TAMPEREEN
AMMATTIKORKEAKOULU

OPINNÄYTETYÖ

VERSIONHALLINTA OHJELMISTOTUOTANNOSSA
Toteutus UNES Oy:ssä

Jukka-Pekka Majuri

Tietojenkäsittelyn koulutusohjelma
Lokakuu 2006
Työn ohjaaja: Paula Hietala

TAMPERE 2006

TAMPEREEN
AMMATTIKORKEAKOULU

Tekijä(t) Jukka-Pekka Majuri

Koulutusohjelma(t) Tietojenkäsittely

Opinnäytetyön nimi Versionhallinta ohjelmistotuotannossa

Työn valmistumis-kuukausi ja -vuosi Lokakuu 2006

Työn ohjaaja Paula Hietala **Sivumäärä:** 44

TIIVISTELMÄ

Ohjelmistojen kehittäminen on muuttunut viimeisen kymmenen vuoden aikana entistä vaativammaksi, sillä esimerkiksi ohjelmistojen koot ovat jatkuvasti kasvaneet ja niiden monimutkaisuus on lisääntynyt. Internetin tulon myötä ohjelmistojen korjaamiseen käytettävä aika on pudonnut murto-osaan entisestä. Tämä uudenlainen kehitys on pakottanut yritykset organisoimaan ja tutkimaan uudelleen omia kehitysprosessejaan, joissa eri versioiden järjestelmällinen hallinta on edellytys kehitykselle.

Opinnäytetyö toteutettiin toimeksiantona UNES Oy:ssä. Tarkoituksena oli ottaa yrityksessä käyttöön versionhallintaohjelmisto ja luoda siitä siellä pysyvä käytäntö kehitysprosesseja varten. Koska yrityksessä ei ollut aiemmin ollut käytössä versionhallintaa, tarvittiin myös perehtymistä sen teoriaan, jotta versionhallinnasta saatavat hyödyt olisivat mahdollisimman suuret ja vältettäisiin epäonnistumiset sekä turhat virheet.

Käytännön toteutuksessa asennettiin Microsoft Visual SourceSafe 2005 -ohjelmisto, joka on ainoa versionhallintatyökalu, jolla pystyy hallitsemaan Microsoft Access -ohjelmiston objektien versiointia. Versionhallintaa käytettiin Accessiin integroiduilla toiminnoilla, kuten objektien vienti ja tuonti, sekä myös SourceSafe-ohjelmiston avulla, esimerkiksi järjestelmänvalvojatoimintoihin.

Versionhallinnan käyttö edellyttää kehitystiimissä hyvää suunnittelua sekä kommunikointia, sillä niistä aiheutuvia puutteita hyvä versionhallintaohjelmisto ei korvaa. Mitä suuremmaksi ja monimutkaisemmaksi projekti kasvaa, sitä enemmän suunnittelun osuus korostuu. Kommunikaatio on usein puutteellista, ja siihen tulisi kiinnittää entistä enemmän huomiota. Pelkästään versionhallinnalla ei voida ratkaista koodissa tapahtuvien muutosten aiheuttamia ongelmia, vaan usein ne voidaan selvittää vain yhteistyöllä ohjelmiston kehittäjien välillä.

Avainsanat Versionhallinta SCM Konfiguraation hallinta

TAMPERE
POLYTECHNIC

Author(s) Jukka-Pekka Majuri

Degree Programme(s) Business Information Systems

Title Software Configuration Management in Software Engineering

Month and year October 2006

Supervisor Paula Hietala **Pages:** 44

ABSTRACT

Software development has become more advanced in the past ten years. Complexity and the sizes of software products have substantially increased and the Internet has minimized the repairing time of them. This kind of development process has forced companies to organize and revalue their processes in which systematic version control is prerequisite for further development.

This thesis was made as an assignment to UNES Ltd, in which there was an intention to introduce a version control management system and use it as a permanent part of the company's policy. Hence there has not been a version control management system in the company before; there was a need to get acquainted with the theory of the version control management. Throughout the theory it is intended to get the best possible advantage and minimize the risks of using the version control in the company.

In practice implementation there was installed the Microsoft Visual SourceSafe 2005. That is the only software configuration management tool which can control Microsoft Access' objects' versions. The version control system was used in Access by integrated actions like getting from and taking into the version control system, and with the assistance of the SourceSafe 2005 program, e.g. in administrative actions.

The use of the version control system requires good planning and communication in a team. If there are deficiencies because of insufficient planning or communication, not even the best version control systems can compensate these problems. The bigger and more complex the project grows, the more significant is the planning part of the process. Communication is often insufficient and therefore there is a need to pay more attention to it to get it better. The version control system cannot control all the changes in the code, but often there is a need for conversation between the developers to solve the conflicts.

Keywords Software Configuration Management Version Control Revision Control

Sisällysluettelo:

1 Johdanto

Nykyaikana teknologiat sekä ohjelmistojen kehitys- ja ylläpitoprojektit ovat muuttuneet yhä monimutkaisemmiksi. Ohjelmistojen koot ovat kasvaneet suuriksi, ja niiden kehittämisessä saattaa olla mukana jopa tuhansia ihmisiä. Usein myös kehitystyö tapahtuu eri puolilla maapalloa samaan aikaan. Jotta yritykset pystyisivät selviytymään tässä varsin monimutkaisessa ja sekavan oloisessa tilanteessa, tarvitaan sitä varten versionhallinta, jolla voidaan hallita eri dokumenttien muutoksia.

Versionhallinnalla on erittäin suuri, ellei jopa elintärkeä, merkitys yrityksen projektinhallinnassa (Leon 2000: 9). Mitä suurempi ohjelmisto- ja kehitystiimi on, sitä suuremmaksi versionhallinnan merkitys kasvaa. Internetin aikakaudella kehitykseen ja virheiden korjaamiseen käytettävissä oleva aika on pienentynyt olemattomaksi. Kun virhe havaitaan, sen etsimiseen ei voida käyttää paljon aikaa. Virheen etsiminen vie ajan lisäksi paljon henkilöresursseja ja rahaa. Versionhallinnan avulla virheet voidaan havaita ja korjata nopeasti. Tällöin on mahdollisuus ohjata resurssit tuottavampaan kehitystyöhön, alentaa ohjelmiston ylläpitokuluja sekä parantaa ohjelmistojen laatua.

Versionhallinta mielletään yleensä vain pelkän ohjelmistokoodin hallintajärjestelmäksi, mutta sillä voidaan kuitenkin hallita mitä tahansa dokumentteja ja niiden muutoksia. Läheskään kaikissa ohjelmistoyrityksissä ei ole käytössä versionhallintaohjelmaa, varsinkaan, jos kyseessä on pienempi ohjelmistoyritys.

Olen tehnyt tämän opinnäytetyön toimeksiantona työpaikalleni UNES Oy:lle, jossa ei ole aiemmin ollut käytössä varsinaista versionhallintajärjestelmää eikä yhtenäistä käytäntöä versionhallinnan toteuttamiseksi. Työssä edetään siten, että alkuosassa selvitetään versionhallinnan keskeisiä käsitteitä ja esitellään versionhallinnan teoriaa, minkä jälkeen toteutetaan versionhallinnan käyttöönotto Microsoft Visual SourceSafe 2005-ohjelmistolla. Lopuksi esitetään vielä yhteenveto aiheesta. Versionhallinnassa käytettävä sanasto ja yleisimmät versionhallintaohjelmat on lueteltu työn lopussa olevissa liitteissä.

2 Taustaa

2.1 Toimeksiantaja

Työn toimeksiantajana on UNES Oy (ks. www.unes.fi). Yritys tuottaa ohjelmistoja kiinteistö- ja palkkahallintoon. Yrityksen asiakkaina on pieniä ja keskisuuria yrityksiä, kaupunkeja sekä kuntia. Kiinteistöhallinnon ohjelmista merkittävin on isännöintiohjelmisto, jolla on tuhansia käyttäjiä ympäri Suomea. Muita kiinteistöalan ohjelmia ovat kulutusseuranta ja huoltokirja. Yksi yrityksen tärkeimmistä ohjelmista on palkkahallintoon tehty palkanlaskentaohjelmisto. Sitä käyttävät tilitoimistot, monien alojen yritykset sekä oppilaitokset opetuskäytössä.

Olen työskennellyt yrityksessä ohjelmistosuunnittelijana noin neljä vuotta. Vastaan pääasiassa palkanlaskentaohjelmiston suunnittelusta ja kehittämisestä. Idea opinnäytetyön aiheesta syntyi itseäni ja muita työntekijöitä kohdanneista ongelmista, joita versionhallinnan puute on yrityksessä aiheuttanut.

2.2 Työn tavoitteet

Työn tavoitteena on luoda yritykseen selkeä ja dokumentoitu käytäntö, jolla voidaan hallita ohjelmistojen eri versioita sekä niissä tapahtuneita muutoksia. Dokumentoinnin avulla voidaan esimerkiksi perehdyttää uusi henkilö yrityksen tapaan toimia yrityksen ohjelmistokehityksen eri vaiheissa. Dokumentointi toimii myös nykyisten työntekijöiden apuna silloin, kun siihen on tarvetta.

Versionhallinnalla on tarkoitus päästä eroon turhista virheistä, joita kehittäjä voi aiheuttaa omalla toiminnallaan. Kehittäjän virheitä voivat olla mm. väärän lähdekoodin muokkaaminen ja siten virheellisen ohjelman tuottaminen. Tällaisten virheiden eliminoiminen parantaa tuotteiden laatua merkittävästi ja lisää kilpailukykyä markkinoilla. Lisäksi työn lopputuloksena käyttöönotettavan versionhallintaohjelmiston tavoitteena on ajan ja resurssien säästäminen virheiden ja muutosten jäljittämisessä lähdekoodissa. Säästetty aika ja resurssit voidaan näin ollen ohjata tuottavampaan ohjelmistojen kehitystyöhön.

2.3 Lähdeaineisto

Tutkimusaihetta koskevan lähdeaineiston kerääminen ei ollut kovin yksinkertaista. Tämä johtuu siitä, että suurin osa kirjallisuudesta on myynnissä vain ulkomailla tai osa siitä ei ole enää muuten saatavilla. Varsinaisia aihetta käsitteleviä yleisteoksia on olemassa vain muutama. Kaikki kerätyksi saatu lähdeaineisto on englanninkielistä, mikä omalta osaltaan lisäsi haastetta työtä tehtäessä.

Tutkimuksen primääriaineistona käytettiin viittä teosta: Yleisteoksena käytettiin Alexis Leonin *Software Configuration Management* -teosta, joka on tätä työtä kirjoitettaessa markkinoiden ainoita kattavia aihetta käsitteleviä yleisteoksia. Yleisteoksen rinnalla käytettiin neljää IEEE-standardin kirjaa, jotka määrittelevät versionhallinnan termit, suunnitelmat ja toiminnot. Sekundääriaineistona käytettiin muun muassa useita eri versionhallintatyökaluihin keskittyneitä sekä muita yksitäisiä versionhallintaa käsitteleviä teoksia. Kaikki työssä käytetty lähdeaineisto on koottu työn lopussa olevaan lähdeluetteloon.

3 Versionhallinta

3.1 Historia

Versionhallinta sai alkunsa Yhdysvaltojen puolustusteollisuudessa vuonna 1962. Tuotteet olivat tuolloin pieniä, eivätkä ne olleet kehittyneet vielä monimutkaisiksi. Aluksi niiden kehityksestä pystyi huolehtimaan pieni ydinjoukko tai pelkästään yksi henkilö. Myöhemmin projekteihin osallistui kuitenkin useita henkilöitä, jolloin informaatio ei enää kulkenut henkilöiden välillä, koska käytössä ei ollut minkäänlaista sovittua tapaa dokumentoida suunnitelmia ja niiden muutoksia. (Leon 2000: 3.)

Yhdysvaltojen ilmavoimat määritteli standardin AFSCM 375-1, joka on ensimmäinen konfiguraationhallinnan standardi. Sen tarkoituksena oli korjata ongelmat tuotteiden suunnittelun hallinnassa ja kommunikaatiossa. Myöhemmin Yhdysvaltojen puolustusvoimat määrittelivät useita standardeja, kuten MIL- ja DoD-standardit. (Leon 2000: 3.)

Kun tietokoneet tulivat myöhemmin entistä tärkeämmäksi osaksi yhteiskuntaa, niiden käyttäjämäärät sekä ohjelmistojen määrät kasvoivat entisestään. Tietokoneiden ja ohjelmistojen avulla ihmiset pystyivät automatisoimaan ja helpottamaan tehtäviä, jolloin niiltä alettiin vaatia myös entistä enemmän ominaisuuksia. Ohjelmistojen monimutkaisuus pakotti organisaatiot lisäämään projekteissa työskentelevien ihmisten määrää, jotta kasvaneet vaatimukset pystyttiin täyttämään. Ohjelmistojen koon, monimutkaisuuden ja tiimien koon kasvaessa vaikeutui myös kehitysprosessin hallinta, koska muutoksia ei kyetty enää hallitsemaan ja henkilöiden välillä syntyi kommunikaatioongelmia. (Leon 2000: 3 - 4.)

Ohjelmistokehityksessä lisääntyneiden ongelmien ratkaisemiseksi alkoivat Yhdysvaltojen puolustusvoimat ja kansainväliset organisaatiot, mukaan lukien IEEE, ANSI ja ISO, miettiä ratkaisua ongelmiin. Myöhemmin näistä jokainen määritteli oman standardinsa, joista ANSI/IEEE on maailmanlaajuisesti käytetty. (Leon 2000: 4.)

3.2 Määritelmä

Versionhallinnalle ei ole olemassa varsinaista vakiintunutta tai kansainvälisesti sovittua määritelmää, vaan IEEE:n standardissa (IEEE Std 1042-1987: 9) käytetään termiä *konfiguraationhallinta*. Versionhallinta voi kuitenkin merkitä a) varsinaista konfiguraationhallintaa b) objektienhallintaa c) yksittäisen objektin hallintaa tai d) objektien versionhallintaa, joka käsittää niiden tunnistamisen sekä säilyttämisen. Versionhallinta on yksi kriittisimmistä konfiguraationhallinnan

toiminnoista. Se koostuu ohjelmistokehityksen aikana syntyneistä tiedostoista. (Jonassen Hass, Anne Mette 2002: 27.)

