Aalto-yliopisto
Perustieteiden korkeakoulu
Teknillisen fysiikan ja matematiikan koulutusohjelma

Janrik Öberg

# Räätälöidyn ohjelmiston päivitettävyyden arviointi

Sovelletun matematiikan erikoistyö
Espoo, 16. toukokuuta 2016

# Sisältö

# Luku 1

# Johdanto

Tietokonesovelluksen lähdekoodin laatu ja rakenne ovat ensisijaisen tärkeässä asemassa niin sovelluksen ylläpidettävyyden, virhealttiuden kuin päivitettävyyden kannalta [11]. Erilaista lähdekoodianalyysiä on tehty esimerkiksi tyypillisten ohjelmointivirheiden havaitsemiseen [6], haitta- tai mainosohjelmakoodin havaitsemiseen [9], suorituskyvyn parantamiseksi [5] ja tietoturvallisuuden huomioon ottamiseen [3]. Binkley et. al. [1] esittävät työssään näkemyksensä automatisoidun ohjelmakoodin analyysityökalujen nykytilasta. He esittävät kolmisenkymmentä eri käytännön sovelluskohdetta, joissa lähdekoodianalyysia käytetään jo nyt. Heidän mukaansa lähdekoodianalyysiä tullaan tulevaisuudessa käyttämään yhä laajemmin ja se tulee jatkossakin olemaan tärkeä tutkimusaihe.

Stamelos et. al. [10] esittävät työssään huonon ohjelmakoodin kompastuskiviä sekä joukon teknisiä mittareita ohjelmakoodin laadun tarkasteluun. Ueda et. al. [11] tutkivat työssään lähdekoodin ylläpidettävyyttä etenkin havainnollistamalla ohjelmakoodissa löytyviä toistettuja lohkoja. Tällaiset "kloonilohkot"ovat ylläpidollinen painajainen, sillä jokainen muutos yhteen lohkoon tulee aina huomata toisintaa jokaiseen samanlaiseen lohkoon. Ohjelmakoodin laatu vaikuttaa huomattavasti ohjelmiston kustannustehokkuuteen ja päivitysten vaatimaan työmäärään [2]. Ohjelmakoodiin tehtyjen muutosten vaikutusta ylläpidettävyyteen ja päivitettävyyteen on tutkinut mm. Elshishiny [4].

Tässä työssä tarkastelen erityistapauksena Microsoft Dynamics NAV -toiminnanohjausjärjestelmään tehtyjen ohjelmistoräätälöintien vaikutusta

järjestelmän versionvaihdon vaatimaan työmäärään. Tavoitteenani on muodostaa eri kriteerejä painottamalla indeksi, joka mahdollisimman hyvin ennustaa ohjelmiston versionvaihdon vaatimaa työmäärää.

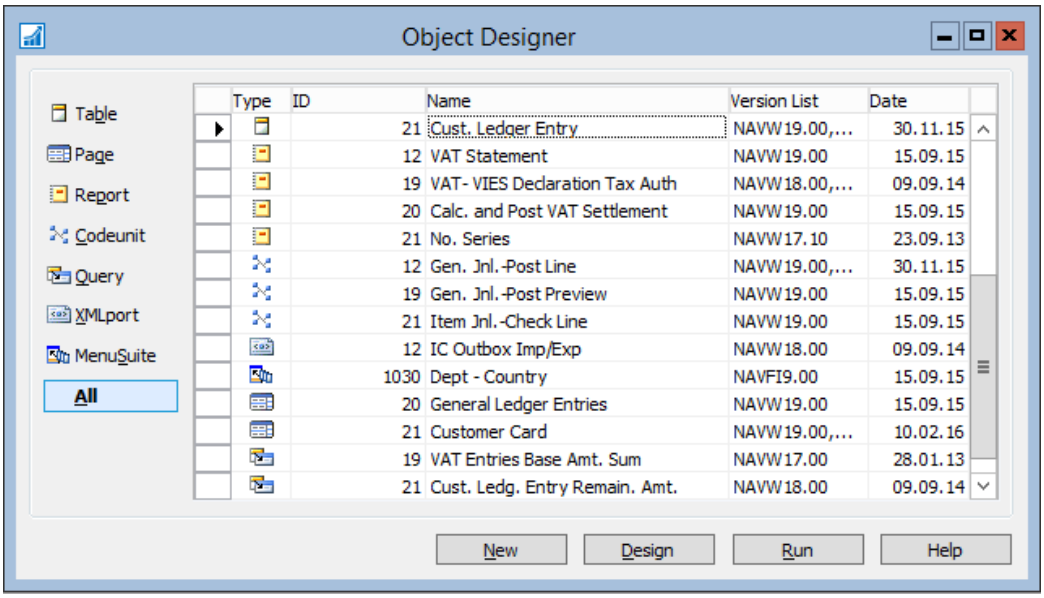
Luvussa 3 esitän oman pohdintani räätälöidyn järjestelmän versionvaihdon tekemisestä ja siinä huomioitavista tekijöistä. Kartoitan ensin oleellisimmat räätälöinnin astetta, eli poikkeamaa perusversiosta, kuvaavat kriteerit. Esittelen myös päivitettävyysindeksin laskemiseksi rakentamani ohjelmistotyökalun.

