Jouni Niemelä

Ohjelmistojen konfiguraationhallinta

Tietotekniikan
Luentomoniste
12. joulukuuta 2006

Jyväskylän yliopisto

Tietotekniikan laitos

Jyväskylä

Sisältö

1 Ohjelmistojen konfiguraationhallinta, mitä se on?

> *"...Tietty määrä sekaannusta on väistämätön osa jokaista ohjelmistoprojektia. Päämääräksi tuleekin asettaa sekaannuksen minimointi, jotta itse työskentelyyn voitaisiin keskittyä täysipainoisesti. Ohjelmistokehityksessä tietyntyyppistä sekaannusta vähentävää taitoa kutsutaan ohjelmistojen konfiguraationhallinnaksi. Konfiguraationhallinta on taito identifioida, organisoida ja kontrolloida rakennettavaan ohjelmistoon tulevia muutoksia. Sen päämääränä on maksimoida tuottavuutta minimoimalla virheiden määrää."*
>
> — *Wayne A. Babich [2]*

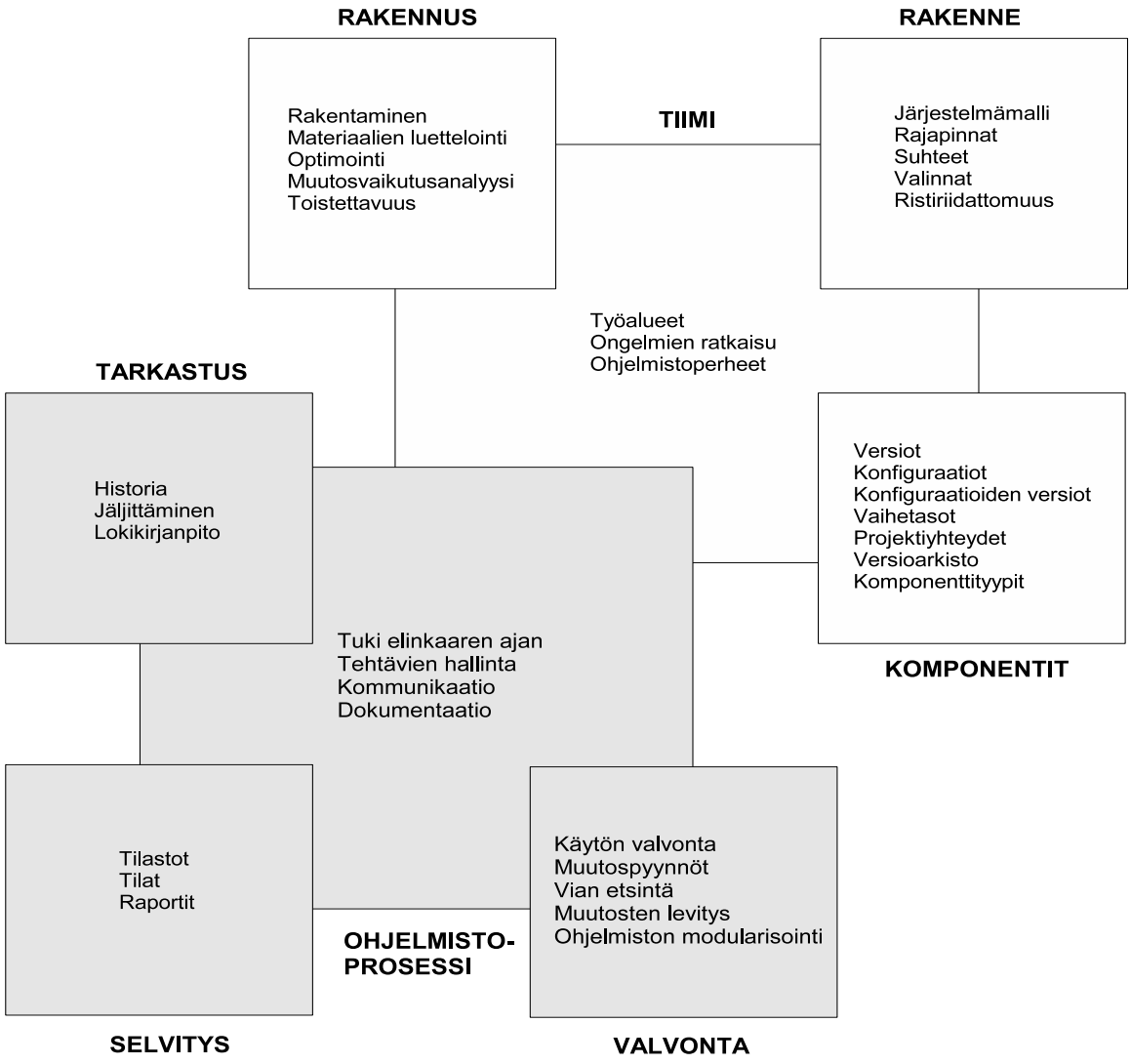
Yleisesti ajateltuna ohjelmistojen konfiguraationhallinta on pohjimmiltaan monimutkaisten ohjelmistojärjestelmien kehityksen ja elinkaaren hallintaa. Hieman käytännönläheisemmin ajateltuna kyseessä on kurinalainen käytäntö, jota systemaattisesti toteuttamalla voidaan kontrolloida muutoksia [2].

Ohjelmistojen konfiguraationhallinta sulautuu jokaiseen ohjelmistokehityksen vaiheeseen. Näiden vaiheiden hallintaan on ohjelmistojen konfiguraationhallinnassa olemassa konsepti nimeltään *vaihetaso* (engl. baseline). Ohjelmistokehityksen aikana hallinta-alkion viimeisintä versiota kutsutaan vaihetasoksi [13]. Käytännössä vaihetasoja syntyy jokaisesta ohjelmistoon kuuluvasta komponentista. Vaihetasoja määritellään ennenkaikkea myös konfiguraatioille, eli toisiinsa yleensä suhteessa oleville komponenteille.

Ohjelmistotekniikassa vaihetaso on eräänlainen virstanpylväs ohjelmiston kehityksessä. Ennen vaihetason saavuttamista hallinta-alkion täytyy yleensä läpikäydä prosessikohtainen formaali katselmointi. Tämän katselmoinnin jälkeen päätetään, hyväksytäänkö vai hylätäänkö katselmoinnin kohteena ollut hallinta-alkio.

Ohjelmistotuotannon eri vastuualueiden henkilöillä voi olla erilainen näkemys konfiguraationhallinnan soveltamiselle. Sen vuoksi myös vaatimukset konfiguraationhallinnan toteutukselle vaihtelevat tarpeiden mukaan. Konfiguraationhallinnan *toiminnalliset alueet* (engl. SCM functionality area) ovat erilaisia näkökulmia ohjelmistojen konfiguraationhallintaan näiden tarpeiden täyttämiseksi. Erilaiset näkökulmat täydentävät toinen toisiaan, ja niistä voidaan muodostaa Susan Dartin kuvassa 1.1 havainnollistama toiminnallisten alueiden verkko [8].

Jokainen laatikko kuvassa 1.1 edustaa tärkeää toiminnallista aluetta. Nämä toiminnalliset alueet voisivat sellaisenaan sisältyä konfiguraationhallintajärjestelmään. Tiimikeskeiset toiminnalliset alueet käsittelevät teknistä lähestymistapaa ohjelmistojen konfiguraationhallintaan, kun taas prosessikeskeiset toiminnalliset alueet käsittelevat lähinnä hallinnollista näkökantaa.



Kuva 1.1: Dartin näkemys toiminnallisten alueiden suhteista [8].

2 Komponenttien toiminnallinen alue

Kehitettävät ohjelmistot sisältävät tyypillisesti useita komponentteja. Ohjelmistojen konfiguraationhallinnan alaisina näitä komponentteja voidaan kutsua hallinta-alkioiksi (engl. configuration item, CI) [14]. Jokainen uusi ohjelmiston rakentamiseen tarvittava komponentti tulee sijoittaa versioarkistoon (engl. repository). Jos komponentin revisio havaitaan riittävän yleiskäyttöiseksi, se voidaan siirtää myös erilliseen komponenttikirjastoon. Sieltä se voidaan myöhemmin noutaa muidenkin ohjelmistoprojektien käyttöön tai jatkokehitettäväksi.

2.1 Komponenttien versiointi

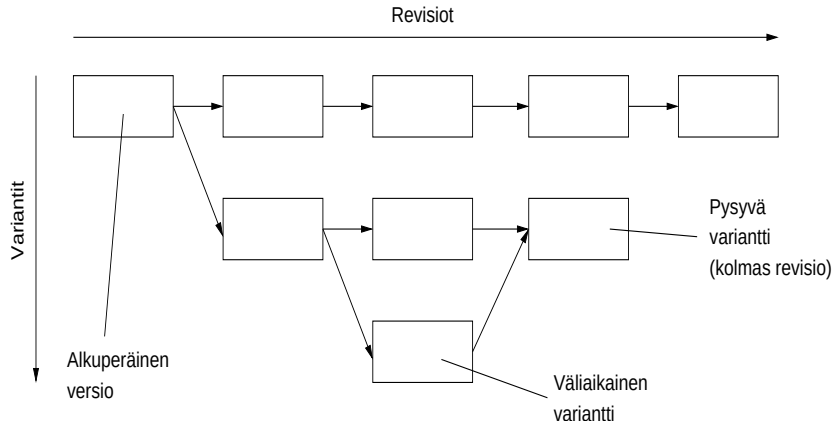
Aina kun komponentin versio syrjäytetään uusilla versioilla, niin näitä uusia versioita kutsutaan revisioiksi (engl. revision). Luotaessa komponentille vaihtoehtoinen versio ei uutta versiota enää kutsutakaan revisioksi vaan variantiksi (engl. variant). Variantin luonti muodostaa versiohistoriaan haaran (engl. branch), joka mahdollistaa komponentin rinnakkaiskehityksen. Pysyviä variantteja (engl. permanent variant) luodaan esimerkiksi silloin, kun ohjelmistoa sovitetaan uuteen kieli- tai laiteympäristöön. Variantteja saatetaan käyttää myös ohjelmiston testaus- ja debuggaustarkoituksiin. Variantti saatetaan ohjelmiston kehityksen edetessä joutua yhdistämään toiseen varianttiin. Näin muodostuvaa uutta revisiota kutsutaan väliaikaiseksi variantiksi (engl. temporary variant).