Työn pohjana olevan aineiston perusteella voidaan kuitenkin todeta, että termien versionhallinta ja konfiguraationhallinta käytössä on havaittavissa ristiriitaisuuksia ja jopa päällekkäisyyksiä. Eri lähteissä niitä käytetään usein harkitsemattomasti toistensa synonyyminä ilman, että niiden välistä merkityseroa on ymmärretty.

Konfiguraationhallinta on joukko aktiviteetteja, joiden tehtävänä on tunnistaa objektit, löytää niiden ominaisuudet, luonteenpiirteet ja riippuvuussuhteet sekä tallentaa ne. Lisäksi konfiguraationhallinnan tehtävänä on valvoa objekteja, hallita niiden muutoksia sekä varmistaa, että ne ovat täydellisiä ja täyttävät kaikki vaatimukset.

IEEE määrittelee konfiguraationhallinnan joukoksi sääntöjä, joiden tehtävänä on tunnistaa ja dokumentoida konfiguraatio-objektien toiminnallisia ja fyysisiä luonteenpiirteitä, hallita niiden muutoksia, tallentaa ja raportoida muutosprosessi, ilmaista tila sekä varmistaa, että ne vastaavat niille asetettuja vaatimuksia. (IEEE Std-610-12-1990: 20 - 21.)

3.3 Komponentit

IEEE (IEEE Std 828-1990: 8 - 11) ryhmittelee versionhallinnan toiminnot neljään eri vaiheeseen eli aktiviteettiin: 1) tunnistaminen, 2) muutoksen hallinta, 3) tilan seuranta sekä 4) arvioinnit ja tarkistukset.

Tunnistaminen on prosessi, jossa pyritään yksilöllisesti tunnistamaan komponentit versionhallintaa varten. Tällaiset komponentit ovat versionhallinnassa konfiguraatio-objekteja, jotka voivat olla muun muassa suoritettavaa koodia, lähdekoodia, käyttäjädokumentaatioita, testi- sekä projektisuunnitelmia. Koska jokaisen objektin tunnistaminen on yksilöllistä, tulee ne myös nimetä yksilöllisesti. Hyvä nimeämistapa kuvaa objektin tilaa, joka helpottaa esimerkiksi tunnistettavuutta varastoinnissa, uudelleen käytettäessä, jäljitettäessä ja tiedon palauttamisessa. (IEEE Std 828-1990: 10.)

Muutoksen hallinta on versionhallinnan toiminnoista useimmin tapahtuva toiminto, jonka tehtävänä on valvoa konfiguraatio-objekteihin kohdistuvia muutoksia. Muutoksen hallinta on joukko prosesseja, joilla varmistetaan tuotannon laatu ja se, että objektit on testattu, tarkistettu ja dokumentoitu. (Leon 2000: 104 - 106.)

Konfiguraatio-objektien tilan seurannalla tarkoitetaan informaation tallentamista ja raportointia. Prosessin avulla saadaan selville objektin versioiden historia, tehdyt muutokset, henkilö tai henkilöt, jotka

ovat tehneet ne, sekä milloin ja miksi muutokset on tehty. Saaduista tiedoista on hyötyä esimerkiksi ongelman selvittämisessä. (Keyes 2004: 13.)

Arvioinneilla ja tarkistuksilla vahvistetaan, että objekti vastaa vaadittuja fyysisiä ja toiminnallisia luonteenpiirteitä. Arvioinneilla varmistetaan myös, että ohjelmisto sisältää oikeat komponentit ja että ohjelmiston kehitys on toteutettu oikein. (Keyes 2004: 14.)

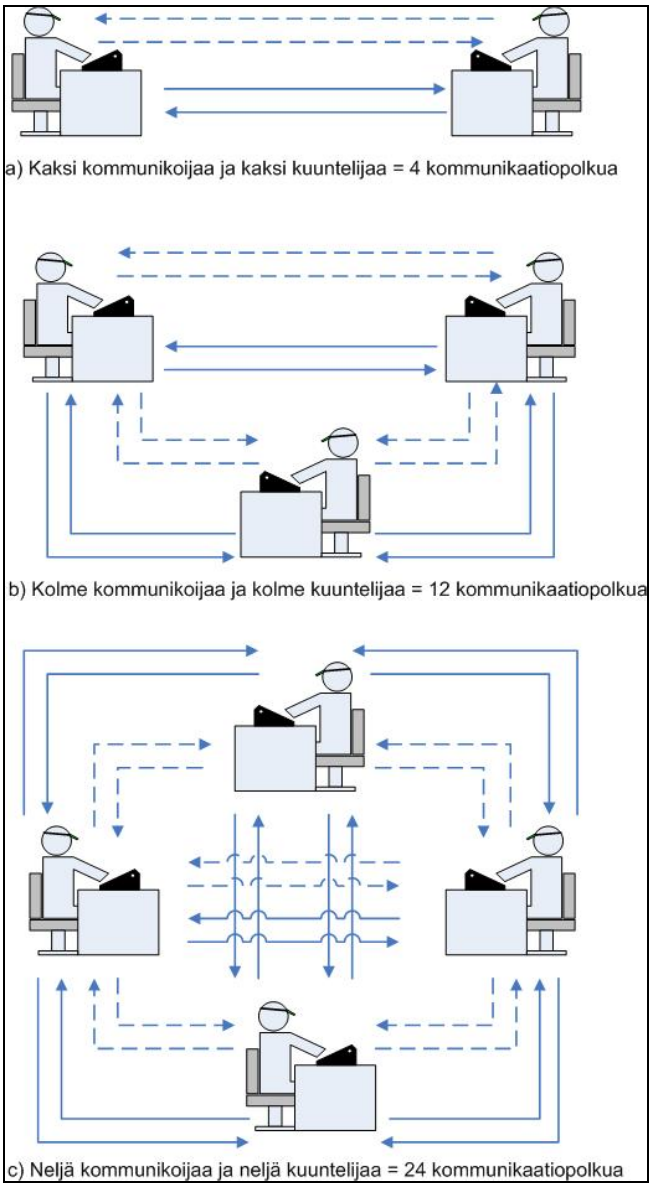
3.4 Merkitys

3.4.1 Ongelmat ilman versionhallintaa

Kommunikaatio-ongelma

Nykyisessä ohjelmistokehityksessä käytettävät työkalut, prosessit ja niiden vaatimukset muuttuvat jatkuvasti. Kehitysprosessin aikana syntyy useita versioita tuotannossa olevista ohjelmistoista, esimerkiksi virheiden korjauksien takia. Samalla myös kehitetään toiminnallisuudeltaan erilaista, uutta versiota ohjelmasta. Rinnakkainen kehitystyö, jossa on osallisena useita henkilöitä, luo monimutkaisuutta, jonka hallitseminen ilman selkeää suunnitelmaa on mahdotonta. (Crnkovic, Asklund & Persson Dahlqvist 2003: 8.)

Ohjelmistokehityksessä työskentelee useita eri henkilöitä. Tällöin henkilöiden välisellä kommunikaatiolla on suuri merkitys. Kun ryhmän koko kasvaa, kasvaa myös kommunikaatiokatkoksien riski. Jos ohjelmiston kehittäjiä on vain yksi, kommunikaatiopolkujakin on vain yksi, eikä kommunikaatio-ongelmaa synny. Kehittäjien määrän kasvaessa kahteen kommunikaatiopolkujen määrä nousee neljään. Tämä tarkoittaa sitä, että tilanteessa on silloin kaksi viestin lähettäjää ja kaksi vastaanottajaa (Leon 2000: 36 - 37). Kuvassa 1 on havainnollistettu kommunikaatiopolkujen määrä ryhmän jäsenten määrän kasvaessa.



Kuva 1. Kommunikaatiopolkujen määrä

Kohdasta c voidaan havaita, että neljän henkilön ryhmä muodostaa jo varsin monimutkaisen tavan kommunikoida. Kommunikaatiopolkujen määrä nousee 24:ään, joten on todennäköistä, että tilanteesta seuraa ongelmia, ellei ryhmässä ole määritelty hyvin suunniteltua tapaa päästä käsiksi tietoon.

Jaettu data

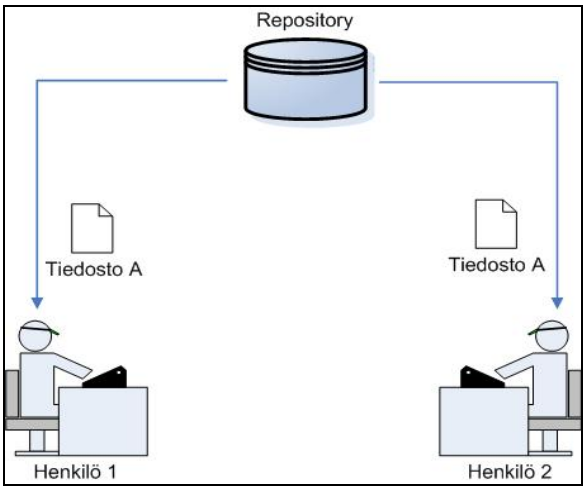
Kun projektissa työskentelee enemmän kuin yksi henkilö, ennen pitkää syntyy tilanne, jossa kaksi tai useampi henkilö muokkaa samaa tiedostoa. Tässä tilanteessa käy niin, että toinen käyttäjä ylikirjoittaa ensimmäisen käyttäjän tiedostoon tekemät muutokset. Tällöin vii-

meksi tiedostoa muokanneen muutokset jäävät voimaan. Tämä on tietenkin tilanne, jota tulee välttää.

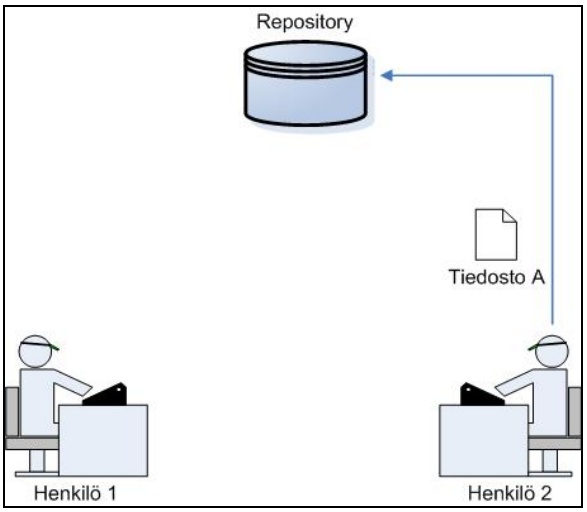
Collins-Sussmanin, Fitzpatrickin ja Pilatonin (2004: 9) mukaan edellä esitettyyn ongelmaan on kaksi ratkaisua: *lock-modify-unlock* (*lukitse-muokkaa-vapauta*) tai *copy-modify-merge* (*kopioi-muokkaa-yhdistä*). Useat nykyaikaiset versionhallintaohjelmistot tukevat lukitse-muokkaa-vapauta-mallia. Malli antaa vain yhden henkilön kerrallaan muokata tiedostoa, jolloin muilla henkilöillä on lukituksen aikana vain lukuoikeus tiedostoon. Muokkauksen jälkeen henkilö vapauttaa lukituksen, jonka jälkeen tiedosto on muiden muokattavissa. Lukitse-muokkaa-vapauta-mallissa on kuitenkin seuraavanlaisia kehitystyötä rajoittavia ongelmia:

- Järjestelmänvalvojatason ongelma. Käyttäjä voi unohtaa vapauttaa lukituksen, jolloin sen vapauttamiseen tarvitaan järjestelmänvalvojan oikeuksia. Lukitus aiheuttaa viivettä kehitystyössä.
- Tiedon serialisointi. Kaksi tai useampi henkilö haluaa muokata samaa tiedostoa, jolloin jokainen joutuu odottamaan vuoroaan. Tilanne voi muodostaa myös ongelman, jossa henkilö luo tiedostosta kopion, jolloin syntyy kaksi toisistaan riippumatonta tiedostoa. Tästä aiheutuu ylläpidollisia ongelmia.
- Turvallisuus. Malli ei pysty valvomaan sisällön turvallisuutta esimerkiksi sellaisessa tilanteessa, että kaksi toisistaan riippuvaa tiedostoa muokataan erillään siten, että niistä tulee yhteensopimattomia.

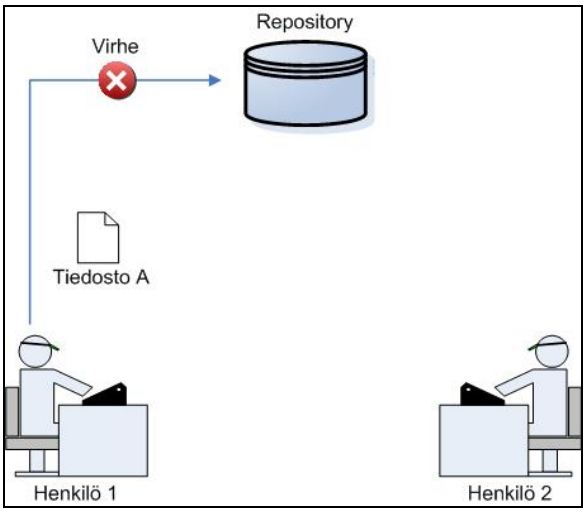
Nykyaikaiset versionhallintaohjelmat tukevat *copy-modify-merge*-mallia. Tässä mallissa jokainen käyttäjä ottaa yhteyden tietovarastoon ja luo itselleen työkopion (Kuva 2.). Työkopio sisältää tietovaraston tiedostot ja hakemistot, jolloin jokainen kehittäjä voi jatkaa työskentelyään omalla työkopiollaan, eikä kenenkään tarvitse odottaa muokkausvuoroaan. Kun halutut muutokset on tehty työkopioon, se viedään takaisin tietovarastoon (Kuva 3.). Tämä ei kuitenkaan tarkoita sitä, että tietovarastoon vieminen aina onnistuisi. Nimittäin sillä hetkellä, kun kopio viedään tietovarastoon, on joku toinen jo saattanut tallentaa sinne oman kopionsa tiedostosta. Tällöin se, joka yrittää viedä tietovarastoon oman kopionsa, saa ilmoituksen siitä, että kopio ei ole ajan tasalla ja että se on muuttunut sen jälkeen, kun kopio on viimeksi haettu (Kuva 4.). Jotta henkilö voisi tallentaa haluamansa muutokset, on hänen haettava viimeisin versio tietovarastosta, jolloin siellä olevat muutokset yhdistetään kopioon ja henkilöllä on jälleen käytettävissään ajan tasalla oleva versio (Kuvat 5 ja 6.).