Lopulta lasken toteutetulla työkalulla käytettävät painokertoimet. Teen sen sovittamalla mallia muutamaan versionvaihtoprojektiin, joiden lopullinen kustannus on tiedossa. Lopulta arvioin mallin toimivuutta sekä lähestymistavan puutteita ja jatkokehitysmahdollisuuksia.

# Luku 2

# Ohjelmiston rakenne

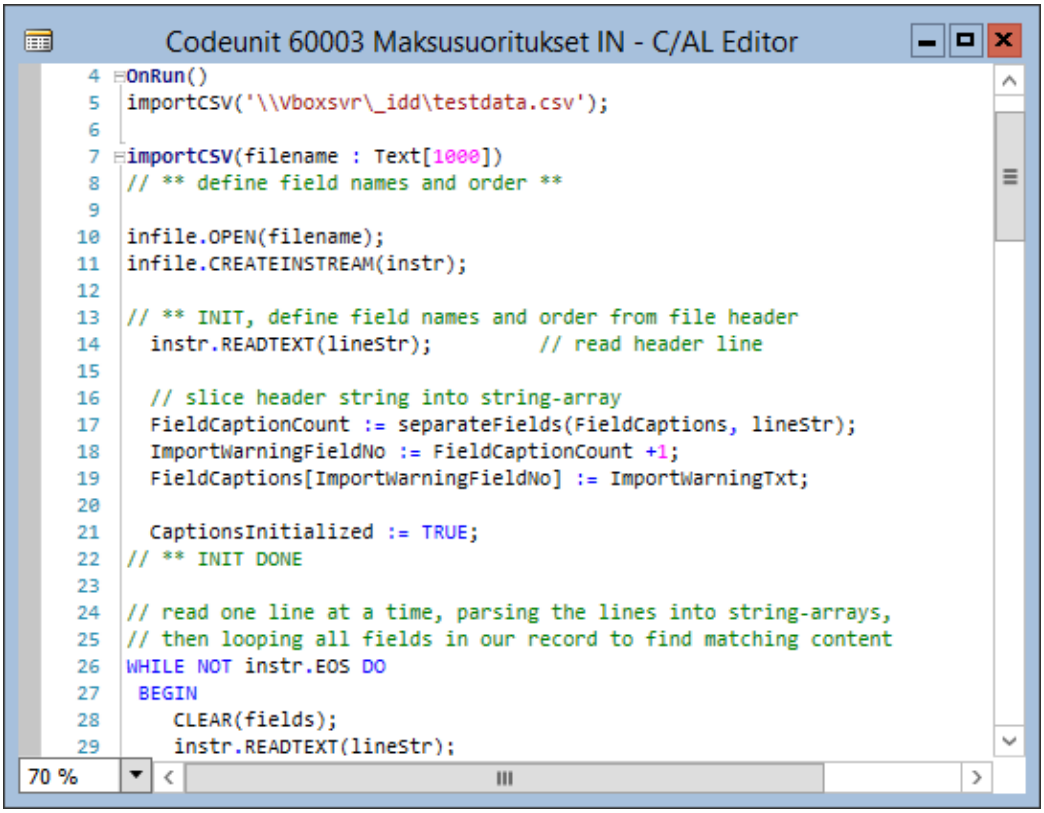
Tässä työssä tarkasteltava, räätälöitävissä oleva osa Microsoft Dynamics NAV -toiminnaohjausjärjestelmän lähdekoodia, löytyy järjestelmän *objektitietokannasta* (lyhyemmin tietokanta). Tietokanta koostuu erityyppisistä *objekteista*. Objekti on tiettyä toiminnallisuutta toteuttava osa järjestelmää. Järjestelmän toiminnan ja käyttäjän kannalta eri tyyppiset objektit eroavat huomattavasti toisistaan.




