Team Foundation Server sovelluskehitysprojektin hallintatyökaluna

Lepistö, Mikael

2010 Leppävaara

Laurea-ammattikorkeakoulu
Laurea Leppävaara

Team Foundation Server sovelluskehitysprojektin hallintatyökaluna

Mikael Lepistö
Tietojenkäsittelyn koulutusohjelma
Opinnäytetyö
Toukokuu, 2010

Laurea University of Applied Sciences
Laurea Leppävaara

Team Foundatation Server as Software development management tool

Mikael Lepistö
Information Technology Programme
Thesis
Toukokuu, 2010

Laurea-ammattikorkeakoulu Tiivistelmä
Laurea Leppävaara
Tietojenkäsittelyn koulutusohjelma

Mikael Lepistö

Team Foundation Server sovelluskehitysprojektin hallintatyökaluna

Vuosi 2010 Sivumäärä 56

Tässä opinnäytetyössä tutkittiin ja kehitettiin Team Foundation Serverin (TFS) käyttöä sovelluskehitysprojektin työvälineenä. Selvitin TFS:n käyttöön liittyviä kehityskohteita omien sekä kohdeyrityksen työntekijöiden käyttökokemuksien avulla ja tein parannusehdotuksia löydettyihin kohtiin. Team Foundation Serverin käyttöön liittyvät ongelmakohdat selvitettiin teemahaastatteluiden ja kyselylomakkeen avulla. Kyselylomake jaettiin 24:lle osaston työntekijälle, joista 18 palautti kyselyn täytettynä. Keskityin etenkin Team Foundation Serverin työkorttien käytön kehittämiseen. Työn tarkoituksena oli myös antaa kattava kokonaiskuva Team Foundation Serveristä.

Opinnäytetyö tehtiin Yritys A:n pyynnöstä. Osasto Yritys A:ssa on vaihtamassa nykyisen sovelluskehitysprojektin hallintajärjestelmänsä Microsoftin Team Foundation Serveriin. Uuteen hallintajärjestelmään siirtyminen tulee tapahtumaan vaiheittain lähimpien 1-2 vuoden aikana osastolla. Yritys A on monikansallinen yritys, joka tuottaa tietojärjestelmiä ja palveluita asiakkailleen. Sovelluskehitystä tehdään maantieteellisesti hajautetusti. Opinnäytetyössä käytetään esimerkkinä erästä kansainvälistä projektia. Virtuaalitiimien avulla saatiin käyttöön eri asiantuntijoiden tiedot ja taidot maantieteellisestä sijainnista riippumatta. Etäisyys voitiin ylittää Team Foundation Serverin tarjoamilla työkaluilla ja yhteisellä versionhallinnalla.

Opinnäytetyön ensisijaisia kohderyhmiä ovat osastomme projektipäälliköt sekä sovellusarkkitehdit. Työstä on myös huomattavaa lisäarvoa testaajille, suunnittelijoille ja versionhallintavastaaville. Kehitysehdotukset on huomioitu yrityksessä, ja osa niistä tullaan ottamaan käyttöön jo seuraavissa projekteissa.

Kohdeyritys on pyytänyt, ettei sen nimeä julkisteta. Myös yritykseen liittyvät tuotteet, henkilöt, asiakkaat ja toimintaympäristön tarkemmat tekniset tiedot pidetään salassa, eikä niitä julkisteta tietoturvariskin ja liikesalaisuuden vaarantumisen takia. Muu osa tutkimuksesta on vapaasti julkaistavissa.

Avainsanat: Team Foundation Server,Sovelluskehitysprojektin hallintajärjestelmä, versionhallinta, Hajautettu ohjelmistokehitys,.NET

Laurea University of Applied Sciences
Laurea Leppävaara
Information Technology Programme

Abstract

Team Foundation Server as a software development management tool

Year 2010 Pages 56

The subject of this thesis was to research and to elaborate on the use of the Team Foundation Server (TFS) as a software development management tool. The development needs were analysed on the basis of the operational experience of the writer and the target company employees and collected by questionnaire. The forms were assigned to 24 persons and 18 people filled out and returned it. Improvement suggestions were made to the relevant points. The focus was especially on the way of working with TFS work items. Interviewing techniques were used to discover the problem areas of the TFS in the department. The secondary purpose of the thesis is to give a good general view about the Team Foundation Server.

The thesis was assigned by Company A. A department in the company is changing its software development management tool from Telelogic Synergy to Microsoft Team Foundation Server. Implementation will be completed step by step in the following 1-2 years in the department. Company A is a multinational enterprise, which produces large information systems for its customers. Software development is geographically divided over the country. I use as an example an international project where the writer was one of the team members is used as an example. Team members' knowledge and skills were pooled despite the physical distance, which was crossed by Team Foundation Server tools and common version control.

The target groups of this thesis are project leaders and system architects in the department. The thesis also delivers useful information about the Team Foundation Server to testers, to developers and to build managers. Improvement suggestions have been considered in the company and part of them will be included in project management in future.

The target company has requested to keep its name in secrecy. Company related products, persons and detailed information about the technical environment will not be published for security risk. The rest of the thesis is a public document.

Sisällys

1 Johdanto

Tämän opinnäytetyön aihe on Microsoft Team Foundation Serverin (TFS) käytön kehittäminen tietojärjestelmiä tuottavassa yrityksessä. Olin mukana projektissa, jossa käytettiin ensimmäistä kertaa Team Foundation Serveriä asiakasprojektissa osastollamme. Projektissa toteutettiin ASP .NET-pohjainen tietojärjestelmä Saksassa toimivalle yritykselle. Selvitin projektin kokemuksien avulla TFS:n käyttöön liittyviä kehityskohteita, ja tein niihin parannusehdotuksia.

Olen töissä monikansallisessa yrityksessä, joka tuottaa laajoja tietojärjestelmiä asiakkailleen. Yritys A on osa suurta konsernia, ja sillä on toimipisteitä yli 30 maassa. Toimin osastolla, joka toteuttaa vähittäiskaupan tietojärjestelmiä. Sovelluskehitystä toteutetaan maantieteellisesti hajautetusti. Hajautettu sovelluskehitys tuo tiimityöskentelyyn useita haasteita liittyen projektinhallintaan, esimerkkinä kasvokkaisviestinnän puuttuminen sekä osaamisen- ja vastuun hajautuminen. Osastomme on vaihtamassa nykyisen sovelluskehitysprojektin hallintajärjestelmänsä Microsoftin Team Foundation Serveriin. Uuteen hallintajärjestelmään siirtyminen tulee tapahtumaan vaiheittain lähimpien 1-2 vuoden aikana. Team Foundation Server otetaan tällä hetkellä käytössä olevan Telelogic Synergyn tilalle.

Käsittelen tässä työssä aluksi ohjelmistosuunnittelun perusteita antaakseni lukijalle hyvän kuvan vaiheista, joita vaaditaan asiakaskohtaisten toiminnallisten vaatimusten toteuttamiseksi testausvalmiiksi ohjelmistoversioksi. Tämän jälkeen esittelen TFS:n toimintaa ja käyttöä sekä teknistä toimintaa yhdessä .NET Frameworkin kanssa teoriapohjalta. Tein opinnäytetyössäni myös kyselylomakkeen, joka jaettiin osastomme työntekijöille. Lomakkeen tarkoituksena oli selvittää TFS:n käyttöön liittyvistä ongelmakohdista. Kyselylomakkeen vastauksia tulkitaan opinnäytetyön empiirisessä osuudessa (luvut 8 - 10). Kysely jaettiin osastollamme 25 henkilölle, jotka ovat käyttäneet Team Foundation Serveriä. Kyselyyn vastasi 18 henkilöä. Haastattelin myös viittä osastomme työntekijää, jotka ovat olleet mukana TFS:n käyttöönotossa ja prosessimallin kehittämisessä.

Työni annetaan eteenpäin osastomme projektipäälliköille sekä sovellusarkkitehdeille. Siitä on myös hyötyä sovelluskehittäjille ja testaajille. Tein opinnäytetyöni osittain työajalla, mutta suurimman osan tein omalla ajalla. Opinnäytetyön kustannuksia aiheuttivat kirjoittamiseen ja tutkimiseen käytetty työaika ja työstä saatu palkka. Yritys ei käyttänyt opinnäytetyöhöni ylimääräistä rahaa, sillä kirjoitus tapahtui suurilta osin omalla ajalla tai muiden töiden ohella, sekä opinnäytetyön lisäksi tuotettu muu dokumentaatio jää yrityksen sisäiseen käyttöön. Opinnäytetyön kirjoittamisen aikana pidettiin useita palavereja sekä koulun että etenkin Yritys A:n opinnäytetyön ohjaajien kanssa. Palavereissa käytiin läpi tilannekatsaus sekä varmistettiin, että työni etenee aikataulussa.

Yritys A on pyytänyt, ettei sen nimeä julkisteta. Myös yritykseen liittyvät tuotteet, henkilöt, asiakkaat, TFS:n ohjeistus osastomme projekteja varten ja toimintaympäristön tarkemmat tekniset tiedot pidetään salassa, eikä niitä julkisteta tietoturvariskin ja liikesalaisuuden vaarantumisen takia. Muu osa tutkimuksesta on vapaasti julkaistavissa.

1.1 Työn tavoitteet, rajaus ja riskit

Uuden sovelluskehitysprojektin hallintajärjestelmän tarkoituksena on tehostaa sovelluskehityksessä käytettäviä prosesseja niin laadullisesti kuin ajallisesti. TFS:n avulla voidaan toteuttaa sovelluskehityksen kaikki osa-alueet arkkitehtuurin suunnittelusta testaukseen. Projektia voidaan näin ollen hallita koko sovelluskehitysprojektin elinkaaren ajan.

Työni tavoitteena on kehittää TFS:n hyödyntämistä sekä edistää TFS-osaamista osastollamme. Omiin oppimistavoitteisiini kuuluu sovelluskehityksen elinkaaren ymmärtäminen.

1.2 Aiheen rajaus

TFS:n käyttöönotto yrityksessä voidaan jakaa Sovelton mukaan seitsemään osa-alueeseen:

1. Team Foundation Server asennus
2. Sovelluskehitystiimin muodostaminen
3. Prosessimallit ja niiden mukauttaminen
4. Work Item Tracking
5. Lähdekoodin hallinta
6. Build-prosessin hallinta
7. Portaali ja raportointi

(Team Foundation Server asennus ja käyttöönotto, 2008).

Yleisesti ohjelmiston tuotantoprosessiin liittyvät osa-alueet ovat: laatujärjestelmä, projektinhallinta, dokumentointi, tuotteenhallinta, laadunvarmistus, testaus, määrittely, suunnittelu, toteutus, käyttöönotto ja ylläpito. (Haikala&Märijärvi, 2004 s.12). TFS ja ohjelmiston tuotantoprosessi ovat kokonaisuutena todella laaja aihealue ja tehtävässä olikin syytä keskittyä vain tiettyihin osa-alueisiin rajallisen aikataulun vuoksi. Keskityn etenkin TFS:n projektin hallintatyökalujen kuten Work Item -mallien kehittämiseen. Pyrin kehittämään myös muita projektinhallintaan liittyviä osa-alueita resurssien sallimissa rajoissa.

1.3 Riskit

Riskit on jaettu kolmeen eri ryhmään: viivästymiset, alhainen vastausprosentti TFS-kyselyyn sekä kyselyyn liittyvät ongelmat.

Mahdollinen riski	**Riskin syy**	**Todennäköisyys**	**Riskin vaikutus**	**Toimenpide-ehdotus**
Viivästyminen	Teoria aineiston tulkitseminen lisäarvoa tuottavalla tavalla kestää odotettua kauemmin tai asiakasprojektit vievät liian ison osan työajasta	Keskisuuri	Vakava	Projektin etenemistä tehostettava jatkossa
Alhainen vastausprosentti TFS:n liittyvään kyselyyn	Kyselyyn vastanneilla ei ole aikaa / halua vastata kyselyyn	Keskisuuri	Vakava	Henkilökohtaiset haastattelut
TFS-kyselystä jää tulkintaa tehdessä oleellisia kysymyksiä puuttumaan	Teoriapohjan kasvaessa tulee mieleen kysymyksiä, joita olisi ollut hyvä olla kyselylomakkeessa	suuri	Riippuen kysymyksen oleellisuudesta kokonaismerkitykselle ja tulosjoukolle	Henkilökohtaiset haastattelut, pahimmassa tapauksessa uusi kysely. Uusi kysely voi kohdistua myös pelkästään tietylle kohderyhmälle.

Taulukko 1 Projektin riskienhallinta

2 Tutkimusmenetelmät

Opinnäytetyössä on käytetty useita eri tutkimusmenetelmiä. Menetelmistä käytetyin on konstruktiivinen tutkimusote, jota tukivat systeemiajattelu, tapaustutkimusote sekä haastattelumenetelmät ja dokumenttianalyysi. Haastattelutapoja on useita, joista hyödynsin teemahaastattelua. Teemahaastattelu on keskustelunomainen tilanne, joissa valmiiden yksityiskohtaisten kysymysten sijasta keskitytään teemoihin. (Teemahaastattelu 2000)

Järvinen ja Järvinen määrittelevät teoksessa Tutkimustyön metodit konstruktiivisen tutkimusotteen. Konstruktiivisen työotteen avulla ratkaistaan reaalimaailman ongelmia ja luodaan lisäarvoa sovellettavalle tieteenalalle. Konstruktiivinen tutkimus on empiiristä tutkimusta. Konstruktiivinen tutkimus perustuu käytännön ja teorian väliseen vuoropuheluun. Sitä käytetään, kun tutkitaan suunnitellun ja käyttöönotetun rakenteen arvoa välineenä. Rakennetta voidaan tarkastella havainnollisesti, testata ja kehittää. (Järvinen & Järvinen 2004, 102). Ojasalon mukaan konstruktiivinen tutkimus on käytännönläheistä. Sen pohjalla on teoreettista tietoa, jota sovelletaan käytännön työssä. Konstruktiivinen tutkimus kehittää ratkaisuja tutkimusongelmiin. (Ojasalo, Moilanen, Ritalahti. 2009. 65.)

Hyödynsin systeemiajattelua ymmärtämisen apuvälineenä. Systeemiajattelun avulla voidaan kuvata yksinkertaistamalla järjestelmien vaikuttavia osia ja näiden toiminnan yhteistulosta. Prosesseja voidaan kuvata kaavioilla, ja saada järjestelmän toiminta havainnollistetuksi ilman järjestelmän olemassaoloa. Systeemiajattelu eroaa siis perinteisestä analyysista siten, että se tutkii järjestelmän eri osien vaikutusta kokonaisuuteen sen sijaan, että jaettaisiin tutkittava kohde erillisiin osiin. (Mielonen, 2009).

Mielonen jakaa järjestelmän kuvauksen systeemiajattelun avulla seuraaviin vaiheisiin:

1. Lue järjestelmän toiminnasta kertova kuvaus.
2. Pohdi mitkä elementit kuvauksessa vaikuttavat muihin elementteihin.
3. Mitä asioita tapahtuu kuvauksen taustalla ja miten ne ilmenevät kuvauksessa?
4. Mitä prosesseja (tapahtumia, muutoksia) järjestelmässä esiintyy?
5. Listaa järjestelmän eri osat siten kuin ne kuvauksessa vahvistuvat tai heikentyvät.
6. Luo toisiinsa vaikuttavien osien välille linkkejä kuvaamaan näiden välisiä suhteita.
7. Anna nuolille vahvistusta tai heikennystä kuvaava merkki (plus tai miinus).
8. Luo linkitettyjen osien välille takaisinkytkentää (nuoli toiseen suuntaan), mikäli tällainen suhde on löydettävissä.
9. Merkitse viive niiden vaikutussuhteiden kohdalle, jotka vaikuttavat tapahtumien itsensä jälkeen ja tekevät siten vuorovaikutuksen vaikeammaksi havaita.
10. Tarkista vastaako vuorovaikutusmalli todellisen järjestelmän kuvausta.

Mielosen mukaan systeemiajattelua voidaan käyttää useaan eri tarkoitukseen. Systeemiajattelua voidaan hyödyntää, kun halutaan jäsentää toimivaa kokonaisuutta niin, että vaikuttavat prosessit ja niiden väliset merkitysevät suhteet tunnistetaan. Prosessit jaetaan vaikuttaviin ja tukeviin prosesseihin. Systeemimallien avulla halutaan löytää ne prosessit, joihin on tarkoitus vaikuttaa saavuttaakseen muutoksia järjestelmään. Vuorovaikutusmallien avulla selvitetään prosessissa ongelmia aiheuttava syy. (Mielonen 2009.)

Tapaustutkimuksen avulla etsitään tietoa nykytilassa tapahtuvasta toiminnasta todellisessa toimintaympäristössä. Tapaustutkimus tuottaa yksityiskohtaista tietoa tutkittavasta kohteesta. Tapaustutkimusote rajaa tutkittavan kohteen funktionaalisesti. Kohteesta erotetaan tietty yksittäinen prosessi tarkasteltavaksi. (Ojasalo ym. 2009 53.)

3 Sovelluskehitys

Yleisesti ohjelmiston tuotantoprosessiin liittyvät osa-alueet ovat: laatujärjestelmä, projektinhallinta, dokumentointi, tuotteenhallinta, laadunvarmistus, testaus, määrittely, suunnittelu, toteutus, käyttöönotto ja ylläpito. Näistä määrittely, suunnittelu, ohjelmointi, testaus ja käyttöönotto sekä ylläpito ovat sovelluskehityksen avainprosesseja. Edellä mainitut prosessit toimivat ns. vesiputousmallissa peräkkäisenä ketjuna, mutta suuntaus on kehittymässä jatkuvasti ns. iteratiivisen kehityksen suuntaan. Sovelluskehityksen tavoitteena on saattaa asiakkaan tarve toimivaksi tietojärjestelmäksi asiakkaan käyttöön. Tämä tavoite saavutetaan erilaisten elinkaarimallien avulla. Elinkaarimalleilla pyritään kontrolloimaan tuotantoprosessien kehittymistä tehokkaammalla mahdollisella tavalla. (Haikala, I., Märijärvi, J, 35.)

3.1 Sovelluskehityksen perusperiaatteita

Sovelluskehitys tapahtuu yleisimmin projektityöskentelynä. Projektisuunnittelussa projekti jaetaan aktiviteetteihin eli projektin päävaiheisiin. Päävaiheet jaotellaan tehtäviksi ja tehtäville annetaan työmääräarviot. Mitä pienempiin tehtäviin päästään, sen luotettavammaksi muodostuu projektisuunnitelma ja sen seuranta. (Haikala, Märijärvi, s.54.)

Ohjelmistosuunnittelun tärkeimpiä periaatteita on minimoida järjestelmän monimutkaisuus. Tietojärjestelmät ovat aina monimutkaisia, koska ne mallintavat monimutkaisia toimintoja. Tietojärjestelmät ovat usein myös erittäin laajoja järjestelmiä, jotka saattavat sisältää miljoonia rivejä koodia. Järjestelmän monimutkaisuus voidaan minimoida huolellisella suunnittelulla eli pitämällä eri komponenttien väliset rajapinnat selkeinä ja yksinkertaisina. Turhia rajapintoja on myös syytä välttää. Projekteissa käytettävät menetelmät tulevat olla skaalautuvia. Organisaatiolla on oltava tapa, jolla projektit viedään läpi. Suuria projekteja ei voida saattaa loppuun ilman harkitusti valittuja prosesseja, projektisuunnitelmaa, huolellista dokumentaatiota, viestintää ja kunnollista projektinohjaamista. Miljoonia rivejä sisältävän tietojärjestelmän toteuttaminen vaatii suuren, usein maantieteellisesti hajautetun, ryhmän panosta vuosien ajan. Vaikeuksiin joutunutta projektia ei pystytä pelastamaan yksittäisen toteuttajan panoksella. Myös henkilöstöresurssien lisääminen suuren projektin loppuvaiheilla usein jopa hankaloittaa ja viivästyttää projektin läpivientiä. (Haikala, Märijärvi, s.27-29.)
Nykyaikaisessa kehitystyössä ohjelmistosuunnittelijan on oltava monipuolinen osaaja. Järvisen mukaan ohjelmakoodin kirjoittaminen ei enää pelkästään riitä vaan suunnittelijan on osattava ottaa kantaa useaan komponenttiin kuten järjestelmäarkkitehtuuriin, tietokantojen toiminnallisuuteen ja hallintaan sekä web-sovellusten rakenteeseen. (Järvinen, s.228).