Esimerkinomaisen komponentin kehityshistoriaa havainnollistetaan kuvassa 2.1 versiograafin avulla. Versiograafista voidaan päätellä version tyypin olevan revisio, väliaikainen variantti tai (pysyvä) variantti vasta sitten, kun komponentin kehitys on edistynyt pidemmälle tai lopetettu kokonaan.

2.2 Komponenttien tallennus

Komponentit tallennetaan versiotietoineen versionhallintatyökalulla hallinnoitavaan versioarkistoon. Komponenttien revisioita ei tallenneta sellaisenaan, sillä se kuluttaisi aivan liikaa tallennustilaa. Kustakin komponentin revisiosta tallennetaan ainoastaan muutostieto eli *delta* johonkin toiseen revisioon. Komponenttien revisioiden välisten deltojen hallinnointiin on kehitetty useita erilaisia menetelmiä.

Komponenteista tallennetaan versioarkistoon deltan sisällön lisäksi paljon erilaisia tunnisteita eli attribuutteja. Jokainen komponentin revisio sisältää yksikäsitteisen tun-



Kuva 2.1: Eri versiotyypit versiograafissa [20].

nisteen, jonka avulla komponentti voidaan identifioida. Lisäksi revisioilla voi olla attribuutteina muun muassa tallentajan nimi, tallennusajankohta, lukitusstatus, leima / vaihetasotieto ja kuvaus tehdystä muutoksesta. Käytettävien attribuuttien lukumäärä ja nimeäminen on versionhallintatyökalukohtaista.

2.3 Varianttien hallinta ohjelmallisesti

Versioarkistossa komponenttien variantteja hallitaan tallentamalla ne versiograafissa eri haaroihin. Ohjelmointikielet tarjoavat vaihtoehtoisen menetelmän hallita yksittäisen komponentin variantteja. Esimerkiksi C-kieli tarjoaa mahdollisuuden tuottaa variantteja esikääntäjän (CPP) ohjauskomentojen avulla (esim. `#if ... #endif ... #end`) [18].

3 Rakenteen toiminnallinen alue

Rakenteessa esitetään kaikki ohjelmiston tarvitsemat komponentit ja niiden väliset rajapinnat (engl. interface), versiot ja konfiguraatiot. Komponenteista valitaan rakenteeseen mukaan vain ne, jotka ovat toistensa kanssa yhteensopivia rakennettavassa ohjelmistotuotteessa [7, 8].

3.1 Konfiguraatiotyypit

Järjestelmämallia tarkastelemalla voidaan päätellä, mitkä komponenteista kuuluvat järjestelmään. Vastaavasti kun tarkastellaan tiettyyn käyttötarkoitukseen räätälöityjen komponenttien joukkoa, puhutaan konfiguraatiosta [20]. Konfiguraation sisältöä voidaan kuvailla valintasääntöjen avulla. Sääntöjen avulla valitut konfiguraatiot voidaan ryhmitellä kolmeen konfiguraatiotyyppiin:

- **Sidottu konfiguraatio** (engl. bound configuration): Sidottu konfiguraatio on täysin identtinen jokaisella valintakerralla käytettäessä samaa konfiguraation muodostusmenetelmää [9]. Sidottua konfiguraatiota käytetään tyypillisesti identifioimaan asiakkaalle toimitettava tuotekokonaisuus [20].
- **Yleiskonfiguraatio** (engl. generic configuration, partially bound): Yleiskonfiguraatiota käytetään tyypillisesti ohjelmiston kehitysvaiheessa esim. valittaessa komponentit uusimman revision säännöllä.
- **Abstrakti konfiguraatio** (engl. abstract configuration): Abstrakti konfiguraatio muodostuu, kun konfiguraation muodostuksessa käytettävät säännöt ovat moniselitteiset. Näin ollen itse konfiguraation määritelmä on epätäydellinen. Abstraktilla konfiguraatiolla on siten havaittavissa samankaltaisuutta olio-ohjelmoinnista tuttuun abstraktiin yliluokkaan (engl. abstract super class) [20].

3.2 Konfiguraation valinta

Konfiguraation valinta voidaan toteuttaa kahdella eri tavalla. Ensimmäinen ja helpompi tapa toteuttaa konfiguraation valinta on asettaa konfiguraatiota osoittava leima (engl. label, tag) kaikkiin konfiguraatioon kuuluviin komponenttiversioihin ja noutaa tätä leimaa vastaavat komponenttien versiot.

Konfiguraation valinta voidaan toteuttaa myös boolean-tyyppisten tunnistekyseilyiden avulla (engl. boolean attribute queries) [9]. Nämä tunnisteet voivat olla esimer-

kiksi version tunniste, muutospäivämäärä, komponentin kehitystila (status), vaihetaso tai variantin tunniste jne. Linux-konfiguraation tuottava tunnistekysely voisi olla esimerkiksi tämän kaltainen:

```
OS = linux AND (status = testattu OR lukittu = Jussi)
```

4 Rakentamisen toiminnallinen alue

Ohjelmiston rakentamisessa komponentit käännetään lähdekoodista ohjelman suoritusympäristön ymmärtämään muotoon. Esimerkiksi C-kielen .c-tiedostot käännetään .obj-tiedostoiksi ja Java-kielen .java-tiedostot käännetään virtuaalikoneen ymmärtämiin .class-tiedostoihin. Järjestelmän rakentamiseen liittyvää informaatiota liitetään rakennustapahtumassa käytettävään skripti-tiedostoon [20]. Yksinkertaisimmillaan tämä skripti määrittää, kuinka konfiguraatioon kuuluvat komponentit käännetään. Nykyaikaiset ohjelmiston rakennusmenetelmät sisältävät lisäksi automatisoituja toimintoja, kuten konfiguraatioon kuuluvien komponenttien noutamisen versioarkistosta, optimoidun kääntämisen, ohjelmistotuotteen paketoinnin jne.

4.1 Lisäysrakentaminen

Lisäysrakentaminen käyttää ohjelmiston rakennuksen aikana hyödyksi riippuvuussuhteita lähdekoodisten ja käännettyjen komponenttien välillä [17]. Eräs varhaisimmista, mutta silti edelleen laajalti käytetty lisäysrakentamista hyödyntävä työkalu on Feldmanin vuonna 1979 Unix-ympäristöön kehittämä MAKE [11]. Siinä Makefile-makrotiedostossa kuvataan järjestelmän rakentamisessa tarvittavat tehtävät, komponentit ja niiden väliset riippuvuudet.

Kehittyneemmät rakennusohjelmat, kuten ANT [1] sisältävät usein integraation versionhallintaohjelmistoihin. Ohjelmistoa rakennettaessa voidaan samalla komentotiedostolla (engl. script file) näppärästi noutaa konfiguraatioon kuuluvat tiedostot tyhjälle työalueelle ja suorittaa lähdekooditiedostojen käännös binäärimuotoon.

4.2 Versioidun ohjelmiston rakentaminen

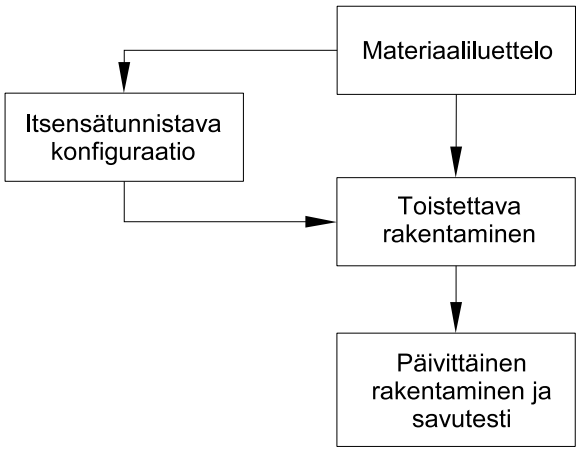
Ennemmin tai myöhemmin hyödylliseksi koettuun ohjelmistoon kohdistuu muutospaineita [20]. Esimerkiksi ohjelmiston vanhaan versioon tarvitaan uusia ominaisuuksia tai siitä löydettyjä virheitä halutaan korjata. Jotta ohjelmiston vanhaa versiota voidaan päivittää, niin vanhan version konfiguraatio on pystyttävä palauttamaan ja rakentamaan uudelleen täsmälleen samalla tavalla kuin aiemmin. Rakennetun ohjelmiston konfiguraatioon kuuluvien komponenttiversioiden leimaus mahdollistaa konfiguraation palautuksen versioarkistosta aina tarvittaessa. Se siis luo yhden edellytyksen täsmälleen samanlaisen ohjelmistoversion rakentamiselle. Lisäksi tarvitaan luettelo kaikista ohjelmiston rakentamisessa käytetyistä ohjelmista ja niiden versioista. Näiden tietojen pe-

rusteella voidaan muodostaa ohjelmiston konfiguraatiosta *materiaaliluettelo* (engl. Bill Of Materials, BOM). Täysi varmuus *materiaaliluettelon* oikeellisuudesta saadaan vasta, kun sen avulla rakennustapahtuma on toistettu onnistuneesti eli sovellettu *toistettavan rakentamisen kehystä* (engl. reproducible build) [3].