Kuva 2.1: Esimerkki tietokannasta erityyppisine objekteineen

Siinä missä Page-tyyppinen objekti näkyy käyttäjälle tietoruutuna, josta hän voi nähdä jonkin tuotteen varastosaldon tai johon hän voi syöttää uuden tilauksen, Report-tyyppinen objekti koostaa tietoa, kuten myyntitilastoja, tulostettavaksi raportiksi. Codeunit-tyyppiset objektit suorittavat toimintoja, esim. varastosaldon ylläpitoa, käyttäjältä piilossa.

Kuvassa 2.1 on esitetty eri objektityypit ja muutama erillinen objekti. Koodianalyysissä ja muutosvertailussa voimme käsitellä kaikkia objekteja samalla tapaa, sillä ne rakentuvat kaikki tekstimuotoisesta *lähdekoodista*.




Kuva 2.2: Osittainen lähdekoodiesimerkki Codeunit-tyyppisestä objektista

Objekteja on järjestelmän versiosta riippuen muutamasta tuhannesta noin viiteen tuhanteen kappaletta. On huomioitava, että jokainen objekti on tyyppinsä sisällä uniikisti numeroitu - ei voi olla kahta samantyyppistä objektia samalla tunnistenumerolla. Kuvassa 2.2 on esimerkin vuoksi esitetty osittainen lähdekoodi Codeunit-tyyppisestä objektista, joka lukee järjestelmään ulkopuolisen tahon toimittamia maksusuorituksia.

# Luku 3

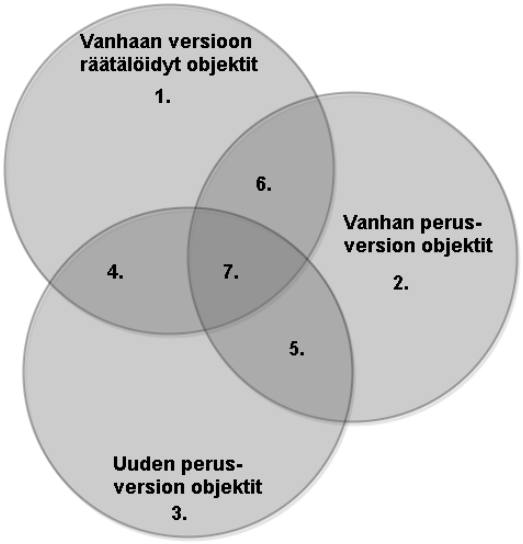
# Tietokannan versionvaihto

Räätälöimättömän toiminnanohjausjärjestelmän, eli Microsoftin perusversion, päivitys tai versionvaihto on työmäärältään pieni, mutta työmäärä kasvaa huomattavasti riippuen siitä, kuinka paljon järjestelmää on muokattu. Tässä kappaleessa esitän oman pohdintani räätälöidyn järjestelmän versionvaihdon tekemisestä.

## 3.1 Versionvaihdossa huomioitavaa

Järjestelmän versionvaihdossa tavoitteena on muodostaa uusi tietokanta, jossa on kaikki uuden perusversion objektit sekä kaikki vanhaan versioon tehdyt räätälöinnit. Arvioitaessa muutosten laajuutta, tulee vertailuun ottaa vanhan perusversion objektit, vanhaan versioon tehdyt räätälöinnit ja uuden version objektit. Joissakin tilanteissa joudumme yhdistelemään eri versioiden objektien lähdekoodeja, toisissa voimme käyttää jonkin version objektia sellaisenaan uudessa tietokannassa.