3.2 Sovelluskehityksen elinkaari

Elinkaarimallilla tarkoitetaan ohjelmistokehityksen vaiheistuksen jakamista. Niin sanottu vesiputousmalli on yksinkertaisin elinkaarimalli. Jokaisessa elinkaarimallissa on kuitenkin tunnistettavissa seuraavat kolme vaihetta:

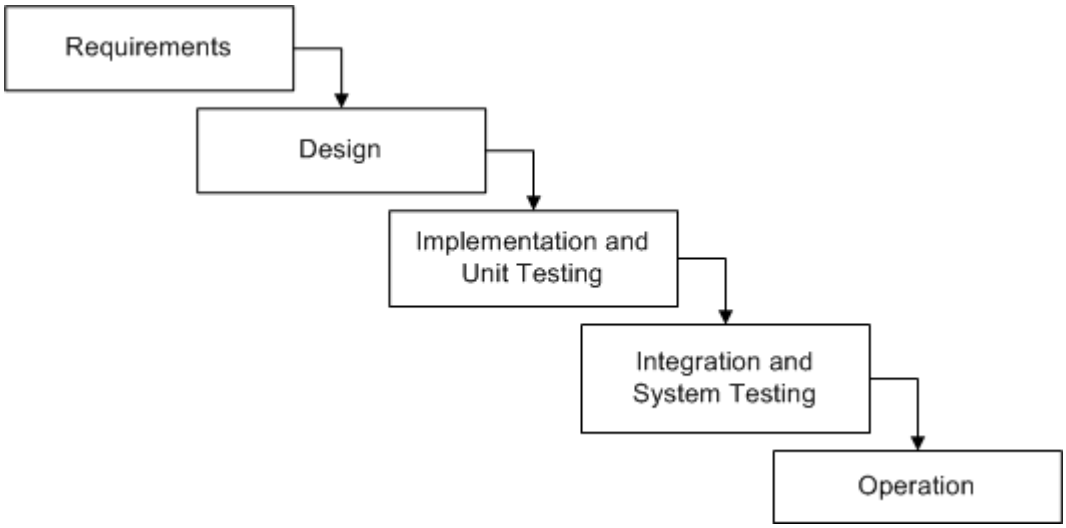
1. Määrittely
2. Suunnittelu
3. Toteutus

Määrittelyvaiheessa selvitetään ja dokumentoidaan asiakkaan vaatimukset tietojärjestelmälle. Määrittelyvaiheen tuloksena syntyvät toiminnalliset määritykset (functional specification). Toiminnallinen määritys kuvaa järjestelmän toimintaa kuten toteutettavat ominaisuudet, käyttöliittymä ja yhteensopivuus muiden järjestelmien kanssa sekä joitain ei-toiminnallisia vaatimuksia kuten suoritustehoa, vasteaikoja ja käytettävyyttä. (Haikala, Märijärvi, 39).

Järjestelmän toimintaa kuvataan käyttötapauksien avulla. Käyttötapaukset voivat olla graafisia tai tekstimuotoisia. Käyttötapaukset hyväksytetään usein asiakkaalla ennen suunnitteluvaihetta. (Haikala&Märijärvi 2004, 40).

Suunnitteluvaihe sisältää määrittelyvaiheessa kuvattujen toimintojen suunnittelun tekniseltä näkökulmalta. Suunnitteluvaihe jakaantuu arkkitehtuuri- ja moduulisuunnitteluun ja sen tuloksena syntyy tekninen määritys. (Haikala&Märijärvi 2004, 40).

Mikkosen ja Koskimiehen mukaan arkkitehtuurisuunnittelu määrittelee järjestelmän ytimen, joka tulee olemaan järjestelmän perusta kehityksen ja ylläpidon aikana. Arkkitehtuuri perustuu ohjelmiston jakamiseen pienempiin osakokonaisuuksiin. (Koskimies, Mikkonen, 19). Moduulisuunnittelu rakentuu arkkitehtisuunnittelun päälle. Jokaisen moduulin sisäinen rakenne suunnitellaan. Moduulin voidaan ajatella olevan esimerkiksi yksittäinen toiminto. Moduuli sisältää tietomäärittelyitä ja kyseenomaista dataa käsitteleviä funktioita. Toteutus eli ohjelmointivaiheessa sovellus kirjoitetaan ohjelmakoodilla ja käännetään valmistuneeksi ohjelmistokokonaisuudeksi. (Haikala&Märijärvi, 40.) Kuviossa 1 on esitetty vesiputousmalli.

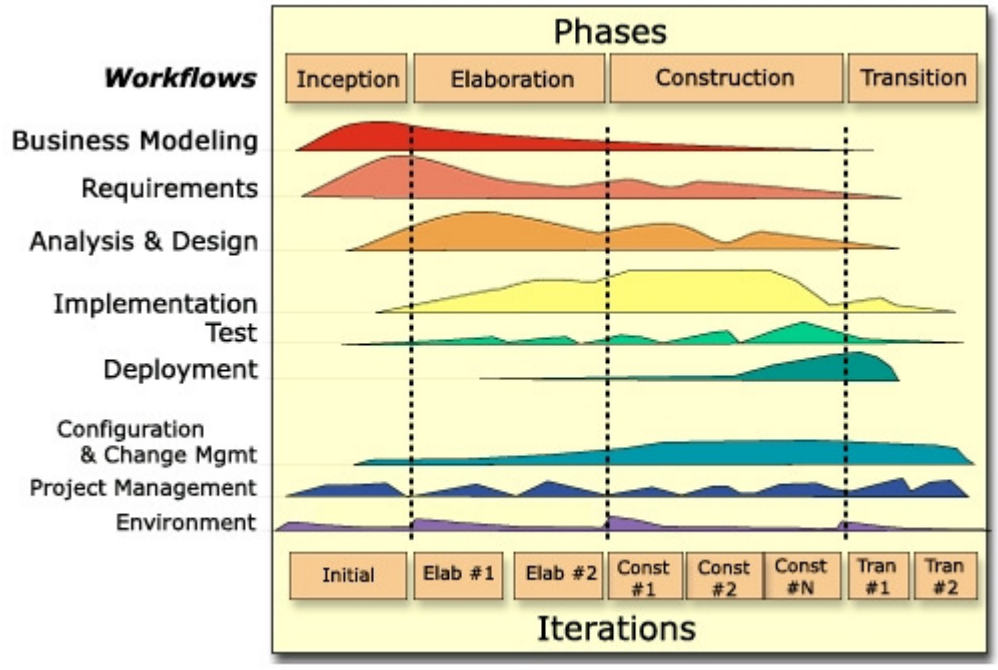


.

Kuvio 1 Vesiputousmallin mukainen elinkaari (Software Development Life Cycle Models, 2005)

Vesiputousmallin lisäksi käytettyjä elinkaarimalleja ovat erilaiset protoilumallit ja inkrementaaliset (Evomallit) mallit. Protoilumalleihin liittyy vahvasti tuotteen toteutus osittain ennen varsinaisen tuotteen rakennusta. Prototyyppi voi sisältää esimerkiksi käyttöliittymän, jossa on korkealla tasolla kuvattu järjestelmän toimintaa. Valmiin prototyypin perusteella aloitetaan ohjelmistoprojekti uudelleen tai toteutetaan valmis tuote valmiin prototyypin päälle. Prototyypeistä on koettu olevan eniten lisäarvoa käyttöliittymäsuunnittelussa. Ongelmaksi on koettu asiakkaan virheellinen käsitys jo lähes valmiista tuotteesta. (Haikala&Märijärvi, 42 -43.)

Inkrementaalisilla malleilla tarkoitetaan ohjelmistokehitystä, jossa ohjelmistoa kehitetään pieninä inkrementteinä yhden projektin sisällä. Määritetty lopputulos saavutetaan valitulla määrällä kierroksia, joiden lopputulos on toimiva lopputuote. Hyvä esimerkki inkrementaalisesta elinkaarimallista on Rational Unified Process eli RUP. Kuvan 2 mukaan RUP perustuu peräkkäisiin iteraatioihin, joista jokainen sisältää oman vesiputouksensa mallinnuksesta toteutukseen ja projektin ylläpitoon. Ohjelmiston kehitys on jaettu neljään vaiheeseen: *Inception (aloitus), elaboration (täsmennys), construction (rakennus) ja transition (siirtymä)*. Jokainen vaihe koostuu yhdestä tai useammasta iteraatiosta.



Kuvio 2 Ohjelmistokehityksen vaiheet RUP:n mukaisesti (RUP in the dialogue with Scurm, 2005)

Inception-vaihe sisältää alustavat suunnittelumallit ja arkkitehtuurit. Tuotteen ominaisuudet ja onnistumiskriteerit selvitetään asiakkaan kanssa. Inception-vaiheessa tehdään myös alustava projektisuunnitelma ja riskianalyysi sekä tarvittaessa toteutetaan järjestelmästä prototyyppi. Vaihe painottuu asiakasvaatimusten selvittämiseen. *Elaboration*-vaiheessa toteutetaan toimiva perusarkkitehtuuri järjestelmälle. Suunnittelumallit, arkkitehtuurikuvaus ja onnistumiskriteerit täydennetään projektin edetessä saavutetusta tiedosta. Riskit kartoitetaan uudelleen ja seuraava vaihetta varten tehdään projektisuunnitelma. Järjestelmää varten tehdään myös alusta käyttöohje. *Construction*-vaiheessa suunnittelumallit ja arkkitehtuurikuvaus ovat jo lähes täydelliset. Järjestelmästä toteutetaan Beta-versio. Käyttöohje kirjoitetaan myös valmiiksi, koska kaikki toiminnalliset ominaisuudet yksityiskohtineen on selvitetty. Onnistumiskriteerit ja riskianalyysi päivitetään ja tehdään projektisuunnitelma seuraavaa vaihetta varten. *Transition*-vaiheessa tiimillä on installointivalmis ohjelmisto. Arkkitehtuurikuvaus ja suunnittelumallit ovat täydelliset. (Haikala, Märijärvi, 41-43.)

RUP-mallin mukaisen sovelluskehityksen avulla päästään jo projektin varhaisessa vaiheessa toteamaan kriittisten päätösten kuten arkkitehtuurivalintaa toteutuksen näkökulmasta toisin kuin vesiputousmallissa, missä tehtyjen määrittelyvirheiden korjaaminen projektin loppupäässä voi tulla erittäin kalliiksi. (Haikala, Märijärvi, 44-45).

Iteratiivinen elinkaarimalli on viety vielä pidemmälle ns. ketterissä (agile) menetelmissä. Ketterissä menetelmissä iteraatiot ovat hyvin lyhyitä. Sovelluskehittäminen on suorastaan jatkuvaa, koska ketterissä menetelmissä käytetään välitöntä testausta joko testaajan tai automaattitestauksen puolesta. (Haikala, Märijärvi, 44-45).

4 .NET Framework

Microsoftin .NET Framework on ohjelmointiympäristö ja ajoympäristö Windows-pohjaisille sovelluksille. .NET-sovellusten käyttäminen vaatii Windowsiin .NET ajoympäristön. Ajoympäristö löytyy nykyään jo lähes kaikista Windows-koneista. Windows Vistasta alkaen .NET on ollut osana käyttöjärjestelmää ja vanhempiin käyttöjärjestelmiin .NET-komponentti on lisätty automaattisten päivitysten yhteydessä. (Järvinen, 24.)

.NET-sovellukset ovat ns. managed-koodia. Managed-koodi on .NET-kielillä tehtyä ja -kääntäjillä käännetty koodiä. .NET-sovellukset pyörivät erityisen virtuaalikoneen alaisuudessa, joka on nimeltään Common Language Runtime (CLR). Kun sovellus suoritetaan, CLR kääntää ohjelmakoodin tietokoneen prosessorin ymmärtämäksi natiiviksi koodiksi. (C# ja .NET Framework ohjelmointi, 18.)

.NET tukee useita eri sovellustyyppejä:

- Graafiset sovellukset, Windows Forms
- Web-sovellukset, HTML- tai XML-liittymin
- Windowsin palvelut (Service Application)
- Mobiilisovellukset
- Konsolisovellukset

Web-sovellusten nimi on .NETssä ASP .NET. ASP .NET-sivut rakentuvat olio- ja tapahtumapohjaisuuden varaan, joten niiden tekeminen muistuttaa osittain enemmän Windows-sovelluksen tekemistä kuin aikaisempien web-tekniikoiden kuten ASP- tai PHP-sivujen toteuttamista. ASP .NET-alustalla kielenä voi käyttää mitä tahansa .NET-kielistä. ASP.NET-sovelluksia ajetaan Microsoftin Internet Information Server –palvelimella (IIS) ja sovelluksia voidaan käyttää kaikilla selaimilla. (C# ja .NET Framework ohjelmointi, 21.)

ASP .NET -sovellus rakentuu useista tiedostoista, joita kehitystyössä on syytä pitää versionhallinnassa. Tärkeimmät tiedostot ovat ratkaisutiedosto eli solution-tiedosto (.Sln), ohjelmointikielestä riippuen ohjelmakoodin sisältävä .vb (Visual Basic) tai .cs (C#), konfigurointitiedosto web.config, yhteiset luokkatiedostot (dll) ja web-sivun grafiikan esittävä .aspx. (TFS Guide, 43.)

ASP NET-sovellukset suoritetaan kuten muutkin .NET-sovellukset eli .NET-ajoympäristön kautta. IIS-palvelimelle lähetetään http-muotoinen pyyntö käyttäjän selaimelta. Kun pyyntö hyväksytään, alkaa varsinainen koodin suorittaminen ja hyödynnetään mahdollisesti tietokantoja tai muita ulkoisia palveluita. Kun koodi on suoritettu ja käännetty, palautetaan käännetty tulos IIS-palvelimen kautta käyttäjän selaimeen. (C# ja .NET Framework ohjelmointi, 21.)

5 Microsoft Visual Studio Team System ja Team Foundation Server

Microsoftin mukaan Visual Studio Team System (VSTS) esiteltiin ensimmäisen kerran Visual Studio 2005:n julkaisun yhteydessä. VSTS on Microsoftin kehittämä elinkaaren hallintaan erikoistunut tuoteryhmä (ALM, Application Life-Cycle Management). VSTS koostuu seuraavista tuotteista:

1. Visual Studio Team System 2008 Team Suite
2. Visual Studio Team System 2008 Architecture Edition
3. Visual Studio Team System 2008 Database Edition
4. Visual Studio Team System 2008 Development Edition
5. Visual Studio Team System 2008 Test Edition
6. Visual Studio Team System 2008 Team Foundation Server

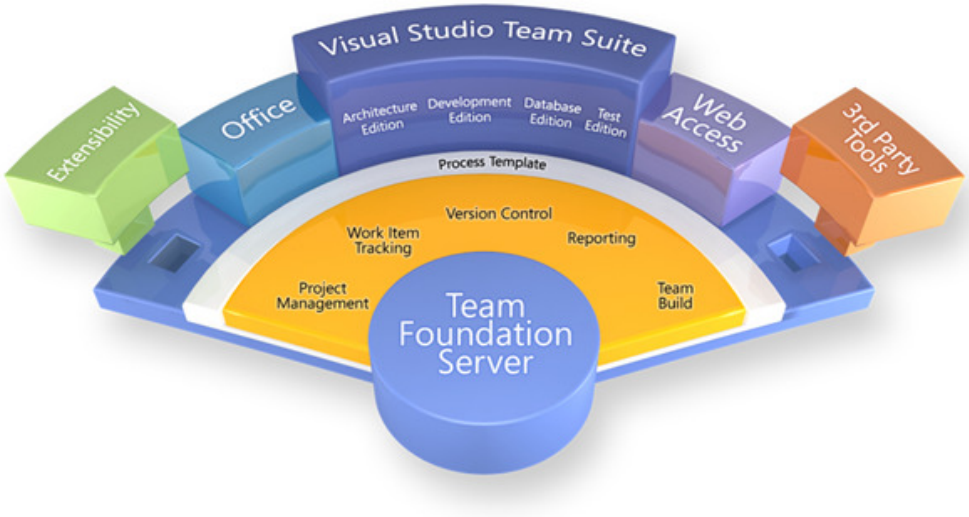
VSTS sisältää seuraavat ominaisuudet:

1. versionhallinta
2. kehitysympäristön ylläpidon
3. build-prosessin hallinta
4. projektinhallintatyökalut
5. prosessimallit
6. raportointityökalut

(Visual Studio Team System, 2010)

Team System on suunniteltu sovelluskehitysprosessin hallintaan. Perinteiset RAD-kehitysympäristöt (Rapid Application Development) ovat suunniteltu yksittäisen kehittäjän käyttöön ja niistä puuttuu tuki esimerkiksi versionhallintaan, laaduntarkkailuun ja projektinjohtamiseen. Toisin sanoen, organisoitu hajautettu kehitystyö ilman edellä mainittuja esimerkkejä on hyvin vaikeaa. RAD-kehitysympäristöksi voidaan mieltään esimerkiksi Visual Studio ilman Team Systemia. (Järvinen, 228.)

Team Systemin oleellisin osa on Team Foundation Server (TFS). Ohjelmisto toimii palvelimena asiakasohjelmille, joita ovat Visual Studio -ohjelmistot versiosta 2005 lähtien.(Järvinen 4-5).



Kuvio 3 VSTS:n rakenne (VSTS:n rakenne, 2010)

Kuviossa 3.0 on kuvattu Visual Studio Team Systemin rakenne. Team Foundation Server toimii palvelimena rakenteen pohjalla ja sen päällä toimivat projektinhallintatyökalut (Project Management, Work Item Tracking, Reporting), versionhallinta (Version Control) ja Team Build-toiminto. Ylimmällä tasolla ovat Client-ohjelmistot kuten Visual Studio, Microsoft Office, TFS Web Access sekä laajennusmahdollisuuden myötä tulevat kolmannen tahon ohjelmistot.

5.1 Prosessimallit

Edellisessä kappaleessa mainittujen työkalujen käyttöön vaikuttavat VSTS:n mukana tulevat prosessimallit (Process Template). Team Foundation Serverin mukana tulevat mallit ovat *MSF for CMMI Process Improvoment* sekä *MSF for Agile Software Development*. CMMI on tarkoitettu suuriin projekteihin ja MSF Agile pienempiin, 5-20 henkilön, projekteihin(TFS Guide, 114)

Prosessimalleja voi itse muokata. Prosessimallit ovat xml-muotoisia tiedostoja, jotka tarjoavat määrittelyt ja rakenteet projektityölle. Esimerkiksi työkorttien kentät ja kenttien valittavana olevat arvot periytyvät prosessimallista. Samoin kyselyjen rakenteet sekä mm Sharepointin asetukset ovat kuvattuna xml-tiedostostoissa. (TFS Guide, 114.)

5.2 Versionhallinta

Versionhallinnalla tarkoitetaan tekniikkaa, jolla pidetään yllä tietoa tiedostoihin tehdyistä muutoksista. Kohteena voivat olla lähdekoodien lisäksi myös dokumentit. Järvisen mukaan versionhallinnasta on paljon hyötyä pienissäkin kehitystiimeissä. Tiedostojen samanaikainen muokkaaminen ilman kunnollista versionhallintaa on vaikeaa ja lisää päällekkäisen työn riskiä. Versionhallinnan avulla on mahdollista palata tiedostojen tasolla historiassa taaksepäin. Muita etuja ovat koodiversioiden (branch) vertailu, tallentaminen sekä tiedostojen yhdistäminen (merge). (Järvinen, 252.) Termit Branch ja Merge käsitellään tarkemmin kappaleissa 7.4.2 ja 7.4.3.

Järvisen mukaan Team Foundationin versionhallinta perustuu SQL-Server-tietokantaan kuten muutkin TFS-palvelimen ominaisuudet. Team Foundationin versionhallinta tukee hajautettua käyttöä. Tietoliikenne on http-pohjaista, mikä takaa hallittavuuden palomuurien osalta. Heikkoja verkkoyhteyksiä varten on mahdollista ottaa käyttää erityinen välityspalvelin, joka osaa kopioida osan versionhallinnan koodista lähemmäs hajautettua tiimiä. (Järvinen, 234.)

Team Foundationin versionhallinta on integroitu osaksi Visual Studiota. Näin ollen kehittäjä voi suoraan kehitysympäristöstään käsin hallita versionhallinnan tiedostoja. Visual Studion Solution Explorer eli ratkaisuikkuna näyttää kuvakkeiden avulla tiedostojen tilan. Tiedosto voi olla esimerkiksi kirjattu ulos (Checkout) tai lukittu (locked). Tiedostojen hallinta ja siihen liittyvät termit käsitellään tarkemmin luvussa 7.4.