Ohjelmiston komponentit sisältävät versioarkistossa ollessaan versiotiedon. Versioarkiston ulkopuolella ne eivät kuitenkaan enää ole versionhallinnan piirissä eli sisällä versiotietoa. Komponenteilla tulisi kuitenkin olla tunniste, jonka avulla niiden versio voidaan tunnistaa myös versioarkiston ulkopuolella. *Itsensätunnistava konfiguraatio* (engl. self-identifying configuration) on kehys, joka pyrkii mahdollistamaan komponenttien tunnistuksen versioarkiston ulkopuolella.

Ohjelmiston *savutestaus* voidaan vielä liittää näiden ohjelmiston rakentamiseen liittyvien kehysten joukkoon *päivittäinen rakentaminen ja savutesti* -kehyksen (engl. dayly build and smoke test) muodossa. Siinä ohjelmisto rakennetaan toistuvasti päivittäin sekä sen perustoiminnot testataan pikaisesti. Näin ohjelmiston kehitystilasta on aina olemassa jonkinlainen käsitys. Päivittäisen rakentamisen hyöty on lisäksi se, että ohjelmiston kehittäjien ei tarvitse muistaa, milloin ohjelmisto rakennetaan seuraavan kerran. Lisäksi päivittäisen rakentamisen ja testauksen rytmi nostaa kehittäjien moraalia, sillä ohjelmisto periaatteessa toimii päivä päivältä paremmin [3].

Kuvassa 4.1 havainnollistetaan näiden neljän ohjelmiston rakentamiseen liittyvän kehyksen välisiä suhteita.



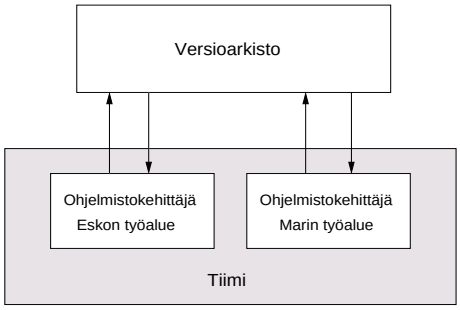
Kuva 4.1: Ohjelmiston rakentamiseen liittyvien kehysten suhteet [3].

5 Tiimin toiminnallinen alue

Ohjelmistokehitys etenee yleensä ohjelmistoa tuottavan tiimin sisällä rinnakkaisena työskentelynä. Konfiguraationhallinnassa jokaiselle kehittäjälle tarjotaan tähän tarkoitukseen oma työalue (engl. workspace), joka eristää työskentelyn muiden työskentelystä [7]. Aina aika ajoin ohjelmistokehittäjät jakavat työnsä tulokset muiden tiimin jäsenten käytettäväksi. Tämä tiedostojen vaihto tulisi suorittaa hallitusti työalueiden välillä käyttämällä projektin versioarkistoa [19].

Kontrolloimaton muutosten siirto työalueiden välillä aiheuttaa häiriöitä tiimityöskentelyyn ja johtaa helposti ristiriitatilanteisiin, jotka täytyy selvittää työtiedostojen liitostyöllä (engl. merge) [12]. Rinnakkaisessa ohjelmistokehityksessä kehittäjien välistä yhteistyötä voidaan koordinoida yhteistyöstrategioiden avulla [20]. Näillä pyritään linjaamaan toimintamalli, jolla hallitaan komponenttien samanaikaista kehitystä.

Kuvassa 5.1 havainnollistetaan, kuinka ohjelmistokehitystiimissä työskentelevien Eskon ja Marin työalueet on eristetty toisistaan ja työalueiden välinen tiedostojen vaihto tulisi tapahtua ainoastaan projektin versioarkiston välityksellä.



Kuva 5.1: Esimerkki, kuinka tiimin jäsenten työalueet on eristetty toisistaan.

5.1 Yhteistyöstrategiat

Komponentin rinnakkaiskehitysvaiheisiin sovelletaan konfiguraationhallinnassa yhteistyöstrategioita. Näiden strategioiden käytöllä koordinoidaan komponentteihin kohdistuvia muutoksia.

Konservatiiviset strategiat pyrkivät Zellerin mukaan välttämään ristiriitatilanteita yksinkertaisilla lukitussäännöillä. Komponentin kehittäjä lukitsee komponentin tietyn revision itselleen muokattavaksi. Komponentin ollessa lukittuna muut kehittäjät eivät voi luoda tästä komponentista uusia revisioita. He voivat kuitenkin noutaa komponen-

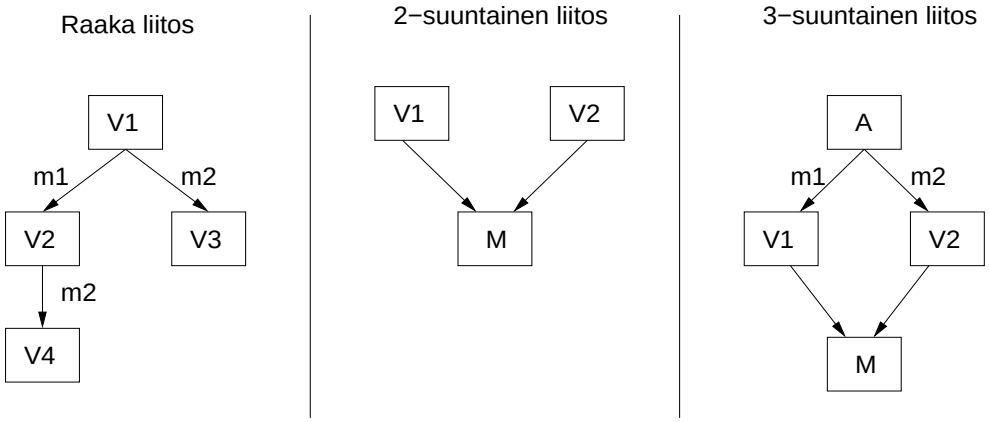
tin lukitun revision kehitettäväksi ja luoda siitä väliaikaisen variantin.

Optimistiset strategiat sallivat konservatiivisista strategioista poiketen oletusarvoisesti rinnakkaisen komponentin kehityksen. Jokaiselle kehittäjälle luodaan optimistista strategiaa käytettäessä oma väliaikainen variantti, jota hän voi työstää omalla työalueellaan.

5.2 Liitosongelmat

Riippumatta siitä, onko käytössä konservatiivinen vai optimistinen strategia, niin rinnakkaisen ohjelmistokehityksen tuloksena tulevat muutokset joudutaan yleensä ennemmin tai myöhemmin yhdistämään pääkehityshaaraan. Tehtävät liitokset voidaan jakaa kuvan 5.2 mukaisesti kolmeen ryhmään [5]:

- **Raaka liitos** (engl. raw merging) kohdistuu johonkin tiettyyn asiayhteyteen. Kuvan 5.2 muutokset $m1$ ja $m2$ toteutettiin toisistaan riippumatta. Myöhemmin muutos $m2$ yhdistettiin muutokseen $m1$. Tuloksena saatiin uusi revisio $V4$.
- **Kaksisuuntainen liitos** (engl. two-way merge) yhdistää kaksi versiota yhdeksi. Kuvan 5.2 esimerkin revisiot $V1$ ja $V2$ yhdistetään uudeksi revisioksi M.
- **Kolmisuuntainen liitos** (engl. three-way merge) hyödyntää alkuperäistä (usein vaihetasoon liittyvää) revisiota liitoksen teossa. Jos muutos koskee ainoastaan toista liitettävänä olevaa revisiota, niin se sisällytetään uuteen liitosversioon automaattisesti. Muussa tapauksessa (esim. päällekkäinen muutos molemmissa revisioissa $V1$ ja $V2$) ristiriitatilanne joudutaan yleensä ratkomaan manuaalisesti.



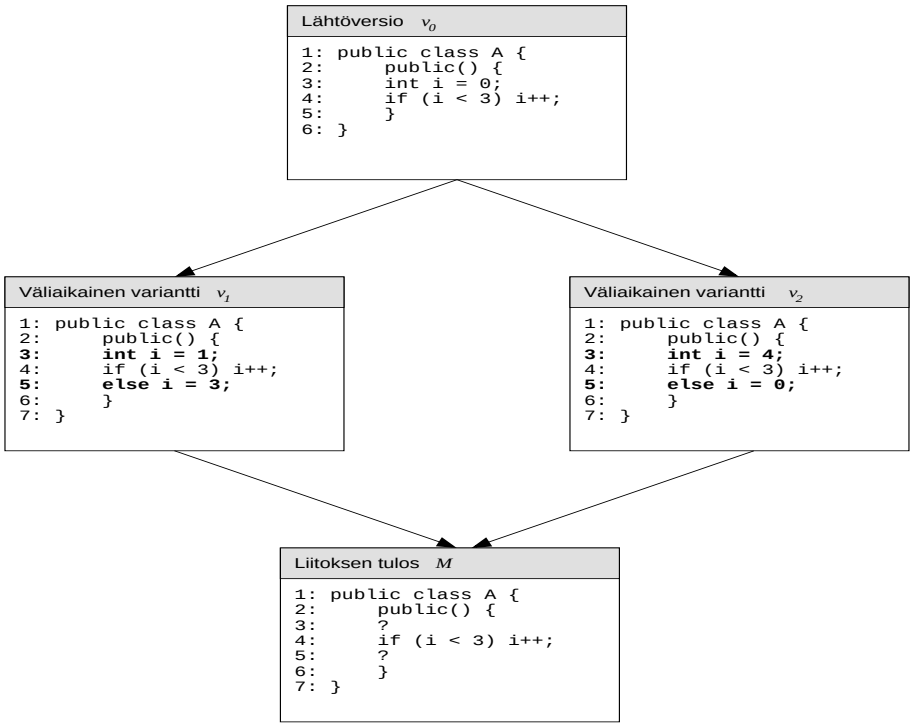
Kuva 5.2: Erilaiset liitostyypit [5].

