenssikaaviona.



Kuva 6 Sekvenssikaavio käyttötapauksesta

Kuvassa 7 on kuvattuna ympäristön väliset yhteydet. Kuvasta on jätetty pois Atlassian Confulence ja JIRA, koska, kuten kappaleessa 8.1.3 mainitsin, kyseiset ohjelmat eivät kuulu tämän opinnäytetyön piiriin vaikka ne ovatkin osa ympäristöä. Kuten kuvata selviää ei käyttäjät ole varsinaisesti suoraan yhteydessä kehitys-, testi- tai tuotantoympäristöön, koska kaikkien ohjelmien siirrot hoidetaan DotNetSiirto-ohjelmalla.



Kuva 7 Ympäristön yhteyksien kuvaus

13 Yhteenveto

Opinnäytetyön tekemistä helpotti huomattavasti se, että kirjoittaja työskentelee Handelsbank CD-FI osastolla, joten yrityksen ympäristö sekä työskentelytavat olivat hänelle entuudestaan tuttuja. Opinnäytetyön tekeminen aloitettiin tutustumalla markkinoilla oleviin ALM-järjestelmiin ja haastattelemalla CD-FI osastopäällikköä, joka toimii myös tämän opinnäytetyön tilaajana.

Haastattelun avulla saatiin vahvistettua niitä ajatuksia mitä kirjoittajalla oli .net-kehitysympäristön vaatimuksista, ottaen huomioon CD-FI:n valmiin ympäristön sekä työntekijät. Haastattelussa selvisi, että ympäristön tulee painottaa käytettävyyden helppoutta, matalaa oppimiskynnystä, hyvää muokattavuutta sekä ympäristön tulee integroitua Microsoftin Visual Studioon.

ALM-järjestelmien erojen selvittämiseen käytetty Kepner Tregoe-päätöksentekomalli helpotti parhaan järjestelmän valitsemista. Kepner Tregoe-mallista löytyi hyvin suoraviivaiset ohjeet sekä määritykset, kuinka malli toimii. Haasteellisinta oli mahdollisten negatiivisten vaikutusten määritteleminen ympäristöjen osalta, mutta näiden pisteytys oli helppoa Kepner Tregoe-mallilla.

Haasteellista työn tekemisessä oli ALM-järjestelmien kartoittaminen. ALM-järjestelmät ovat jääneet, kuten luvussa seitsemän todettiin, ensimmäisten versioiden lunastamattomien lupausten takia epätietoisuuteen. ALM-järjestelmistä on toki kirjoitettu paljon, mutta järjestelmien hyötyjä oli vaikeata hahmottaa näiden kirjoitusten perusteella. Markkinoilla olevia järjestelmiä tutkittaessa oli helppo löytää niiden vahvuudet sekä heikkoudet. VersionOne:n ehdoton vahvuus oli sen helppo integroiminen muihin ohjelmiin, sekä järjestelmän hyvät muokkausmahdollisuudet. Koska Visual Studio on merkittävässä osassa ympäristöä, ajateltiin pysytellä Microsoftin tuoteperheessä ja tutkia Microsoftin ALM-järjestelmää nimeltä Team Foundation Server. Team Foundation Server osoittautui kuitenkin aivan liian raskaaksi järjestelmäksi, sillä se on rakennettu usean sadan käyttäjän ympäristöjä silmällä pitäen. VersionOne osoittaa tässäkin joustavuutensa tarjoamalle erilaisia paketteja erikokoisille asiakkaille, alkaen kymmenelle käyttäjälle ilmaisesta Team Editionista. Team Editionin avulla on yrityksen helppoa tutustua järjestelmään ennen ostopäätöksen tekoa. Jokaisesta VersionOne:n versiosta on saatavilla myös 30 päivän testiversio.

Tämän opinnäytetyön lukemalla jokainen voi varmistua siitä tarvitseeko hänen organisaationsa ALM-järjestelmää, sekä millaisia asioita järjestelmän suunnittelussa tulee ottaa huomioon. Aihetta voisi syventää VersionOne ALM-järjestelmän osalta selvittämällä

mitä vaatimuksia ohjelmistolla on kolmannen osapuolten ohjelmistojen yhdistämisessä järjestelmään. Tässä opinnäytetyössä oli avattu ainoastaan Agile Developmentin tekemän tutkimuksen tuloksia, jotka on saatu haastattelemalla yrityksiä, jotka käyttävät VersionOne ALM-järjestelmää. Jatkotutkimuksen aiheena voisi hankkia tutkimustuloksia useammasta ALM-järjestelmästä ja verrata, eroavatko yritysten vastaukset sen mukaan mitä ALM-järjestelmää he käyttävät.