5.3 Käyttäjäryhmät

MSF Agile prosessimallissa on käytössä neljä eri käyttäjäryhmää. Lukijat (Readers) -käyttäjäryhmällä on oikeus vain lukea projektiin liittyviä tiedostoja ja työkortteja. Toteuttajat (Contributors) -ryhmällä on lukuoikeuden lisäksi oikeus lisätä, muokata ja poistaa dataa sekä kirjata ulos ja sisään tiedostoja. Versionhallintavastaavilla (Build Services) on Toteuttaja-ryhmän oikeuksien lisäksi oikeus käynnistää build-prosessi. Administraattori (Project Administrators) -oikeuksilla on oikeus kaikkiin edellä mainittuihin toimintoihin. (TFS Guide, 123.)

5.3.1 Tiimiprojektit

Team Foundationin versionhallinta sitoutuu tiimiprojekteihin (team projects). Tiimiprojekti muodostaa versionhallinnan hakemistopuun juuren (tree root). Tiimiprojekti perustetaan Visual Studion kautta ohjatulla toiminnolla. Ensimmäisessä vaiheessa annetaan projektin nimi.

Toisessa vaiheessa kysytään projektissa käytettävää prosessimallia. Kolmannessa vaiheessa kysytään portaalin (Portal) nimeä. Portaalilla tarkoitetaan Sharepoint-palvelua, joka on Microsoftin Collaboration-palvelu. Sharepointista on pääsy mm Web Accessiin sekä projektin dokumentaatioon. Neljännessä vaiheessa konfiguroidaan hakemistopuun (tree) asetukset. Team Foundation Server – versionhallinnassa hakemistopuun juurta kuvataan dollarimerkillä ($). Juuren alla olevat hakemistot kuvataan kauttaviivalla (/) kuten web-osoitteissakin. Tiimiprojektille luodaan automaattisesti uusi haara versionhallintaan. Esimerkiksi TestProject-nimisestä tiimiprojektista tulee $ / TestProject.
Viimeisessä vaiheessa näytetään yhteenveto käyttäjän tekemistä valinnoista. Käyttäjän hyväksyttyä tiimiprojektin asetukset, Team Foundation Server –palvelin lataa valitun prosessimallin, luo uuden haaran versionhallintaan sekä tarvittaessa perustaa uuden portaalisivuston SharePoint-palvelimelle. Tiimiprojektin perustamisen jälkeen on mahdollista ottaa versionhallinta käyttöön. (Järvinen, 255.)

Tiimiprojektin hakemistopuu replikoituu sekä asiakas- että Server-puolelle. Web-projektissa tyypillinen hakemistopuu voi olla esimerkiksi seuraavan mallin mukainen:

$TestProject
/Main
/Source
/MyApp1
/Source
/ClassLibrary1
/MyApp1Web
/UnitTests
/ClassLibrary1Tests
/MyApp1WebTests
/MyApp2
/Source
/ClassLibrary2
/MyApp2Web
/UnitTests
/ClassLibrary2Tests
/MyApp1WebTests
/SharedBinaries
UI.dll
BusinessLogic.dll
DataAccess.dll
/SQL
/Config
/SharedSource
MasterPage.aspx
/Docs
/Func
/Tech
/Testing
/Delivery_Notes
/Tests
/FunctionalTests
/PerformanceTests

/SecurityTests
/TeamBuildTypes
Build
/BuildType1
/BuildType2

(TFS Guide, 423).

Main-kansio on projektin yläkansio, joka sisältää lähdekoodit, dokumentaation, testitapaukset ja Team Build –toiminnan tuottamat tiedostot. Sovelluskansiot sisältävät Visual Studio solution–tiedoston (.sln). Solution–tiedosto eli ns. ratkaisutiedosto kokoaa yhteen viittaukset ratkaisuun kuuluvista web-projekteista. Esimerkissä sovelluskansiot ovat MyAppl1 ja MyAppl2. Projektitiedostot (.vsproj tai .vbproj) sijaitsevat polussa /Main/Source/MyApp1/Source. Projektitiedostot ovat XML–muotoisia koodin kääntämiseen käytettäviä tiedostoja, jotka sisältävät viittauksen projektiin kuuluvista tiedostoista sekä projektiin liittyvistä asetuksista. Shared-Binaris-kansio sisältää projektin yhteisiä dll-tiedostoja (Dynamic Link Library). Dll-tiedostot ovat ajonaikaisia jaettuja kirjastoja, jotka jakavat ohjelmakoodia sekä dataa useiden sovelluksien kesken. SQL–kansioon kerätään sovelluksessa käytettävät SQL–proseduurit. Config–kansioon kerätään .NET-sovellusten käyttämät konfigurointitiedostot (.config-päättyiset tiedostot). (Järvinen, 203-204.)

Yksikkötestit (Unit test) kerätään Unit Tests–kansioon. Yksikkötestit ovat kehittäjän toteuttamia ohjelmakoodin eheyteen ja logiikkaan liittyviä testejä. Varsinaisia testaajan testejä varten on perustettu kansiot FunctionalTests, PerformanceTests ja SecurityTests, jotka sisältävät testisuunnitelman ja testitapaukset automaatiotestausta varten. (Tfs Guide, 38)
TeamBuildTypes-kansio luodaan automaattisesti ensimmäisen Team Buildin yhteydessä, ja kansioon lisätään Team Build–toiminnon tuottamat tiedostot eli käännetyt sovellukset. (Tfs Guide, 37).

5.3.2 Työalueet

TFS:n hakemistopuu on kopio kehittäjien työaseman työalueesta. Vastaavanlainen hakemistopuu löytyy siis jokaisen kehittäjän työasemalta. Tiedostomuutokset kehittäjän omalla työasemalla eivät päivity automaattisesti TFS:n työalueelle vaan kehittäjä itse päättää, mitkä tiedostot ovat mukana ja mitä versiota tiedostosta käytetään versionhallinnassa. (TFS Guide, 394.)

5.4 Tiedostojen hallinta

Muutosjoukot (Changeset) ovat Team Systemin versioinnin perusyksikkö. Muutosjoukko koostuu tiedostoista, kommenteista ja työkorteista. Muutosjoukolla on tunnistenumero eli ”id”, jonka avulla muutosjoukkoon voidaan viitata. Muutosjoukkoon voi kohdistua yksi tai useampi työkortti. Versionhallintaan tallennettujen muutosjoukkojen ansiosta projekti edistyy; lähdekoodien määrä kasvaa ja tehtäviä kuitataan valmiiksi. Kun työkortit on valittu liitettäväksi muutosjoukkoon, voidaan kullekin työkortille määritellä erikseen sisäänkirjaustoiminto (Check-in action). Sisäänkirjaustoiminnon avulla on mahdollista asettaa muutosjoukkoihin sisältävien työkorttien statuksen muuttaminen. Esimerkiksi Virhekirjauksen voi asettaa resolved-tilaan, kun muutosjoukko kirjataan sisään. (Järvinen, 260.)

TFS Guiden mukaan on erityisen suositeltavaa, että muutosjoukkoon liitetään Tehtävä (Task) tai Defect eli virhekirjaus, jotta buildin rikkoutumistilanteissa päästään pian jäljille, mikä ja kenen tekemä osa koodista on rikkonut buildin. Buildin rikkoutumisella tarkoitetaan tilannetta, jossa lähdekoodeja ei pystytä onnistuneesti kääntämään ns. managed-koodiksi. Tällöin lähdekoodissa voi olla esimerkiksi syntaksi- tai semantiikkavirhe. Koodien kääntäminen tapah-

tuu Visual Studion Build-toiminnon avulla. Yksi versionhallinnan perussäännöistä onkin, että versionhallintaan kirjataan ainoastaan toimivaa koodia. (TFS Guide, 188.)

Team Systemin versionhallinnan tiedostoja hallitaan Visual Studion ratkaisuikkunan (Solution Explorer) ja Pending Changes –muutosikkunan avulla. Molemmat ikkunat näyttävät tiedostoja, joita kehittäjä on muokannut viimeisen sisäänkirjauksen jälkeen, poistanut tai lisännyt hakemistopuuhun. (Järvinen, 258.)

5.4.1 Tiedostojen kirjaaminen sisään ja ulos

Normaalissa käyttötilanteessa kehittäjä avaa projektin ratkaisutiedoston (.sln) omalta levyltään. Ensimmäiseksi on syytä varmistua, että käytettävissä on viimeisimmät lähdetiedostot projektista, jotka ovat versionhallintaan tallennettu. Visual Studion Source Control –ikkuna ilmoittaa Latest–sarakkeessa, onko tiedoston versio uusin mahdollinen. (Järvinen, 261-263).

Tiedosto kirjataan ulos, kun kehittäjä tekee siihen muutoksia. Tiedoston kirjaaminen ulos tapahtuu Source Control –ikkunassa klikkaamalla hiiren oikeata näppäintä tiedoston kohdalla ja valitsemalla pikakomennon Check-out. Ennen kuin tiedosto kirjataan ulos, kehittäjän tulee valita tiedostolle sopiva lukitusominaisuus. Lukitusominaisuus on tiedostokohtainen ja voimassa, kunnes tiedosto kirjataan sisään (Check-in). Käyttäjä voi valita esimerkiksi, että kukaan muu ei voi muokata tiedostoa, kun tiedosto on käyttäjällä uloskirjattuna. Kun muutokset tiedostoon on tehty, tiedosto voidaan kirjata takaisin sisään. Ennen tiedoston kirjaamista sisään, tehdyt muutokset kommentoidaan kommenteille varattuun kenttään. (TFS Guide, 338.)

5.4.2 Samanaikaisten muutosten käsittely

Kehittäjien on mahdollista työskennellä samanaikaisesti samojen tiedostojen parissa. Jokaisella kehittäjällä on oma kopionsa muokattavasta tiedostosta omalla koneellaan. Kun rinnakkaiset muutokset kirjataan sisään versionhallintaan, TFS huolehtii muutosten tallennuksen onnistumisesta. Jos kehittäjä on sallinut uloskirjaamisen toisille käyttäjille uloskirjatessaan (Check-out) tiedoston, tiedostoa on mahdollista muokata samaan aikaan. Kun ensimmäinen kehittäjistä on saanut muutoksensa valmiiksi tiedostoon, hän kirjaa tiedoston takaisen (Check-in) versionhallintaan. Kun toinen kehittäjä on saanut omat muutoksensa tehtyä tiedostoon, hän kirjaa myös oman versionsa tiedostosta versionhallintaan. Tällöin TFS ilmoittaa jälkimmäiselle kehittäjälle, että alkuperäinen tiedosto ja palvelimella uusimpana oleva tiedosto eivät enää täsmää. Tiedoston versioiden välille on nyt syntynyt ristiriita (Conflict). Ristiriita pitää ratkaista. Visual Studio osaa ratkaista itse useimmat ristiriitatilanteet AutoMerge-toiminnon avulla, mutta kaikkia tilanteita ei voi eikä pitäisikään ratkaista kyseisen toiminnon kautta. (Järvinen, 267- 271.)

Tiedostojen versioiden erilaisuutta on mahdollista tarkastella Compare-toiminnolla. Toimintoa avaa ruudulle Differences Viewer-ikkunan, jonka avulla voidaan verrata kahta eri tiedostoversiota. Oletusasetuksena verrataan kehittäjän paikallista versiota palvelimella olevaan versioon. Vertailu voidaan tehdä myös palvelimella olevien tiedostoversioiden välillä. (How to: Show Differences between Two Files or File Versions, 2010).

5.4.3 Branch

Laajassa kehitystyössä törmätään usein tilanteisiin, joissa tiedostoja tai polkuja on haarautettava. Tyypillisiä tapauksia haarautumiselle ovat uuden kehitystyön aloittaminen ennen edeltävän työn valmistumista sekä asiakaskohtaisten räätälöintien tekeminen. Tiedostojen haarauttaminen tapahtuu ohjatulla toiminnolla Source Control Explorer -ikkunan kautta. Kehittäjän on mahdollista valita, aloitetaanko haaroittaminen viimeisimmästä tiedoston tasasta vai aikaisemmasta kyseiseen tiedostoon kohdistuvasta muutosjoukosta tai tietystä päivämäärästä. Uudet koodiversiot ladataan kehittäjän valitsemaan tiedostopolkuun. Kansio ei saa olla nykyi-

sen projektin alikansio, vaan sen täytyy olla irrallaan nykyisestä projektista. (Understanding Branching, 2010.)

5.4.4 Merge

Haarautuneiden versioiden yhdistäminen tapahtuu Merge-toiminnolla. Kuten Branch-toimintokin, Merge tapahtuu ohjatun toiminnon avulla. Ohjatussa toiminnossa valitaan sekä haluttu lähde- että kohdehaara tai halutut kaksi tai useampi tiedosto, joiden välillä yhdistäminen tapahtuu. (Järvinen, 274).

Yksinkertaisimmallaan Merge-toiminto yhdistää halutut kaksi eri tiedostoa toisiinsa. Työkalu vertailee ensin tiedostojen eroja, ja liittää tiedostojen sisällöt uuteen, kolmanteen, tiedostoon. (Merging in Team Foundation Version Control, 2010).

5.4.5 Shelving

TFS Guiden mukaan Team Foundation Serverin versiohallinnassa on mahdollista hyllyttää koodia (shelving). Hyllyttämisen avulla kehittäjä voi tallentaa tekemänsä keskeneräiset koodimuutokset talteen TFS-palvelimelle. Koodin hyllyttämistä voidaan siis ennemmin ajatella keskeneräisen koodin varmuuskopioimisena kuin toimimattoman koodin hylkäämisenä. Hyllytetylle koodiversiolle perustetaan oma haara versionhallintaan. Hyllytettyä haaraa kutsutaan hyllytetyksi muutosjoukoksi (Shelveset). Kehittäjä voi asettaa säännön, että tiedostoista voidaan ottaa hyllytettyyn muutosjoukkoon kohdistuvat muutokset päivittäiset varmistukset tai jopa jatkuvia varmistuksia. (TFS Guide, 398.)

Etenkin pitkissä kehitystehtävissä koodin hyllyttämistä suositellaan. Kovalevyn rikkoutuminen voi tuhota tehdyn työn. Näin ollen varmuuskopiointia vähentää riskiä turhan työn tekemisestä. Hyllyttäminen mahdollistaa myös toisten kehittäjien pääsyn hyllytettyyn koodiin. Sairas- tai muissa poissaolotapauksissa toinen kehittäjä voi tarpeen vaatiessa jatkaa keskeneräistä sovellusta eteenpäin. (Järvinen, 265-266.)

Hyllytetty koodi voidaan ottaa jälkeenpäin versionhallinnan käyttöön Unshelve-toiminnolla milloin tahansa. Hyllytetty muutosjoukko jää talteen palvelimelle, ellei sitä erikseen poisteta. Hyllytetyt muutosjoukot on hyvä poistaa sen jälkeen, kun muutokset on saatu valmiiksi ja tallennettu versionhallintaan aitoina muutosjoukkoina. (Järvinen, 267.)

5.5 Team Foundation Build

Build-prosesessista käytetään yleisemmin nimeä koodin kääntäminen. Visual Studion kääntäjä muodostaa tietokoneohjelmassa käytetyn ohjelmakoodin perusteella konekielisen ajettavan binääritiedoston eli toisin sanoen kääntää ohjelman tietokoneen ymmärtämään kieleen. (C# ja .NET Framework ohjelmointi, 34).

Järvisen mukaan Visual Studiossa pienin käännettävä kokonaisuus on projekti. Projektiin kuuluu yksi tai useampi tiedosto sekä viittauksia toisiin tiedostoihin, kuten yhteisiin kooditiedostostoihin. (Järvinen, 198). Visual Studiossa kirjoitetut sovellukset on ensin käännettävä, jotta niitä voidaan ajaa, virheenjäljittää eli debugata ja testata. Isoissa projekteissa on syytä määritellä käännösasetuksia. Esimerkiksi pääsovellus voi tarvita tietyn luokkakirjaston toimiakseen, jolloin luokkakirjasto on käännettävä ennen pääsovellusta. Kehittäjä voi myös valita Debug-asetuksen, jolloin koodit käännetään virheenjäljitysaikana ja virheenjäljityksen kannalta tärkeitä symbolitiedot liitetään mukaan käännökseen. Release-asetus taas on käytössä, jolloin lähdekoodeista muodostetaan kokonaisuus esimerkiksi asiakastestejä tai tuotantoa varten. Tällöin turhat symbolitiedot jätetään pois käännetyistä tiedostoista. Käännöksen ai-

kaisiin asetuksiin vaikuttavat myös itse projektin asetukset, joissa on määritelty mm Target Framework eli hyödynnettävä versio .NET Frameworkista ja Assemblyn nimi. (Järvinen 209.)

Visual Studion kääntäjän kääntämistä lähdekoodeista muodostuu kiintolevylle uusia tiedostoja. Käännöskonfiguraatiosta riippuen, projektikansion alle muodostuvat joko bin/debug- tai bin/release -kansiot. Bin-kansioon on käännösten lopputulosten tallennuspaikka, johon muodostuvat exe- tai dll.päätteiset tiedostot. (Järvinen, 218.)

5.6 Work Item

Work Item eli työkortti on hyvin oleellinen osa hajautettua kehitystyötä Team Systemissä. Työkortit ovat lomakkeita, joihin kerätään erityyppistä tietoa kohteesta riippuen. Jokaisella työkortilla on merkityksensä projektin kannalta. Esimerkiksi MSF Agile -prosessimallissa työkortti voi olla:

1. Ohjelmavirhe eli Bug
2. Yleinen tehtävä eli Task
3. Laatuvaatimus eli Requirement
4. Riski eli Risk
5. Skenaario eli Scenario

Osa työkorteista perustetaan jo tiimiprojektin perustamisen yhteydessä. Tällaiset työkortit ovat usein projektipäällikön tehtäviä, kuten vaikka kehittäjien tehtävien perustaminen ja käyttöoikeuksien määrittäminen projektin jäsenille.

Työkorttien ominaisuudet kuten elinkaari, kentät ja pakolliset tiedot ovat administrator-käyttäjän muokattavissa Visual Studion kautta. Työkortteja ja työkorttien kenttiä voidaan luoda lisää ja muokata. Visual Studio 2008 Käsikirjan mukaan työkorteilla on seuraavan taulukon mukaiset yleiset kentät:

KENTTÄ	SUOMEKSI	TARKOITUS
Title	Otsikko	Työkortin nimi, josta käy ilmi tiivis kuvaus esimerkiksi ohjelmistovirheestä.
Type	Tyyppi	Määrittää esimerkiksi, minkä tyyppisestä laatuvaatimuksesta on kyse. Vaatimus voi liittyä suorituskykyyn tai tietoturvaan.
Severity	Vaikutus	Määrittää esim. kuinka suuresta (teknisestä) riskistä on kyse. Vaikutus voi olla sovellukselle vaikkapa kriittinen, suuri tai pieni.
Area	Alue	Kertoo, mihin tekeillä olevan projektin osa-alueeseen esim. riski liittyy. Web-sovelluksessa osa-alue voi olla tietokanta-, sovellus- tai käyttöliittymäkerros.
Iteration	Iteraatio	Kertoo, mihin projektin välitavoitteeseen työkortti liittyy
Assigned To	Annettu tehtäväksi	Henkilö, jolle tehtävä on osoitettu
State	Tila	Työkortin tila, esim. virhekirjauksessa started tai resolved

Reason	Syy	Kuvaus, minkä takia työkortti on merkitty suljetuksi. Syy voi olla, että Defect on korjattu, duplikaatti tai aiheeton.
Triage	Kiireellisyysaste	Kertoo kiirellisyysasteesta suhteessa mahdollisiin seuraauksiin.
Priority	Prioriteetti	Työkortin suoritusjärjestys, voi olla pieni, normaali tai suuri.
Rank	Tärkeys	Kertoo numeerisesti, kuinka tärkeä työkortti on verrattuna muihin työkortteihin.

Taulukko 2 Työkorttien yleisiä kenttiä (Järvinen, 251).