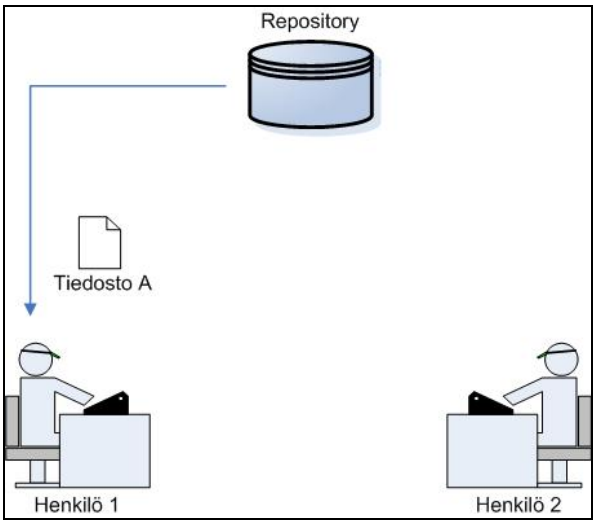
Kuva 2. Työkopion ottaminen



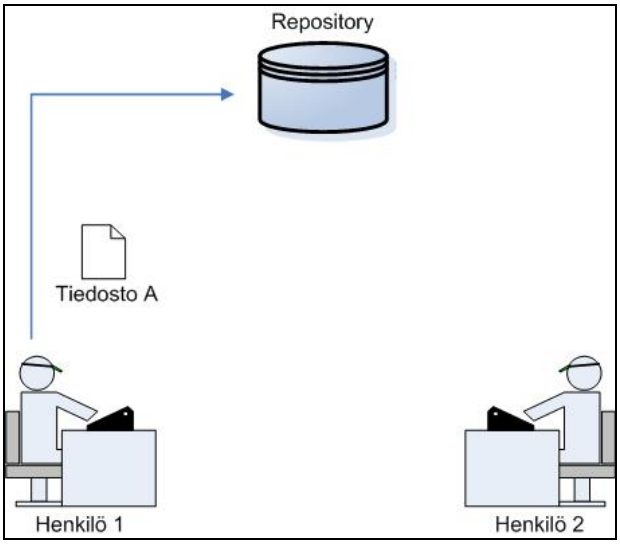
Kuva 3. Muutoksen vieminen tietovarastoon



Kuva 4. Tiedosto on muuttunut kopioinnin jälkeen.



Kuva 5. Muutosten yhdistäminen työkopioon



Kuva 6. Yhdistetyn version vienti tietovarastoon

Collins-Sussman ym. (2004: 12) korostavat kommunikaation merkitystä versionhallinnassa. Osa versionhallinnassa syntyneistä konflikteista voidaan selvittää vain kommunikoimalla toisen henkilön kanssa. Konflikteja ei kuitenkaan voida välttää täysin, eikä edes tiedon lukitseminen ehkäise konfliktien syntymistä. Päinvastoin, lukitseminen hidastaa kehitystyötä.

Lukitseminen ei kuitenkaan ole täysin poissuljettu vaihtoehto versionhallinnassa, ja oikein käytettynä sillä on oma merkityksensä. Tiedostoja, joita ei pystytä muokkaamaan eikä yhdistämään versionhallinnan keinoin, voidaan lukita. Yleensä tällaisia tiedostoja ovat binääri-, ääni- ja kuvatiedostot.

3.4.2 Versionhallinnan hyödyt

Ohjelmistokehityksen prosessi on jatkuvassa muutoksessa. Siihen osallistuu usein suuri määrä henkilöitä, jotka työskentelevät suunnittelun, kehitystyön, testauksen tai projektin johtotehtävissä. Pelkästään henkilöstö muodostaa prosessissa oman vaativuutensa, mutta myös ohjelmistojen lisääntynyt monimutkaisuus ja laajuus lisäävät tarvetta hallita jatkuvaa muutosprosessia.

Leon (2000: 46) jakaa versionhallinnasta saatavat hyödyt seitsemään eri osaan:

1. lisääntynyt tuottavuus
2. alemmat ylläpitokulut
3. parempi laatu
4. virheiden määrän vähentyminen
5. nopeampi virheiden ja ongelmien tunnistaminen
6. prosessista riippuva kehitys
7. tiedon oikeellisuus.

Kommunikaatio- ja jaetun datan ongelmat ovat ongelmia, jotka kuluttavat paljon aikaa, aiheuttavat tiedon kahdentumista ja lisäävät monimutkaisuutta. Ongelmien lisääntyessä myös tuottavuus vähenee. Kun kommunikaatio on hyvin suunniteltu ja organisaatiossa tiedetään miten muutokset voidaan käsitellä, säästyy aikaa, ongelmat vähenevät ja tuottavuus nousee. (Leon 2000: 48)

Koska ohjelmistokehitys on jatkuva prosessi, olemassa olevaan ohjelmakoodiin tehdään muutoksia jatkuvasti. Kun ohjelmiston suunnittelu ja kehitys tehdään järjestelmällisesti sekä kontrolloidusti, alentaa versionhallinta kehitystyöstä aiheutuneita ylläpitokuluja. (Leon 2000: 49)

Virheitä syntyy aina kehityksen aikana, eikä niistä päästä koskaan täysin eroon. Virheiden syntyminen tulisi kuitenkin pyrkiä estämään etukäteen laadunvarmistuksen avulla. Hassin (2002: 6) mukaan versionhallinta toimii vuorovaikutuksessa laadunvarmistuksen kanssa. Virheen jäljitys tulee aloittaa heti, kun objekti on viety versionhallintaan. Kun virheet on jäljitetty ja korjattu, täytyy olla olemassa myös mekanismi, jolla voidaan selvittää, mitä ja miksi muutoksia on tehty, sekä kuka ne on tehnyt. Virheistä tehdään raportti, joka viedään myös versionhallintaan. Raportin avulla voidaan myöhemmin nopeasti selvittää ongelma, ja näin virheen korjaukseen käytettävä aika pienenee. (McConnell 1998: 129)

Leonin (2000: 52) mukaan prosessipohjaisessa kehityksessä ohjelmistokehitys riippuu itse prosessista eikä ihmisistä. Ihmisestä riippuvainen kehitys on vaarallista, koska esimerkiksi henkilön lähtiessä

yrityksestä pois, voi poislähtijä viedä koko projektia koskevan tiedon mukanaan. Ongelman vakavuutta lisää edelleen se, että mikäli projektin dokumentaatio ei ole ajan tasalla, saattavat muut joutua aloittamaan koko työn alusta.

Koska ohjelmistot ovat jatkuvassa muutosprosessissa koko niiden kehityksen ajan, täytyy olla olemassa keino, jolla varmistetaan, että ne vastaavat juuri niihin kohdistuvia vaatimuksia ja muutoksia. Dokumentointiprosessilla varmistetaan, että muutokset ovat oikeita sekä suunnitelman mukaisia, ja näin ollen taataan tiedon oikeellisuus. (Leon 2000: 53)

3.5 Riskit

Kuten Keyes (2004: 16) toteaa, ohjelmiston kehitysprosessiin liittyy joukko riskejä, eikä versionhallinta ole tässä asiassa poikkeus. Riskien ottaminen on osa versionhallinnan luonnetta, sillä ilman riskejä ei tapahdu kehitystä. Riskit itsessään eivät ole vaarallinen asia, vaan olennaista on se, että ne pitää osata tunnistaa oikein ja reagoida niihin ennalta. Keyes (2004: 16) jakaa versionhallinnan riskienhallinnan kuuteen aktiviteettiin:

1. tunnistaminen
2. analysointi
3. suunnitelma
4. seuranta
5. hallinta
6. viestintä.

Jotta riskejä voitaisiin hallita, ne täytyy ensin tunnistaa. Riskien tunnistaminen ajoissa vähentää sellaisten ongelmien syntymistä, jotka voisivat olla haitaksi projektille. Kun riskit on tunnistettu, ne täytyy osata analysoida. Analyysissä riskit muunnetaan päätöksentekoinformaatioksi, jonka avulla voidaan tehdä suunnitelma. Suunnitelmassa riski-informaatiosta luodaan päätökset ja toiminnot riskienhallintaa varten. Suunnitelmaan kuuluu myös toimenpiteet riskien tunnistamista varten sekä riskien priorisointi. Seurannalla puolestaan tarkkaillaan riskien tiloja ja niiden vähentämistä. Hallinnan avulla taas pyritään korjaamaan poikkeavuudet, jotka eivät kuulu riskien suunnitelman mukaisiin toimintoihin. Tämä tarkoittaa sitä, että kun riskit ovat kontrolloituna, voidaan niissä havaitut muutokset korjata suunnitelman mukaiseksi. Viimeisen aktiviteetin, viestinnän, merkitys on ratkaiseva koko riskienhallinnan kannalta, sillä ilman riittävää viestintää ei voida toteuttaa koko riskienhallintaa. (Keyes 2004: 16.)

Riskienhallinnan suunnitelma tunnistaa riskit kolmella osa-alueella: a) liiketoiminta, b) ihmiset ja c) teknologia. Liiketoimintariskeihin kuuluvat:

- kustannukset
- yrityskulttuuri
- sitoutuminen.

Versionhallintaohjelmiston kustannukset eivät rajoitu pelkästään lisenssikustannuksiin, vaan riskienhallinnassa täytyy ottaa huomioon myös henkilöistä ja resursseista aiheutuvat kustannukset. Jokaiselle yritykselle muodostuu oma yrityskulttuurinsa, jossa kunkin versionhallinnasta vastaavan henkilön täytyy tuntea yrityksen periaatteet ja tavat toimia, jotta versionhallinta saataisiin yhteensopivaksi yrityksen kulttuurin kanssa. Versionhallintaprosessi on monimutkainen, ja varsinkin sitä perustettaessa yritysten on usein vaikea löytää syitä siihen, miksi sitä tulisi käyttää. Jotta prosessissa onnistuttaisiin, täytyy sitoutumisen olla mahdollisimman vahva, sillä pitkäjänteisellä työllä saavutetut hyödyt ovat yritykselle kiistattomat. (Keyes 2004: 16.)

Ihmisiin liittyviin riskeihin kuuluvat:

- huijaaminen
- mieleiset työkalut
- vastustus.

On aina vaarana, etteivät kaikki kehittäjät sitoudu määritettyyn tapaan toimia, vaan yrittävät sisällyttää koodia suoraan versioon ilman, että se menisi läpi koko versionhallintaprosessin. Usein kehittäjillä on myös omat mieleisensä työkalut, jotka eivät ole yrityksen käyttämiä työkaluja. Monesti kehittäjillä on lisäksi puutteelliset tiedot versionhallinnasta, eikä sen vaikutuksia pitkällä aikavälillä tunneta riittävän hyvin. Puutteellisten tietojen ja tiettyihin työkaluihin kohdistuvasta mieltymyksestä aiheutuvasta vastustuksesta voidaan päästä eroon koulutuksen ja kommunikaation avulla ja varmistaa, että versionhallinnasta tulee osa yrityksen kehitysprosessia. (Keyes 2004: 16.)

Teknologiaan liittyvät riskit:

- kontrollin kadottaminen
- turvallisuus
- skaalautuvuus.

Versionhallinnan riskienhallinnalla on tarkoitus löytää ja tunnistaa etukäteen mahdolliset riskit, jotta ne eivät pääsisi yllättämään ja aiheuttaisi vakavia ongelmia. Viestinnällä voidaan pienentää kontrollin kadottamisesta johtuvia riskejä, joten yrityksen sisällä pitäisi kommunikoida mahdollisimman paljon. Versionhallinnan tietokantaan,

tiedostoihin ja dokumentteihin pääsyn turvallisuutta täytyy puolestaan hallita, jotta ne säilyttäisivät luotettavuutensa. (Keyes 2004: 16.)

Ohjelmistoprojekteilla on tapana kasvaa ja monimutkaistua. Aikanaan valittu ja käyttöön otettu versionhallintatyökalu saattaa olla jo alun perin vaatimaton ominaisuuksiltaan, joita kehittynyt ohjelmisto ja muuttuva organisaatio vaativat. Työkalua hankittaessa tämä skaalautumattomuudesta johtuva riski pitää ottaa myös huomioon. (Keyes 2004: 16.)

3.6 Perusteet

3.6.1 Vaihetaso

Usein versionhallinnassa puhutaan *baseline*-termistä. Leonin (2000: 58) mukaan baselinen eli vaihetason rooli on yksi tärkeimmistä rooleista versionhallinnassa. IEEE (IEEE Std 610.12-1990: 12) määrittelee vaihetason seuraavasti: Tuote, joka on tarkastettu ja hyväksytty, toimii jatkokehityksen perustana, ja sitä voidaan muuttaa vain tietyn muutosprosessin kautta. Käytännössä tämä tarkoittaa sitä, että kun objektiin tehdään muutos, se tapahtuu läpi koko versionhallinnan prosessin ja kaikki muutokset tallennetaan. Aina, kun muutos tapahtuu, syntyy uusi vaihetaso. Vaihetaso tarjoaa tietyt muutokset tietyssä tilanteessa, ja niihin voidaan palata missä tahansa kehityksen vaiheessa. Bellagio ja Milligan (Bellagio & Milligan 2005: 6 - 7) jakavat vaihetason merkityksen kolmeen osaan:

- uudelleen käytettävyys
- jäljitettävyys
- raportointi.

Uudelleen käytettävyydellä tarkoitetaan sitä, että tiettyyn versioon voidaan palata uudelleen ja tuottaa siitä uusi julkaisu. Jotta kaikki projektia koskevat tiedot saataisiin selvitettyä, tulisi myös projektin hallinnan dokumentit, suunnitelmat, testit ja ohjelmistoon liittyvät tuotokset olla versionhallinnassa eli näin ollen myöhemmin jäljitettävissä. Raportoinnilla puolestaan saadaan myöhemmin selville esimerkiksi muutokset kahden eri version välillä. (Bellagio & Milligan 2005: 6 - 7.)