Liitostyökalut voidaan jaotella liitosoperaatiossa käytettävän semanttisen tason mukaan kolmeen ryhmään:

- **Tekstipohjainen liitos** (engl. texual merging) on yleisimmin käytetty liitosmenetelmä. Tekstipohjaisen liitoksen tuloksena tuleva uusi versio ei ole aina syntaktisesti oikea. Esimerkiksi yhdistettäessä kaksi toimivaa C-kielistä tiedostoa tulos saattaa olla C-kielen kieliopin vastainen. Kuvassa 5.3 havainnollistetaan kolmisuuntaista liitosta, jossa on selvä ristiriitatilanne väliaikaisten varianttien välillä.

- **Syntaksiin perustuvat liitosmenetelmät** (engl. syntactic merging) tuottavat varmasti kieliopillisesti oikeanlaisia liitosten tuloksia [5]. Visual Comparer -liitostyökalu[1] on eräs harvoista syntaksin tarkastuksen sisältävistä liitostyökaluista.

- **Semantiikkaan perustuvat liitosmenetelmät** (engl. sematic merging) huomioivat liitoksen merkityksen koko rakennettavassa ohjelmistossa. Semanttisia liitoksia tukevan työkalun tulisi suorittaa analyysejä liitoksen ohjelmistoon mahdollisesti aiheuttamista ristiriidoista [5]. Semanttisten integraatioiden suorittaminen on vaativa ongelma. Käytännössä semantiikkaan perustuvia liitosohjelmiakaan ei juuri ole saatavilla.



Kuva 5.3: Väliaikaiset variantit ja niiden liittäminen uudeksi versioksi.

[1]URL: `http://www.visual-comparer.com/`, tark. 25.6.2006.

6 Ohjelmistoprosessikeskeiset toiminnalliset alueet

Prosessikeskeiset toiminnallisuusalueet käsittelevät lähinnä ohjelmistojen konfiguraationhallinnan hallinnollista puolta.

6.1 Tarkastuksen toiminnallinen alue

Konfiguraation tarkastuksella varmistetaan ohjelmistotuotteen sisältävän kaikki tarpeelliset komponentit, sekä tuotteen luvatunlainen toiminta. Tarkastusprosessissa katselmoidaan, että koko ohjelmistojärjestelmä vastaa sille asetettuja vaatimuksia [15].

Konfiguraationhallinnan tarkastustustoimenpiteet ja testausprosessi ovat pitkälti kytköksissä toisiinsa, sillä molemmat pyrkivät samaan päämäärään: ohjelmiston hyvän laadun *tarkastukseen* (engl. verification) ja *varmistukseen* (engl. validation) [16]. Taulukossa 6.1 on mm. Pressmanin kirjasta löytyvä Boehmin näkemys tarkastuksesta ja varmistuksesta.

Tarkastus :	Rakennammeko tuotetta oikein?
Varmistus :	Rakennammeko oikeaa tuotetta?

Taulukko 6.1: Määritelmä tarkastukselle ja varmistukselle (engl. V & V) [16].

6.2 Selvityksen toiminnallinen alue

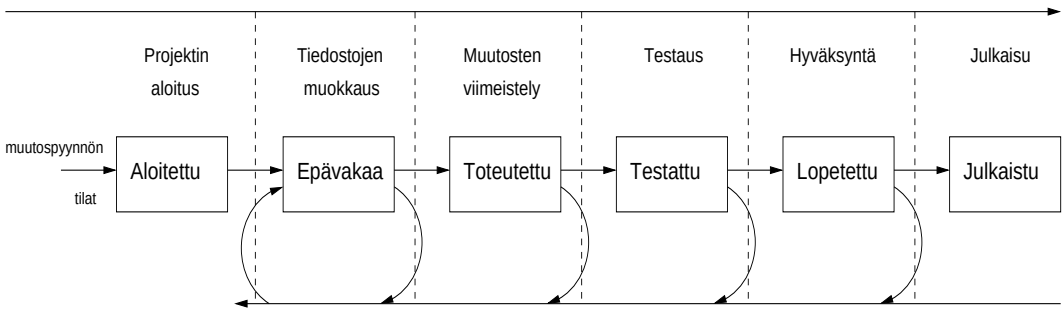
Konfiguraation tilan selvitys (engl. configuration status accounting) on ohjelmistossa tapahtuvien muutosten jäljitykseen käytettävä prosessi. Siinä huolehditaan vaihetasoon liittyvien tekemättömien ja tehtyjen tehtävien tilakirjanpidosta, valvonnasta ja raportoinnista [15].

Versionhallintatyökalut mahdollistavat tilan selvitykseen tarvittavan tiedon keräyksen ja raportoinnin. Tämä muutosprosessi-informaatio pitää sisällään muun muassa tiedon muutoksen ajankohdasta, tekijästä, syystä ja muutoskuvauksen [15]. Ohjelmoijalle komponentin historiatieto on kullanarvoisen tärkeää ongelmatilanteissa. Usein ohjelmistoprojekteissa käy niin, että viime viikolla toiminut ohjelma ei enää toimikaan halutulla tavalla. Helpoin tapa etsiä syy tähän on tutkia, mitä muutoksia ohjelmistoon on tapahtunut sitten viime viikon.

6.3 Valvonnan toiminnallinen alue

Muutoksia kontrolloidaan konfiguraationhallinnassa muutospyynnöillä (engl. Change Request, CR). Nämä muutospyyntödokumentit sisältävät kuvailun muutostarpeesta: uuden toiminnallisuuden esittelyn, olemassa olevan toiminnon parannusehdotuksen, ilmoituksen selvästä virheestä, ohjelmistoprosessin parannusehdotuksen, jne. [6].

Aika ajoin muutoksenhallintalautakunta (engl. Change Control Board, CCB) käsittelee tehdyt muutosehdotukset ja tekee päätökset, mitkä niistä toteutetaan [15]. Muutospyyntö läpikäy kehitysprosessin eri tiloja ennen kuin sen realisoima muutos on mahdollisesti valmis julkaistavaksi eli käyttöönotettavaksi ohjelmistotuotteessa. Kuva 6.1 havainnollistaa näiden tilojen läpikäyntiä ohjelmiston kehityksessä.



Kuva 6.1: Muutospyynnön (CR) kehitysprosessi [6].

6.4 Ohjelmistoprosessin toiminnallinen alue

Ohjelmistojen konfiguraationhallinta on ohjelmistotuotteen valmistumista auttava tukitoimi tai tukiprosessi, joka sulautuu jokaiseen ohjelmistokehityksen vaiheeseen. Konfiguraationhallinta sopii hyvin tukitoimeksi esimerkiksi sovellettaessa vesiputous- tai inkrementaalisen kehityksen ohjelmistoprosessia [15]. Susan Dartin mukaan konfiguraationhallinnan menetelmät tukevat prosessien näkökannalta ohjelmistokehitystä seuraavasti [8]:

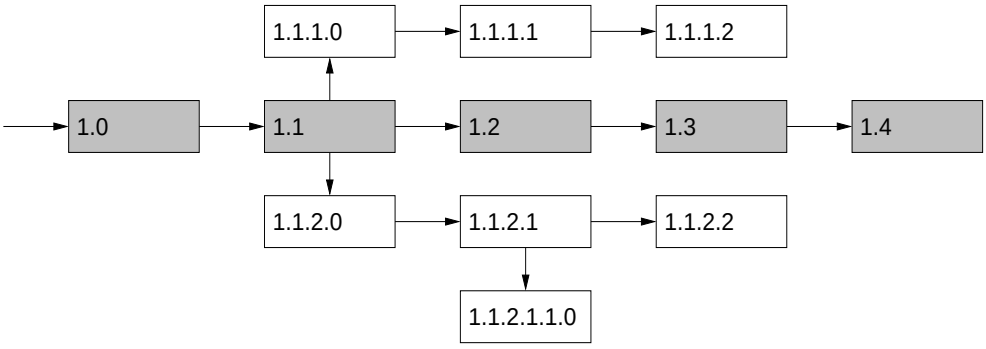
- Tukemalla elinkaarimallia ja organisaation käytäntöjä.
- Kyvyllä identifioida suoritettavat tehtävät, niiden suoritusajankohta ja valmistuminen.
- Helpottamalla tehtävän suorittamiseen liittyvän tiedon jakamista sitä tarvitseville henkilöille.
- Helpottamalla ohjelmistotuotteeseen liittyvän tietotaidon dokumentointia.

7 Versiomallit

Konfiguraationhallinnassa versiomalli (engl. version model) määrittelee, kuinka tieto (yleensä tiedostot tai niiden joukot) versioidaan, versiot identifioidaan ja organisoidaan, sekä operaatiot olemassa olevien versioiden noutamiselle ja uusien luonnille [4]. Versiomallit voidaan jakaa karkeasti kahteen mallisuuntaan: versio- ja muutospohjaisiin malleihin.

