TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Riho Koppel 120921

MS DYNAMICS AX RAKENDUSE VERSIOONIUUENDUSTE JUURUTAMISE METOODIKA VÄLJA TÖÖTAMINE TELEKOMMUNIKATSIOONIETTEVÕTTES

Bakalaureusetöö

Juhendajad: Riina Maigre
Doktorikraad
Kristjan Gerndorf
Bakalaureusekraad

Tallinn 2018

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Riho Koppel

20.05.2018

Annotatsioon

Käesoleva bakalaureusetöö eesmärgiks on kirjeldada protsessi, mille abil on võimalik MS Dynamics AX 2012 versiooniuuendusi läbi viia ning tuua välja võimalikud vead, mis antud protsessi käigus tekkida võivad. Lisaks sisaldab töö hinnangut rakenduse versiooniuuenduse järgsele testimismahule. Töö on kirjutatud ühe konkreetse Dynamics AX 2012 rakendust kasutava ettevõtte kontekstis.

Töö tulemusena valmib põhjalik protsessi kirjeldus versiooniuuenduste läbiviimiseks Dynamics AX 2012 rakenduses koos lahendustega vigadele, mis selle protsessi käigus esineda võivad. Töös valminud protsessi on võimalik ettevõttes kasutada ka tulevaste Dynamics AX 2012 versiooniuuenduste läbiviimiseks.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 48 leheküljel, 10 peatükki, 23 joonist, 3 tabelit.

Abstract

Methodology for Implementing MS Dynamics AX Version Updates in a Telecommunications Company

The goal of this thesis is to describe a methodology for implementing MS Dynamics AX version updates and to identify possible errors that might occur during the process. The thesis also includes a recommendation for the amount of testing required for the system after the update. The thesis explores the version update implementation process in the context of a certain telecommunications company.

The thesis results in a thorough description of the update process with solutions to the problems that occurred during the process and can be re-used for updating Dynamics AX 2012 to future versions.

The thesis is written in Estonian and contains 48 pages of text, 10 chapters, 23 figures, 3 tables.

Lühendite ja mõistete sõnastik

Auto-merge	Programmikoodi automaatne ühendamine.
AX2012	Axapta 2012, lühend äritarkvara täisnimetusest
CIL	*Common Intermediate Language*, .NET raamistikul põhinev keel
CU,CU7, CU13	*Cumulative Update*, Dynamics AX 2012 rakenduse versiooniuuendus. Number tähistab versiooni.
ERP	*Enterprise Resource Planning*, majandus- ja juhtimistarkvara
Modelstore	Dynamics AX 2012 rakenduse juurde kuuluv andmebaas, kus hoitakse rakenduse elementide definitsioone.
SQL	*Structured Query Language,* keel andmebaasi päringute kirjutamiseks.
SSRS	*SQL Server Reporting Services*, Microsofti poolt loodud programm aruandluse loomiseks.

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Käesolev bakalaureusetöö on kirjutatud eesmärgiga luua ühes konkreetses ettevõttes metoodika, mis aitaks antud ettevõttel oma äritarkvara versiooniuuendusi läbi viia. Töö autor on osalenud metoodika välja töötamise protsessis nii analüütiku kui ka arendaja rollis.

1.1 Taust ja probleem

MS Dynamics AX 2012 on Microsofti poolt pakutav tarkvaraline ERP süsteem keskmistele ja suurtele ettevõtetele. Antud ERP süsteem on kasutuses ühes Eesti telekommunikatsioonіettevõttes, kus sai aastal 2015 juurutatud rakenduse versioon CU7. Rakendus on ettevõttes kasutuses siiani ning seda on nii juurutuse käigus kui ka sellele järgnenud ajaperioodil suures mahus muudetud rahuldamaks konkreetse ettevõtte ärilisi vajadusi.

Praeguseks on Microsoft antud tootele väljastanud mitmeid *Cumulative Upgrade* (edaspidi CU) tarkvarauuendusi, millest viimane on versiooninumbriga 13. CU puhul on tegu tarkvarauuenduste pakiga, mis hõlmab endas lisaks uutele funktsionaalsustele ka kriitilisi veaparandusi rakenduse funktsionaalsuse, töökindluse ning jõudluse osas.