Tässä kolmen version objektien vertailussa päädymme jokaisen objektin kohdalla yhteen seitsemästä tapauksesta, joita kuva 3.1 havainnollistaa.




Kuva 3.1: Venn-diagrammi eri versioiden objektien vertailusta

1. Kyseinen objekti löytyy vain räätälöidystä kannasta, eli se on lisätty vanhan perusversion tietokantaan.
   - Objekti ei ole ristiriidassa toisen kanssa, eli se voidaan ottaa se sellaisenaan uuteen tietokantaan.
2. Objekti löytyy vanhan perusversion kannasta, mutta se ei eroa uuden perusversion objektista. Objektiin ei siis ole tullut muutoksia uuden ja vanhan perusversion välillä, eikä sitä olla räätälöity.
   - Sama kuin 1.
3. Uuteen perusversioon lisätty objekti, jota ei löydy vanhasta perusversiosta ja jonka numerointi ei ole ristiriidassa minkään räätälöidyn objektin kanssa.
   - Sama kuin 1. ja 2.
4. Uuteen perusversioon lisätty objekti, jonka tunniste on ristiriidassa räätälöityyn kantaan lisätyn objektin kanssa.
   - Joudutaan tekemään räätälöidylle objektille *uudelleen numerointi*.

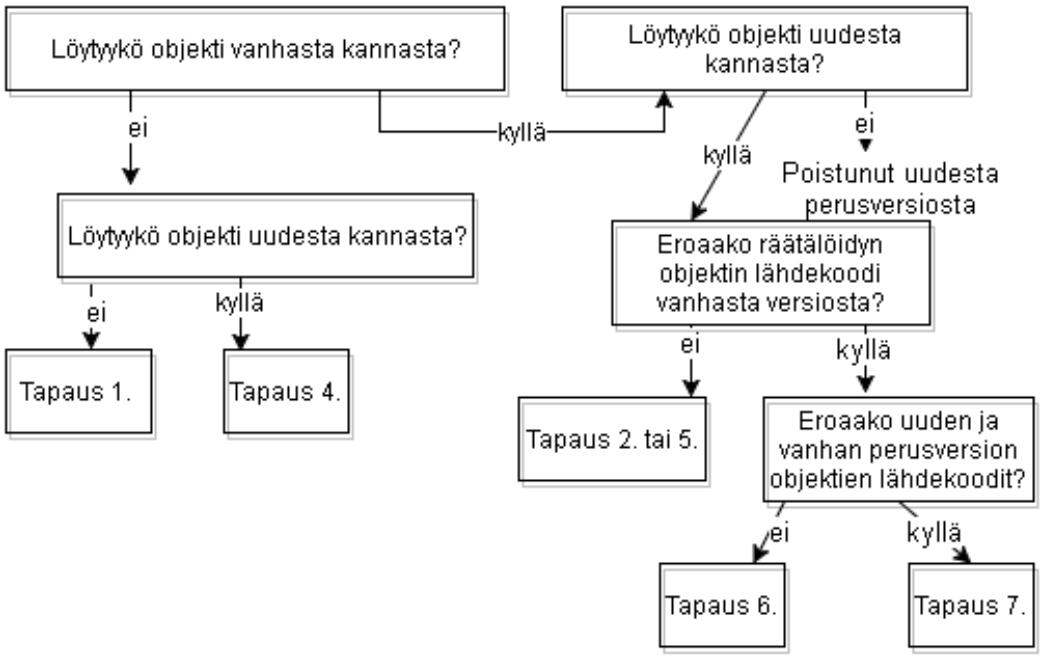
5. Vanhan ja uuden perusversion objektit eroavat toisistaan, mutta kyseistä objektia ei ole räätälöity.
   - Voimme ottaa lopulliseen tietokantaan uuden tietokannan objektin, sillä Microsoft on jo tehnyt objektin päivitykset.
6. Vanhan ja uuden perusversion objektit eivät eroa toisistaan, kyseiseen objektiin on tehty räätälöityjä muutoksia.
   - Voimme ottaa räätälöidyn objektin uuteen tietokantaan, sillä perusversioiden objektit ovat identtiset.
7. Vanhan ja uuden perusversion objektit eroavat toisistaan, jonka lisäksi kyseisen objektin vanhaan versioon on tehty muutoksia.
   - Joudumme tekemään objektien yhteenliittämisen (merge), jossa yhdistämme vanhaan versioon tehdyt räätälöinnit uuden version objektiin. Tämä on työläin tapaus.