7.1 Nouda ja tallenna -malli

Nouda ja tallenna -malli perustuu yksittäisten komponenttien siirtoon työalueen ja versioarkiston välillä. Uusi komponentti luodaan aluksi työalueella tiedostoksi. Muokkausten jälkeen komponentti *tallennetaan* (engl. checkin) versioarkistoon, jolloin sinne muodostuu uusi versiograafi. Aluksi komponentin versiograafissa on ainoastaan yksi revisio. Versiograafi kuitenkin täydentyy uudella revisiolla aina jokaisen tallennusoperaation jälkeen. Komponentin versiograafista voidaan valita sopiva komponentin revisio ja *noutaa* (engl. checkout) se työalueelle [9]. Kuvassa 7.1 havainnollistetaan esimerkinomaisen komponentin versiograafia.

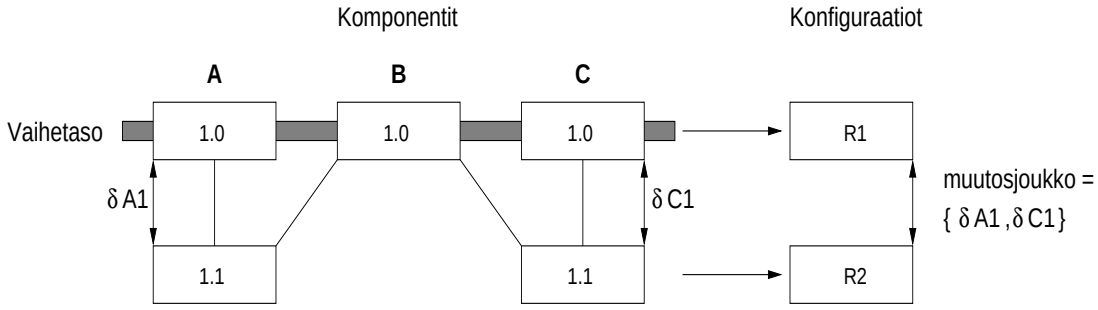


Kuva 7.1: Komponentin versionumerointi versiograafissa.

Versiopohjaisia malleja käyttävät versionhallintaohjelmistot tukevat yleensä *nouda ja tallenna* -mallin mukaisesti yksittäisten tiedostojen käsittelyä. Esimerkiksi CVS mahdollistaa *nouda ja tallenna* -mallin mukaisesti yksittäisten tiedostojen noudon ja tallennuksen.

7.2 Muutosjoukkomalli

Muutosjoukkomalli keskittyy tukemaan ohjelmiston konfiguraatioissa tapahtuvia loogisia muutoksia. Konfiguraatio muodostetaan *muutosjoukkomallissa* vaihetason komponenteista ja muutosjoukoista. Konfiguraatioiden muutosjoukko on kahden konfiguraatioversion välinen ero. Tämä ero saadaan kokoamalla konfiguraatioversioiden sisältämien komponenttien erot muutosjoukoksi. Kuvassa 7.2 havainnollistetaan konfiguraatioversioiden $R1$ ja $R2$ välisen muutosjoukon muodostumista [9]. Kuvassa konfiguraation $R1$ muutosjoukko sisältää komponenttien A ja C muutokset. Komponentin B muutosta ei tarvita, sillä se ei ole muuttunut.



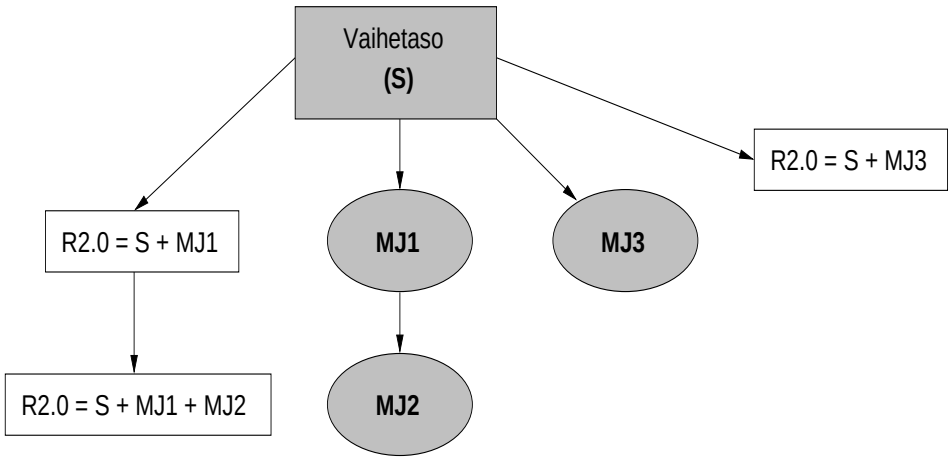
Kuva 7.2: Konfiguraatioiden välisen muutosjoukon muodostuminen [9].

Kuva 7.3 havainnollistaa versiograafimuotoon piirrettynä muutosjoukkomallisten konfiguraatioiden muodostusta. Samassa kuvassa on piirrettynä konfiguraatioiden versiograafi laatikoilla ja muutosjoukkojen vastaava graafi ellipseillä. Kuvassa konfiguraatio $R2.0$ on muodostettu kolmella eri tavalla vaihetasosta S. Ensimmäisessä vaihtoehdossa $R2.0$ kootaan vaihetason S ja muutosjoukon $MJ1$ yhdisteenä. Seuraavaksi $R2.0$ kootaan lisäämällä edelliseen vielä muutosjoukko $MJ2$. Kolmannessa vaihtoehdossa $R2.0$ kootaan lisäämällä vaihetasoon S muutosjoukko $MJ3$. Tässä esimerkissä muutosjoukko $MJ2$ on riippuvainen muutosjoukosta $MJ1$, joten esimerkiksi muutosjoukko $MJ3$ ei välttämättä toimisi ristiriidattomasti $MJ2$:n kanssa.

7.3 Kokoonpanomalli

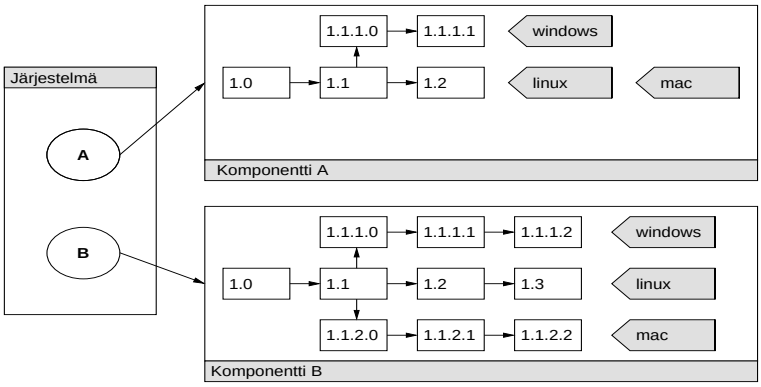
Kokoonpanomallissa käytetään versioarkistoa ja työaluetta sekä lukitusmekanismeja samanaikaisten muutosten hallintaan samoin kuin nouda ja tallenna -mallissakin. Tämä malli luottaa komponentin versiograafista löytyvään informaatioon oikean version valinnassa, joten kokoonpanomallia voidaan luonnehtia nouda ja tallenna -mallin sivutuotteeksi. *Kokoonpanomallissa* konfiguraatiot ovat itsenäisiä versionhallinnanalaisia kokonaisuuksia, jotka koostuvat järjestelmämallista ja version valintasäännöistä [9].

Kokoonpanomallissa järjestelmämalli kuvaa järjestelmän rakennetta ja luettelee kaikki järjestelmän rakentamiseen tarvittavat komponentit. Oikeat komponenttien ver-



Kuva 7.3: Konfiguraation versiograafi sekä sen sisältämät muutosjoukot [9].

siot valitaan säännöillä [9]. Järjestelmämalli ja valintasäännöt yhdessä toimivat siis konfiguraation sisällön määrittelevänä *materiaaliluettelona*. *Kokoonpanomallissa* määritellään ensin järjestelmään kuuluvat komponentit ja tämän jälkeen valitaan näiden revisiot. Kuvassa 7.4 havainnollistetaan järjestelmää, johon kuuluu kaksi komponenttia A ja B.



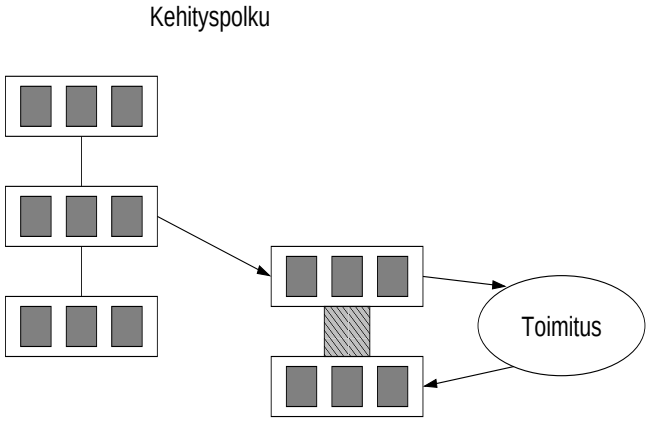
Kuva 7.4: Konfiguraatioon kuuluvan komponentin valinta.