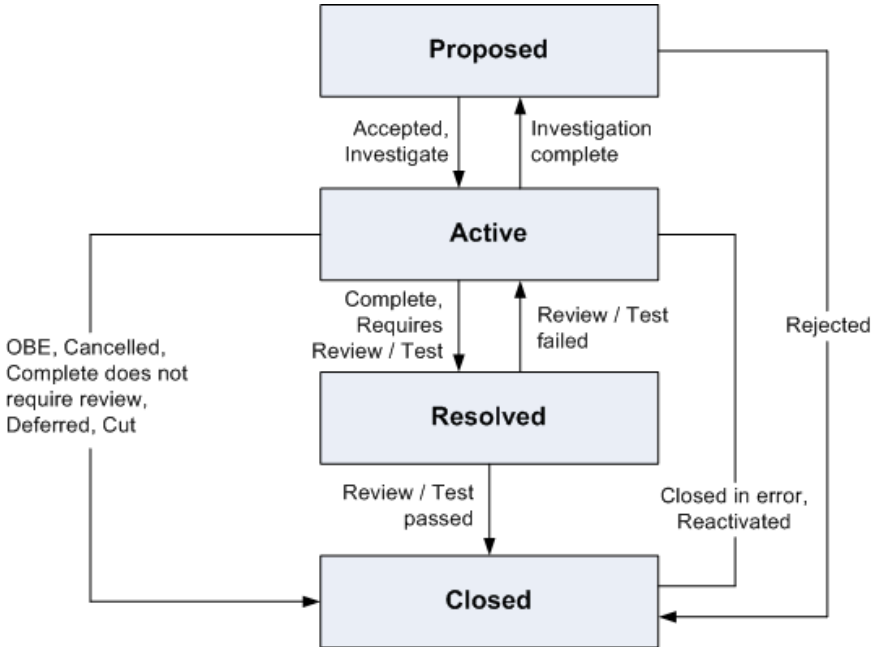
TFS Guiden mukaan työkorttien kenttiä käytetään hakujen ja projektin raportoinnin pohjalla. Osa työkorttien kentistä on tämän takia pakollisia. Työkorttien huolellinen täyttäminen helpottaa projektin edistymisen seurantaa. Kentillä täytyy siis olla kunnollisia ja ajan tasalla olevia arvoja, jotta raportointi onnistuisi. Työkortteja käytetään pääsääntöisesti Visual Studion tai TFS Web Accessin kautta, mutta niiden käsittely on mahdollista myös Excelin tai MS Projectin kautta, jolloin suuren työkorttijoukon samanaikainen editoiminen on mahdollista. (TFS Guide, 130.)

Työkortti koostuu viidestä välilehdestä. Description-välilehdellä on työkortin tarkka, vapaamuotoinen kuvaus. Työkortin muutosten historiaa kerätään History-välilehdelle. Linkitykset muihin työkortteihin, muutosjoukkoihin ja web-linkkeihin löytyvät välilehdeltä Links. Liitteisiin voi lisätä dokumentteja, kuvia ja muita eri tiedostomuotoja. Liitteitä käsitellään Attachments-välilehdellä. Details-välilehdellä kerätään tietoa Build-versiosta, joissa virhe on sekä huomattu että korjattu. (Järvinen, 252-253.)

5.6.1 Työkorttien elinkaari

Jokaisella työkortilla on oma elinkaarensa. Esimerkiksi MSF CMMI prosessimallissa Bug-tyyppisen työkortin elinkaari on nelivaiheinen:

1. Proposed
2. Active
3. Resolved
4. Closed



Kuvio 4 Bug-työkortin elinkaari MSF CMMI prosessimallissa (TFS Guide, s.108).

Closed-tilaan edetään vaiheittain Resolved ja Closed –tilojen kautta. Bug voidaan palauttaa myös vaiheen taaksepäin, jos korjaus virheeseen ei ole tuonut haluttua muutosta, jolloin se palautetaan Active-tilaan ja annetaan uudelleen tehtäväksi, usein samalle henkilölle. (TFS Guide, 108.)

Visual Studio 2008 käsikirjan mukaan Bug on New-tilassa, kun se on perustettu eikä henkilö, jolle työkortti on annettu tehtäväksi (assigned to), ole vielä aloittanut tutkimaan työkortin kuvaamaa virhesekvenssiä. Active-tila tarkoittaa, että Virhekirjaus on työn alla. Resolved-tila tarkoittaa, että Virhekirjaus on korjattu ja korjatut tiedostot on tuotu versionhallintaa. Closed-tila tarkoittaa, että korjattu virhekirjaus on verifioitu. Bugin asettaa Closed-tilaan testaaja tai henkilö, jolle on annettu tehtäväksi tarkistaa uuteen buildiin tehdyt muutokset. Virhekirjaus tulee siis verifioida testiympäristössä eikä kehitysympäristössä. (Järvinen, 252.)

Työkortit pitävät yllä historiatietoa niihin tehdyistä muutoksista. Historiatietoon tallentuu kaikki työkorttiin tehdyt muutokset, muutoksen tekijä ja aikaleima tapahtumalle. Historiatiedoista näkee mitä kenttiä on muutettu. (Järvinen, 252.)

5.6.2 Kyselyt

Visual Studiossa on mahdollista perustaa tiimikyselyjä (Team Queries) ja omia kyselyjä (My Queries). Kyselyiden avulla on mahdollista etsiä suuria joukkoja työkortteja samanaikaisesti sekä saada kokonaiskuva esimerkiksi avoinna olevista virhekirjauksista tai omista työkorteista. Tiimikyselyt ovat näkyvissä kaikille tiimin jäsenille. Omat kyselyt ovat henkilökohtaisia eikä niitä muut tiimin jäsenet näe. (Järvinen, 280.)

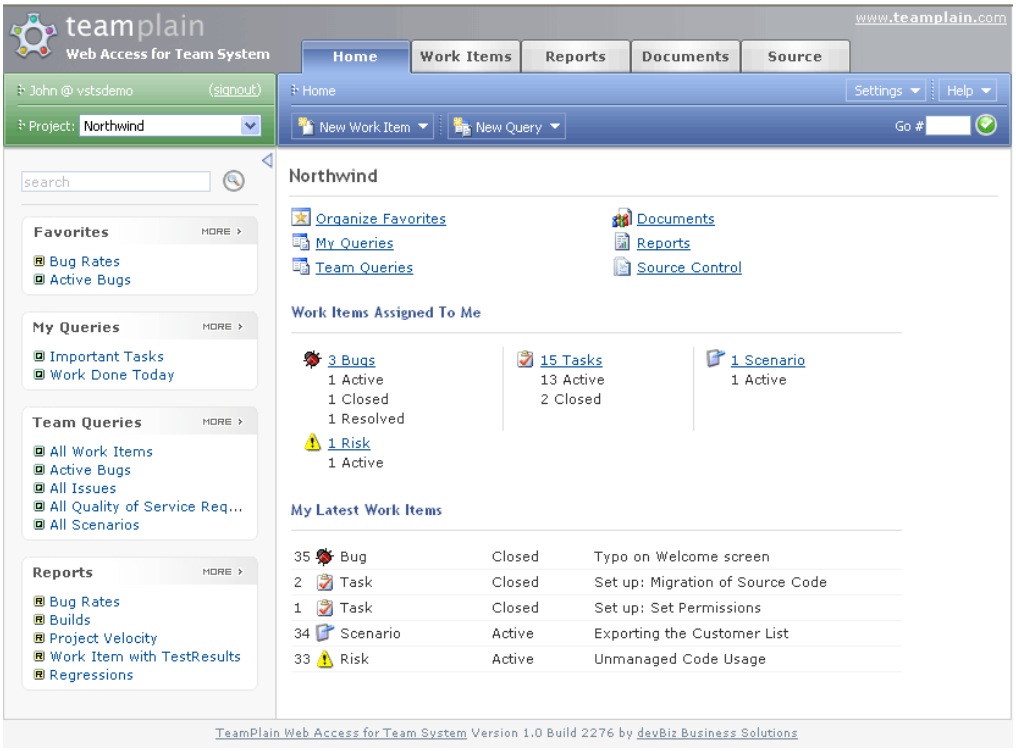
5.7 Web Access

MSDN:n mukaan Web Access on Team Foundation Serverin selainpohjainen käyttöliittymä. Web Access tarjoaa lähes kaikki hallinnolliset projektin toiminnot mitä Visual Studiokin:

- Luoda, katsella, tehdä kyselyjä työkortteihin
- Hallita projektin dokumentaatiota
- Katsella versionhallinnan tiedostoja ja niihin liittyvää informaatiota

- Hallita build-prosessia ja seurata buildin etenemistä
- Luoda ja katsella raportteja projektin tilanteesta

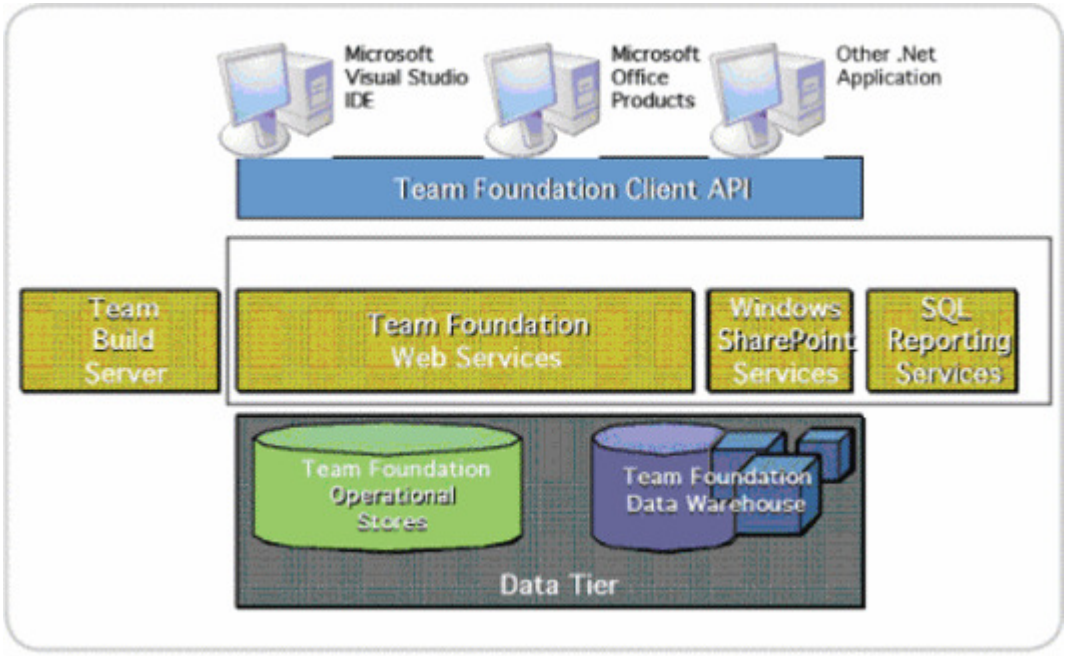
Web Accessin etusivu on muokattavissa käyttäjän haluamaksi. Etusivulla voidaan näyttää viimeiseksi muokatut työkortit, henkilölle suunnatut työkortit sekä käyttäjän itsensä tekemät. kyselyt. (Team System Web Access, 2010.)



Kuvio 5 Web Access (Team System Web Access, 2010).

6 Team Foundation Serverin tekninen kuvaus

Team Foundation Server rakentuu ASP.NET –pohjaisista web service –palveluista. Team Foundation Server tallentaa kaiken datan SQL Server –tietokantaan. Kuvassa 4.0 on VSTS:n arkkitehtuuri on jaettu kolmeen kerrokseen: Data Tier, Application Tier ja Client Tier:



Kuvio 6 VSTS arkkitehtuuri (VSTS:n rakenne, 2010).

TFS Guiden mukaan varmuuskopiointistrategiaa tulee jo miettiä Team Foundation Serverin asennuksen yhteydessä. Tietokantakerros (Data Tier) on rakennettu SQL Serverin päälle, joten varmuuskopio otetaan tietokannasta. Tiimin koko ratkaisee miten usein varmuuskopiota kannattaa ottaa (TFS Guide, 144.)

Microsoftin mukaan tietokantakerroksella ylläpidetään data työkorteista, versionhallinnasta, buildista sekä raporteista. Tietokantaan ei ole suoraan pääsyä Client-ohjelmiston kautta vaan kaikki kommunikaatio tapahtuu Web-Servicen kautta sovelluskerroksella. Web-Servicet jakaantuvat kahteen ryhmään: Team Foundation Data Services ja Team Foundation Integration Services. Data Services kommunikoi suoraan tietokantakerroksen kanssa ja ylläpitää sekä muokkaa tietokannassa olevaa dataa. Integration Services ei ole yhteydessä tietokantaan vaan tarjoaa integrointi- ja automaatiopalvelut. Palveluihin kuuluu käyttäjien hallinta, Team Foudation Serverin monitorointi ja tietoturvapalvelut. Ylimmällä kerroksella eli Client-kerroksella ovat asiakasohjelmistot, joita ovat muun muassa Microsoft Visual Studio ja Microsoft Office. (Team Foundation Server Architecture, 2010.)

7 Hajautettu ohjelmistokehitys Yritys A:ssa

Yritys A:lla on Suomessa viisi toimipistettä. Suomen toimipisteillä on yhteiset tietoverkot sekä versionhallinta. Etäyhteys-ohjelmien (Remote Desktop, NetOP, Office Communicator, Puhelinkonferenssit) ja VPN-yhteyden avulla voidaan ottaa yhteys esimerkiksi Helsingissä sijaitsevaan testilaboratorioon tai asiakaskentällä toimivaan tietojärjestelmään. Kommunikointi eri toimipisteiden välissä tapahtuu pääasiallisesti sähköpostitse, mutta usein myös puhelimitse. Muiden paikkakuntien toimipisteiden työntekijät käyvät myös useita kertoja vuodessa Helsingin toimipisteessä. Vierailut liittyvät usein projektien käynnistykseen tai yrityksen sisäisiin koulutuksiin.

Osastomme tuotteita kehitetään useissa eri maissa. Tuotteisiin ja toimialaan liittyvää osaamista on kolmella paikkakunnalla. Kasvokkaisviestintää eri maiden työntekijöiden välillä on hyvin vähän. Lähes kaikki kommunikointi tapahtuu sähköpostitse tai muiden verkossa toimivien viestimien kuten VoIP:n tai verkkoportaalien kautta.

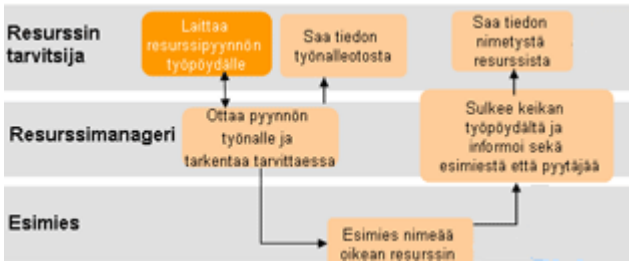
Projektit ovat usein jaettu eri maiden välille, sillä tuottamamme tietojärjestelmät ovat erittäin laajoja ja koostuvat usein kahdesta tai useammasta rinnakkain toimivasta järjestelmästä. Projektiryhmä voi koostua eri maassa työskentelevistä henkilöistä tai projekti voidaan jakaa kahteen erilliseen projektiin, jolloin yksi projektiryhmä toimii esimerkiksi Suomessa ja toinen ryhmä toisessa maassa.

Työskentely osastollamme on projektiluonteista: projektipäälliköt varaavat työntekijöitä tarpeen ja tilanteen mukaan. Projektin vetäjänä toimii projektikohtainen projektipäällikkö. Työntekijällä on kuitenkin hallinnollinen esimies eli ryhmävetäjä. Työntekijä voi siis olla useammassa projektissa samanaikaisesti, ja työtehtävät saattavat vaihdella henkilöstöllä hyvinkin paljon. Projekteissa käytetään erilaisia rooleja. Ohjelmistosuunnittelijan tittelillä työskentelevä henkilö saattaa toimia arkkitehtinä projektissa, jossa toimintaympäristö on hänelle erittäin tuttu, ja hänellä on vankka kokemus kyseisen toimintaympäristön piirteistä.

Työntekijöiden varaaminen projektiin tapahtuu resursointiprosessin avulla. Yrityksen sisäisen materiaalin mukaan resursoinnin tavoitteena on:

- Löytää ja ohjata sopivat osaavat henkilöt projekteihin.
- Varmistaa asiantuntijoiden riittävä, mutta kohtuullinen asiakastyön osuus ja pyrkiä tasaamaan eri henkilöiden välisiä asiakastyön määrän vaihteluja.

Kuviossa 7 on esitetetty yrityksen resursointiprosessi. Yrityksen sisäinen materiaalin mukaan resursointiprosessin keskeiset elementit ovat resurssien ohjaus yksikön sisällä, yksiköiden välinen yhteistyö, rekrytointi sekä osaamisen kehittäminen.

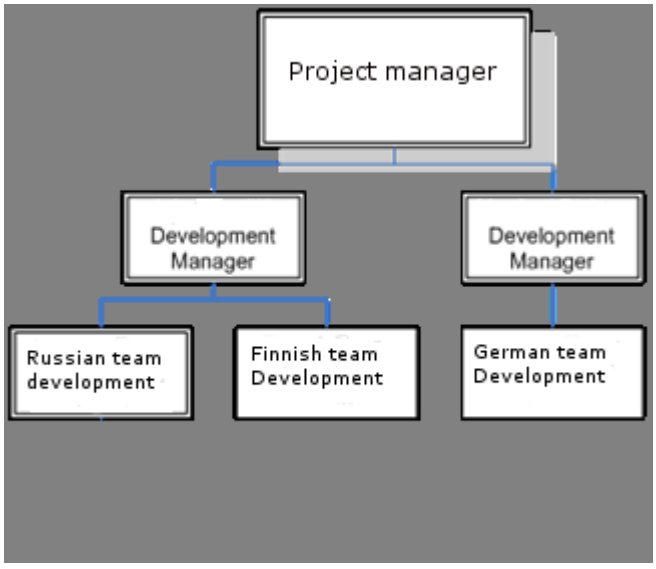


Kuvio 7 Resursointiprosessi

Resurssin tarvitsija on yleensä projektipäällikkö, joka hakee projektiin resurssimanagerin kautta tiettyyn tehtävään, esimerkiksi suunnittelutyöhön, oikean henkilön. Asiantuntijaryhmän esimies valitsee oikean henkilön, joka nimetään projektiin tiettyyn tehtävään.

8 Projekti X, Lähtötilanne

Saksassa toimiva vähittäiskaupanalan yritys tilasi Yritys A:lta ASP .NET-pohjaisen tietojärjestelmän. Tilattu tietojärjestelmä koostui kahdesta rinnakkain toimivasta ohjelmistosta. Tietojärjestelmän kehitys jaettiin Saksan, Suomen ja Venäjän toimipisteiden välille. Saksan toimipisteen tehtävänä oli suunnitella ja toteuttaa tietojärjestelmän businesskriittinen ohjelmisto. Suomen ja Venäjän tiimien tuli taas suunnitella ja toteuttaa taustajärjestelmät eli hallinnolliset toiminnot ja operatiivista tietojärjestelmää tukevat taustasovellukset, kuten käyttäjien hallintaan, datan siirtoon ja järjestelmän avaukseen liittyvät toiminnot. Suomen osalta projekti käynnistyi jo vuonna 2008, jolloin aloitettiin asiakkaan vaatimusten määritteleminen toiminnallisella ja teknisellä tasolla. Suomen apuna toimi toteutuspuolella Venäjän toimipiste, eli maantieteellisesti tietojärjestelmän suunnittelua, kehitystä ja testaamista toteutettiin kolmessa eri maassa. Saksan toimipiste toimi asiakasrajapinnassa ja oli samalla päävastuussa tietojärjestelmän toteuttamisesta, käyttöönotosta ja kommunikaatiosta asiakkaan kanssa.



Kuvio 8 projektin organisaatio

Projektissa otettiin ensimmäistä kertaa käyttöön Team Foundation Server 2005 sekä TFS:n käyttöä tukeva iteratiivinen prosessimalli. TFS ja sen työkalut olivat Suomen ja Venäjän tiimien jäsenten yhteisessä käytössä, joko Visual Studion tai Web Accessin kautta. Suomen ja Venäjän tiimien projektipäällikkö perusti yhdessä arkkitehtien kanssa TFS:ään kehitys- ja testaustehtävät, joista osa annettiin suoraan tietyille henkilöille hoidettavaksi. Isoin osa tehtävistä jätettiin kuitenkin niitä varten perustettuun Backlogiin, josta tiimin jäsenet poimivat tehtäviä vastuulleen prioriteettinumeron mukaisesti.

Team Foundation Server käyttökokemukset –kyselyn mukaan suunnittelijat ja versionhallintavastaava käyttivät Team Foundation Serveriä sekä Visual Studion ja Web Accessin kautta projektissa X. Testaajat ja projektipäällikkö käyttivät pääasiallisesti Team Foundation Serveriä vain Web Accessin kautta.