3.6.2 Versio ja variantti

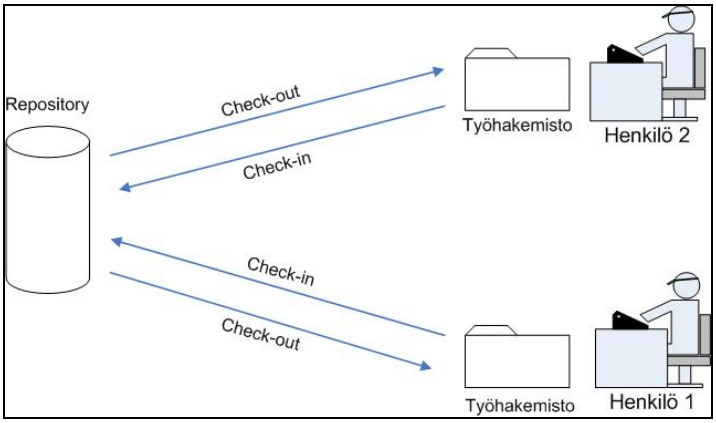
IEEE:n (IEEE Std 610.12-1990: 81) mukaan termillä *versio* tarkoitetaan alustavaa tai uudelleenjulkaisua konfiguraatio-objektista. Versio on ilmentymä järjestelmästä, joka poikkeaa muista ilmentymistä toiminnallisuuden tai ominaisuuden osalta.

Variantti voi olla version kaltainen, ja se voi sisältää kaiken saman toiminnallisuuden, kuin versiokin. Variantti eroaa kuitenkin versiosta siten, että se on suunniteltu käytettäväksi kokonaan eri ympäristöissä, kuten esimerkiksi eri käyttöjärjestelmissä tai laitteistoissa. (Leon 2000: 62.)

3.6.3 Check-in–Check-out-prosessi

Luvussa 3.4.1.2 todettiin, että henkilön täytyy ottaa työkopio tiedostoista, joita hän haluaa muokata. Halutut tiedostot haetaan komennolla *check-out*, jolloin versionhallintaohjelma tuo paikalliset kopiot tiedostoista muokattavaksi työhakemistoon. Mason (2006: 12) huomauttaa, että *check-out*-prosessi on tehtävä, vaikka henkilö työskentelisikin samalla työasemalla, jossa tietovarasto sijaitsee. Mason mainitsee myös, että check-out-prosessi varmistaa sen, että henkilö saa käyttöönsä viimeisimmät ja päivitetyt kopiot tiedostoista.

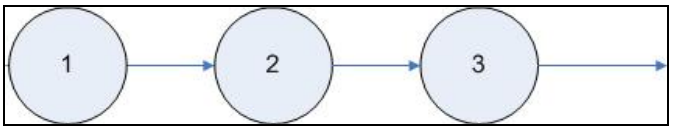
Kun tiedostoihin on tehty halutut muutokset ja niiden toiminta on testattu, pitää ne viedä takaisin tietovarastoon. Tätä prosessia kutsutaan nimellä *check-in*. Kuva 7 kuvaa molemmat prosessit.



Kuva 7. Check-out- ja check-in-prosessit

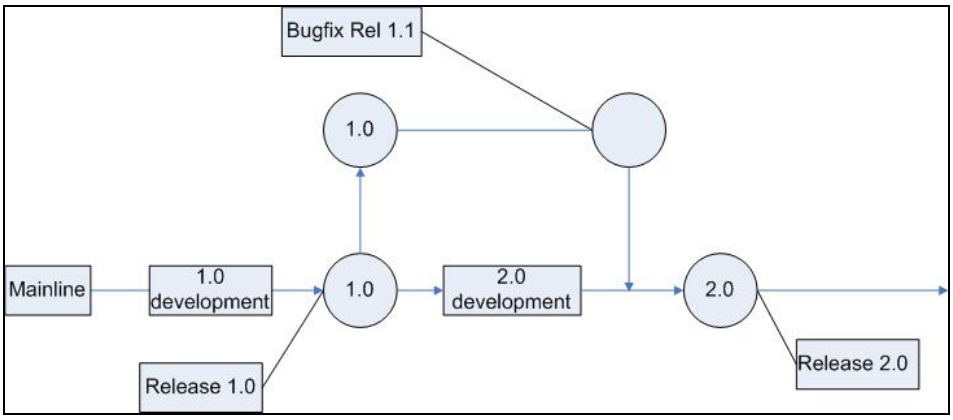
3.6.4 Haarautuminen

Haarautuminen eli *branching* on version erottamista omaksi versiokseen, jota voidaan kehittää riippumatta muista versioista ja johon tehdyt muutokset eivät vaikuta muihin kehityslinjoihin. Baysin (2004: 18) mukaan haarautuminen on yksi monimutkaisimmista toiminnoista versionhallinnassa, ja se vaatii hyvän suunnittelun ja strategian ennen toteuttamistaan. Normaalisti version kehitys on kuvan 8 mukaisesti lineaarinen.



Kuva 8. Version lineaarinen kehitys

Lineaarinen kehitys ei ole kuitenkaan aina mahdollista, eikä se ole aina edes järkevää. Usein versionhallinnasta puhuttaessa tulee esiin termi *version jäädyttäminen*, joka tarkoittaa sitä, että koodiin ei saa tehdä muita muutoksia kuin kriittisiä korjauksia, ennen kuin versio on valmis julkaistavaksi. Berczukin (2002: 13) mukaan koodin jäädyttämisellä on olemassa negatiivisia vaikutuksia, kuten turhautumista ja resurssien hukkaamista. Esimerkiksi, kun koodi on jäädytettynä pitkän aikaa, päiviä tai viikkoja, on vaarana, että kehittäjät alkavat vaihtaa tiedostoja keskenään ohittaen kokonaan versionhallintaprosessin. Jotta voitaisiin taata turvallinen ja tehokas tapa jatkaa kehitystä, on hyvä käyttää haarautumista koodin jäädytyksen sijaan. Tällöin kehittäjillä on mahdollisuus jatkaa kehitystä version julkistamisesta huolimatta ja välttää turhautumista sekä versionhallintaprosessin ohittamisesta johtuvia ongelmia. Kuvassa 9 on kuvattuna haarautuminen, jossa versiosta 1.0 on tehty oma haaransa virheen korjausta varten ja kehitys versioon 2.0 voi jatkua samaan aikaan. Kun virhe on korjattu, korjattu koodi yhdistetään version 2.0 kehityslinjaan, jotta virhe ei toistuisi uudessa versiossa.



Kuva 9. Haarautuminen

Moreira (2004: 82 - 83) jakaa haarautumat kuuteen tyyppiin:

1. pääkehityslinja (mainline tai trunk)
2. julkaisu (release branch)
3. integraatio (integration branch)
4. jaettu (shared branch)
5. yksityinen (private branch)
6. korjaus (bugfix branch).

Pääkehityslinjasta käytetään usein termejä *mainline* tai *trunk*. Pääkehityslinja on kehityksen perusta, josta haarautumia lähdetään tekemään ja johon haarautumien muutokset yhdistetään. (Moreira 2004: 82.)

Julkaisua varten luodaan haarautuma, johon siirretään kaikki koodi, joka on testattu ja valmis asennuksen luontia varten. Integraatiota taas voidaan käyttää julkaisuhaarautuman kanssa, jossa kaikki projektin jäsenet kokoavat työnsä integraatiohaarautumaan. Integraatiolla varmistetaan, että kaikki julkaistavat koodit ja komponentit toimivat yhdessä. Kun integraatio todetaan toimivaksi, se yhdistetään julkaisun haarautumaan. (Moreira 2004: 82.)

Jaettu haarautuma luodaan integraatiohaarautumasta, mikäli yksittäinen henkilö haluaa välttää muiden tekemät muutokset. Kun muutokset on testattu ja todettu toimiviksi, ne yhdistetään takaisin integraatiohaarautumaan. Yksityinen haarautuma luodaan joko integraatio- tai jaetusta haarautumasta, jossa yksittäinen henkilö tekee muutoksia ilman, että se vaikuttaisi muihin. Kun muutokset todetaan valmiiksi, ne testataan ja yhdistetään haaraan, josta se on luotu. Korjaushaarautuma on nimensä mukaisesti eristämistä omaksi haarakseen virheen korjausta varten. Korjauksen jälkeen koodin toiminta testataan, ja muutokset yhdistetään takaisin pääkehityslinjaan. (Moreira 2004: 83.)

3.6.5 Nimeäminen

IEEE (IEEE Std 828-1998: 6 - 7) määrittelee nimeämisen, eli miten eri versiot tulee yksilöllisesti tunnistaa. Tunnistamismetodien tulisi sisältää nimeämissäännöt, versionumerot ja kirjaimet. Standardissa on kuvattu metodit eri objektien tiloille, kuten tallennukselle, jäljitykselle, jakelulle, korjaamiselle ja jatkokehitykselle.

Versionumerointi on välttämätön versionhallinnassa, mutta se ei välttämättä yksinään kuvaa, mitä milläkin vaiheella on tarkoitus tehdä. Kuten Berczuk (2002: 119 - 120) mainitsee, jokaisella koodilinjalla on tarkoituksensa, ja kehittäjä tunnistaa sen nimestä. Hyvässä nimeämistavassa käytetään siis numeroita ja kirjaimia, joilla osoitetaan objektin asema versionhallinnassa. Esimerkiksi ”Release 1.5” kertoo sen merkityksestä paremmin kuin pelkkä numerosarja ”1.5” ja että se on luotu version ”1.0” jälkeen, mutta ennen versiota ”2.0”. Nimeämistavassa tulee kiinnittää huomiota siihen, ettei nimistä synny duplikaatteja eli kahteen kertaan esiintyviä nimiä. Tämä nimittäin johtaisi siihen, ettei tunnistaminen olisi enää yksilöllistä eikä jäljitettävyys enää toimisi. (Leon 2000: 99.)

3.6.6 Julkaisut

Luokittelu

Julkaisut eli *releases* ovat osa versionhallinnan prosessia, jossa tuote toimitetaan muutosten huolellisen testaamisen jälkeen asiakkaiden saataville. Julkaisut jaetaan viiteen luokkaan merkitystensä mukaan:

1. Alpha Release
2. Beta Release
3. Final Release
4. Update ja Upgrade Release
5. Patch ja Emergency Fix.

Alpha-julkaisu on tarpeellinen, kun tuotteeseen on lisätty tärkeimmät toiminnallisuudet. Julkaisu tuodaan vain tarkoin rajatulle määrälle asiakkaita, jotka voivat testata tuotteen ominaisuuksien toimivuutta ja antaa palautetta niistä. Alpha-julkaisuiden tarkoituksena on myös kerätä tietoa virheistä ja ongelmista, joita ne voivat sisältää paljon tässä vaiheessa kehitystä. (Bays 1999: 127 - 128.)

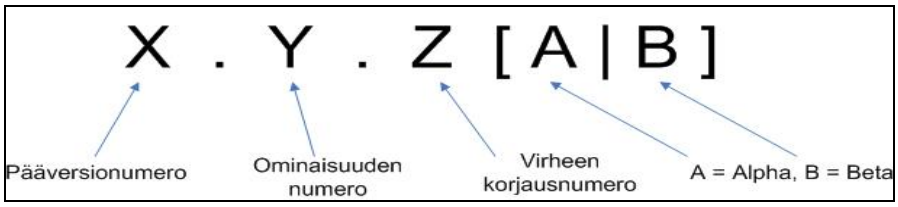
Beta-julkaisu on ominaisuuksiltaan ja toiminnallisuudeltaan usein jo lähellä lopullista julkaistavaa versiota, johon lisättävät muutokset ovat enää muodollisia tai pieniä virheen korjauksia. Beta-version jakaminen voi vaihdella siten, että se voi olla täysin julkinen, jossa jokainen halukas voi testata sitä, tai sitten tarkkaan rajattu ja valvottu jakelu. Beta-julkaisun aikana pyritään keräämään riittävästi tietoa esimerkiksi virheistä, jotta voitaisiin päättää lopullisen version julkaisusta. Virheiden määrä ja vaikeusaste voivat viivästyttää lopullista version valmistumista huomattavasti. Lopullinen versio eli *final release* on kaikille niille asiakkaille toimitettava versio, jotka päättävät hankkia sen. (Bays 1999: 130 - 131.)

Kun versio on julkaistu, aletaan nykyisen version pohjalta tehdä uusia ominaisuuksia ja korjata edellisen version virheitä. Tällaista uutta versiota kutsutaan termillä *upgrade release,* ja se sisältää sekä kaikki edellisten versioiden ominaisuudet että uudet ominaisuudet, jolloin vanhoilla asiakkailla säilyvät kaikki asetukset päivitettäessä uuteen versioon. Usein myös aiempien versioiden omistajia kannustetaan päivittämään uuteen versioon hinnoittelemalla se alemmaksi kuin tuotteen täysi hinta. Ohjelmistotuotteissa esiintyy virheitä koko niiden elinkaaren ajan, eikä niistä päästä täysin eroon koskaan. Kaikki asiakkaat eivät halua uusinta versiota, mutta jo olemassa oleviin ohjelmistoihin joudutaan tekemään virheiden takia päivityksiä. Tällaisia päivityksiä kutsutaan termillä *update release.* Päivityksissä on korjattu vain virheitä, eivätkä ne sisällä uuden version sisältämiä uusia ominaisuuksia tai toimintoja. (Bays 1999: 131.)

Ajoittain virheet saattavat olla kriittisiä, eikä niiden korjauksen ajoitusta voida siirtää normaaliin kehitysprosessiin, vaan niistä joudutaan mahdollisimman pian tekemään erillinen korjauspäivitys, jota kutsutaan termillä *patch* tai *emergency fix*. Kriittiset korjaukset on luonnollisesti huomioitu myöhemmissä normaaleissa julkaisuissa. (Bays 1999: 131 - 132.)