Näiden lisäksi on olemassa harvinainen erikoistapaus, jossa vanhan perusversion objektia on muokattu, mutta kyseistä objektia ei enää ole uudessa perusversiossa. Näitä muutoksia ei voida tuoda uuteen versioon.

Koska lopputuloksena halutaan tietokanta, joka sisältää sekä kaikki uuden version muutokset, että vanhaan versioon tehdyt räätälöinnit, eroaa eri tapausten käsittely toisistaan. Tietokannan analysoimisen vaiheet:

1. Tuodaan kaikkien tietokantojen kaikkien objektien lähdekoodit tekstitiedostoihin.
2. Erotellaan lähdekoodi siten että jokaista objektia vastaa yksi tiedosto.
3. Käydään kuvan 3.2 mukaisesti yksitellen läpi räätälöidyn kannan objektit.
4. Kun päädytään tapaukseen 7. tehdään lähdekoodin vertailu, kuten kuvassa 3.3

Sillä oleellisin yksittäinen tekijä muutosten kokoluokkaa arvioitaessa on lähdekoodirivien määrä, joille täytyy tehdä yhteenliittäminen, on tapaus 7.




Kuva 3.2: Päättelypolku objektien käsittelystä

kiinnostavin. Eroavien lähdekoodirivien tarkasteluun käytän tässä työssä avoimen lähdekoodin GNU Diffutils (diff) työkalua.[1]

Työkalu ottaa syötteenä kahden objektin lähdekoodit, joita se vertailee keskenään antaen tulosteena näiden erotustiedoston. Tästä tulosteesta pystymme näkemään alkuperäiseen tiedostoon nähden tehdyt lisäykset (uudet rivit) sekä alkuperäisestä poistetut rivit. Muuttunutta riviä kuvataan alkuperäisen poistona ja uuden lisäyksenä.

Perättäiset muuttuneet rivit tulkitaan yhdeksi muuttuneeksi lohkoksi. Tämä on oleellista, sillä viisi perättäistä muutettua riviä - eli yksi lohko, on paljon helpompi siirtää uuteen ohjelmaversioon kuin viisi täysin erillistä riviä, joiden välissä on muuta ohjelmakoodia.

Kuvassa 3.3 nähdään yhden Codeunit-tyyppisen objektin erotustiedosto uuden ja vanhan tietokannan välillä. Plus-merkillä diff kuvaa lisättyä riviä, miinusmerkillä poistettua. Lohkot on erotettu @@-merkinnällä, jolloin näemme helposti, että muuttuneita lohkoja on neljä kappaletta.

[1] http://www.gnu.org/software/diffutils/diffutils.html

```
--- \\fileshare\orig\COD22.TXT  2016-04-15 13:39:33 +0300
+++ \\fileshare\newf\COD22.TXT  2016-04-15 13:40:07 +0300
@@ -5,3 +5,4 @@
-    Date=28.01.13;
-    Time=12:00:00;
-    Version List=NAVW17.00;
+    Date=12.01.14;
+    Time=20:17:03;
+    Modified=Yes;
+    Version List=NAVW17.00,M011;
@@ -1860 +1861 @@
-    ItemLedgEntry."Entry Type"::Consumption,
+    //ItemLedgEntry."Entry Type"::Consumption,  // M011
@@ -1865 +1866,3 @@
-    THEN
+    THEN BEGIN
+      //MESSAGE(Text005,ItemLedgEntry."Item No.");
+     //EXIT;
@@ -1866,0 +1870 @@
+    END;
```