Lähteet

Borland StramTeam Internet-sivut. Viitattu 10.10.2010.
http://www.borland.com/us/products/starteam/index.html

Collins-Sussmann Ben, Fitzpatrick Brina W., Pilato C. Michael. 2008. Version Control with Subversion: For Subversion 1.5: (Compiled from r3305). Viitattu 28.7.2010.
http://svnbook.red-bean.com/en/1.5/svn-book.pdf

Decide Guide Internet-sivut. Viitattu 10.10.2010. http://decide-guide.com/kepner-tregoe/

decision-making-confidence Internet-sivut. Viitattu 10.10.2010
http://www.decision-making-confidence.com/kepner-tregoe-decision-making.html

deJong Jennifer. 2008. Mea culpa, ALM toolmakers say. Viitattu 12.5.2010.
http://www.sdtimes.com/SearchResult/31952

Desmedt Serge. 2006. Building a Development Environment Part 1: Managing your sourcecode with Subversion. Viitattu 5.8.2010. http://sdesmedt.wordpress.com/2006/11/04/building-a-development-environment-part-1-managing-your-sourcecode-with-subversion/

DevTopics. 2008. What is .Net?. Viitattu 25.7.2010. http://www.devtopics.com/what-is-net/

Erickson Jonathan. 2009 Programmer Productivity. Viitattu 9.8.2010.
http://nelli.laurea.fi:2107/pqdlink?did=1875242131&Fmt=6&clientId=29499&RQT=309&VName=PQD

Gardner Dana. 2007. ALM 2.0 era ushers application development into a managed business process. Viitattu 1.7.2010. http://www.zdnet.com/blog/gardner/alm-20-era-ushers-application-development-into-a-managed-business-process/2456

Handelsbanken Central Data Finland:n osastopäällikkö Halonen Petri 20.5.2010, Helsinki. Tekijän hallussa.

Handelsbanken Internet-sivut. Viitattu 15.7.2010. www.handelsbanken.fi

Jaideep. 2008. 5 essentials while building test environment for software testing. Viitattu 17.7.2010. http://itknowledgeexchange.techtarget.com/quality-assurance/5-essentials-while-building-test-environment-for-software-testing/

Küng Stefan, Onken Lübbe, Large Simon. 2010. TortoiseSVN. Suomentaja Kari Granö. Viitattu 10.6.2010.
http://ftp.heanet.ie/disk1/sourceforge/t/project/to/tortoisesvn/Documentation/1.6.10/TortoiseSVN-1.6.10-fi.pdf

Michigan Department of Technology, Management and Budget:n Internet-sivut. Viitattu 15.7.2010. http://www.michigan.gov/cybersecurity/0,1607,7-217-34415---,00.html

Mircosoft Corporation. 2007. Key Benefits of Microsoft Visual Studio Team System. Viitattu 8.6.2010. http://www.microsoft.com/downloads/details.aspx?FamilyID=af18b941-1c70-4ca7-bb9c-a4a2bdc0adfb&displaylang=en#filelist

Microsoft Team Foundation Server:n Internet-sivut. Viitattu 10.10.2010.
http://msdn.microsoft.com/en-us/vstudio/dd408378.aspx

Moonsoft:n Internet-sivut. Viitattu 11.8.2010. http://www.moonsoft.net

Orcanos:n Internet-sivut. Viitattu 15.5.2010. http://www.orcanos.com/why_alm_20.htm

TechTarget:n Internet-sivut. Viitattu 27.7.2010.
http://searchsoftwarequality.techtarget.com/sDefinition/0,,sid92_gci936502,00.html

VersionOne, LLC. 2007. Agile Development: Results delivered. Viitattu 1.7.2010.
http://www.versionone.net/pdf/AgileDevelopment_ResultsDelivered.pdf

Kuvat

Taulukot