Numerointi

Versionumeron avulla voidaan tunnistaa yksilöllisesti tuotteen toiminnallisuuden tila, joka vaihtuu julkaisusta toiseen. Versionumero koostuu kahdesta elementistä: tieto-osasta, joka on yleensä numero, sekä erotinosasta, joka ilmaistaan pisteellä. Versionumero voi koostua useista tieto-osista, kuten kuvasta 10 voidaan todeta. (Bays 1999: 136.)



Kuva 10. Numeroinnin syntaksi

Pääversionumero ilmaisee tuotteen toiminnallisuuden kasvun. Esimerkiksi asiakas voi hyödyntää sitä päättäessään päivittää versionsa uudempaan. Pääversionumeron kasvu voi merkitä täysin erillistä tuotetta, riippuen yrityksen lisensoinnin tavasta. Se merkitsee myös pääsääntöisesti uutta haarautumaa tai kehityslinjaa. (Bays 1999: 137.)

Ominaisuuden tieto-osasta käy ilmi, mitä muutoksia ominaisuuksiin on tehty, mitä uusia ominaisuuksia on lisätty tai mitä merkittäviä muutoksia tuotteeseen on tehty. Virheiden korjausnumeron tieto-osa puolestaan ylläpitää numerointia, jota joudutaan kasvattamaan virheitä korjattaessa. Version ollessa vielä alpha- tai beta-kehitysvaiheessa voidaan versionumeroinnissa ilmaista sitä kirjaimilla A tai B. On myös mahdollista, että alpha- tai beta-versiota ilmaisevan kirjaimen perään liitetään vielä numero, joka kuvaa kunkin vaiheen iteraatiota, mikäli alpha- tai beta-vaiheessa syntyy useita versioita. (Bays 1999: 137 - 141.)

3.7 Suunnitelma

Versionhallinnan suunnitelma määrittelee, miten versionhallinta toteutetaan käytännössä projektin aikana. Suunnitelman tarkoituksena on dokumentoida, miten ja mitkä eri aktiviteetit suoritetaan, sekä kuka ne suorittaa. Hyvän suunnitelman laatiminen vaatii perehtyneisyyttä versionhallinnan prosesseihin, toimintoihin, standardeihin, työkaluihin, itse projektiin sekä organisaatioon. (Leon 2000: 171.)

Versionhallinnan prosessien toiminnot tapahtuvat ohjelmistokehityksessä joko suunnitellusti tai suunnittelemattomasti. Suunnittelu tekee prosessista kuitenkin tehokkaamman, ja sitä varten tehtävä suunnitelma dokumentoi IEEE-standardin (IEEE Std 828-1998: iii.) mukaan seuraavat asiat:

- versionhallinnan aktiviteetit
- keinot aktiviteettien suorittamiseksi
- vastuuhenkilöt
- aikataulu
- resurssit.

IEEE (IEEE Std 828-1998: 2) jakaa suunnitelman kuuteen luokkaan:

1. johdanto (introduction)
2. hallinta (management)
3. aktiviteetit (activities)
4. aikataulut (schedules)
5. resurssit (recources)
6. suunnitelman ylläpito (plan maintenance).

Johdannon tarkoituksena on kuvata suunnitelman tarkoitusta ja laajuutta niille henkilöille, jotka osallistuvat versionhallinnan prosessiin. Johdannosta käyvät ilmi myös suunnitelmassa käytetyt avaintermit sekä viittaukset standardeihin, terminologiaan, sääntöihin ja suunnitelmassa käytettyihin dokumentteihin. (IEEE Std 828-1998: 3.)

Hallintaluokan tehtävänä on kuvata henkilön tai organisatorisen yksikön velvollisuuksia ja valtuuksia versionhallinnan aktiviteetteihin. Organisatoriset yksiköt voivat koostua muun muassa toimittajista, asiakkaista, hankkijoista ja alihankkijoista. (IEEE Std 828-1998: 4.)

Aktiviteettien informaatio kuvaa versionhallinnan aktiviteetteja, jotka on kuvattu aiemmin luvussa 3.3. Aikataulujen tarkoituksena on puolestaan kuvata aktiviteettien järjestys, riippuvuudet ja suhteet projektin tapahtumiin tai välitavoitteisiin. Aikataulut käsittävät myös suunnitelman keston sekä kaikkien aktiviteettien välitavoitteet. Välitavoitteet sisältävät vaihetasojen perustamisen, versionhallinnan järjestyksen toteuttamisen sekä arviointien aloitus- ja lopetuspäivät. Aikatauluja voidaan myös kuvata erilaisilla kaavioilla, joilla voidaan korostaa sen merkitystä. (IEEE Std 828-1998: 10.)

Resurssien informaation tehtävänä on tunnistaa työkalut, tekniikat, infrastruktuuri, henkilökunta sekä harjoittelu, joita tarvitaan aktiviteettien toteuttamista varten. Versionhallinta voidaan toteuttaa missä tahansa ympäristössä, ja käytetyt työkalut voivat olla joko erityisiä versionhallinnan työkaluja tai sitten integroituna projektiin, varta vas-

ten projektia varten hankittuja työkaluja. Infrastruktuuria suunniteltaessa tulisi ottaa huomioon toiminnallisuudet, kuten suorituskyky, turvallisuus, tilat, laitteet sekä aikarajoitteet. (IEEE Std 828-1998: 11.)

Suunnitelman ylläpidon tehtävänä on turvata sen jatkuvuus koko projektin elinajan, ja siihen kirjataan myös koko sen historia muutoksineen. Suunnitelmaa tulisi alkaa tarkastella heti jokaisen projektin alettua, muuttamaan sitä projektin mukaisesti sekä jakamaan sitä projektin jäsenille. (IEEE Std 828-1998: 11.)

4 Toteutus

4.1 Yritys

UNES Oy on kehittänyt ohjelmistoja 1980-luvulta saakka, jolloin ohjelmistot olivat vielä yksinkertaisia ja asiakasmäärät pieniä. Asiakkaiden määrä ja ohjelmistojen monimutkaisuus lisääntyivät tasaisesti läpi 1990-luvun, ja tultaessa uudelle vuosituhannelle asiakkaiden määrä lähti jyrkästi nousuun. Samalla myös vaatimukset ohjelmistoja kohtaan kasvoivat. Tilanne on johtanut siihen, että ohjelmakoodin muutosten hallinta ei enää ole ollut mahdollista, koska yrityksessä ei ole aiemmin ollut käytössä versionhallintaohjelmaa eikä yhteisiä pelisääntöjä sen noudattamiseksi.

Yrityksessä ei tällä hetkellä tehdä täysin uusia ohjelmistoja, vaan toiminta keskittyy aiemmin valmistettujen ohjelmistojen kehittämiseen ja ylläpitoon. Samasta ohjelmistosta syntyy useita muutos- ja kehitysversioita, joiden välisiä eroja työntekijän täytyy tällä hetkellä hallita ilman versionhallintaohjelmistoa. Normaalisti ylläpitoa ja kehitystä hoitaa yksi henkilö yhtä ohjelmistoa kohden, mutta ajoittain on tilanteita, jolloin projektiin tarvitaan lisäksi toinen kehittäjä. Koska tulevaisuutta on vaikea ennustaa, on versionhallintaa suunniteltaessa otettava huomioon myös sellainen seikka, että henkilöstöä joudutaan lisäämään tai tiettyyn projektiin tarvitaan lisää henkilöresursseja.

Versionhallinnan käyttöönotolla on tarkoitus saada ylläpidettyä ohjelmistojen muutoksia koko kehitysprosessin ajan. Versionhallinnan huolellisella suunnittelulla pyritään siihen, että versionhallinta tulisi osaksi ohjelmistojen kehitysprosessia ja sen käyttäminen olisi vaivatonta ja kannustavaa. Lisäksi versionhallinnan käyttöönotolla halutaan parantaa ohjelmistojen laatua. Tätä opinnäytetyötä käytetään lähtökohtana versionhallinnan käyttöönoton ajan, ja sen tarkoituksena on opastaa versionhallinnan käytössä sekä sen perusteissa. Lisäksi tätä työtä hyödyntämällä yrityksen on mahdollisuus kehittää edelleen versionhallintaprosessia muuttuviin tilanteisiin tai teknologioihin sopivaksi. Mikäli tulevaisuudessa, esimerkiksi .NET Frameworkin myötä, tulee tarve siirtyä toiseen versionhallintaohjelmaan, on siihen siirtyminen kuitenkin helppoa, koska kaikissa versionhallintaohjelmissa toimintaperiaate on pääosin samanlainen.

4.2 Työkalun valinta

Versionhallintaohjelmistoa kartoitettaessa lähtökohdaksi oli otettava nykyinen sovelluskehitin Microsoft Access XP. Sovellukset on tehty siten, että ohjelmaversio ja tietokanta on eriytetty toisistaan ja yhteys tietokantaan toimii linkityksen avulla. Ongelmaksi työkaluja kartoitettaessa muodostui se, että Accessin objektit täytyi saada tietokan-

nasta erilleen ja talletettua vielä tekstitiedostoksi. Ilman tätä toimintoa esimerkiksi koodin muutosten vertailu olisi mahdotonta kahden objektin kesken. Ainoa työkalu, jolla käytännössä pystytään versionhallintaan Accessin kanssa, on Microsoft Visual SourceSafe. Tästä ohjelmasta valittiin versioksi versio 2005. Objektien muuttaminen tekstitiedostoksi muuhun kuin Visual SourceSafeen olisi ollut mahdollista, mutta sen toteuttamiseksi olisi täytynyt tehdä erillinen ohjelma, jonka tekeminen ei olisi ollut mielekästä. Nyt valittu työkalu integroituu Accessiin täysin ja osaa tehdä tarvittavat toiminnot, joten valinta on ollut siksikin luontevin.

4.3 Hallinta

Versionhallinnan käyttöönotto aloitetaan järjestelmänvalvojatoiminnoilla, joilla pyritään turvaamaan, että vain määritetyillä henkilöillä on oikeudet ja vastuut päästä käsiksi versionhallinnan tietoihin. Järjestelmänvalvojaohjelmalla voidaan luoda uusia ja poistaa vanhoja käyttäjiä sekä antaa näille erilaisia oikeuksia projekteihin. Visual SourceSafe ei kuitenkaan pysty suojaamaan tietovarastoa fyysisesti, koska jokainen käyttäjä, jolla on pääsy esimerkiksi verkkoresurssiin, voi tuhota tietovaraston. Tällöin tietovarasto on suojattava Windowsin omilla käyttöoikeuksien määritysoperaatioilla. Yksi hyvä keino tietojen suojaamiseksi on luoda kaksi ryhmää: versionhallinnan järjestelmän valvojat sekä käyttäjät ja lisätä kuhunkin ryhmään sinne kuuluvat henkilöt. Tämän jälkeen voidaan tietovaraston hakemiston ominaisuuksien tietoturvavälilehdeltä antaa näille ryhmille niille kuuluvat oikeudet. Järjestelmänvalvojan ohjelma käynnistetään kuvan 11 mukaisesti *ohjelmat*-valikosta.

Kuva 11. Järjestelmänvalvojan ohjelman käynnistäminen

4.3.1 Käyttäjät

Kun ohjelma on käynnistynyt, ensimmäinen tehtävä on lisätä siihen käyttäjät, joilla on suunnitelman mukaisesti oikeudet ja velvollisuudet käyttää versionhallintaa ja siihen kuuluvia toimintoja (Kuva 12). Käyttäjän nimi saa olla enintään 31 merkkiä pitkä, eikä se saa alkaa välilyönnillä eikä loppua välilyöntiin. Lisäksi nimessä ei saa käyttää pistettä. Salasanaa määritettäessä pituus saa olla enintään 15 numeroa pitkä, ja se saa sisältää mitä tahansa kirjaimia ja merkkejä. Käyttäjää lisättäessä ruudulla on myös valittavana käyttäjälle *Read-only*-

valinta, joka tarkoittaa sitä, että käyttäjä voi vain lukea tietovaraston tietoja, mutta ei tehdä muutoksia sinne.

Kuva 12. Käyttäjän lisääminen

4.3.2 Oikeudet

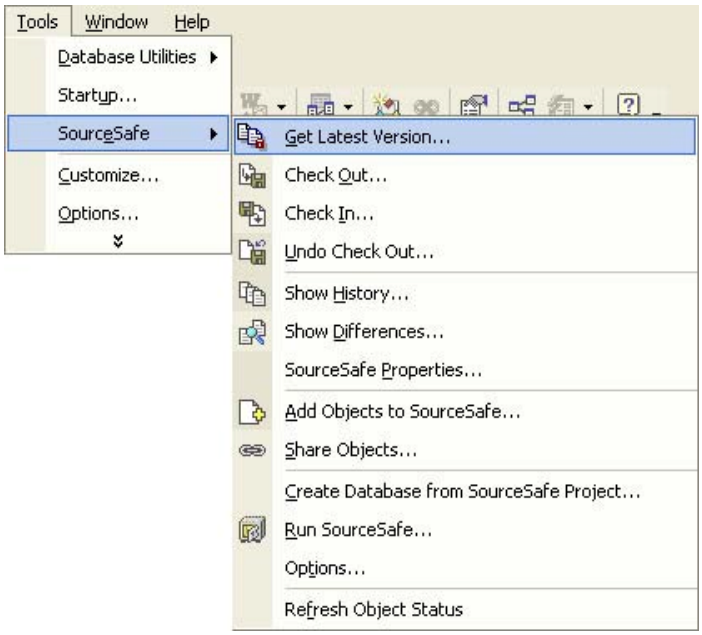
Järjestelmänvalvojan toiminnoissa voidaan määrittää käyttäjälle yksilöllisesti erilaisia oikeuksia valitun tietovaraston projekteihin. Määriteltäviä oikeuksia ovat: 1) *Read* 2) *Check Out/Check In* 3) *Add/Rename/Delete* ja 4) *Destroy*. Destroy mahdollistaa *rollback*-toiminnon käytön, jolla koko muutostapahtuma voidaan peruuttaa, eikä tietoja tallenneta tietovarastoon. Read-oikeus antaa mahdollisuuden vain lukea objekteja, mutta ei muuttaa niitä. Check Out/Check In-oikeuden avulla objektit voidaan hakea tietovarastosta muokattavaksi ja viedä ne muutoksen jälkeen takaisin. Add/Rename/Delete-oikeus antaa oikeuden lisätä, nimetä uudelleen ja poistaa tiedostoja. *Add Assignment* -painikkeella voidaan lisätä henkilölle oikeudet valittuun projektiin. Mikäli yksittäisiä projekteja ei haluta lisätä, henkilölle voidaan antaa oikeudet kaikkiin projekteihin kerralla valitsemalla *$/*-tunnuksella oleva kohta (Kuva 13).