Kuva 3.3: Diff-ohjelman tuottama vertailu kahdesta tiedostosta

## 3.2 Toteutettu aputyökalu

Osana tätä erikoistyötä käytin noin yhden viikon työpanoksen kehittäessäni työkalun, joka automaattisesti tekee kaikki työssä kuvatut vaiheet. Toteutettu ohjelma siis:

1. Erottelee objektien lähdekoodit.
2. Käy läpi kuvassa 3.2 esitetyn päättelypolun.
3. Vertailee lähdekooditiedostojen sisällön keskenään.
4. Koostaa objekteittain tiedot muuttuneista riveistä ja koodilohkoista.
5. Laskee indeksin käyttäen vapaavalintaisia painoja.

Ko. ohjelmaa käytän työn yhteenvedossa esittämieni projektien analysoimiseen ja painokerrointen laskemiseen. Ohjelmaa ja sen antamia tuloksia ollaan ottamassa koekäyttöön tarjousten laatimisen tueksi versionvaihtoprojekteja toteuttavassa IT-alan yrityksessä.

# Luku 4

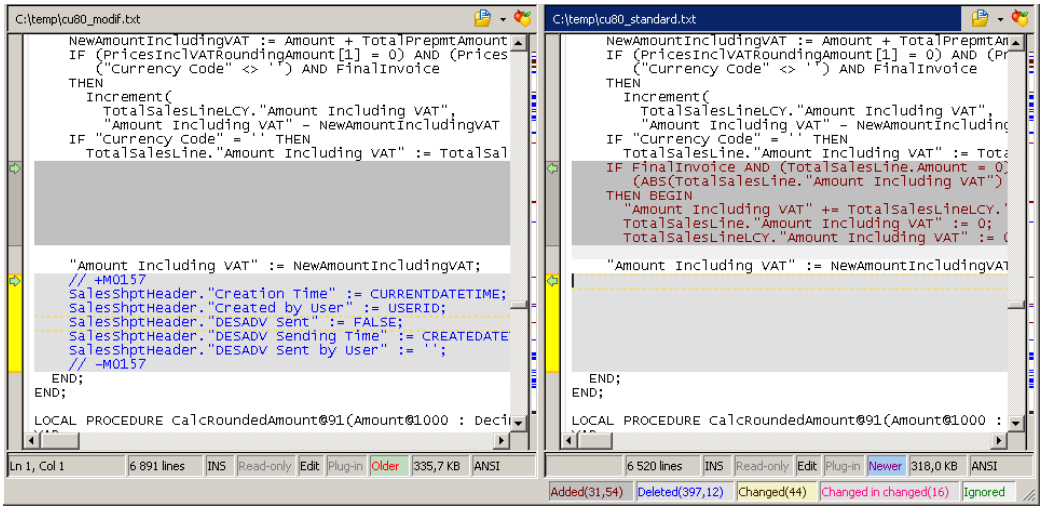
# Kriteerit

Mahdollisia kriteerejä versionvaihdon työmäärän arvioimiseksi ovat muuttuneiden ohjelmakoodilohkojen, lisättyjen tai poistettujen rivien ja muutettujen objektien määrä. Lisäksi olisi mahdollista painottaa eri objektityyppien muutoksia eri kertoimella.

Valittaessa kriteerejä tulee huomioida miten eri versioiden objektien yhteenliittäminen käytännössä tehdään. Haastattelemieni ohjelmakehittäjien ja oman kokemukseni perusteella tehokkaimmaksi ja toimivimmaksi tavaksi on todettu lähdekooditietostojen käsittely rinnakkain tekstinkäsittelyohjelmassa, joka havainnollistaa tiedostojen eroavaisuudet. Lorente [7] et. al. esittävät kirjassaan vaihtoehtoisen tavan käyttämällä MergeTool-ohjelmaa.

Yleensä on tapana pitää vasemmalla puolella räätälöidyn vanhan version lähdekoodia ja oikealla uuden perusversion lähdekoodia. Uuden perusversion lähdekoodia työstetään eroavuus kerrallaan kohti lopullista muotoa, jossa on säilytetty kaikki uuden version toiminnallisuus ja lisätty vanhaan versioon tehdyt muutokset. Tällä tavalla työskenneltäessä on helppo siirtää kokonaisia muuttuneita ohjelmakoodilohkoja vanhasta versiosta uuteen.