Projekti jaettiin neljään iteraatiovaiheeseen, joista jokaisesta toimitettiin asiakkaalle ns. drop eli tiettyyn asteeseen toteutettu toimiva osa järjestelmästä. Iteraatiovaihe saattoi sisältää useita buildeja eli testausvalmiita väliversioita, joita testattiin virhekorjausten ja tehtävien valmistuessa. Iteraatiovaiheet jaettiin asiakkaan päävaatimusten perusteella. Esimerkiksi ensimmäiseen toimitukseen oli toteutettu perustuotteesta tulevat järjestelmältä vaaditut

perustoiminnot, jotka muokattiin asiakkaan järjestelmää varten. Toisessa ja kolmannessa dropissa olivat mukana järjestelmän raportit ja raportointiin liittyvät ominaisuudet. Neljäs drop käsitti loput järjestelmän toiminnot, jotka sisälsivät erityispiirteitä, ja joiden toiminnallista ja teknistä määritystä tarkennettiin projektin edetessä. Projektille asetettiin myös iteraatioihin perustuvat virstanpylväät (project milestones), jotka on esitelty taulukossa 3:

NIMI	KOHTEET	HYVÄKSYMISKRITEERIT
Projekti hyväksytään	Sopimus, Projektisuunnitelma, Dokumentaatio	Projektin sopimusasiat hyväksytään Yritys A:n ja asiakkaan välillä. Asiakas hyväksyy toiminnallisen ja teknisen määrityksen
Perustoiminnot	Järjestelmän perustoiminnot	1. drop toimitetaan asiakkaalle, asiakas hyväksyy toiminnallisuuden (vaikka tekisikin yksittäisiä korjauspyyntöjä)
Perusraportit	Järjestelmän tuottamat perusraportit	2. drop toimitetaan asiakkaalle, asiakas hyväksyy raportit ja raportteihin liittyvän toiminnallisuuden
Lisäraportit	Järjestelmän tuottamat lisäraportit	3. drop toimitetaan asiakkaalle, asiakas hyväksyy raportit
Erikoistoiminnot	Järjestelmän erikoispiirteiset toiminnot	4. drop toimitetaan asiakkaalle, asiakas hyväksyy toiminnallisuuden
Projektin luovutus	Lähdekoodien luovutus, dokumentaation luovutus, toimitustiedote	Asiakkaan hyväksymistestien läpäisy, dokumentaation hyväksyminen
Projekti suljetaan	Projektien sulkeminen, arviointi, testausraportti, loppuraportti	-

Taulukko 3 Projektin virstanpylväät (project milestones)

Yksi iteraatiovaihe kesti noin seitsemän viikkoa. Päättyneen iteraatiovaiheen jälkeen katselmoitiin saatu tulos ja tehtiin asiakkaalle toimitustiedotus. Toimitustiedotteessa lueteltiin valmistuneet osa-alueet ja listattiin niin havaitut kuin suljetut sekä edelleen aktiiviset virhekirjaukset (Defect).

TFS:n käyttöä varten oli tehty koulutussuunnitelma, jota toteutettiin luentomaisten koulutustilaisuuksien muodossa. Suomen tiimiä varten koulutus oli jaettu kahteen osaan. Ensimmäisellä kerralla kerrottiin Team Foundation Serverin työkaluista, ja jälkimmäisellä kerralla aiheena oli uusi prosessimalli, jota hyödynnettiin TFS:n käytössä. Team Foundation Server käyttökokemukset –kyselyyn vastanneista lähes kaikki olivat olleet TFS-koulutuksessa.

Projektipäällikkö seurasi projektin etenemisestä viikoittaisten tiimipalaverien sekä tehtävien tilapäivitysten avulla. Viikoittaiset tiimipalaverit pidettiin Suomessa Helsingin toimipisteessä. Muissa Suomen toimipisteissä toimivat tiimin jäsenet sekä satunnaisilla kerroilla myös Venäjän tiimi oli mukana viikkopalavereissa LiveMeetingin eli puhelinkokouksen mahdollistavan järjestelmän avulla.

Projekti oli niin sanottu Add-on -projekti, mikä tarkoittaa, että asiakasta varten tehtävä ohjelmakoodi liitetään jo olemassa olevan perustuotteen päälle. Osa järjestelmän vaatimuksista oli jo toteutettu perustuotteen kehityksen yhteydessä. Näihin toimintoihin lisättiin vain asiakaskohtaiset vaatimukset. Osa toiminnoista oli täysin uusia, ja ne vaativat sovituksen perustuotteen toiminnallisuuden kanssa.

8.1 Hajautetun kehitystyön tuomat haasteet projektissa

Kolmeen maahan jaettu projekti tiedettiin etukäteen lähtökohdiltaan haastavaksi. Kasvokkaisviestinnän vähäisyys projektiryhmien välillä vähensi viestinnän tehokkuutta ja riskiksi tiedettiin etukäteen ryhmien välisessä viestinnässä tapahtuvat viiveet. Projektin kansainvälisyydestä johtuvia haasteita olivat kulttuuriset erot ja kielelliset vaikeudet. Myös projektin laajuudesta johtuen dokumentaation hallinta tiedettiin haastavaksi ennen projektin alkua. Lisäksi kahden rinnakkain toimivan järjestelmän sulauttaminen projektin loppuvaiheessa aavistettiin monimutkaiseksi toteuttaa, ja sitä varten oli varattu reilusti kalenteriaikaa integraatio- ja regressiotestaamiselle ja korjauksien tekemiseen.

8.1.1 Viestintä ja vastuun jako

Virtuaalitiimien avulla saatiin eri asiantuntijoiden tiedot ja taidot käyttöön maantieteellisestä sijainnista huolimatta. Etäisyys voitiin ylittää Team Foundation Serverin tarjoamilla työkaluilla ja yhteisellä versionhallinnalla.

Viestintä toimi hyvin ryhmien sisällä. Suomen tiimi piti viikoittaisia palavereja säännöllisesti ja mahdollisti avoimen keskustelun ryhmän sisällä. Projektipäällikkö koosti jokaisesta palaverista tiivistelmän, joka jaettiin sähköpostin välityksellä tiimin jäsenille. Suurin osa Suomen ryhmän jäsenistä toimi samassa toimitilassa, mikä edesauttoi sovelluskehityksen ja testaamisen edistymistä. Tiimin jäsenet pystyivät pyytämään apua toisiltaan tarvittaessa ja mahdolliset epäselvyydet määritysdokumenteissa pystyttiin ratkaisemaan keskustelemalla.

Ryhmien välinen viestintä ei toiminut toivotulla tavalla. Kasvokkaisviestintä tiimien välillä ei ollut mahdollista johtuen pitkistä etäisyyksistä. Ryhmien välistä viestintää varten oli perustettu projektin Sharepoint-palveluun uutisryhmätyylinen keskustelupalsta (Team Discussion) yleisistä projektiin liittyvistä asioista. Palstalle oli tarkoituksena kerätä tietoa projektissa käytetyistä toimintatavoista, TFS:ään sekä Visual Studioon liittyvistä kysymyksistä ja projektiin liittyvistä tapahtumista sekä aikatauluista. Keskustelupalsta jäi kuitenkin hyvin vähäiselle käytölle: viestejä kertyi palstalle alle kymmenen kappaletta lähinnä projektipäällikön ja kahden arkkitehdin toimesta. Ryhmien välinen viestintä tapahtuikin lähinnä sähköpostitse. Tämä hidastutti tiedon saamista ja levittäytymistä. Pahimmillaan sähköpostiviestintä kätki kaikille tärkeän tiedon vain viestiketjussa mukana olevien jäsenten sähköpostilaatikoihin. Saksan tiimi toimi asiakasrajapinnassa, joten tieto välittyi aina heidän kauttaan sekä Suomen että Venäjän tiimille. Tieto liikkui usein Venäjän tiimille vasta Suomen tiimin kautta. Yhden asian ratkaisemiseksi ja kysymykseen suoran vastauksen saamiseksi saattoi kulua päiviä. Sähköpostiviestintää hidastutti myös toimiminen vieraalla kielellä, joka johti siihen, että yhdenkään tiimin jäsenistä kukaan ei toiminut kotikielellään.

Henkilöstölle tehtyjen teemahaastattelujen mukaan Sharepointissa oleva projektin keskustelupalsta jäi vähäiselle käytölle, koska vastauksen saaminen esitettyyn kysymykseen keskustelupalstan kautta oli hidasta, sillä palstaa ei käytetty aktiivisesti. Esimerkiksi yksikään haastelemistani Suomen tiimin jäsenistä ei käynyt palstalla päivittäin. Ongelman voisi ratkaista siten, että aina uuden viestin saapuessa keskustelupalstalle, Sharepoint lähettäisi automaattiviestin jokaisen tiimin jäsenen sähköpostiin. Haastateltavat pitivät sähköpostin käyttöä helpompana ja tutumpana tapana viestiä. Sähköpostin avulla pystyy myös valitsemaan vastaanottajat, joille kysymys suunnataan. Maantieteellisesti hajautetuissa projekteissa on myös syytä kerätä tärkeitä kirjeenvaihtoja ylös kaikkien nähtäville. Sharepointin keskustelupalsta olisi

oikea paikka säilyttää kyseisiä viestinvaihtoja. Tällä menetelmällä tärkeää tietoa ei kätketä ihmisten sähköpostilaatikoihin vaan saadaan kerralla kaikkien tiimin jäsenten näkyville. Työkaluun tai muuhun kaikille yhteiseen komponenttiin liittyvien ongelmien, ja etenkin niiden ratkaisujen, kuvaus vähentää turhan työn määrää.

Projektin koordinointi vaati paljon aikaa kehityspäälliköiltä (development manager). Kalenteriajassa mitattuna projekti kesti useita kuukausia, ja Suomen tiimissä henkilöstöä toimi samaan aikaan 4-20 henkilöä. Team Foundation Serveriin perustetut tehtävät oli aikataulutettava ja projektissa tapahtuvista muutoksista, esimerkiksi juuri aikatauluja koskevista muutoksista, oli tiedotettava myös tiimin jäsenten hallinnolliselle esimiehelle.

8.1.2 Dokumentaation hallinta

Projektin dokumentaatiota ylläpidettiin Team Foundation Serverillä. Dokumentaatioon oli pääsy Visual Studion, TFS Web Accessin ja Sharepoint-palvelun kautta. Kaikilla tiimin jäsenillä oli oikeus uloskirjata (check-out) dokumentteja ja tehdä niihin muutoksia.

Dokumentaation hallittavuutta vaikeutti dokumentoinnin versioituminen. Taustajärjestelmän toiminnallisuutta määrittelevä dokumentaatio jakaantui niin sanotusti alkuperäiseen, Saksassa tuotettuun, dokumentaatioon ja uuden prosessimallin mukaiseen, Suomessa tuotettuun, dokumentaatioon. Sama dokumentaatio tuotettiin siis kahteen kertaan. Syynä ratkaisuun oli halu tarkentaa dokumentaation kuvaamaa taustajärjestelmän toiminnallisuutta keräämällä siitä uuden prosessimallin mukaisesti tietoa. Dokumentaatio projektissa jakaantui taulukon 4 mukaisesti.

Segmentti	Dokumentit	Tehtävät	Työtavat
Alkuperäinen toiminnallinen määritys (Saksan tiimin tuottama)	Use-Case Specifations, Layout Samples	Määrittelee toteutettavan tietojärjestelmän, havainnollistaa	-
Uuden prosessimallin mukainen toiminnallinen määritys (Suomen tiimin tuottama)	Use-Case Specifations, Use-Case Work Flow, Graphical User Interface Specification	Määrittelee toteutettavan tietojärjestelmän, kuvaa käyttäjät, käyttötapaukset sanallisesti ja kaavioiden avulla	Dokumentit tuotettiin Saksan tiimin kirjoittamista toiminnallisen määrityksen kuvauksista arkkitehtien ja suunnittelijoiden toimesta
Tekninen määritys (Suomen tiimin tuottama)	Architechture Design, Component Design	Kuvaa järjestelmä arkkitehtuurin, kuvaa arkkitehtuurin komponentit, järjestelmän eri toimintojen teknisen rakenteen kuten käytettävät sql-proseduurit, web configin jne..	Dokumentit tuotettiin perustuotteen ja toiminnallisen määrityksen pohjalta
Testaus (Suomen tiimin tuottama)	Test Plan, Unit test documnents, Test Report	Kuvaa testitapaukset, testitulokset, yksikkötestien tulokset	Dokumentit tuotettiin perustuotteen testisuunnitelman ja toiminnalisen sekä teknisen määrityksen pohjalta

Hallinto (Suomen ja Saksan tiimin tuottama)	Project management plan, Project plan, Resources, Work Estimates, Iteration Handling, Project risks	Kuvaa projektin hallinnan, käytettävät resurssit, iteraation hallinnan, projektissa havaitut riskit	Dokumentit tuotettiin projektin johdon tapaamisista
Ohjeet (Suomen tiimin tuottama)	Installation, Defect Guide, Task Guide, Work Hour reporting guide, Quick start Guide	Tukea kehitys&testaustyötä projektissa. Ohjeet virtuaaliympäristön konfigurointiin	Tuotettiin arkkitehtien ja suunnittelijoiden toimesta
Toimitustiedotteet (Suomen ja Saksan tiimin tuottama)	Delivery Notes, Defect List, Change List	Kuvaa toimitetun dropin sisällön,aktiiviset&suljetut virhekirjaukset	Versionhallintavastaavan ja projektipäällikön toimesta

Taulukko 4 Projektin virstanpylväät

Toiminnalliseen ja etenkin tekniseen määritykseen jouduttiin tekemään kesken toteutuksen paljon muutoksia. Osa toiminnoista ja toimintoihin liittyvistä poikkeustilanteista tarkentui projektin edetessä, mistä johtuen dokumentaatiota jouduttiin päivittämään usein. Kaikista muutoksista tehtiin TFS:ään Muutospyyntö-työkortti (Change). Näin asiakkaan puolelta tulleet muutospyynnöt aikaisemmin sovittuun toiminnallisuuteen pystyttiin huolehtimaan projektin laskutuksessa.

8.2 Projektin roolit

Suomen tiimissä projektissa toimi henkilöitä viidessä eri roolissa: kehityspäällikkönä, arkkitehteina, versionhallintavastaavina, kehittäjinä ja testaajina. Kehityspäällikkö (Development manager) toimi projektin vetäjänä ja vastasi Suomen ryhmän tuloksesta projektipäällikölle. Varsinaiseen toteutustyöhön tai toiminnalliseen sekä tekniseen määritykseen kehityspäällikkö ei puuttunut, vaan hänen tehtävänään oli välitavoitteiden suunnittelu, tehtävien suunnittelu ja projektin etenemisen seuranta.

Arkkitehdit suunnittelivat järjestelmän tärkeimmät osat korkealla tasolla. Heidän tehtäviinsä kuului toiminnallisen ja teknisen määrityksen lisäksi toteutukseen osallistuminen ja toteutuksen tukeminen. Arkkitehdit vastasivat myös työmääräarvioiden laatimisesta sekä olivat projektipäällikön apuna tehtävien perustamisessa.

Versionhallintavastaava piti huolta versionhallinnassa olevasta koodista ja koodin buildaamisesta, testijärjestelmän perustamisesta, testijärjestelmän päivityksestä ja droppien toimituksesta asiakkaalle. Hänen tehtävänään oli toisin sanoen koota versionhallintaan siirretyt koodit toimivaksi järjestelmän osaksi.

Kehittäjät osallistuivat toteutuksen lisäksi toiminnalliseen ja tekniseen määritykseen. Kehittäjän vastuisiin kuuluivat myös katselmointi, yksikkötestit ja virhekirjausten korjaaminen sekä mahdollisten muutospyyntöjen toteuttaminen. Katselmoinnilla tarkoitetaan määriteltyä prosessia noudattavaa dokumentin läpikäyntiä.

Testaajat tarkistivat järjestelmän toimivuuden. Testaajat tekivät myös toiminnallisen ja teknisen määrityksen pohjalta testisuunnitelman. Testaajat tekivät virhekirjauksia (Defect) TFS:ään ja raportoivat mm TFS:n kautta testijärjestelmässä havaituista virheistä. Testaajat

tekivät myös toiminnallisen ja teknisen määrityksen pohjalta testisuunnitelman. Projektin päätyttyä vastuu-alueeseen kuului myös testausraportin laadinta.

9 Team Foundation Serverin käyttö osaston projekteissa

Osaston tapa käyttää TFS:ää poikkeaa Järvisen esittelemistä MSF *for CMMI Process Improvement* sekä *MSF for Agile Software Development* -prosessimalleista. Tiimiprojekti voi sisältää useita aliprojekteja. Toisin sanoen, samaan tiimiprojektiin on niputettu useita asiakkuuksia johtuen asiakkuuksien laajuudesta, ja samaan asiakkuuteen voi kohdistua useita samanaikaisia projekteja. Tällöin työkortteihin on lisätty kentät Customer ja Project erottamaan projektit toisistaan.

Perustuotteen lähdekoodeja ei toistaiseksi pidetä Team Foundation Serverin versiohallinnassa, vaan versiohallintaa käytetään toistaiseksi vain Add-on -projektien lähdekoodien hallintaan. Kaikissa projekteissa Team Foundation Serverin versionhallintaa ei tulla vielä hyödyntämään. Vanhoja järjestelmiä ylläpidetään kentällä ja tästä syystä niitä kehitetään edelleen. Näiden järjestelmien lähdekoodit ovat edelleen Visual Source Safe:ssa (VSS) tai Telelogic Synergyssa. Näin ollen näihin järjestelmiin liittyvissä projekteissa TFS:stä tullaan hyödyntämään vain projektinhallintaominaisuudet. VSS on Microsoftin edellinen versionhallintatuote. Mikään ei kuitenkaan estä lähdekoodien siirtämistä TFS:n versionhallintaan, ja myöhemmin tulevaisuudessa lähdekoodit tullaankin sinne siirtämään.

9.1 Työkorttien käyttö

Osastolla on käytetty lähinnä tehtävä eli Task-, Virhekirjaus eli Defect- ja Muutospyyntö eli Change -tyyppisiä työkortteja. Työkorttien pohja eli eri työkorttityypeissä käytetyt kentät ja kenttien valintalistat ovat käyttämämme prosessimallin mukaisia.

Työkortteja muokataan Visual Studiossa vastaamaan kulloisenkin projektin tarpeita. Eri projekteissa saattaa esiintyä siis toisistaan hieman erilaisia työkortteja. Erot ovat erilaisissa kentissä ja kenttien sisällöissä. Esimerkiksi Customer- eli Asiakas-kentän tarjoama sisältö on aina projektikohtainen.

Defectin, Changen ja Taskin perustamisesta pidettiin lähes kaikkien vastanneiden toimesta helppona ja nopeana: pakollisia kenttiä oli täytettävänä vain kaksi. Useiden vastanneiden mielestä pakollisia kenttiä olisi kuitenkin tarvittu enemmän. Lisäarvoa tuottavan tilastodatan saamiseksi raporteille, tulisi pakollisiksi kentiksi lisätä ainakin Severity eli kiireellisyysaste, Iteration Path eli iteraatiovaihe ja Application eli sovellus tai sovelluksen osa. Kenttien pakollisuutta perusteltiin raportoinnin ja kyselyiden helpottamiseksi.
Kyselyn lisäkommenteissa harmiteltiin, että työkortin kopioiminen ei ollut mahdollista. Tämä olisi nopeuttanut suuren työkorttimäärän tekemistä kerralla. Suuren työkorttimäärän tekeminen ei ole mahdollista Web Accessin tai Visual Studion kautta, mutta on mahdollista toteuttaa esimerkiksi Excelin kautta.

Risks, Requirements ja Quality of Service –työkortit eivät ole olleet käytössä osastomme projekteissa. Yrityksen ohjeistus kehottaa käyttämään kyseisiä työkortteja, mutta niitä ei ole hyödynnetty. Risks-työkortteja ryhmän jäsenet voisivat vapaasti perustaa, kun huomaavat projektiin liittyvät riskin. Näin ollen tieto riskistä saadaan kaikkien esille eikä tieto kätkeydy sähköposteihin tai kasvokkain käytyihin keskusteluihin. Requirements-työkortteihin olisi kerätty tietoa toteutettavan toiminnon vaatimuksista. Työkortin sisältö perustuisi toiminnalliseen määritykseen. TFS Guiden mukaan Requirements-työkortti muistuttaa Tehtävää (Task), mutta olisi sisällöltään konkreettisempi eli siinä olisi kuvattu suoraan ominaisuudet, joita toiminnon on pidettävä sisällään. Quality of Service -työkorttiin kerätään järjestelmään liittyviä laatuvaatimuksia kuten suorityskyky ja tietoturvallisuus.

9.1.1 Task

Kehityspäällikkö perusti tehtävät Team Foundation Serveriin. Yhden toiminnon toteutusta varten oli kolme eri Tehtävä-työkorttia eli kehitystehtävät jaoteltiin kolmeen tasoon:

1. Moving Use Case from specification_agreed to verified
2. Realize Use Case
3. Develop Components

Ensimmäinen vaihe oli kahta muuta tehtävää alustava tehtävä. Kehittäjän tehtävänä oli siinä tutustua mm toiminnon toiminnalliseen määritykseen (Use-Case Specifations, Use-Case Work Flow, Graphical User Interface Specification). Tehtävässä kerrottiin myös tehtävän osa-alueet (Developer tasks: Analyze, Design, Document, Implement, Unit test the function) sekä tehtävässä kehitettävän toiminnon laajuus (esim. 3 henkilötyöpäivää). Realize Use Case -tehtävässä kehittäjä tunnisti implementoitavat komponentit, kirjoitti teknisen määrityksen (Component Design) toiminnosta sekä yksikkötestisuunnitelman (Unit Test). Develop Component - tehtävässä kehittäjä toteutti itse toiminnon. Tehtävän osa-alueet ovat Create the report, Execute Unit tests, Update design document Maintain source control.