Kuva 13. Oikeuksien lisääminen projekteihin

4.4 Microsoft Access

VisualSourceSafe integroituu Microsoft Accessiin täysin ja tarjoaa tarvittavat toiminnot versionhallinnan käyttämiseen, kuten muun muassa tietovaraston luomisen, objektien lisäämisen ja poistamisen, sekä Check in/Check out-prosessit. Kehittäjän näkökulmasta versionhallinnan käyttö tapahtuu suurimmaksi osaksi kehitystyökalun kautta eli Microsoft Accessista. On syytä huomioida, että versionhallinnan käyttö Accessissa poikkeaa sen rakenteen vuoksi paljon normaalista ohjelmistojen versionhallinnasta, jossa lähdekoodi on aina tekstimuodossa. Accessissa *.mdb*-päätteisen tietokannan sisältämät objektit, kuten kyselyt, lomakkeet, raportit, makrot ja koodimodulit, muutetaan tekstitiedostoiksi versionhallintaohjelman hakemistoon, jossa niihin ei saa koskaan itse tehdä muutoksia, vaan ne tehdään aina kehitystyökalun kautta. Mikäli kehittäjä on entuudestaan käyttänyt versionhallintaa muissa projekteissa, mahdollisesti muilla versionhallintatyökaluilla, voi Visual SourceSafe -versionhallintatyökalun käyttö Accessin yhteydessä olla aluksi hieman hämmentävää. Kuitenkaan Access-ohjelman yhteydessä käytetyt versionhallinnan toiminnot eivät poikkea muissa ohjelmistoprojekteissa käytettävien versionhallintaohjelmien toiminnoista.

Kehitysympäristössä versionhallinnan toiminnot löytyvät työkaluriviltä (Kuva 14), valikosta ja pikavalikosta tietokantaikkunassa.



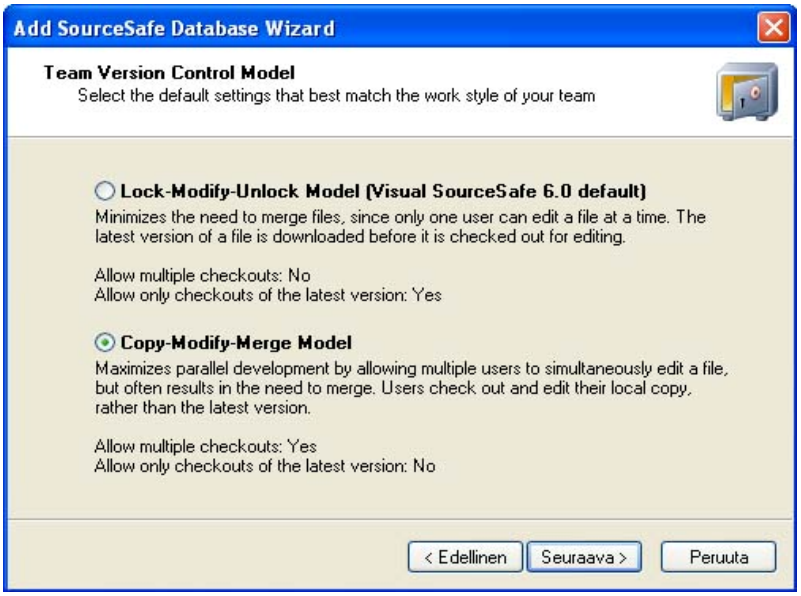
Kuva 14. Versionhallinnan komennot Microsoft Access ohjelmassa

Tietokannan lisääminen

Jotta versionhallinnan toimintoja voitaisiin käyttää tietokannan kanssa, täytyy se lisätä versionhallintaan. Tietokannasta voidaan luoda joko uusi tietovarasto tai se voidaan lisätä jo olemassa olevaan tietovarastoon. Tietovarastoja voidaan luoda miten paljon tahansa, mutta etukäteen tehtävän suunnittelun avulla pyritään pitämään versionhallinnan rakenne mahdollisimman yksinkertaisena.

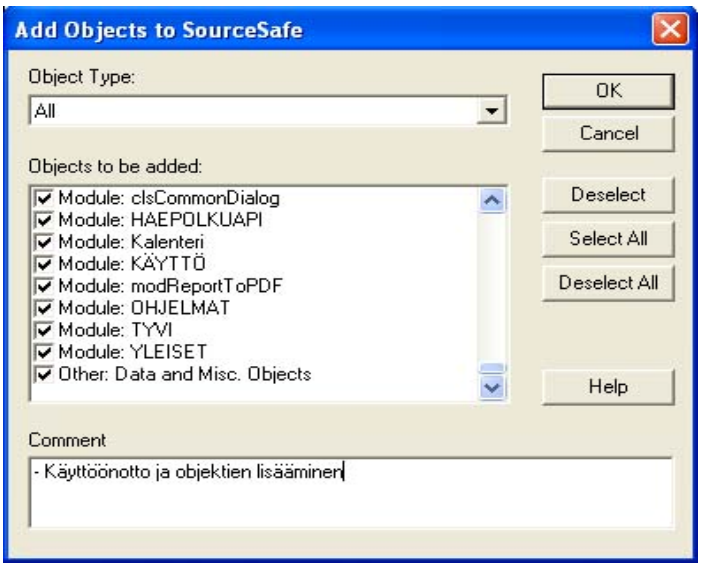
Uusi tietovarasto luodaan valikosta *Tools –> SourceSafe –> Create Database from SourceSafe Project….* Tämän jälkeen avautuu kirjautumisikkuna, josta voidaan valita haluttu tietovarasto. Tässä vaiheessa ei kuitenkaan kirjauduta, vaan valitaan *Browse-painike*, jonka avulla päästään seuraavaan ikkunaan, josta uuden tietovaraston luonti suoritetaan painamalla *Add-painiketta*. Tämän jälkeen Visual SourceSafe käynnistää wizard-toiminnon, jossa käydään vaiheittain läpi tietovaraston lisääminen. Seuraavassa vaiheessa wizard kysyy vielä, lisätäänkö uusi tietovarasto (*Create a new database)* vai liitetäänkö tietokanta olemassa olevaan tietovarastoon (*Connect to an existing database)*. Tässä tapauksessa valitaan ensimmäinen vaihtoehto, eli lisätään uusi. Valinnan jälkeen kysytään hakemistoa, johon tietovarasto luodaan.

Hakemisto voi sijaita myös verkkolevyllä, mikäli versionhallintaa on tarkoitus käyttää jaetusti verkon yli. Wizard haluaa myös tiedon siitä, minkälaista mallia halutaan käyttää tiimissä. Vaihtoehtona on kaksi mallia: a) *Lock-Modify-Unlock* ja b) *Copy-Modify-Merge* (Kuva 15). Paras valinta on *Copy-Modify-Merge*, eli kopioi-muokkaa-yhdistämalli, riippumatta siitä työskenteleekö henkilö yksin tai ryhmässä. Valittu työskentelymalli poistaa lukituksesta aiheutuvat ongelmat, kuten luvussa 3.4.1.2 jo todettiin. Mikäli henkilö työskentelee yksin, mallilla ei ole suurtakaan merkitystä, paitsi siinä tilanteessa, että projektiin liittyy myöhemmin muita henkilöitä. Kun edellä valittu kopioi-muokkaa-yhdistä-malli on jo valmiiksi valittuna, ei muiden liittyminen aiheuta mitään toimenpiteitä tai muutoksia. Mallin valinnan jälkeen tietovaraston lisääminen on valmis, ja seuraavaksi tehtäväksi jää käytettävän ohjelmatietokannan lisääminen siihen.



Kuva 15. Mallin valinta

Ohjelmatietokanta lisätään luotuun tietovarastoon valikosta *Tools –> SourceSafe –> Add Database to SourceSafe.…*. Tämän jälkeen valitaan kirjautumisruudulta luotu tietovarasto ja painetaan *OK-painiketta*. Viimeisin vaihe ennen lopullista käyttöönottoa on ohjelmatietokannassa olevien objektien vienti versionhallintaan (Kuva 16). Objekteista valitaan pudotusvalikosta kaikki (*All)* sekä kirjoitetaan kommentti lomakkeen alareunassa olevaan tekstikenttään ja lopuksi painetaan *OK*. Visual SourceSafe ilmoittaa, kun objektien lisääminen on valmis ja versionhallintaa voidaan alkaa käyttää normaalisti ohjelmiston kehityksen aikana.



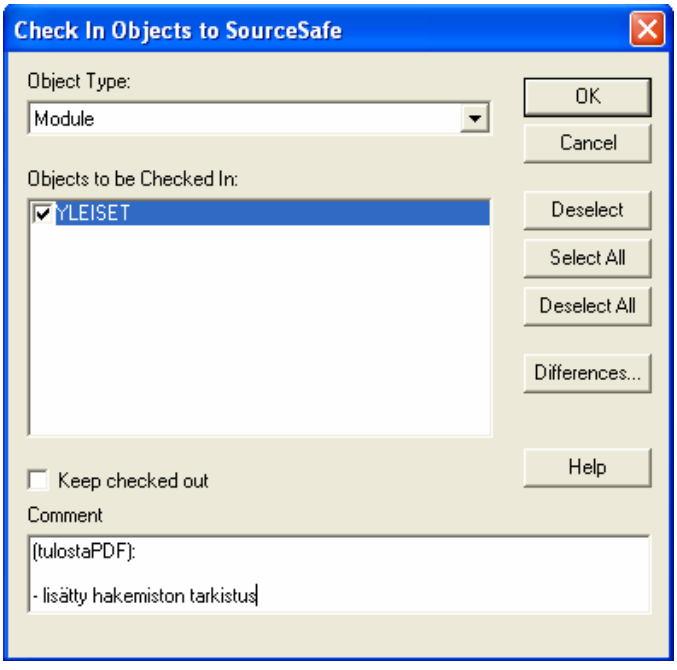
Kuva 16. Objektien lisääminen versionhallintaan

Check-in ja Check-out

Objektien muokkaaminen ei ole enää mahdollista niiden tietovarastoon lisäämisen jälkeen ilman, että objekti otetaan muokattavaksi *Check-out*-komennolla. Jokaisen objektin viereen on ilmestynyt lukkoa kuvaava ikoni, joka ilmaisee sen, ettei objektia ole otettu muokattavaksi. Vastaavasti, kun objekti otetaan muokattavaksi *Check-out*-komennolla, lukkokuvakkeen tilalle vaihtuu oikeinmerkki merkiksi siitä, että objektia voidaan muokata (Kuva 17). Puutteena voidaan havaita, että *Check-out*-toiminnossa ei ole mahdollisuutta lisätä kommenttia, joka tulisi hyvän tavan mukaisesti tehdä, jotta saataisiin käsitys siitä, miksi objekti on otettu muokattavaksi.

Kuva 17. Tilanne ennen ja jälkeen Check-out komennon

Kun objekti tai objektit on otettu muokattavaksi ja tarvittavat muutokset tehty, ne pitää viedä takaisin tietovarastoon komennolla *Check-in.* Viennin yhteydessä täytyy laittaa aina kommentti sille varattuun kenttään siitä, mitä on muutettu, jotta esimerkiksi muut kehittäjät näkisivät tehdyt muutokset. Myös myöhemmin historiatietojen tarkastelussa nähdään tehdyt muutokset kommentteineen, mikä helpottaa esimerkiksi virheen jäljitystä. Kuvasta 18 nähdään, kuinka objekti viedään kommentin kanssa tietovarastoon.

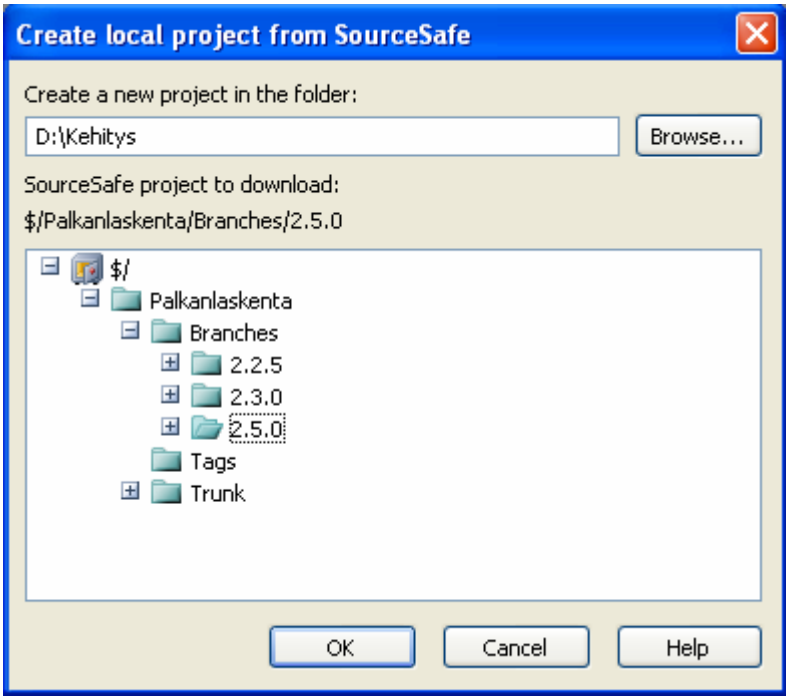


Kuva 18. Objektin vienti tietovarastoon

Liittyminen projektiin

Ohjelmistojen tekemiseen tarvitaan niiden laajuuden ja monimutkaisuuden vuoksi nykyään useita henkilöitä. Koska tietojen turvallisen muokkaamisen täytyy tapahtua versionhallinnan prosessien kautta, on projektiin liittyvien henkilöiden luotava itselleen työhakemisto omaa työkopiota varten. Työkopio on, kuten luvussa 3.4.1.2 todettiin, paikallinen kopio projektin tiedostoista, joita kehittäjät voivat itsenäisesti muokata muista kehittäjistä riippumatta.