Kuvassa 4.1 nähdään esimerkki kahden lähdekooditiedoston rinnakkaisesta käsittelystä. Esimerkin kuvassa uuteen perusversion objektiin (oikealla) on tullut yksi kuusi riviä pitkä lohko ohjelmakoodia lisää verrattuna räätälöityyn vanhan version objektiin (vasemmalla). Räätälöityyn objektiin on tämän kohdan alle lisätty yksi seitsemän riviä pitkä lohko ohjelmakoo-




Kuva 4.1: Esimerkki lähdekoodien yhteenliittämisestä käytännössä

dia. Yhteenliittämisessä tämä räätälöityyn objektiin lisätty lohko kopioidaan uuden perusversion objektiin. Tämän perusteella vaikuttaa järkevältä ottaa *muuttuneiden ohjelmakoodilohkojen määrä* yhdeksi kriteeriksi.

Kuten kappaleessa 3 esitetään, täysin uusien räätälöityjen objektien tuominen uuteen versioon on vaivatonta. Lisäksi, koska yhteenliittämisprosessissa käsitellään tekstitiedostoja, joissa on koko kannan kaikkien objektien lähdekoodit peräkkäin, ei muuttuneiden objektien lukumäärä suoraan vaikuta lähdekoodien yhdistämisen työmäärään. Näistä syistä johtuen muuttuneiden objektien lukumäärä ei suoraan ole vaikuttava tekijä päivitysprosessin työläyttä arvioitaessa. Kuitenkin, kaikki perusobjekteihin tehdyt muutokset lisäävät hankalien ristiriitatilanteiden riskiä versionvaihdossa. Tällainen voi tapahtua esimerkiksi, jos jokin toiminnallisuus perusversioiden välillä muuttuu niin radikaalisti, että koko toiminnallisuutta, johon on tehty räätälöinti, ei enää ole uudessa versiossa olemassa.

Ylläpidettävyyden ja päivitettävyyden kannalta Microsoftin suositus heidän ohjelmakehityskursseillaan [8] sekä oppimateriaalissa on, että standardiobjekteita muokataan mahdollisimman vähän. Kirjassaan Lorente [7] et. al. vahvistavat näkemyksen, että muokattujen perusobjektien määrä kuvastaa räätälöinnin astetta, ja että pienempi määrä on toivottavampi. Näistä syistä näen järkeväksi sisällyttää indeksiin kriteeriksi myös *muokattujen perusobjektien määrän*. Tällä tavalla indeksi kuvastaa myös ohjelmakehityksen laatua.

Lisättyjen tai poistettujen rivien määrää ei ole tarpeen valita erillisiksi kriteereiksi, sillä muuttuneiden lohkojen määrä on työmäärän arvioinnin kannalta oleellisempi kriteeri. Ohjelmakoodin laatu ei myöskään riipu suoraan rivimäärästä. Paremmin tehty, selkeämpi ja ylläpidettävämpi ohjelmakoodi saattaa päinvastoin olla huonoa ja epäselvää koodia pidempi [2].

Kaikki objektityypit Report-tyyppiä lukuunottamatta yhdistellään samalla tapaa. Report-tyyppi on graafinen tuloste, jonka ulkomuodon pienikin räätälöinti, kuten tulostettavien tietojen siirtäminen eri kohtaan tulostetta, voi aiheuttaa kymmenien rivien muutoksen objektin lähdekoodiin. Report-objekteja ei käytännössä voida yhdistää tekstivertailulla ja niiden työmäärä arvioidaan erikseen. Tässä työssä jätän Report-tyypin pois indeksistä, jolloin emme tarvitse objektityypeille omia painokertoimia.