Testaustehtävät jaoteltiin yhdelle tasolle:

1. Execute tests

Tehtävä sisälsi testisuunnitelman mukaisten testitapausten läpikäynnin. Alun perin ajatuksena oli, että testaustehtävät olisi jaoteltu kolmelle eri tasolle kuten kehitystehtävätkin. Ajatuksesta luovuttiin kun huomattiin, että kahdesta alustavasta tehtävästä ei olisi ollut lisäarvoa kuin kehitystehtävissä.

Projektissa käytettiin myös hallinnollisia tehtäviä, jotka oli lähinnä tarkoitettu projektipäällikölle. Kehityspäällikölle suunnatuissa työkorteissa saattoi olla tehtäviä kuten ”Perusta tehtävä toiminnolle X”, ”Selvitä asiakkaalta toiminnon Y tietty poikkeustilanne B” tai ”Iteraatio Vaihe 1 katselmointi”.

Vaikka tehtävät olivat jaoteltu erikseen eri rooleille, jaottelu tapahtui vain otsikkotasolla: otsikko kuvasi, oliko kyseessä testaus-, kehitys- vai hallinnollinen tehtävä, mutta työkortissa ei ollut kenttää, mikä kuvaisi tehtävän tyypin. Raporteissa (reports) ja kyselyissä (query) ei näin ollen pystynyt erottelemaan tehtävätyyppejä toisistaan. Jotta erottelu onnistuisi raporteille tai kyselyille, tulee tehtäväkorttiin perustaa uusi kenttä, tehtävätyyppi (Task Type). Osastomme projekteja varten riittäisi, että kentän arvoiksi voitaisiin valita Develop, Testing tai Management.

Tehtäville oli määritelty viisiosainen elinkaari. Start-tilassa oleva tehtävä on uusi ja aloittamaton tehtävä. Tehtävä siirretään Defined-tilaan, kun tiimin jäsen henkilö siirtää tehtävän itselleen ja aloittaa tehtävän analysoinnin. Estimated-tilaan siirrytään, kun tiimin jäsen hyväksyy työkortissa esitetyn työmääräarvion, joka löytyy työkortin Details-välilehdeltä. Tehtäväkortti asetetaan Scheduled-tilaan, kun tiimin jäsen tietää, milloin hän pystyy aloittamaan tehtävän suorittamisen. Kun tehtävä on aloitettu ja käynnissä, tehtäväkortti siirretään In Progress -tilaan. Tehtävä siirretään Completed-tilaan, kun toiminto on toteutettu, dokumentaatio päivitetty vastaamaan toteutusta ja lähdekoodit sekä päivitetty dokumentaatio on siirretty versionhallintaan. Testaustehtävä on valmis, kun tehtävässä annettu testisuunnitelman mukaiset testitapaukset on suoritettu. Huomioitavan arvoista on, että testaustehtävä on valmis, vaikka kaikki testitapaukset eivät menisikään onnistuneesti läpi. Testaustehtävään tulee linkittää testitapauksien ajossa huomatut ja kirjatut virhekirjaukset (Defect). Projektissa tuli vastaan tilanteita, jolloin kaikkia testitapauksia ei ollut mahdollista suorittaa. Esimerkiksi Saksan tiimin toteuttamasta dropista saattoi puuttua taustajärjestelmän tietyn toiminnon vaatima komponentti. Tällöin testaustehtävä jätettiin In Progress -tilaan. TFS:n raporteille ja projektin etenemisen seuraamisen kannalta tämä näkyi vääristävänä tekijänä. Asia voidaan ratkaista lisäämällä tehtävän elinkaareen yksi tilaa lisää: ”Waiting”. Tehtäväkortti tulisi voida asettaa Waiting-tilaan, kun työkortin käsittelijästä riippumattomasta syystä tehtävä on jätettävä kesken. Tilanteen ratkaisemiseksi voidaan ottaa mallia myös MSF CMMI -prosessimallista. MSF CMMI -prosessimallissa on käytössä työkorttityyppi Issue. Issue -työkorttiin kuvataan tilanne, joka estää tehtävän jatkamisen. Issue-työkorttiin linkitetään työkortit, jotka ovat kyseisestä syystä keskeytetty.

Tehtävä-työkortille oli määritelty vain yksi pakollinen kenttä, joka oli kortin otsikko. Päätöksen taustalla oli ajatus siitä, että työkortteja pystytään luomaan nopeasti. Edellisessä projektinhallintaohjelmistossa Telelogic Synergyssä oli koettu rasitteeksi monen pakollisen kentän täyttäminen. Muut projektin tehtäväkorteissa käytettävät kentät olivat:

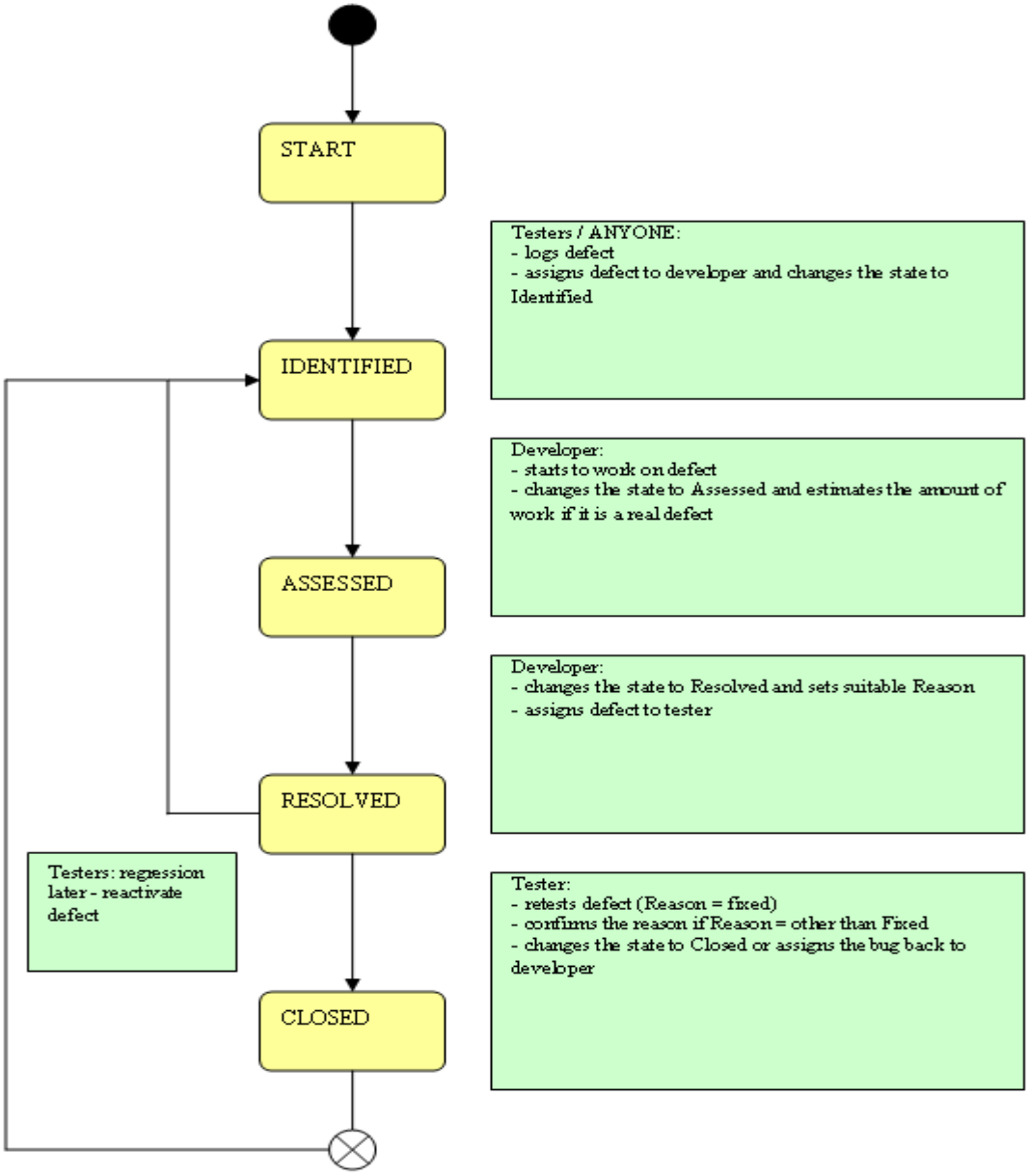
- Current State
- Area Path
- Iteration Path
- Assigned To
- Priority
- Estimated Effort

Description-välilehdelle kirjoitettiin vapaamuotoisesti tehtävän tarkka kuvaus. Area Path -kenttään annettiin valintalistasta toiminnon nimi, johon tehtävä liittyi. Iteration Path -kentän arvot olivat projektissa käytettyjen iteraatiovaiheiden mukaisia. Vaihtoehdot ovat Startup, Basic Functions, Basic Reports, Additional Reports, Special Functions ja Handover. Tehtävätyökorteille oli määritelty Priority-kenttä, jota kuvattiin järjestysluvulla. Mitä pienempi järjestysluku, sitä suurempi prioriteetti tehtävällä on. Projektipäällikkö siirsi tärkeimmät tehtävät työntekijöille. Muut tehtävät siirrettiin TFS:n Backlogiin, joka on kokoelma aloittamattomia työkortteja. Backlog on hyvä työkalu, mutta sen sisältöä etenkin kehittäjät unohtivat seurata tarpeeksi usein. Projektissa Backlogiin liitetyt tehtävät unohtuivat usein ja projektipäällikkö joutui huomauttamaan niistä.

9.1.2 Defect

Virhekirjaukset ovat kaikista työkorttityypeistä eniten käytössä. Virhekirjauksia tehtiin pääasiassa integraatio- ja regressiotesteissä testaajien toimesta, mutta kaikilla tiimin jäsenillä oli velvollisuus kirjata virhekirjauksia ohjelmavirheen huomatessaan. Virhekirjaus kannattaa kirjata myös tilanteissa, joissa henkikö ei ole varma onko kyseessä ohjelmavirhe vai ohjelman piirre. Virhekirjaus voidaan hylätä, jos se osoittautuu aiheettomaksi. Tällöin kuitenkin kaikkien näkyville tulee kyseisestä tapauksesta tieto esille, eikä samaa asiaa jouduta jatkossa ihmettelemään toisen henkilön toimesta.

Virhekirjauksilla on kuvan 9 mukainen elinkaari.



Kuvio 9 Virhekirjauksen elinkaari projektissa

Elinkaari koostuu viidestä vaiheesta. Testaaja tai kuka tahansa tiimin jäsen luo virhekirjauksen tekemästään havainnosta ja ohjaa sen ratkaistavaksi projektitiimissä toimivalle suunnittelijalle. Virhekirjaus sisältää pääasiallisesti vain yhden ohjelmavirheen. Näin virhekirjauksen elinkaarenhallinta helpottuu. Jos virhekirjaus sisältää useamman ohjelmavirheen, elinkaaren hallittavuus vaikeutuu tilanteissa, joissa osa virheistä virhekirjauskortissa on korjattu, mutta osa virheistä toistuu edelleen. Sharepoint oli konfiguroitu lähettämään automaattiviesti henkilölle, jolle kohdistetaan uusi Virhekirjaus. Näin varmistettiin, että henkilö oli tietoinen testijärjestelmässä tapahtuneesta virheestä. Virhekirjaukset kohdistuvat järjestelmän tiettyyn osaan ja vaiheeseen Area Path ja Iteration Path -kenttien avulla. Muita kenttiä ovat:

- Current State
- Assigned To
- Priority
- Severity

- Reason

Description-välilehdelle syötetään mahdollisimman tarkka kuvaus havaitusta virheestä. Osastomme ohjeistus kehottaa lisäämään kenttään seuraavat väliotsikot:

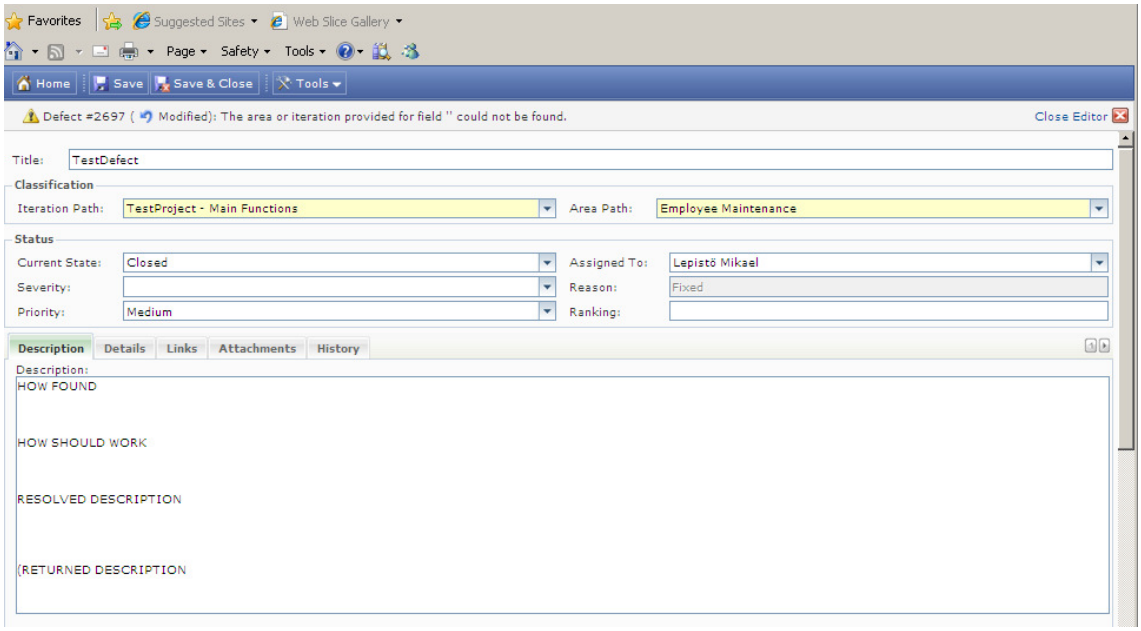
- HOW FOUND
- HOW SHOULD WORK
- RESOLVED DESCRIPTION
- (RETURNED DESCRIPTION)

How Found -otsikon alle kirjoitetaan kuvaus miten virhe löydettiin. Tyypillisesti käytetään menetelmää, jossa kuvataan tarkka sekvenssi siitä mitä toimenpiteitä tehtiin, jotta virhe ilmenee. Virhe toistuu säännöllisesti eli on toistettavissa samalla sekvenssillä, jos virhekirjauksen tekijä ei ole toisin maininnut. Korvaava tapa menettelylle olisi lisätä Details-tietoihin laatikko Repeated Sequently, joka olisi oletusasetuksena valittuna. How Should Work -otsikon alle kirjoitetaan kuvaus, miten ohjelman tulisi toimia tilanteessa, joka johti virheeseen. Resolved Description -otsikon alle suunnittelija kirjoittaa, miten hän ratkaisi virhetilanteen. Otsikon alle tulee täyttää, mitä tiedostoja muutettiin. Kyseinen kohta täytetään, kun virhekirjauksen tila muutetaan vaiheeseen Resolved. Returned Description -otsikko on lisätty, kun virhe ei ole korjaantunut ja Resolved-tilassa oleva virhekirjaus joudutaan palauttamaan takaisin kehittäjälle. Otsikon alle kirjoitetaan syy, minkä takia työkortti palautetaan. Virhekirjauksen tila muutetaan vaiheeseen Identified. Kyselylomakkeen tulosten mukaan, olisi selkeämpää jos palautetulle virhekirjaukselle olisi oma tilansa, esimerkiksi Returned. Näin ollen projektin etenemisen seuraaminen onnistuisi paremmin, koska saataisiin tietoa toimimattomista korjauksista. Samalla välittyisi tiimin jäsenille tietoa, siitä minkä takia tietynlainen korjaus ei ollut toiminut ja vältettäisiin saman virheen toistamista jatkossa.

Kun virhekirjaus on luotu, virhekirjauksen käsittelyn hoitaakseen ottanut suunnittelija siirtää sen Identified-tilaan. Tällöin tehdään päätös virhekorjauksen aiheellisuudesta. Jos virhekirjaus on aiheeton, se hylätään antamalla Reason-kenttään arvo Obsolete ja muutetaan tilaksi Resolved. Jos virhekirjaus on aiheellinen, siirretään se Assessed-tilaan. Ratkaistu virhekortti siirretään Resolved-tilaan. Kehittäjän tulee muistaa Details-välilehdeltä valita sopiva arvo kentälle Build, johon korjaus kohdistuu. Build-kentän arvosta tiedetään, missä ohjelmistoversiossa korjaus on mukana. Kentän täyttäminen tulisi asettaa pakolliseksi, koska projektin aikana se jäi usein täyttämättä.

Yleinen ongelma aikaisempien projektihallintajärjestelmien tukeman virhekirjausmallien kanssa on ollut virhekirjausten linkittämien eri projektien välillä. Samaa perusohjelmistoa tarjotaan usealle asiakkaalle asiakaskohtaisesti muokattuina versiona. Sama virhetilanne voi tapahtua eri ohjelmistoversioissa. Näin ollen virhekirjauksen lisätietoihin tarvitsisi ristiviittauksen projektien välille. Perustuotteessa havaitut virheet toistuvat kyseistä versiota käyttäviin projekteihin, jolloin virhekirjaukset ja samalla yhteen korjaukseen käytettävät työtunnit kertaantuvat. Työkortit on mahdollista linkittää toisiinsa, jos ne sijaitsevat samassa tiimiprojektissa. Osastollamme on aloitettu projekteja, joissa tiimiprojekti voi sisältää useita projekteja. Näin ollen samoista ongelmista voidaan jakaa tietoa työkorttien linkityksen kautta. Työkortin linkitys tapahtuu työkortin Links-välilehdeltä ohjatulla toiminnolla.

Kuvassa 3. on projektissa käytetty Defect-työkortti. Työkortti-editori osaa varoittaa käyttäjää, kun siihen yritetään syöttää virheellisiä arvoja: tässä tapauksessa sekä Iteration Path että Area Path -kenttien arvoiksi on annettu olemattomat arvot yrityksen ja projektin asiakkaan pitämiseksi salassa.



Kuvio 10 Projektissa käytetty työkortti

Team Foundation Server -käyttökokemukset -kyselyyn vastanneiden enemmistön mukaan Defect-työkortilla on liikaa mahdollisia tiloja. Yli kolmasosa vastanneista haluaisi yhdistää Defectin Identified ja Assessed -kentät. Palautetut virhekirjaukset siirrettiin takaisin Identified-tilaan, mikä ei indikoinut siihen että virhekirjaus on korjattu väärin. Samalla ei ollut varmuutta siitä, oliko tiimin henkilö jolle virhekirjaus on kohdistettu, tietoinen tekemänsä korjauksen virheellisyydestä, koska Sharepointin lähettämät automaattiviestit olivat konfiguroitu huomioimaan vain Start-tilassa olevat Virhekirjaukset. Selkein ratkaisu toistaiseksi ongelmaan oli ohjeistaa henkilöstö palauttamaan Defectit Start-tilaan. Sharepoint voidaan myös konfiguioida reagoimaan Start-tilan lisäksi Returned-tilaan.

9.1.3 Change

Change eli Muutospyyntö on tärkeä työkortti muutostenhallinnan näkökulmasta. Muutospyyntö tehdään asiakkaan toimesta; itse työkortti kirjataan kuitenkin tiimin jäsenten toimesta. Muutospyyntö tarkoittaa asiakkaan haluamaa muutosta jo hyväksyttyyn toiminnallisuuteen ja määritykseen. Näin varmistetaan, että projektissa tehty lisätyö huomioidaan laskutuksessa. Samalla estetään tilanne, jossa asiakas pääsisi sanelemaan jatkuvasti muutoksia toteutettavaan järjestelmään ilman velvoitteita.

Muutospyyntökorteissa on käytettävissä seuraavat kentät:

- Current State
- Area Path
- Iteration Path
- Assigned To
- Priority

Description-välilehdelle syötetään mahdollisimman tarkka kuvaus halutusta muutospyynnöstä. Muutospyyntökortin kuvauksen täyttämistä ei ole formalisoitu mitenkään. Linkkeihin tai liitteisiin tulisi ainakin liittää tekninen/toiminnallinen määritys, jota tullaan muuttamaan tai johon viitataan muutospyynnön johdosta.