Microsoft Accessissa työkopio tehdään luomalla tietokanta projektista, johon kehittäjän tarvitsee liittyä komennolla *Tools –> SourceSafe –> Create Database from SourceSafe Project…*. Tämän jälkeen avautuvasta ikkunasta valitaan tietovarasto, jossa tarvittava projekti sijaitsee (Kuva 19). Kun tietovarasto on valittu, Visual SourceSafe aloittaa objektien siirtämisen Accessiin. Siirron jälkeen kehittäjällä on oma kopionsa tietovarastossa sijaitsevista tiedostoista ja mahdollisuus tehdä muutoksia haluttuihin objekteihin määritetyillä oikeuksilla, jotka antavat siihen mahdollisuuden.

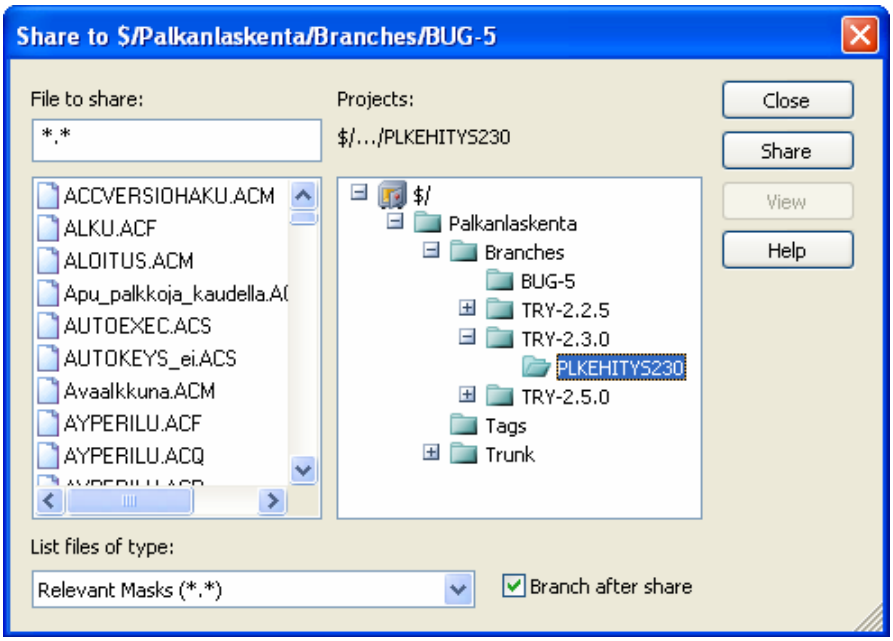


Kuva 19. Liittyminen projektiin

Haarautuman lisääminen

Kehitysprojektin aikana tulee usein tarve tehdä täysin erillinen kehitysversio normaalista pääkehityslinjasta. Tällaisia tilanteita ovat muun muassa virheenkorjaus, uuden ominaisuuden kehittäminen tai julkaisuversion tekeminen. Kuten aiemmin todettiin, haarautuman tarkoituksena on jatkaa version kehittämistä erillään pääkehityslinjasta, eikä haarautumaan tehtävien muutoksien haluta vaikuttaa siihen ennen kuin eri versiot päätetään yhdistää.

Visual SourceSafessa haarautuman luominen on varsin helppo operaatio, joka tehdään Accessin sijasta itse versionhallintatyökalun Explorer-ohjelmassa. Ohjelma käynnistyy joko Accessin valikosta *Tools –> SourceSafe –> Run SourceSafe…* tai Windowsin ohjelmatvalikosta *Microsoft Visual SourceSafe –> Microsoft Visual SourceSafe*. Haarautuma lisätään valitsemalla haluttu kohta versiopuusta ja lisäämällä uusi projekti komennolla *File –> Create Project* sekä antamalla projektille nimi. Tämän jälkeen valitaan versiopuusta juuri luotu projekti ja valitaan valikosta *Versions –> Share to $...*, jonka jälkeen valitaan, minkä projektin tiedostot jaetaan uuteen haarautumaan (Kuva 20). Valinnan jälkeen painetaan *Share*-painiketta. On erityisen tärkeää, että valintaruutu *Branch after share* on valittuna, koska silloin versionhallintaohjelma katkaisee linkit projektiin, josta tiedostot haarautettiin. Jos se ei olisi valittuna, kaikki muutokset haarautumassa aiheuttaisivat muutoksen välittömästi myös vastaavissa tiedostoissa projektissa, josta haarautuma luotiin.



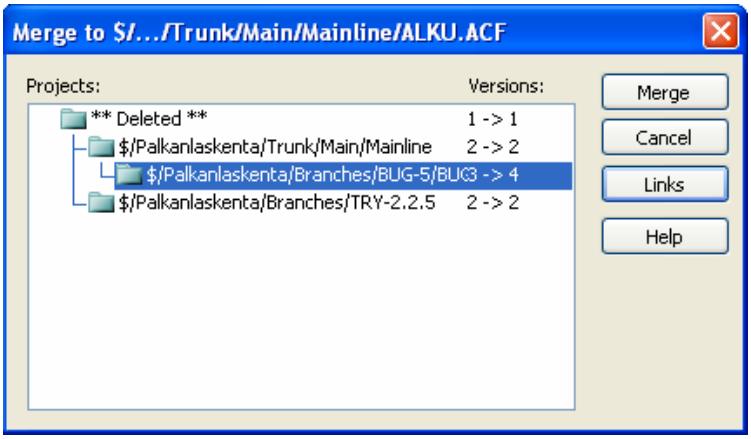
Kuva 20. Haarautuman luominen

Tiedostoista voitaisiin haarauttaa myös yksittäisiä tiedostoja, mutta käytettäessä Accessia tiedostoja voi muokata vain kyseisessä ohjelmassa. Lisäksi erillään muutettujen tiedostojen testaaminen vaatisi aina niiden yhdistämistä johonkin kokonaiseen projektiin ennen niiden mahdollista yhdistämistä pääkehityslinjaan. Näin ollen on syytä valita aina kaikki tiedostot haarautettavasta projektista.

Koska haarautuma perii kaikki tiedot valitusta versiosta, täytyy käyttäjän muokata itse ohjelman nimeä kahdessa objektissa, joiden tiedostopääte on *.ACB* ja *.ACN*. Mikäli nimen muutosta ei tehdä, perityn objektin nimi aiheuttaa sen, että haarautumasta muokkausta varten uutta tietokantaa työhakemistoon luotaessa on ohjelmatietokanta jo olemassa samalla nimellä. Jos jo olemassa oleva ohjelmatietokanta korvattaisiin epähuomiossa uudella, ei varsinaista vahinkoa pääsisi vielä syntymään, koska korvattu tietokanta sisältää täsmälleen samat objektit kuin korvaavakin. Versionhallintaohjelma kuitenkin kysyy korvauksen vahvistusta, joten viimeistään tässä vaiheessa on projektin nimi vaihdettava kahteen edellä mainittuun objektiin. Kun nimi muutetaan halutuksi, voidaan Accessissa luoda uusi projekti valitsemalla haarautuma versiopuusta, kuten kappaleessa ”Liittyminen projektiin” mainittiin. Luotu projekti ilmestyy nimettynä Access-tietokantana hakemistoon.

Haarautuman yhdistäminen

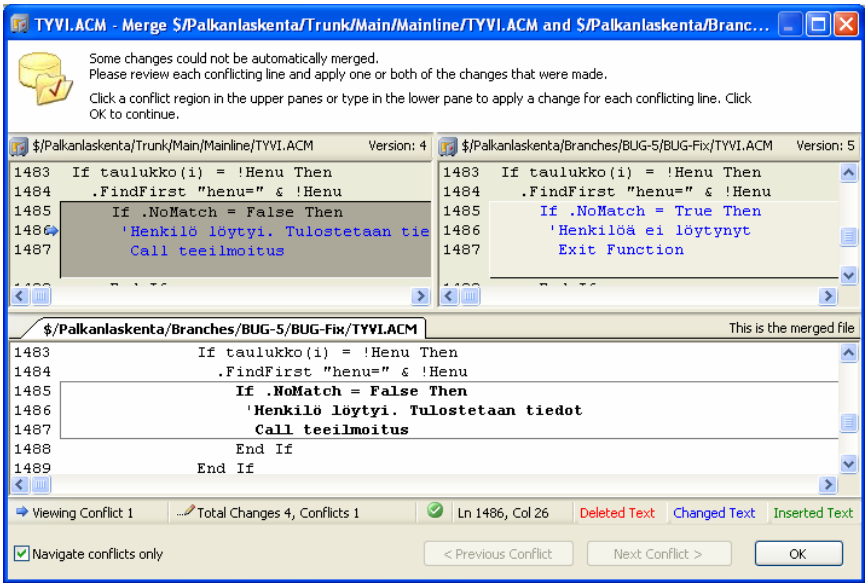
Kehitettäessä uusia ominaisuuksia tai korjattaessa virheitä ohjelmassa täytyy näistä haarautumista saada huolellisen ja perusteellisen testauksen jälkeen yhdistettyä pääkehityslinjaan vain muutokset. Yhdistäminen tapahtuu haarauttamisen tavoin Visual SourceSafe Explorer -ohjelmassa, jossa valitaan versiopuusta se projekti, johon muutokset halutaan yhdistää. Kun projekti on valittu, kaikki projektissa olevat objektit listautuvat oikeanpuoleiseen ikkunaan, josta valitaan se objekti, johon yhdistäminen tapahtuu. Tämän jälkeen valitaan valikosta *Versions –> Merge Branches…*, jolloin versionhallintaohjelma näyttää listan haarautumista, josta valitaan se projekti, josta muutokset tuodaan (Kuva 21).



Kuva 21. Muutosten yhdistäminen haarautumasta pääkehityslinjaan

Kun projekti on valittu, painetaan *Merge*-painiketta, jonka jälkeen lisätään vielä kommentti siitä, mitä ollaan tekemässä. Ennen lopullista yhdistämistä versionhallintaohjelma kysyy: *”Have any conflicts in object.xxx been properly resolved?”* Kysymys voi johtua siitä, että kaksi tai useampi käyttäjä on ottanut objektin muokattavaksi. Mikäli objektia ei ole otettu muokattavaksi, kysymys on merkityksetön ja siihen vastataan *Yes*.

Erityinen tilanne voi esiintyä, kun kaksi tai useampi henkilö on muokannut samaa tai samoja rivejä koodissa samaan aikaan. Tällöin Merge-komennon jälkeen avautuu koodi-ikkuna, joka on jaettu kolmeen osaan (Kuva 22). Konflikti selvitetään siten, että valitaan vierekkäisistä alueista klikkaamalla rajatun alueen sitä puolta, jossa on haluttu muutos. Valitun alueen koodi siirtyy automaattisesti alla olevaan ikkunaan, jossa nähdään lopullinen yhdistettävä versio. Tämän jälkeen yhdistäminen tapahtuu painamalla *OK*-painiketta. Ohjelma ilmoittaa vielä: *“All conflicts has been resolved, would you like to save the file?”* Tässä vaiheessa yhdistäminen voidaan vielä peruuttaa, mutta pääsääntöisesti se hyväksytään *Yes*-painikkeella.

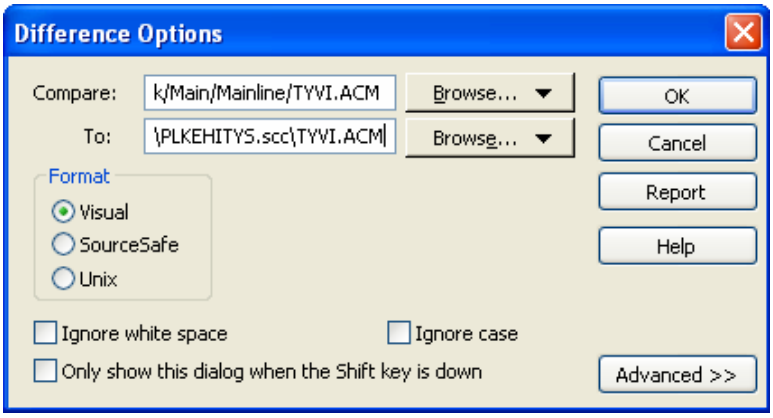


Kuva 22. Konflikti muutoksen yhdistämisessä

Muutosten vertailu

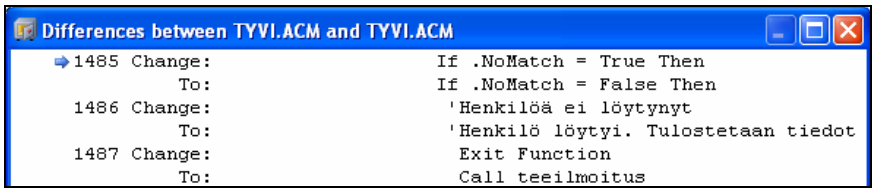
Muutosten vertailu eri versioiden välillä on usein tarpeellista, koska varsinkin kehitystyössä, jossa on osallisena useita henkilöitä, halutaan tutkia, mitä muutoksia lähdekoodiin on tehty. Muutoksien tutkiminen on myös tarpeellista, vaikka projektiin osallistuisi vain yksi henkilö, koska itse tehtyjen muutosten muistaminen voi olla vaikeaa tai jopa mahdotonta.

Tehtyjä muutoksia voidaan tutkia kehitystyökalun välineillä esimerkiksi valitsemalla haluttu objekti ja sen jälkeen valitsemalla valikosta *Tools –> SourceSafe –> Show Differences…*. Ensimmäisessä tekstikentässä (*Compare:)* on se versio, jota halutaan verrata, ja toisessa tekstikentässä (*To:)* versio, johon sitä halutaan verrattavan. *Browse*-painikkeilla voidaan valita haluttu versio (Kuva 23).