Painotettaviksi kriteereiksi valitaan siis:

1. Muuttuneiden ohjelmakoodilohkojen määrä.
2. Muokattujen perusobjektien määrä.

# Luku 5

# Yhteenveto

Olettaen että versionvaihdon kustannusta estimoiva indeksi voidaan muodostaa summaamalla painotetut kriteerit, laskin pienimmän neliösumman menetelmällä sellaiset painokertoimien arvot, jotka parhaiten vastaavat kolmen projektin A, B ja C toteutunutta kustannusta. Nämä käyvät ilmi taulukosta 5.1. Näiden kolmen projektin kohdalla malli näyttää antavan melko hyvin projektien toteutuneita kustannuksia vastaavat ennusteet.

| | Muokatut perusobjektit | Ohjelmakoodi-lohkojen määrä | Toteutunut kustannus | Indeksi | Mallin antama kustannusarvio |
|---|---|---|---|---|---|
| **Projekti A** | 146 | 898 | 3 564,00 € | 776,928 | 3 535,02 € |
| **Projekti B** | 179 | 1438 | 5 480,00 € | 1235,301 | 5 620,62 € |
| **Projekti C** | 87 | 734 | 3 100,00 € | 629,833 | 2 865,74 € |
| **Painokertoimet** | 0,161 | 0,839 | | 4,55 | |

Kuva 5.1: PNS-menetelmällä lasketut arvot painokertoimille ja niiden antamat kustannusarviot.

Työssä käyttämäni suoraviivainen tekstivertailumenetelmä versionvaihtoon toimii hyvin, kun päivitettävä tietokanta edustaa uusinta Dynamics NAV -sukupolvea. Käytännössä esitelty menetelmä soveltuu päivitettäessä 2009r2-versiota uudempia tietokantoja. Tätä vanhempien versioden teknologia ja perusobjektit eroavat niin paljon toisistaan, että näin suoraviivaista versionvaihtoa ei voi tehdä. Työssä en myöskään huomionut Report-tyyppisten objektien vaikutusta versionvaihdon työmäärään.

# Lähdeviitteet


[1] David Binkley. Source code analysis: A road map. In *2007 Future of Software Engineering*, FOSE '07, pages 104–119, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-2829-5. doi: 10.1109/FOSE.2007.27. URL http://dx.doi.org/10.1109/FOSE.2007.27.

[2] B. W. Boehm, J. R. Brown, and M. Lipow. Quantitative evaluation of software quality. In *Proceedings of the 2Nd International Conference on Software Engineering*, ICSE '76, pages 592–605, Los Alamitos, CA, USA, 1976. IEEE Computer Society Press. URL http://dl.acm.org/citation.cfm?id=800253.807736.

[3] T.W. Burrell, J.E. Caves, L. Lafreniere, and R.M. Shupak. Code property analysis for security mitigations, March 24 2015. US Patent 8,990,930 https://www.google.com/patents/US8990930.

[4] H.E. Elshishiny, S. Sabry, G.H. Selim, and O. Shokry. Analysis of source code changes, March 17 2015. US Patent 8,984,485 https://www.google.com/patents/US8984485.

[5] L.M. Hines. Optimizing applications using source code patterns and performance analysis, July 15 2014. US Patent 8,782,613 https://www.google.com/patents/US8782613.

[6] Gerard J Holzmann. Static source code checking for user-defined properties. In *Proc. IDPT*, volume 2, 2002.

[7] Cristina Nicolàs Lorente Laura Nicolàs Lorente. Implementing microsoft dynamics nav 2013, 2013. https://books.google.fi/books?id=QmoIU1gR8B8C&lpg=PT375&ots=bDUHAa_3pM.

[8] Microsoft. C/side solution development in microsoft dynamics® nav. https://www.microsoft.com/en-us/learning/course.aspx?cid=80437.

[9] M. Siman. Detecting malicious advertisements using source code analysis, April 3 2014. WO Patent App. PCT/IB2013/058,741 https://www.google.com/patents/WO2014049504A1?cl=en.

[10] Ioannis Stamelos, Lefteris Angelis, Apostolos Oikonomou, and Georgios L Bleris. Code quality analysis in open source software development. *Information Systems Journal*, 12(1):43–60, 2002.

[11] Yasushi Ueda, Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue. Gemini: Maintenance support environment based on code clone analysis. In *Software Metrics, 2002. Proceedings. Eighth IEEE Symposium on*, pages 67–76. IEEE, 2002.