Muutospyyntökortin elinkaari on kuusiosainen: Start, Requested, Specified, Assessed, Implemented ja Verified. Uusi muutospyyntö on Start-tilassa.

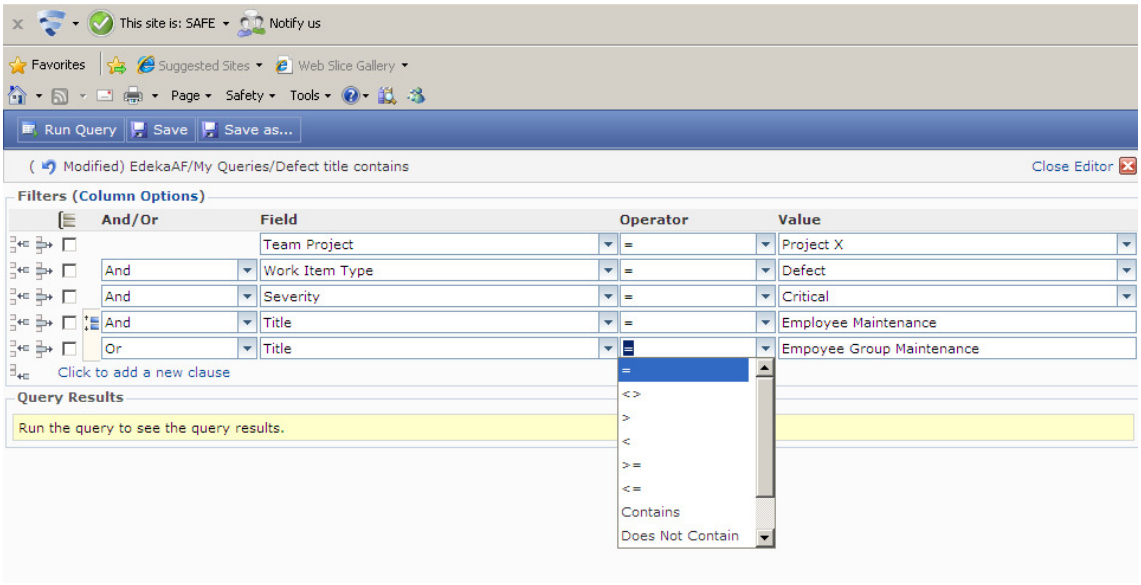
Team Foundation Server -käyttökokemukset -kyselyn mukaan defectin, changen ja taskilla on liikaa tiloja (state). Changen-tiloissa koettiin myös, että mahdollisia tiloja liikaa. Toiveena esitettiin usean vastanneen toimesta, että käyttöön otettaisiin tilat:

- Start
- Specified
- Implemented
- Verified

9.2 Kyselyt ja raportointi

Tiimikyselyt ovat kaikille tiimin jäsenille näkyviä yhteisiä kyselyjä. Tiimikyselyt ovat tyypillisesti hyvin yleisiä kyselyjä kuten ”Resolved Defects” ja ”Open Defects”. Valmiit, kaikille tiimin jäsenille näkyvät, kyselyt säästävät työntekijöiden aikaa ja vaivaa, jota kuluisi uutta kyselyä tehdessä. Osastomme projekteissa tiimikyselyjä (Team Query) ovat luoneet lähinnä testaajat. Uusi tiimikysely on mahdollista tehdä vain administrator-oikeuksin, jotka annettiin tarpeen vaatiessa testaajille. Käyttäjäryhmien jaottelua tulisi tehdä uusiksi, jotta esimerkiksi kaikilla testaajilla olisi oikeus tehdä tiimikyselyjä.

Osastomme henkilöstö koki helpoksi uusien kyselyjen muodostamisen. Kyselyjä voidaan muodostaa sekä Visual Studion että Web Accessin kautta.



Kuvio 11 Kyselyn muodostaminen

Kuvassa 11 on ikkuna, jonka avulla voidaan tehdä uusia kyselyjä. Uusia työkortin kenttiin kohdistuvia hakuehtoja voidaan lisätä klikkaamalla nuolen kuvaa. Field-alasvetolaatikon arvo vastaa työkortin kentän nimeä ja Value-alasvetolaatikko vastaa kyseisen kentän arvoa. Kuvas-

sa tehdyssä kyselyssä on käytetty = (yhtä suuri kuin) operaattoria. Operaattorina voidaan käyttää myös muita yleisesti tunnettuja operaattoreita kuten < (pienempi kuin) ja <> (eri suuri kuin) sekä merkkijonoja palauttavissa hauissa operaattoreita kuten ”Contains” (sisältää merkkijonon) ja ”Does Not Contain” (ei sisällä merkkijonoa). Kuvassa on annettu neljä kenttää, joihin haku kohdistuu: Team Project, Work Item Type, Severity ja Title. Kenttä Title on lisätty kahdesti, koska on haluttu kyselyn kohdistuvat kahteen eri toimintoon: Employee Maintenance ja Employee Group Maintenance. Näin ollen kentät on täytynyt lisätä samaan ryhmään. Ryhmitys tapahtuu ruksaamalla kentän vasemmassa laidassa oleva ruksilaatikko halutuista kentistä ja sen jälkeen klikataan ryhmitys-painiketta (Filters-otsikon alla oleva painike, vasemmassa yläkulmassa). Kuvassa oleva kysely listaa kriittiset virhekirjaukset, jotka kohdistuvat Employee Maintenance- ja Employee Group Maintenance -toimintoihin. Listassa on oletuksena vain Title- ja Severity-kentät. Uusia kenttiä voi lisätä listanäkymään Column Options -linkin avulla (kuvassa vasemmassa yläkulmassa).

Uusien raporttipohjien teko on epäselvää henkilöstölle eikä yrityksen materiaalista löydy asiaan dokumentaatioa. Yli kaksi-kolmasosaa TFS käyttökokemukset -kyselyyn vastanneista ei ollut juuri tutustunut Team Foundation Serverin raportointi-toimintohin. Lähinnä raportteja ottivat projektipäälliköt sekä testaajat. Yksikään vastaajista ei ollut tehnyt itse uusia raporttipohjia vaan projektissa käytettiin ohjelmiston mukana tulleita valmiita raporttipohjia. Kyselyn kommenttiosuudessa oli myös useita mainintoja, että ei ole tiedossa miten uusia raporttipohjia tehdään.

Uusia raporttipohjien teko on työläämpi prosessi verrattuna kyselyjen tuottamiseen. TFS Guiden mukaan uusia raporttipohjia tehdään omina projekteinaan Visual Studion kautta. Projektimalliksi (template) valitaan Report Server Project. Solution Explorer -ikkunan avulla luodaan uusi jaettu tietokantayhteydet (Shared Data Source) Team Foundation Serveriin. Itse raportin tuottaminen tapahtuu ohjatulla toiminnolla. Valmis rdl-päätteinen raporttitiedosto siirretään palvelimelle. (TFS Guide, 442-445.)

9.3 Sovelluskehitys

Osastomme käytössä on Visual Studio 2008 Development Edition. Sovelluskehitystyö projektissa toteutettiin virtuaaliympäristössä. Osastollamme on käytössä VMWaren ohjelmistot WMware Workstation ja WMware Player. VMware Workstation on maksullinen ohjelmisto virtuaalikoneiden luontiin sekä ajamiseen. Player taas on ilmainen ohjelmisto, sillä voidaan ajaa muilla VMware-ohjelmistoilla luotuja virtuaalikoneita, mutta ei voida luoda niitä itse. (WMWare Products, 2010)

VMware-ohjelmistoja käyttäviä käyttöjärjestelmiä kutsutaan isäntäkoneeksi ja käyttöjärjestelmän sisällä toimivia virtuaalikoneita ja niiden käyttöjärjestelmiä kutsutaan vastaavasti vieraskoneiksi. Vieraskone toimii isäntäkoneen käyttöjärjestelmässä omassa ikkunassa, mutta tarjoaa käyttäjälle ikkunassa vieraan omat käyttöjärjestelmäikkunansa ja valikkonsa. Vieraskone tunnistaa isäntäkoneen fyysiset resurssit kuten kiintolevyt, näytön, Internet-yhteyden, äänikortin ja hiiren sekä näppäimistön. (WMWare Workstation User Guide, 53-54.)

Projektia varten oli luotu virtuaalikoneimage. Virtuaalikoneimage tarkoittaa WMware Workstationilla luotua virtuaalikonetta. Image oli kopioitavissa osastomme verkkolevyltä. Virtuaalikoneen käyttöjärjestelmänä oli Windows XP Professional ja siihen oli asennettu valmiiksi kaikki sovelluskehitystyössä tarvittavat ohjelmistot kuten Visual Studio Developer Edition ja Sql Server Management Studio. Virtuaalikoneen käyttöönottoon vaadittiin MSDN-lisenssit, koska useimmat virtuaalikoneen sisältämät sovelluskehitysalustat kuten Visual Studio ja Windows-käyttöjärjestelmä on luonnollisestikin maksullisia tuotteita.

Virtuaalikoneiden avulla sovelluskehitys tapahtuu sekä tehokkaammin että turvallisemmin. Virtuaalikoneesta on helppo ottaa varmuuskopio ja tarvittaessa on mahdollista palauttaa koko käyttöjärjestelmä sovelluksineen. Virtuaalikoneen siirtäminen toiseen koneeseen on myös mahdollista ja onnistuu varsin vaivattomasti. Virtuaalikoneen avulla kehittäjät voivat pitää

työasemansa erillään kehitysympäristöstä. Virtuaalikoneista on myös mahdollista ottaa ns. snapshot, eli koneen sen hetkinen tila tallennetaan kokonaisuudessaan tiedostoon ja tähän tilaan voidaan palata takaisin milloin tahansa. Monien snapshotien ottaminen tekee etenkin VMware Workstationista kätevän työkalun esimerkiksi monimutkaisen ohjelmiston esittelyyn tai ohjelmistokehityksen testiympäristöksi. (WMWare Workstation User Guide, 183-184)

Virtuaalikoneen Visual Studiosta oli mahdollisuus ottaa yhteys Team Foundation Serverin tarjoamiin palveluihin. Ensimmäinen vaihe kehitystyössä oli kopioida TFS-palvelimelta työalue (Work area) kehittäjän virtuaalikoneelle. Kopioiminen tapahtui Visual Studion Source Control Explorer -ikkunan kautta. Työalue oli hyvä päivittää myös Source Control Explorer -ikkunan kautta, saman tien kun lähdekoodit oli saatu kopioitua työalueelle. Työalueen kopiointi muodosti virtuaalikoneelle seuraavan hakemistorakenteen:

$Project X

/Main

/Build
/Config
/Graphics
/Help
/Installpackages
/Systemfunctions
/Report
/SharedBinaries
/Sql

/TeamBuildTypes

Build

/BuildType1

Kun käytössä oli toimiva työalue ja viimeisimmät lähdekoodit, oli uuden toiminnon aloittaminen mahdollista. Projektille annettiin nimi ja sijainti hakemistorakenteessa. Projektiin valittiin kehitystyötämme varten suunniteltu prosessimalli. Mallipohjan avulla vältetään itseään toistavien vaiheiden tekemistä kerta toisensa jälkeen.

Koska projekti oli niin sanottu Add-on -projekti, tuli kehittäjän lisätä uuteen projektiin viittaukset (references) perustuotteen dll-tiedostoihin. Viittauksen lisääminen projektiin tapahtui Visual Studion Solution Explorer -ikkunan avulla. Tarvittavat Dll-tiedostot sijaitsivat Shared-Binaries-kansiossa. Perustuotteen Dll-tiedostostoihin tarvittiin viittaus, jotta projekti pystytään kääntämään onnistuneesti (build).

Versionhallinnalle oli asetettu Check-in Policy -määritykset. Jokaiseen sisäänkirjattuun (Check-in) tiedostoon tuli liittää työkortti sekä erillinen kommentti. Näin selvitettiin nopeasti Buildin rikkoutuessa muutosjoukko, josta ongelma johtuu. Useamman työkortin liittäminen muutosjoukkoon sallittiin, koska yksittäinen toiminto koostui useammasta työkortista: kolme Task-työkorttia ja mahdolliset Virhekirjaukset. Tiedoston uloskirjaamiseen (check-out) asetettiin sääntö, joka sallii tiedoston päivittämisen uusimpaan mahdolliseen versioon ennen tiedoston muokkausta. Tiedoston uloskirjaamisessa oli valittava lukitusmetodi:

1. Unchanged – Keep any existing lock
2. None – Allow shared checkout
3. Check-out – Prevent other users from checking out and checking in
4. Check-in – Allow other users from check out but prevent them from checking in

Vaihtoehto 2-None sallii kaikkien käyttäjien muokata tiedostoa vapaasti eli toisin sanoen, lukitusta ei ole varsinaisesti käytössä. Vaihtoehto 3-Check-out estää muita käyttäjiä kirjaamasta tiedostoa ulos. Vaihtoehto 4-Check-in sallii muille käyttäjille tiedoston uloskirjaamisen, mutta estää tiedoston kirjaamisen sisään. Vaihtoehto 1 käyttää aikaisemmin määriteltyä lukitsemisvalintaa.

Projektissa ei hyödynnetty lähdekoodien haarauttamisen (branch) mahdollisuutta. Projektin alussa syyksi katsottiin, että toiminnosta ei olisi saatu lisäarvoa, koska kyseessä oli add-on -projekti. Sen lisäksi todettiin, että Team Foundation Serverin brach-toiminto on hyvin raskas prosessi verrattuna esimerkiksi Telelogic Synergyn vastaavaan toimintoon. Kehitystahti oli myös hyvin nopeaa ja testatessa haluttiin käyttää aina viimeisintä versiota lähdekoodeista. Saksaan toimitettuihin väliversioihin (drop) pystyttiin palamaan, koska msi-paketit pidettiin tallessa verkkolevyllä sekä oli sovittu, että testilaboratoriossa oleva testijärjestelmä pidetään aina samalla tasolla kuin Saksaan toimitettu drop. Väliversioihin olisi voitu palata myös tekemällä jokaisesta väliversiosta branch. Tekemällä jokaisesta toiminnosta oma haaransa, olisi voitu palata takaisin versiossa yksittäisten toimintojen tasolla.

Team Foundation Server käyttökokemukset –kyselyyn vastanneiden kehittäjien mukaan kehittäjät kokivat Team Systemin tarjoamat palvelut helpoiksi. Versionhallinta on integroitu Visual Studioon, joten varsinainen kehitystyö tapahtui kuten ennenkin. Team Explorer-, Source Control Explorer- ja Pending Changes -ikkunoiden käyttö koettiin helpoksi. Lisäkommenttina vastauksiin oli usealla vastaajalla, että aluksi ikkunoiden tarjoamia mahdollisuuksia ei osattu hyödyntää, vaan tarvittavat toimenpiteet tehtiin muuta, kuten Menu-valikon, kautta. Ikkunoiden käyttö oli kuitenkin helppo omaksua, kun sitä ensimmäisiä kertoja käytti. Tiedoston historian ja tiedostojen vertailuun tarkoitetut toiminnot saivat myös kiitosta: niitä pidettiin helppoina ja nopeina käyttää. Moni vastaajista kommentoi versionhallinnan logiikkaa aluksi erikoisena verrattuna Telelogic Synergyn tiedostojen hallintaan. Telelogic Synergyssä yhteen tehtävään kohdistui tiedostoja. Näin pystyi myöhemmin näkemään, mitä kaikkia tiedostoja oli muutettava tai toteuttaa tietyn ohjelmistopiirteen toteuttamiseksi. Team Foundation Serverissä muutosjoukkojen hallinta toimi ikään kuin päinvastoin; muutosjoukkoon kohdistui yksi tai useampi työkortti. Kommentiosuuteen oli kahden työntekijän toimesta huomautettu koodin hyllyttämiseen liittyvästä ominaisuudesta; ominaisuutta ei ole esitelty yrityksen materiaalissa tai koulutustilaisuudessa, joten ominaisuus oli useilta kehittäjiltä huomioimatta.

9.4 Testaaminen

Testijärjestelmät olivat virtualisoitu kuten kehitysympäristötkin. Jokaisella testaajalla oli oma virtuaalinen testijärjestelmänsä sekä näiden lisäksi Helsingin testilaboratoriossa sijaitsi yhteinen testijärjestelmä. Virtuaalikonetta käytettiin joko WMware Workstationilla ja WMware Playerilla. Laboratorion testijärjestelmää käyttivät Suomen tiimin jäsenet Oulusta: järjestelmään sai etäyhteyden VPN-yhteydellä ja Remote Desktop -ohjelmistolla. Testaajat pystyivät itse päivittämään testijärjestelmänsä, kun korjauksia virhekirjauksiin saatiin valmiiksi. Päivittäminen tapahtui komentorivillä komennoilla, jotka purkivat build-prosessin projektin lähdekoodeista tekemät msi-paketit tietyillä parametreilla. Järjestelmän asennuksen ja päivityksen testaaminen sisältyi systeemitesteihin. Projektin testisuunnitelma kattoi yksikkötestit, integraatiotestit ja systeemitestit. Testaaminen tapahtui Suomen tiimin toimesta. Venäläisten testaaminen rajoittui kehitysympäristöllä toteutettuihin yksikkötesteihin. Tästä seurasi usein tilanteita, jolloin tietty järjestelmän toiminto, esimerkiksi uuden työntekijän tallennus, onnistui kehittäjän koneella, mutta ei taas toiminutkaan halutulla tavalla testijärjestelmässä.

Testaamisen hallinta käsiteltiin Compuwaren QADirectorissa. Testisuunnitelmat, testiskriptit ja testauksen raportointi hoidetaan ainoastaan kyseisen järjestelmän kautta. Toisin sanoen, TFS:stä ei voinut testitapaustasolla seurata, miten testaus etenee. Varsinaisia testituloksia ei pystytty TFS:n kautta seuraamaan. Testaamisen etenemisen seuranta on mahdollista ainoastaan Task- ja Defect-työkorttien välityksellä. Paperista versiota testisuunnitelmasta ei ollut, joten jos tiimin jäsen halusi tietoa testitapauksista ja testituloksista, hän tarvitsi projektikohtaiset oikeudet ja tunnukset QADirectoriin. Paperinen versio testisuunnitelmasta olisi ollut syytä tehdä, koska myöhemmin havaittiin, että se olisi vähentänyt työmäärä testausraportis-

ta. Myös asiakkaalle siitä olisi ollut lisäarvoa, ja olisi varmistuttu siitä, etteivät asiakkaan testit mene täysin päällekkäin omien testiemme kanssa.

Team Systemin Test Edition sisältää samoja ominaisuuksia kuin Compuwaren QADirector. Jani Järvisen mukaan Visual Studio Team System 2008 on erityisesti ohjelmistotestaajien työväline. Mukana on tuki automaatiotestaamiselle. Automaatiotestauksen avulla voidaan vähentää manuaalisesti tehtävän testauksen määrää; nauhoitetut testit voidaan ajaa aina uudelleen koodimuutosten jälkeen. (Järvinen, 240). Test Edition integroituisi projektinhallintaominaisuuksiin ja Visual Studion kehitysympäristöön, mikä lisäisi kehitystyön ja testaamisen laatua sekä edistäisi projektin seurantaa.

10 Team Foundation Server 2010 esikatsaus

ITPro:n mukaan Team Foundation Server 2010 mukana tulee useita paranneltuja ominaisuuksia. Kun tehdään uusi haara (branch), siirtyy uuteen kohdehaaraan myös historia lähdehaarasta. Eri haaroja tutkailtaessa nähdään myös helpommin mihin kaikkialle jokin tietty lähdehaara on haaroitettu eteenpäin. Koodihaaroihin voidaan myös lisätä lisätietoa, kuten esim. koodihaaran omistaja tai sen käyttötarkoitus. Koodihaarojen ja niihin liittyvien työkorttien visualisointia on parannettu huomattavasti. Koska vaatimuksetkin tallennetaan TFS:ään työkortteina, voidaan myös visualisesti seurata milloin jokin tietty vaatimus toteutuu jossain tietyssä koodihaarassa. Asennus on TFS 2010:ssä melko suoraviivaista, ja helpotettu huomattavasti. Lisäksi asennus ja konfigurointi on erotettu kahdeksi erilliseksi tehtäväksi. TFS voidaan asennuksen jälkeen jättää konfiguroimatta, jolloin käyttöönotto on paljon nopeampaa tilanteissa, joissa halutaan pitää TFS -palvelimia varalla. TFS voidaan asentaa täysin ilman SharePointia, mutta tällöin ei ole käytettävissä SharePointin tarjoamia toiminnallisuuksia. Reporting Servicenkin voi jättää TFS -asennuksesta kokonaan pois. Tämä yksinkertaistaa ja nopeuttaa asentamista sekä konfigurointia. ITPro mainitsee uusia ominaisuuksista Team Foundation Server 2010:sta: Gated Check-in -toiminnon avulla käynnistetään palvelimella käännösautomatiikka, jonka avulla voidaan varmistua siitä, että koodia sisäänkuitatessa ei rikota mitään. Tämä tapahtuu ennen tiedostostonkirjausta, jolloin päästään ongelmaan käsiksi paljon aikaisemmin, kuin jos käytettäisi esimerkiksi Continuous Integration -toiminnallisuutta. Toiminnosta tulee olemaan hyötyä osastollomme, uuden buildin tekeminen vie aina aikaa 30 minuutista kahteen tuntiin. Toiminnon avulla tullaan säästämään aika, joka johtuisi turhien buildien teosta.