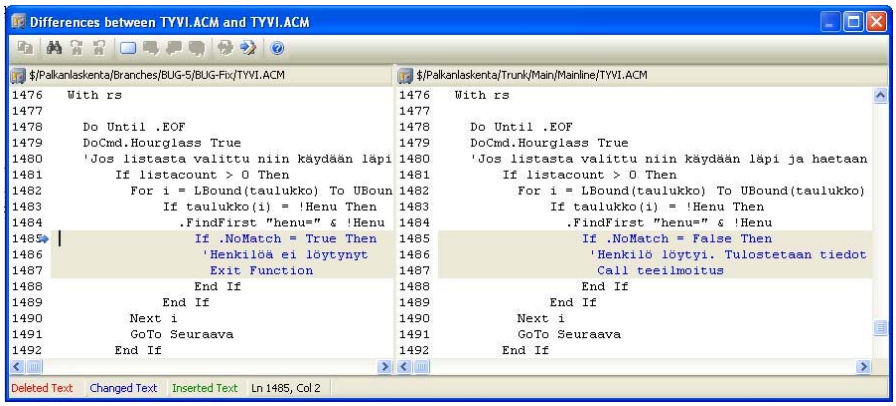
Kuva 23. Vertailtavien versioiden valinta

Muutoksia on mahdollista tarkastella kolmessa eri muodossa. Visuaalinen muoto näyttää muutokset eri versioiden välillä ikkunassa, joka on jaettu kahteen osaan siten, että vasemmalla on vertaileva koodi ja oikealla vertailtava koodi. SourceSafe-valinta tulostaa muutokset tekstipohjaisena, jossa kerrotaan, mitä on muutettu milläkin rivillä (Kuva 24).

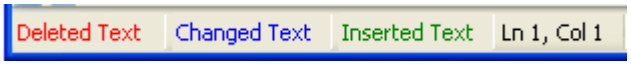


Kuva 24. Muutosten vertailu SourceSafe-muodossa

Visuaalisessa ikkunassa muutokset versioiden välillä on merkitty nuolella sekä maalatulla alueella. Lähdekoodi on merkitty värillä, joka kuvaa tehtyä muutosta (Kuvat 25 ja 26.). Punainen väri tarkoittaa, että tietoa on poistettu, sininen merkitsee sitä, että koodia on muokattu, ja vihreä sitä, että koodia on lisätty. Lähdekoodissa voi olla useita kohtia, joita on muutettu, jolloin seuraavaan muutokseen pääsee *F7*-painikkeella tai valitsemalla työkaluriviltä toinen oikealta olevan painikkeen.



Kuva 25. Muutosten vertailu visuaalisesti

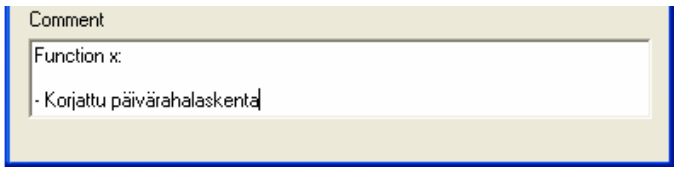


Kuva. 26. Muutosta kuvaavat värit

Yhteenveto perustoiminnoista

Edellisistä luvun 4.4 alaluvuista voidaan todeta, että huolellisen suunnittelun jälkeen versionhallinnan toimintojen käyttö on varsin suoraviivaista ja helppoa. Suurimmaksi osaksi kehitystyössä käytettävien toimintojen käyttö on vain muutaman perustoiminnon, kuten Check-In ja Check-Out, käyttöä.

Check-In-prosessissa on kiinnitettävä erityistä huomiota kommentointiin, kuten esimerkiksi siihen, mitä ja mihin muutoksia on tehty. Kommentoinnin tulee olla selkeää ja ytimekästä, ja kommentin alussa tulee aina ilmaista kohde, johon muutos on tehty sekä sen jälkeen toiminto, mitä on tehty (Kuva 27). Joskus voi tulla tilanne, että muutosta ei haluta viedä versionhallintaan, vaan Check-out-prosessi joudutaan perumaan. Peruminen onnistuu komennolla *Undo Check Out*, jolloin muutokset eivät välity versionhallintaan, eivätkä siten myöskään tallennu sinne.



Kuva 27. Kommentointi Check-Out-prosessissa

Kun projektissa työskentelee useampi henkilö, on ennen koodin muokkaamista syytä ottaa versionhallinnasta viimeisin versio käyttöön. Tämä toiminto vähentää konfliktin mahdollisuutta, eikä henkilö muokkaa omaa kopiotansa koodista, joka saattaa olla jo vanhentunut (Out of date). Viimeisin versio haetaan komennolla *Tools –> Visual SourceSafe –> Get Latest Version…*. Sama toiminto löytyy myös painettaessa hiiren oikealla painikkeella objekti-ikkunan päällä.

5 Yhteenveto

Tämän opinnäytetyön tarkoituksena oli saada luotua yritykseen dokumentoitu ja yhteisesti sovittu tapa hallita ohjelmistojen eri versioita ja siten myös parantaa niiden laatua. Koska kysymyksessä on pieni yritys eikä siinä ole erillistä versionhallintatiimiä, tavoitteena oli, että tämän työn avulla jokainen käyttäjä ymmärtäisi, mistä versionhallinnasta on kysymys, miksi sitä tulisi käyttää ja ottaisi myös vastuun sen toteuttamisesta.

Työ toteutettiin siten, että yrityksen palvelimelle asennettiin Microsoft Visual SourceSafe 2005 -versionhallintaohjelmisto, jota käytettiin yhdessä Microsoft Access 2002 -version kanssa. Ohjelmointityökalu ja siihen sisältyvä lähdekoodi onnistuttiin integroimaan työssä käytetyn versionhallintaohjelmiston kanssa.

Tärkein vaihe koko toteutuksessa oli huolellinen suunnittelu, jonka avulla saatiin luotua versiopuuhun selkeä ja looginen rakenne sekä kuvaava nimeämiskäytäntö. Kun versionhallintaa oli käytetty jonkin aikaa, oli selkeästi havaittavissa, kuinka oleellisesta asiasta siinä on kyse. Kun ohjelmistoprojekti ja siinä olevien objektien määrä kasvaa ajan kuluessa, paljastuu silloin myös, miten versionhallinta on suunniteltu. Jos sitä ei tehdä heti alussa kunnolla, on suuri mahdollisuus päätyä kaaokseen, ja hyödyt versionhallinnasta jäävät tällöin saavuttamatta. Toteutuksen pohjaksi jokaista versionhallinnassa olevaa ohjelmistoa varten luotiin standardin mukainen suunnitelma. Suunnitelmaa ei ollut kuitenkaan mahdollista liittää mukaan tähän työhön, koska se sisältää yksityiskohtaista tietoa ohjelmistosta.

Kommunikaation merkitystä yrityksessä ei voida koskaan korostaa liikaa. Koska versionhallintaohjelmistot on suunniteltu useamman kuin yhden henkilön käyttöön, on kommunikaatio kehittäjien kesken onnistumisen kannalta välttämätöntä. Kehitystyössä muokataan usein samoja lähdekooditiedostoja samaan aikaan, jolloin syntyy konflikteja siitä, kenen tekemä muutos on viimeisin ja oikea. Konflikti voidaan selvittää vain kommunikoimalla muutoksen tehneen henkilön tai henkilöiden kanssa. Tiedon panttaaminen ja itsekkyys lisäävät riskiä epäonnistua versionhallinnassa, ja sitä kautta myös koko ohjelmiston kehitysprojekti vaarantuu.

Haarautumisen käyttämistä pelätään usein yrityksissä, koska siihen liittyy huonoja kokemuksia. Usein nämä huonot kokemukset johtuvat pääsääntöisesti versionhallinnan puutteellisesta suunnittelusta tai haarautumien harkitsemattomasta käyttämisestä. Tässä työssä pyrittiin osoittamaan, miten haarautumia käytetään järkevästi, havaitsemaan niiden hyödyt ja tuomaan esiin, mitä niillä voidaan saavuttaa. Käy-

tännön toteutuksessa todettiin, että haarautumat mahdollistavat uusien ominaisuuksien kehittämisen turvallisesti erillään pääkehityslinjasta vaarantamatta sitä missään vaiheessa.

Pienissä yrityksissä on suuri vaara, että koko kehitystyö kulminoituu vain yhteen tai korkeintaan muutamaan avainhenkilöön, jolloin tietovuodon riski on todennäköinen. Toiminta on siis tällöin henkilöstä riippuvaista. Tällainen tapa toimia voi merkitä yritykselle jopa koko toiminnan päättymisen riskiä tai ainakin merkittävää taantumista, ellei tieto ole oikealla tavalla tallennettuna. Käytettäessä versionhallintaa taataan se, että tiedot on dokumentoituna turvallisesti ja oikein tietovarastossa, jolloin esimerkiksi uusi työntekijä pystyy jatkamaan kehitystä, mikäli työntekijä tai työntekijöitä poistuu yrityksestä.

Versionhallinnan, kuten monen muunkin ohjelmiston käyttö vaatii aluksi paljon harjoittelua. Käyttöönotossa ei ole suotavaa ensimmäisenä toimenpiteenä siirtää nykyistä ohjelmistoprojektia kokonaisuudessaan suoraan versionhallintaan, vaan kannattaa ensin tutustua ohjelmistoon ja harjoitella sen käyttöä. Niiden henkilöiden kannalta, joille versionhallinta on entuudestaan tuntematonta, on parempi, että harjoittelemiseen käytetään riittävästi aikaa. Tällöin on suurempi mahdollisuus päästä sellaiseen tulokseen, että henkilöt sitoutuvat versionhallinnan prosessiin osana ohjelmistokehitystä ja huomaavat siitä saavutettavat hyödyt, eivätkä esimerkiksi liity sen vastustajiin alussa saadun huonon tai sekavan käsityksen perusteella. Versionhallinta on varsin helppoa ja yksinkertaista, kun asiat tehdään alusta lähtien oikein.

Tämän opinnäytetyön aikana yrityksen palkanlaskentaohjelmiston koko kehitysmateriaali koodeineen ja dokumentteineen siirrettiin versionhallintaohjelmistoon. Ohjelmistosta on tehty jo muutama julkaisuversio toimitettavaksi asiakkaille haarauttamalla ne omiksi julkaisuhaarautumikseen tässä työssä edellä kerrotulla tavalla. Opinnäytetyön aikana yrityksessä suoritettiin myös suuri operaatio, jossa huoltokirja- ja kulutusseurantaohjelmisto yhdistettiin yhdeksi ohjelmaksi. Yhdistämisoperaation jälkeen yhdistetty ohjelmisto siirrettiin versionhallintaan, jonka prosessien kautta sen kehitystä jatketaan turvallisesti.

Yrityksen työntekijöille ei ole vielä ehtinyt muodostua mielipiteitä versionhallinnasta, koska se on ollut käytössä vasta varsin vähän aikaa ja siitä saatavat kokemukset saattavat kertyä tilanteesta riippuen hitaastikin. Lisäksi versionhallinnasta saatavat hyödyt voivat näkyä vasta pitkän ajan päästä. Yrityksessä tullaan järjestämään koulutusta, jotta versionhallinnan käyttöönotto tulisi helpommaksi ja henkilöstö saataisiin koulutettua tekemään asiat oikealla tavalla, jolloin versionhallinnasta tulisi yrityksessä pysyvä, turvallinen ja toimiva käytäntö.

Lähteet

Bays, Michael E. 1999. Software Release Methodology. New Jersey: Prentice Hall PTR

Bellagio, David E. & Milligan Tom J. Software Configuration Management Strategies and IBM® Rational® ClearCase® Second Edition A Practical Introduction. Boca Raton London New York Washington, D.C: IBM Press

Ben Collins-Sussman, Brian W. Fitzpatrick & C. Michael Pilato 2004. Version Control with Subversion. O'Reilly Media

Berczuk, Steve 2002. Software Configuration Management Patterns: Effective Teamwork, Practical Integration. Boston: Addison-Wesley

Crnkovic, Ivica & Asklund, Ulf & Dahlqvist, Annita Persson 2003. Implementing and Integraing Product Data Management and Software Configuration Management. Boston: Artech House

IEEE Std 1042-1987, IEEE Guide to Software Configuration Management (ANSI), www.ieee.org.

IEEE Std 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology, www.ieee.org.

IEEE Std 828-1990, IEEE Standard for Software Configuration Management Plans, www.ieee.org.

IEEE Std 828-1998, IEEE Standard for Software Configuration Management Plans, www.ieee.org.

Jonanssen Hass, Anne Mette 2002. Configuration Management Principles and Practice. Boston: Addison Wesley

Keyes, Jessica 2004. Software Configuration Management. Boca Raton London New York Washington, D.C: Auerbach Publications

Leon, Alexis 2000. Guide to Software Configuration Management. Boston: Artech House Publisher

Mason Mike 2006. Pragmatic Version Control Using Subversion 2nd Edition. North Carolina, Dallas: The Pragmatic Bookshelf

McConnell, Steve 1998. Software project Survival Guide. Washington: Microsoft Press

Moreira, Mario E. 2004 Software Configuration Implementation Roadmap. West Sussex: John Wiley & Sons Ltd

Liitteet

Liite 1: Yleisimmät versionhallintaohjelmistot

Ohjelmisto	Kaupallisuus	Valmistaja	Osoite
Subversion	Ilmainen	Tigris.org	www.subversion.tigris.org
RCS	Ilmainen	Gnu.org	www.gnu.org/
CVS	Ilmainen	Nongnu.org	www.nongnu.org/cvs/
Perforce	Maksullinen	Perforce Software Inc.	www.perforce.com
ClearCase	Maksullinen	IBM	www.ibm.com
Visual SourceSafe	Maksullinen	Microsoft	www.microsoft.com
BitKeeper	Maksullinen	BitKeeper	www.bitkeeper.com

Liite 2: Versionhallinnan sanastoa

Branch	haara versiosta, jota voidaan kehittää muista riippumatta
Tag	merkkaus, josta käytetään myös termiä Label
Commit	muutoksen hyväksyntä
ChangeLog	muutosloki, johon kirjataan tehdyt muutokset
Check-out	työkopio haetaan versionhallinnasta
Check-in	muokattu työkopio viedään versionhallintaan
Conflict	ristiriita, joka voi syntyä, kun kaksi tai useampi henkilö muokkaa samaa tiedostoa
Database	tietovarasto, josta voidaan käyttää myös nimitystä repository
Merge	yhdistetään haara(t) versioon
Release	julkaistava versio
Repository	tietovarasto, jossa säilötään versioiden muutokset
Revision	uusi versio, joka syrjäyttää vanhemman
Version	versio ohjelmasta tai tiedostosta
Working folder	hakemisto, johon tiedostot tuodaan check-out-komennolla