7.4 Pitkä toimitus -malli

Pitkä toimitus -malli käsittelee järjestelmässä tapahtuvaa kehitystä yksittäisten muutosten sarjoina. Operointi tapahtuu pääasiassa versioiduilla konfiguraatioilla yksittäisten komponenttien sijaan. Tässä mallissa valitaan aina ensin konfiguraatio, johon muutoksia halutaan toteuttaa. Tehtävistä muutoksista muodostuu toimitustapahtuma (engl. transaction), jonka tulee kestää ohjelmistokehittäjän koko työalueella tapahtuvan työskentelyn ajan. Tästä mallin nimi juontaakin juurensa, sillä istunnon (toimi-

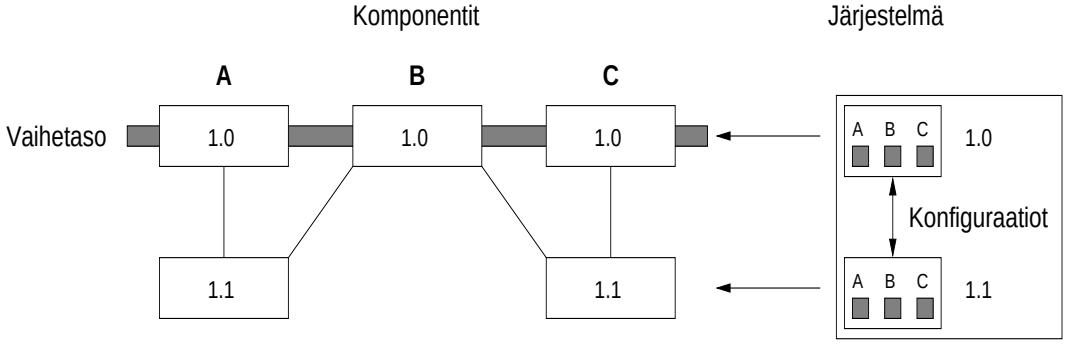
tuksen) pituus voi olla ajallisesti tunteja, päiviä tai jopa kuukausia. Muutokset eivät näy toimituksen ulkopuolelle ennen *suorita*-komennon (engl. commit) antamista.

Suorita-komennon antamisen jälkeen versioarkistoon muodostuu uusi konfiguraatio. Suoritetuista toimituksista muodostuu konfiguraatioiden kehityspolku, joka voi haarautua samalla tavalla kuin komponenttien versiograafitkin. Kuvassa 7.5 havainnollistetaan konfiguraation versiohistoriaa. Siinä pääkehityshaaran toisesta konfiguraatiosta on muodostettu uusi itsenäinen kehityspolku, jonka kehitys etenee päähaarasta riippumattomasti.



Kuva 7.5: Konfiguraation versiohistoria [9].

Tämä malli tukee luonnollisella tavalla uuden konfiguraation luontia vaihetasolla määritetystä konfiguraatiosta [10]. Kuvassa 7.6 havainnollistetaan, kuinka konfiguraation valinnan jälkeen komponenteista valitaan aina automaattisesti konfiguraatioon kuuluvat versiot.

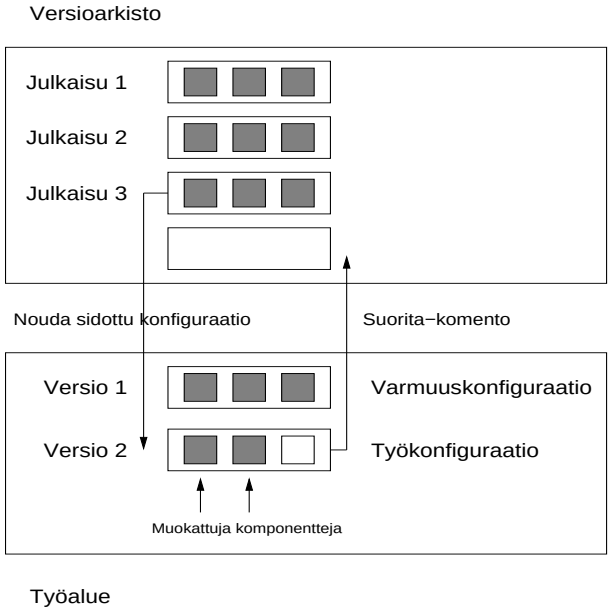


Kuva 7.6: Konfiguraation version valinta [9].

Työalue sisältää työstettävän työkonfiguraation ja joukon varalla olevia varmuuskonfiguraatioita. Työalue alustetaan noutamalla versioarkistosta sidottu konfiguraatio työkonfiguraatioksi tai kopioimalla jokin varmuuskonfiguraatio työstettäväksi konfiguraatioksi. Työkonfiguraatio on työalueen ainoa konfiguraatio, jonka sisältöä voidaan

muokata. Tehdyistä muokkauksista tallentuu historia työalueelle ketjuna jäädytettyjä varmuuskonfiguraatioita. Kuvassa 7.7 on esitetty edellä mainittu tapahtumasarja ja työalueen sisäinen versiohistoria. Nämä varmuuskopiot voivat olla käytännössä virtuaalisia kopioita, joista vain muutokset on tallennettu [9].

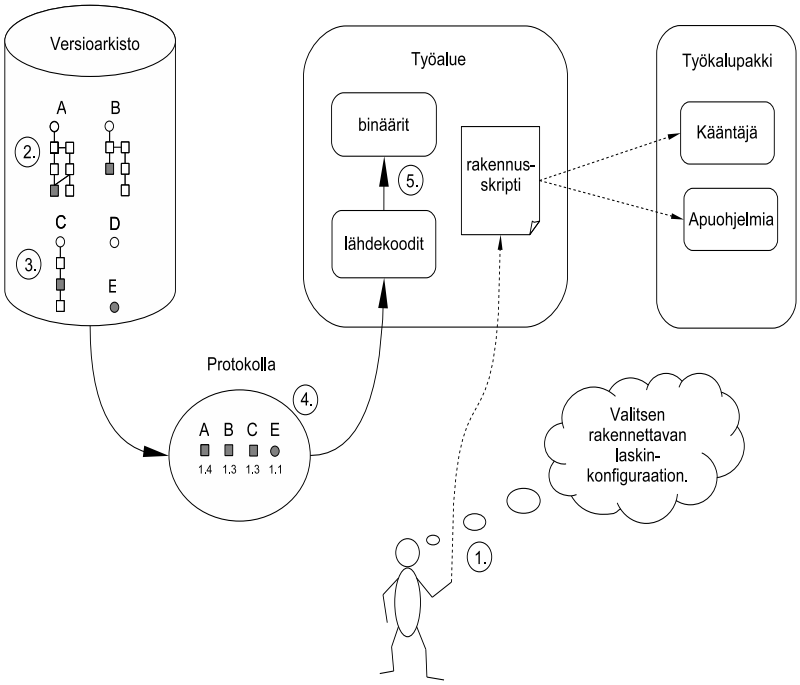
Kaikki tehdyt muutostyöt ovat täysin näkymättömiä työalueen ulkopuolelle ennen konfiguraation toimittamista versioarkistoon *suorita*-komennolla. Kun toimitus on suoritettu loppuun *suorita*-komennon antamisen jälkeen, myös toimitustapahtuma päättyy ja toimitetusta konfiguraatiosta muodostuu uusi revisio versiohistoriaan [9].



Kuva 7.7: Työalue ja historiatiedot [9].

8 Katkelma ohjelmiston rakennusjärjestelmän mallinnuksesta

Mallinnettava järjestelmä koostuu ohjelmiston rakentamiseen tarvittavista elementeistä. Järjestelmän rakennetta ja toimintaa havainnollistetaan kuvassa 8.1. Siinä esitellään järjestelmään kuuluva *versioarkisto*, *työalue* ja näiden väliseen tiedonsiirtoon käytettävä *protokolla*.



Kuva 8.1: Yleiskuva rakennustapahtumasta.

Esitellään aluksi järjestelmän käyttämät abstraktit perustyypit, jotka on tässä nimetty selvyyden vuoksi lisäämällä nimen loppuun ”Tyyppi” ja Z-mäisesti käyttämällä hakasulkuja nimen ympärillä:

1. **[LohkoTyyppi]** on abstrakti perustyyppi, josta *Revision* delta ja *Tiedosto* rakentuvat.

2. **[IDTyyppi]** on abstrakti perustyyppi, joka kuvaa *Revision* sisältämää yksikäsitteistä tunnistetta.

3. **[NimiTyyppi]** on abstrakti perustyyppi, joka kuvaa yksikäsitteisesti *Tiedoston* nimeä ja hakemistopolkua sekä *Komponentin* tunnistetta.

4. **[LeimaTyyppi]** on abstrakti perustyyppi, jonka avulla idenfioidaan konfiguraatioon kuuluvat *Revisiot*.

5. **[VirheTyyppi]** on abstrakti perustyyppi, joka kuvaa järjestelmässä tapahtunutta virheellistä toimintoa.

Seuraavaksi määritellään järjestelmän versioarkistokeskeisen osan sisältämät joukot ja kuvaillaan niiden sisältämät tyypit ja alkiot, sekä hahmotellaan niiden välisiä suhteita:

1. **Versioarkisto** sisältää joukon *Komponentteja* ja *Konfiguraatioita*.

2. **Komponentti** sisältää puumaisessa rakenteessa joukon *Revisioita* ja yksiselitteisen *[NimiTyypin]*.

3. **Revisio** sisältää deltatietona joukon *[LohkoTyyppejä]* ja täsmälleen yhden *[IDTyypin]*. *Revisiolla* on lisäksi "'linkitykset" edelliseen (1 kpl) ja seuraaviin (n kpl) *Revisioihin*. Näiden linkitysten avulla muodostuu *Komponentin* puumuotoinen revisiohistoria.