Koodihaarojen ja muutosten etsiminen on visualisoitu niin, että esim. on paljon helpompaa varmistaa onko jokin tietty korjaus jo viety eteenpäin kaikkiin tarvittaviin koodihaaroihin. Samalla voidaan myös helposti nähdä mihin kaikkiin koodihaaroihin johonkin tiettyyn työkorttiin kirjoitettu.

Enterprise Team Foundation Server -ominaisuuden avulla TFS 2010:ssa voidaan TFS-instanssi skaalata useammalle sovelluspalvelimelle (Application Tier) jolloin yhden sovelluspalvelimen vioittuminen ei keskeytä toimintaa. Tämän ominaisuuden myötä voidaan useita TFS instansseja myös hallita yhtenä kokonaisuutena, ja siirtää projekteja palvelimelta toiselle. Projekteja voidaan myös arkistoida ja palauttaa käyttöön, hieman samaan tyyliin kuin Visual SourceSafessakin aikanaan. (Team Foundation Server 2010:n ominaisuuksia 2008.)

11 Yhteenveto

Opinnäytetyön tarkoituksena oli antaa tietoa Team Foundation Serverin käytöstä Yritys A:n projekteissa ja samalla selvittää henkilöstön tyytyväisyyttä TFS:n projektinhallintaominaisuuksiin. Team Foundation Server-käyttökokemukset -kyselyyn vastanneiden enemmistön mukaan Team Foundation Server on helpottanut projektityötä ja etenkin projektin etenemisen seuraamista. Yli puolet vastanneista oli sitä mieltä, että TFS tulisi ehdottomasti ottaa käyttöön osastomme muissakin projekteissa. Erittäin hyvänä piirtenä Team Foundation Serverissä on pidetty sitä, että myös Venäjän tiimi pystyi käyttämään sitä eri projekteissa. Yhteinen versionhallinta nopeutti ja helpotti kehitystyötä. Team Foundation Serveriä pidetään hyvänä työvälineenä, mutta silti osastomme henkilöstön mielestä etenkin työkorttien käyttämisessä on parantamisen varaa. Pakollisia kenttiä oli määritelty liian vähän, mikä johti siihen että työkortit eivät sisältäneet aina kaikkia tarvittavia tietoja. Työkorttien huolellinen täyttäminen on poikkeuksellisen tärkeä asia, kun työskennellään projektissa, jossa kirjataan yli 1000 Virhekirjausta. Luotettavien kyselyjen ja raporttien muodostaminen muuttuu tällöin lähes mahdottomaksi. Opinnäytetyössä esittämät kehitysehdotukset työkorttien kenttiin ja elinkaaren hallintaan tullaan ottamaan käyttöön tulevissa projekteissa.

Opinnäytetyön aikana löydettiin seuraavat kehitysehdotukset:

1. Lisää pakollisia kenttiä työkortteihin: Severity, Iteration Path, Application.
2. Sharepoint-palvelu generoi automaattiviestin tiimin jäsenille, kun sen keskustelupalstalle saapuu uusi viesti.
3. Risks-, Requirements-, Issue- ja Quality Of Services -työkortit käyttöön.
4. Tehtäväkorttien jaottelussa huomioidaan tehtävän tyyppi. Prosessimalliin lisätään uusi kenttä, Task Type, jonka arvoina voivat olla ”Management”, ”Development” ja ”Testing”.
5. Tehtäväkortin elinkaareen tila ”Waiting”, kun tehtävää ei voida suorittaa toisesta tekijästä johtuen loppuun. Tilaa voisi kuvata työkortti ”Issue”, johon luodaan linkki.
6. Palautetuttuja Virhekirjauksia varten prosessimalliin luodaan tila ”Returned”.
7. Virhekirjauksen Details-tietoihin laatikko Repeated Sequently. Laatikko ruksataan, jos virhetilanne toistuu säännöllisesti. Laatikko on oletusasetuksena valittu.
8. Defectin elinkaaressa on liikaa tiloja. Identified ja Assessed tilat yhdistetään.
9. Testaajien oikeuksiin sallittaisiin tiimikyselyiden teko, nyt tiimikyselyn teko on mahdollista vain administrator-oikeuksin.
10. Visual Studio Test Edition testaamisen työkaluksi; selvitys työkalun käyttöönotosta, toiminnallisista ominaisuuksista sekä kustannuksista.

Opinnäytetyössä esitettyjä kehitysehdotuksia päästään arvioimaan käytössä uusien projektien myötä. 6/10:stä kehitysehdotuksesta liittyy työkorttien käyttöön, jotka ovat projektinhallinnan kannalta oleellinen osa sovelluskehitystä. Laajoissa sovelluskehitysprojekteissa saattaa syntyä jopa yli tuhat virhekirjausta sekä satoja tehtäviä; tällöin työkorttien oikeanlainen jaottelu ja järjestelmällinen käyttö helpottaa projektin seuraamista, tiedon löytämistä ja hyödyntämistä ja siten vähentää päällekkäisen työn riskiä.

Opinnäytetyössä korostui yleisten projektinhallintaperusteiden ymmärtäminen ja niiden soveltaminen ohjelmistotuotantoon. Opinnäytetyön tuottajalle aihe oli oppimismahdollisuus sovelluskehityksenprosessin hallitsemiseen aina määrittelyvaiheesta testausvalmiiden järjestelmien tuottamiseen. Ensisijaisen tärkeää sovelluskehityksessä on ymmärtää, mitkä ovat asiakkaan todelliset vaatimukset järjestelmälle. Ohjelmistokehitystiimin sekä samalla koko projektin henkilöstön on myös varauduttava vaatimuksien muuttumiseen kesken projektin. Muutostenhallintaa seurataan TFS:n Change-työkorttien avulla. Muuttuvat vaatimukset vaikeuttavat

projektinhallintaa monella eri tavalla. Työmäärän tarve ja kustannuksien arviointi on suoritettava uudelleen muuttuneisiin vaatimuksiin kohdistuen. Projektin todellinen eteneminen on tärkeää tiedostaa sovelluskehitysprojektin joka vaiheessa, minkä takia työkorttien pakollisiksi kentiksi on määriteltävä nykytilaa monipuolisimmin kenttiä. Ottamalla työkorttityypit Risk, Requirement, Issue ja Quality Of Services käyttöön, järjestelmän tilasta voidaan luoda laajempia työkorttiketjuja. Usein sovelluskehitysprojektin keskeneräisestä tuotteesta on mahdoton sanoa, mikä on projektin todellinen valmiusaste. Kriittiset järjestelmän ominaisuudet saatetaan huomioida liian myöhään, jolloin valmiusaste putoaa rajusti, ja jo valmiiksi toteutettuja osioita joudutaan muokkaamaan todellisten vaatimusten mukaisesti.

Lähteet

C# ja .NET Framework ohjelmointi. FC Sovelto 2008 (maksullista materiaalia).

Haikala, I& Märijärvi, J., Ohjelmistotuotanto. 2004. Talentum Helsinki.

How to: Show Differences between Two Files or File Versions, 2010. Microsoft. Viitattu 27.2.2010.
(http://msdn.microsoft.com/en-us/library/ms181445(VS.80).aspx)

Merging in Team Foundation Version Control, 2010. Microsoft Viitattu 05.01.2010.
http://msdn.microsoft.com/en-us/library/ms364074(VS.80).aspx#vs05teamsys_topic8

Mielonen, S., Systeemiajattelu.
http://mlab.taik.fi/polut/Yhteiskunnalliset/tyokalu_systeemiajattelu.html

Järvinen Pertti&Järvinen Annikki. 2004. Tutkimustyön metodeista. Opinpajan kirja, 2004 Tampere

Ohjelmistokehityksen vaiheet RUP:n mukaisesti. 2005. Viitattu 16.10.2009.
(http://www.ibm.com/developerworks/rational/library/feb05/krebs)

Ojasalo, K., Moilanen, T. & Ritalahti, J. 2009. Kehittämistyön menetelmät. WSOY-pro. Helsinki

Software Development Life Cycle Models. IBM 2005. Viitattu 16.10.2009.
(http://codebetter.com/blogs/raymond.lewallen/archive/2005/07/13/129114.aspx)

Teemahaastettelu. 2000. Haettu 22.09.2009
(http://www.fsd.uta.fi/menetelmaopetus/kvali/L6_3_2.html)

TFS Guide 2007.08.02.pdf. 2007. Microsoft Viitattu 12.12.2009.
(http://www.microsoft.com/downloads/details.aspx?FamilyID=FF12844F-398C-4FE9-8B0D-9E84181D9923&displaylang=en)

Team Foundation Server Achitecture, 2010. Microsoft Viitattu 27.2.2010.
(http://msdn.microsoft.com/en-us/library/bb668952.aspx)

Team Foundation Server asennus ja käyttöönotto, 2008. Viitattu 10.10.2009
(http://www.fcsovelto.fi/Kurssit/Kurssivalikoima/Pages/Kurssihaku.aspx?ID=1591)

Team Foundation Server 2010:n ominaisuuksia, 2008. Viitattu 03.2.2010
(http://itpro.fi/asiantuntijaryhmat/ohjelmistokehitys/Lists/Posts/Post.aspx?ID=138)

Team System Web Access, 2010. Microsoft. Viitattu 06.01.2010.
(http://msdn.microsoft.com/en-us/library/bb892990.aspx)

Understanding Branching, 2010. Microsoft Viitattu 05.01.2010.
(http://msdn.microsoft.com/en-us/library/ms181424(VS.80).aspx)

Visual Studio Team System, 2010. Microsoft. Viitattu 04.2.2010.
http://msdn.microsoft.com/en-us/library/fda2bad5.aspx

Visual Studio 2008 Käsikirja. Järvinen Jani. WSOYPro 2008 Porvoo

VSTS:n rakenne, 2010. Viitattu 05.02.2010. (http://blogs.msdn.com/ukvsts/archive/2008/10/01/testing-with-visual-studio-team-system.aspx)

VSTS arkkitehtuuri 2010. Viitattu 06.1.2010. Microsoft (http://msdn.microsoft.com/en-us/library/ms364062(VS.80).aspx)

WMWare Workstation User Guide. 2009. Viitattu 02.01.2010.(http://www.vmware.com/support/pubs/ws_pubs.html)

WMWare Products, 2010. Viitattu 02.1.2010 (http://www.vmware.com/products/)

Yrityksen sisäinen materiaali

12 Liitteet

LIITE 1 Team Foundation Serverin (TFS) käyttökokemukset –kysely

Kyselyn tarkoituksena on kerätä yrityksen henkilöstöltä kokemuksia Team Foundation Serverin käytöstä.
Käytän saamiani tuloksia opinnäytetyössäni, joka on julkinen työ. Yrityksen nimeä ei mainita opinnäytetyössäni.

Kyselyyn vastataan anonyymisti. Tulosta dokumentti paperille, rastita / ympyröi mielestäsi oikeimmat ratkaisuvaihtoehdot ja toimita täytetty paperi nimettömänä 25.1.2010 mennessä Mikael Lepistön postilaatikkoon.

1. Työn kuvasi projektissa

   1.Ohjelmistosuunnittelija
   2.Versionhallintavastaava
   3.Testaaja
   4.Systeemiarkkitehti
   5.Projektipäällikkö

2. Olet käyttänyt TFS-järjestelmää

   1 projektissa
   2 projektissa
   3 – 5 projektissa
   Useammassa kuin 5 projektissa

3. Oletko saanut koulutusta TFS:n käyttöä varten?

   Olen saanut koulutusta / En ole saanut koulutusta

****

4. Onko mielestäsi TFS helpottanut vai vaikeuttanut projektinhallintaa tai projektin etenemisen seuraamista?

   1 *2* *3* *4* *5*

   1= vaikeuttanut 5= helpottanut paljon

5. Onko TFS helpottanut tai vaikeuttanut omaa työtäsi?

   1 *2* *3* *4* *5*

   1= vaikeuttanut 5= helpottanut paljon

6. Tulisiko mielestäsi TFS ottaa käyttöön osastomme muissakin projekteissa?

1 *2* *3* *4* *5*

1= Ei missään nimessä 5: Ehdottomasti

7. Oletko käyttänyt TFS:ää Visual Studion vai Web Accessin kautta?

1 *2* *3*

1= Vain Web Accessin kautta 2= Molempien kautta 3= Vain Visual Studion kautta

8. Oletko löytänyt helposti haluamasi työkortit (Work Item) My Queries –kyselyjen avulla?

1 *2* *3* *4* *5*

1= en ollenkaan 5= aina

Jos vastasit 1 tai 2, niin kuvaa lyhyesti syy:

9. Ovatko projekteissa käytetyt Team Queries –kyselyt olleet riittäviä?

1 *2* *3* *4* *5*

1=Riittämättömiä 5= Erittäin riittäviä

Jos et koe riittäväksi (1 tai 2), niin kuvaa lyhyesti syy:

10. Onko mielestäsi uusien virhekirjausten (defect) tekeminen nopeaa ja helppoa?

1 *2* *3* *4* *5*

1= Erittäin vaikeaa ja hidasta 5= Erittäin nopeaa ja helppoa

Jos koet vaikeaksi ja hitaaksi(1 tai 2), niin kuvaa lyhyesti syy:

11. Ovatko projekteissa käytettyjen virhekirjausten (defect) käytettävissä olevat kentät olleet riittäviä?

1 2 3 4 5

1=Riittämättömiä 5= Erittäin riittäviä

Jos et koe riittäväksi (1 tai 2), niin kuvaa lyhyesti syy:

12. Onko projekteissa käytettyjen virhekirjausten (defect) pakollisten kenttien (Title, assigned to)lukumäärä ollut liian pieni?

1 2 3 4 5

1= Liian vähän pakollisia kenttiä 5= Kenttiä on tarpeeksi

Jos vastasit 1 tai 2, niin mitä kenttiä tulisi asettaa pakollisiksi?

13. Onko mielestäsi virhekirjausten elinkaari (start,indentified,assessed,resolved,closed) riittävä?

1 2 3 4 5

1= Liikaa vaihtoehtoisia tiloja 3=Sopiva määrä 5: Liian vähän vaihtoehtoisia tiloja

Jos vastasit 1, 2, 4 tai 5, niin kuvaa lyhyesti syy:

14. Onko mielestäsi uusien muutospyyntöjen (change) perustaminen nopeaa ja helppoa?

1 2 3 4 5

1= Erittäin vaikeaa ja hidasta 3=En ole perustanut uusia muutospyyntöjä 5= Erittäin nopeaa ja helppoa

Jos koet vaikeaksi ja hitaaksi(1 tai 2), niin kuvaa lyhyesti syy:

15. Ovatko projekteissa käytettyjen muutospyyntöjen (change) käytettävissä olevat kentät olleet riittäviä?

1 2 3 4 5

1=Riittämättömiä 5= Erittäin riittäviä

Jos et koe riittäväksi (1 tai 2), niin kuvaa lyhyesti syy:

16. Onko projekteissa käytettyjen muutospyyntöjen (change) pakollisten kenttien (Title, assigned to)lukumäärä ollut liian pieni?

1 2 3 4 5

1= Liian vähän pakollisia kenttiä 5= Kenttiä on tarpeeksi

Jos vastasit 1 tai 2, niin mitä kenttiä tulisi asettaa pakollisiksi?

17. Onko mielestäsi muutospyyntöjen (change) elinkaari (start,requested,specified,assessed,implemented,verified) riittävä?

1 2 3 4 5

1= Liikaa vaihtoehtoisia tiloja 3=Sopiva määrä 5: Liian vähän vaihtoehtoisia tiloja

Jos vastasit 1, 2, 4 tai 5, niin kuvaa lyhyesti syy:

18. Onko mielestäsi uusien tehtävien (task) perustaminen nopeaa ja helppoa?

1 2 3 4 5

1= Erittäin vaikeaa ja hidasta 3=En ole perustanut uusia tehtäviä 5= Erittäin nopeaa ja helppoa

Jos koet vaikeaksi ja hitaaksi(1 tai 2), niin kuvaa lyhyesti syy:

19. Ovatko projekteissa käytettyjen tehtävien (task) käytettävissä olevat kentät olleet riittäviä?

1 2 3 4 5

1=Riittämättömiä 5= Erittäin riittäviä

Jos et koe riittäväksi (1 tai 2), niin kuvaa lyhyesti syy:

20. Onko projekteissa käytettyjen tehtävien (task) pakollisten kenttien (Title, assigned to) lukumäärä ollut liian pieni?

1 2 3 4 5

1= Liian vähän pakollisia kenttiä 5= Kenttiä on tarpeeksi

Jos vastasit 1 tai 2, niin mitä kenttiä tulisi asettaa pakollisiksi?

21. Onko mielestäsi tehtävien elinkaari (start,defined,estimated,scheduled,assessed,in progress,completed) riittävä?

1 2 3 4 5

1= Liikaa vaihtoehtoisia tiloja 3= Sopiva määrä 5= Liian vähän vaihtoehtoisia tiloja

Jos vastasit 1, 2, 4 tai 5, niin kuvaa lyhyesti syy:

22. Onko mielestäsi dokumentaation ja dokumenttiversioiden hallittavuus helppoa TFS:ssä?

1 *2* *3* *4* *5*

1= Vaikeaa 5= Helppoa

Jos vastasit 1 tai 2, niin kuvaa lyhyesti syy:

23. Oletko käyttänyt TFS:n Raportointi-toimintoja (Reports)?

1 *2* *3* *4* *5*

1= en ollenkaan 5= usein

*Jos et ole käyttänyt lainkaan (1) Reports -toimintoja, niitä koskeviin kysymyksiin ei tar
vitse vastata

24. Oletko tehnyt uusia Raportti-pohjia?

1 *2* *3* *4* *5*

1= en ollenkaan 5= usein

25. Ovatko projekteissa käytetyt raporttipohjat olleet riittäviä?

1 *2* *3* *4* *5*

1=Riittämätön 5= Erittäin riittävä

Jos et koe riittäväksi (1 tai 2), niin kuvaa lyhyesti syy:

26. Ovatko projekteissa käytettyjen raporttipohjien esittämä informaatio esitetty selkeästi?

1 2 3 4 5

1=Informaatio on esitetty vaikeaselkoisesti 5= Informaatio on esitetty selkeästi

Jos et koe selkeäksi (1 tai 2), niin kuvaa lyhyesti syy:

TFS VISUAL STUDIO

*Jos et ole käyttänyt TFS:ää Visual Studion kautta, näihin kysymyksiin ei tarvitse vastata.

27. Onko tiedostojen versionhallinta helpompaa ja ymmärrettävämpää TFS:n avulla kuin sen edeltäjällä Telelogic Synergyllä?

1 2 3 4 5

1= Vaikeampaa 5: Helpompaa ja ymmärrettävämpää

Jos koet vaikeammaksi1 tai 2), niin kuvaa lyhyesti syy:

28. Onko tiedostojen historian seuraamiseen ja vertailuun tarkoitetut toiminnot riittäviä?

1 2 3 4 5

1=Riittämättömiä 5= Erittäin riittäviä

Jos et koe riittäväksi (1 tai 2), niin kuvaa lyhyesti syy:

29. Koetko Team Explorer –ikkunan käytön helpoksi?

1 2 3 4 5

1= Erittäin vaikeaa 5: Erittäin helppoa

Jos koet vaikeaksi 1 tai 2), niin kuvaa lyhyesti syy:

30. Koetko Source Control Explorer –ikkunan käytön helpoksi?

1 2 3 4 5

1= Erittäin vaikea 5: Erittäin helppo

Jos koet vaikeaksi 1 tai 2), niin kuvaa lyhyesti syy:

31. Koetko Pending Changes –ikkunan käytön helpoksi?

1 2 3 4 5

1= Erittäin vaikea 5: Erittäin helppo

Jos koet vaikeaksi (1 tai 2), niin kuvaa lyhyesti syy:

Kommentoi halutessasi vapaasti TFS:n käyttöä osastomme projekteissa sekä mielipidettäsi kohdissa 4,5 ja 6:

Kiitos Vastauksistasi