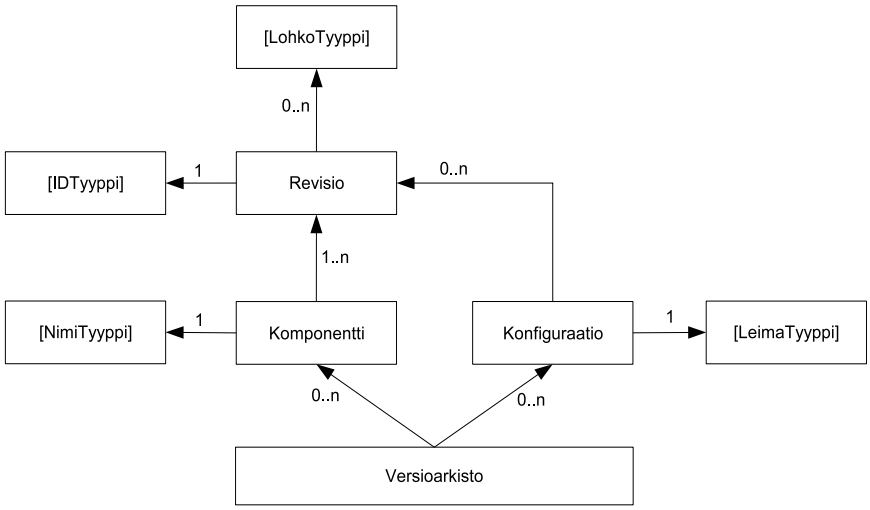
4. **Konfiguraatio** sisältää yksikäsitteisen *[LeimaTyyppi]*-tunnisteen ja joukollisen *Revisio*-linkityksiä.

Versioarkiston mallinnus koostuu *Versioarkisto*-, *Konfiguraatio*-, *Komponentti*- ja *Revisio*-luokista. Nämä luokat käyttävät lisäksi abstrakteja perustyyppejä *[NimiTyyppi]*, *[IDTyyppi]*, *[LohkoTyyppi]* ja *[LeimaTyyppi]*. Kuvassa 8.2 hahmotellaan näiden luokkien ja abstraktien perustyyppien välisiä suhteita. Tässä hahmotelmassa ei ole kuvattuna *Komponentin Revision* puumaista historiarakennetta. Sen sijaan *Komponentti* esitetään kuvassa *Revisioita* sisältävänä joukkona.

8.1 Mallinnusesimerkkinä Versioarkisto-luokka

Rakenteellisesti kuvissa 8.3, 8.4 ja 8.5 esitettävä *Versioarkisto*-luokka on eräänlainen säiliö *Komponentti*- ja *Konfiguraatio*-luokkien instansseille. Näitä instansseja säilötään sekvenssityyppisissä attribuuteissa: *komponentit* ja *konfiguraatiot*. Järjestelmän käynnistyksen yhteydessä nämä attribuutit täytetään esimerkinomaisen mukaisilla *laskin*-ohjelmiston komponenteilla ja konfiguraatioilla. Tässä yhteydessä tätä alustustapahtumaa ei kuvata alustavassa tilaskeemassa eli *INIT*-operaatiossa, vaan se jätetään abstraktille tasolle. Muiden *Versioarkisto*-luokan operaatioiden toimintaa voidaan suomentaa seuraavalla tavalla:

- **NoudaRevisio**: Tämä operaatio mahdollistaa yhden *[NimiTyyppi]* ja *[IDTyyppi]* -tyyppisillä parametreilla tunnistettavan *Revision* siirron *Tiedostoksi Protokollan* sisälle.



Kuva 8.2: Versioarkiston sisältämien alkioiden ja joukkojen suhteet.

- **TallennaRevisio**: Jos *[NimiTyypillä]* tunnistettava *Komponentti* löytyy jo *komponentit*-sekvenssistä, niin operaatio tallentaa uuden *Revision* revisiohistoriassa *[IDTyyppi]*-tyyppisellä parametrilla tunnistettavan *Revision* seuraajaksi. Vastaavasti mikäli *Komponenttia* ei vielä löydy *komponentit*-sekvenssistä, niin operaatio luo uuden *Komponentin* ja jatkaa sekvenssiä sillä.

- **NoudaKonfiguraatio**: Tässä operaatiossa etsitään *[LeimaTyyppi]*-tyyppisellä parametrilla identifioitu *Konfiguraatio* ja siirretään sen sisältämät *Revisiot Tiedostoina Protokollan* sisälle.

- **LuoKonfiguraatio**: Operaatiossa luodaan uusi *[LeimaTyyppi]*-parametrilla tunnistettava konfiguraatio ja lisätään se *konfiguraatiot*-sekvenssiin.

- **EtsiKonfiguraatio**: Tämä operaatio etsii *konfiguraatiot*-sekvensistä *[LeimaTyyppi]*-tyyppistä parametria vastaavan *Konfiguraatio*-luokan instanssin.

- **LisaaLeima**: Operaatio etsii parametrina saatua *[LeimaTyyppiä]* vastaavan *Konfiguraation* hyödyntämällä operaatiota *EtsiKonfiguraatio*. Mikäli *Konfiguraation* instanssi löytyy, niin toisena parametrina saatu *Revision* instanssi lähetetään edelleen löydetyn *Konfiguraation* operaatiolle *LisaaRevisio*.

- **PoistaLeima**: Kuten *LisaaLeima*-operaatiossakin, niin tässäkin etsitään aluksi parametrina saatua *[LeimaTyyppiä]* vastaava *Konfiguraatio*. Löydetystä *Konfiguraatiosta* poistetaan parametrina saatu *Revisio* lähettämällä se edelleen *Konfiguraatio*-luokan operaatiolle *PoistaRevisio*.

Versioarkisto

$komponentit : \text{seq}\, Komponentti$
$konfiguraatiot : \text{seq}\, Konfiguraatio$

NoudaRevisio

$nimi? : [NimiTyyppi]$
$id? : [IDTyyppi]$
$revisio : Revisio$
$protokolla! : Protokolla$
$i : \mathbb{N}$

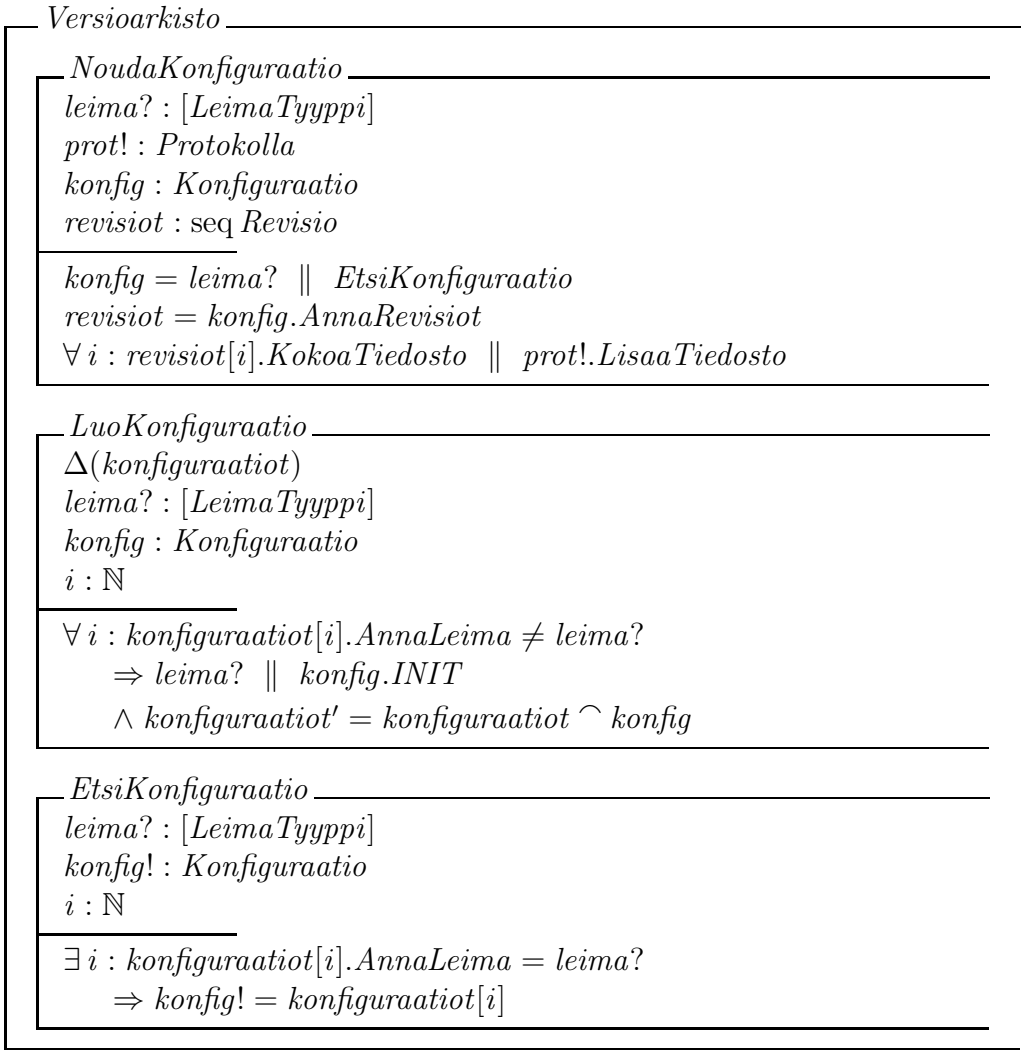
$\exists\, i : \{komponentit[i].AnnaNimi = nimi?\}$
$\quad \Rightarrow revisio = \{id? \parallel komponentit[i].EtsiRevisio\}$
$\quad \wedge\, revisio.KokoaTiedosto \parallel protokolla!.LisaaTiedosto$

TallennaRevisio

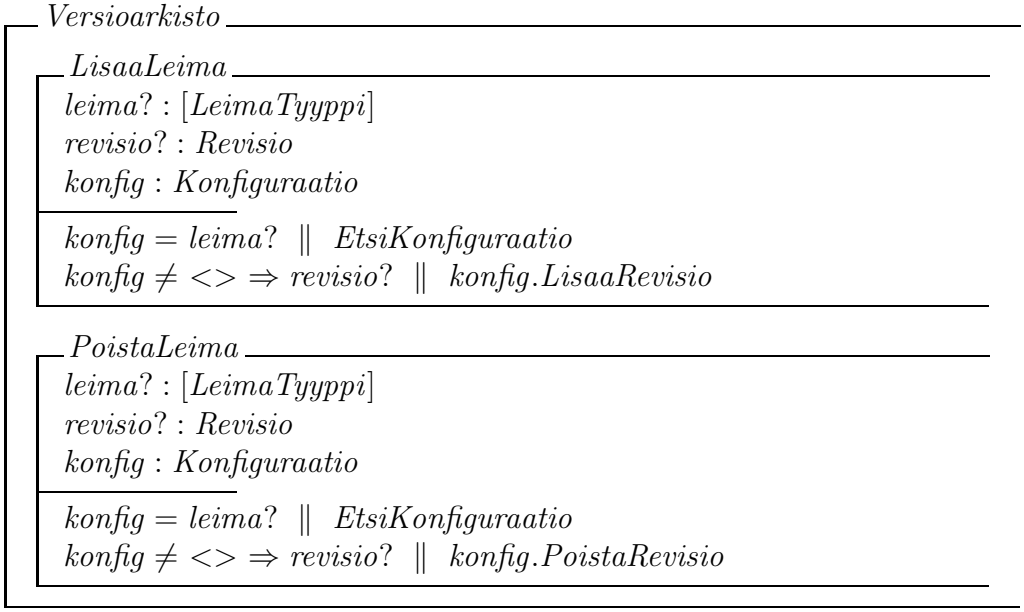
$id? : [IDTyyppi]$
$protokolla? : Protokolla$
$juurirevisio : Komponentti$
$i : \mathbb{N}$
$revisio : Revisio$
$tiedosto : Tiedosto$

$tiedosto = protokolla?.PalautaTiedosto$
$\exists\, i : \{komponentit[i].AnnaNimi = tiedosto.nimi\}$
$\quad \Rightarrow revisio = \{id? \parallel komponentit[i].EtsiRevisio\}$
$\quad \wedge\, tiedosto.sisalto \parallel revisio.LuoUusi$
$[]$
$\nexists i : \{komponentit[i].AnnaNimi = tiedosto.nimi\}$
$\quad \Rightarrow \{tiedosto.nimi \wedge this\} \parallel juurirevisio.INIT$
$\quad \wedge\, tiedosto.sisalto \parallel juurirevisio.LuoUusi$
$\quad \wedge\, komponentit' = komponentit \frown juurirevisio$

Kuva 8.3: Versioarkistoa kuvaava luokka, osa 1.



Kuva 8.4: Versioarkistoa kuvaava luokka, osa 2.



Kuva 8.5: Versioarkistoa kuvaava luokka, osa 3.

9 Lopuksi

Tämän luentomonisteen esityksen antia ohjelmistokehitykselle voidaan kiteyttää ohjenuoriksi, joita noudattamalla on mahdollista vähentää toistettavaan ohjelmiston rakentamiseen liittyvää sekaannusta:

1. Pidä kaikki ohjelmiston kehitykseen liittyvät komponentit versionhallinnassa ja tee kaikesta ohjelmiston rakentamiseen liittyvästä informaatiosta *materiaaliluettelo*.

2. Kerää ohjelmiston muutostarpeet muutospyynnöiksi ja suunnittele ohjelmistoon toteutettavat muutokset etukäteen.

3. Tallenna tekemäsi muutokset säännöllisesti versioarkistoon. Komponenttien vaihto tiimin jäsenten välillä on järkevintä suorittaa versioarkiston välityksellä.

4. Tee komponentin revisiohistoriaan vain perusteltavissa olevia haaroja, sillä revisiohistorian haaraumat voivat johtaa liitosongelmiin. Harkitse myös varianttien toteutusta ohjelmallisesti.

5. Ole huolellinen tehdessäsi komponenttien revisioiden välisiä liitoksia. Väärin toteutettu liitos voi johtaa ohjelman vääränlaiseen toimintaan tai jopa lähdekoodin kääntymättömyyteen väärän syntaksin takia

6. Sido julkaistavan ohjelmiston versiota vastaava konfiguraatio esimerkiksi leimalla. Konfiguraation sitominen tukee ohjelmiston saman version uudelleen rakentamista.

7. Aloita ohjelman julkaisuversion rakennustapahtuma aina tyhjältä työalueelta. Vanhat tai ylimääräiset tiedostot työalueella saattavat sotkea rakennustapahtumaa tai aiheuttaa ohjelmistoon ei toivottuja toiminnallisuuksia.

8. Rakenna ensimmäinen ohjelmiston versio heti kun se on mahdollista. Ohjelmiston komponenttien välinen integraatio voi onnistua helpommin, jos mahdolliset pulmat havaitaan aikaisessa vaiheessa.

9. Rakenna säännöllisesti ja usein pysyäksesi selvillä ohjelmiston kehitystilasta.

10. Automatisoi rakennusprosessi, sillä automatisoinnilla voidaan pienentää rutiininomaisen työn määrää ja keskittyä itse ohjelmiston kehitykseen.

10 Lähteet

[1] Ant Apache. URL: http://ant.apache.org, tark. 25.6.2006.

[2] Wayne A. Babich. *Software Configuration Management; Coordination for Team Productivity.* Addison-Wesley, 1986.

[3] Ralph Cabrera, Brad Appleton, ja Stephen P. Berczuk. Software reconstruction: Patterns for reproducing software builds. Raportti, PLoP '99 Conference, 1999. URL: http://citeseer.ist.psu.edu/cabrera99software.html, tark. 25.6.2006.

[4] Reidar Conradi. Towards a uniform version model for software configuration management. Raportti, Norwegian University of Science and Technology (NTNU), 1997. URL: http://citeseer.ifi.unizh.ch/565181.html, tark. 25.6.2006.

[5] Reidar Conradi. Version models for software configuration management. Raportti, Norwegian University of Science and Technology, Trondheim, 1998. URL: http://portal.acm.org/citation.cfm?id=280280&coll=portal&dl=ACM&CFID=10697426&CFTOKEN=21159027, tark. 25.6.2006.

[6] Ivaca Crnkovic, Magnus Larsson, ja Frank Lüders. Software process measurements using software configuration management. Raportti, Departmanent of Computer Engineering - Mäladelan University, 2000. URL: http://www.mrtc.mdh.se/publications/0133.pdf, tark. 25.6.2006.

[7] Susan Dart. Spectrum of functionality in configuration management systems. Raportti, Software Engineering Institute, Carnegie-Mellon University, 1990. URL: http://www.sei.cmu.edu/publications/documents/90.reports/90.tr.011.html, tark. 25.6.2006.

[8] Susan Dart. Concepts in configuration management systems. Raportti, Software Engineering Institute, Carnegie-Mellon University, 1991. URL: ftp://ftp.sei.cmu.edu/pub/case-env/config_mgt/papers/cm_concepts.ps, tark. 25.6.2006.

[9] Peter H. Feiler. Configuration management models in commercial environments. Raportti, Software Engineering Institute, Carnegie Mellon University, 1991. URL: http://www.sei.cmu.edu/legacy/scm/abstracts/abscm_models_TR07_91.html, tark. 25.6.2006.

[10] Peter H. Feiler ja Grace Downey. Transaction-oriented configuration management. Raportti, Software Engineering Institute, Carnegie Mellon University, 1990. URL: http://www.sei.cmu.edu/publications/documents/90.reports/90.tr.023.html, tark. 25.6.2006.

[11] S. I. Feldman. Make - a program for maintaining computer programs. Raportti, Bell Laboratories, 1979. URL: http://citeseer.ist.psu.edu/feldman79make.html, tark. 25.6.2006.

[12] Karol Frühauf ja Andreas Zeller. Software configuration management: State of the art, state of the practice. Raportti, INFORGEM AG, Informatiker Gemeinschauf für Unternehmensberatung / Universität Passau, Lehrstuhl für Softwaresysteme, 1999. URL: http://www.infosun.fmi.uni-passau.de/st/papers/scm99/scm99.pdf, tark. 25.6.2006.

[13] Ilkka Haikala ja Jukka Märijärvi. *Ohjelmistotuotanto*. Talentum Media Oy, 1995.

[14] IEEE. IEEE Std 610.12-1990, IEEE Standard Glossary of Software Engineering Technology. Raportti, The Institute of Electrical and Electronics Engineering, 1990. URL: http://ieeexplore.ieee.org/xpl/tocresult.jsp?isNumber=4148, tark. 25.6.2006.

[15] Alexis Leon. *A Guide to Software Configuration Management*. Artech House, Inc., 2000.

[16] Roger S. Pressman. *Software Engineering a Practioner's Approach 5th Edition*. McGraw-Hill Publishing Company, 2000.

[17] Jari Vanhanen. Improving configuration management processes of a software product. Pro Gradu -tutkielma, Helsinki University of Technology, Department of Computer Science, 1997. URL: http://www.soberit.hut.fi/~jvanhane/pubs/scmthesis.pdf, tark. 25.6.2006.

[18] David Whitgift. *Methods and Tools for Software Configuration Management*. John Wiley & Sons, Ltd., 1991.

[19] Laura Wingerd ja Christopher Seiwald. High-level best practices in software configuration management. Raportti, Perforce Software, 1998. URL: http://www.perforce.com/perforce/bestpractices.html, tark. 25.6.2006.

[20] Andreas Zeller. *Configuration Management with Version Sets, A Unified Software Versioning Model and its Applications*. Väitöskirja, Technische Universität Braunschweig, 1997. URL: http://www.infosun.fmi.uni-passau.de/st/papers/zeller-phd/, tark. 25.6.2006.