Tehnoloogilise võla vähendamiseks ning rakenduse kasutatavuse parandamiseks on ettevõte otsustanud oma ERP süsteemi viimasele versioonile viia ning plaanib versiooniuuendusi edaspidi teha regulaarselt.

Sellist rakenduse versiooniuuendust viiakse ettevõttes läbi esmakordselt ning seetõttu puudub ettekujutus uuenduse paigaldusprotsessist ning sellele järgnevast testimisprotsessist. Keerukust lisavad asjaolud, et rakendust on suures mahus modifitseeritud ning versiooniuuendus on mahult tavapärasest suurem, kuna rakendus tuleb viia versioonilt 7 otse versioonile 13.

1.2 Ülesande püstitus

Käesoleva bakalaureusetöö eesmärk on nii analüüsi kui praktilise töö käigus luua MS Dynamics AX 2012 versiooniuuenduste juurutamise taaskasutatav metoodika. Töös keskendutakse järgmistele eelanalüüsi käigus välja selgitatud küsimusele:

1. Kuidas toimub versiooniuuenduse paigaldamine ning milliseid probleeme võib selle käigus esineda?

2. Kui mahukat testimist versiooniuuendus vajab?

Töö käigus leitud lahendusi eelpool välja toodud küsimustele võibki tervikuna lugeda metoodikaks, mida annab rakendada ka tulevikus antud ettevõttes MS Dynamics AX 2012 versiooniuuendusi tehes.

1.3 Metoodika

Metoodika väljatöötamine toimub nii analüüsi kui ka praktilise töö käigus. Praktiline osa hõlmab endas arenduskeskkonnas versiooniuuenduse paigaldamist, et tekiks arusaam uuenduse paigaldusprotsessist, selle eelduseks olevatest nõuetest ning paigalduse käigus tekkivatest võimalikest vigadest. Praktiline osa toetub Microsofti poolt välja antud kirjandusele ning bakalaureuseõppe käigus omandatud SQL andmebaasiserveri seadistamise ning haldamise, SQL päringukeele tundmise ning objektorienteeritud programmeerimiskeele kirjutamise oskustele.

Olleks uuenduste paigaldamise edukalt läbinud, saab analüüsida uuenduse käigus tekkinud muudatuste mahtu ning selle põhjal määrata rakenduse testimisvajaduse mahtu.

Lisaks hõlmab töö analüüsiv osa endas MS Dynamics AX2012 tehnilise kirjandusega tutvumist.

1.4 Ülevaade tööst

Käesolev bakalaureusetöö koosneb üheksast peatükist.

Töö teises peatükis defineeritakse tarkvaratoode, räägitakse tarkvaratoote eri tüüpidest ning antakse põgus ülevaade tarkvaratoote elutsüklist. Kolmandas peatükis antakse

ülevaade tarkvarauuendustest, nende kategooriatest ning paigaldusmeetoditest. Töö neljas peatükk keskendub rakenduse Dynamics AX 2012 arhitektuurile, tuues välja süsteemi omadused, mille tundmine on rakenduses versiooniuuenduse läbiviimisel vajalik. Viiendas peatükis räägitakse rakenduse versiooniuuenduse CU13 paigaldusmeetoditest ning kirjeldatakse uuenduse käigus installeeritavaid komponente. Kuuendas peatükis kirjeldatakse versiooniuuenduse üldist paigaldusprotsessi ettevõttes, kus käesoleva töö käigus rakenduse uuendamine läbi viidi. Töö seitsmes peatükk keskendub versiooniuuenduse paigalduse läbiviimisele arenduskeskkonnas. Peatükis kirjeldatakse paigaldusprotsessi läbimiseks vajalikke tegevusi, tuuakse välja uuenduse paigalduse käigus tekkinud vead ning meetodid nende vigade lahendamiseks. Kaheksas peatükk räägib versiooniuuenduse paigaldamisest test- ning toodangukeskkonnas. Töö ühendas peatükis toimub versiooniuuenduse läbinud rakenduse testimismahule hinnangu andmine. Kümnes peatükk on käesoleva töö viimane ning sisaldab endas töö kokkuvõtet.

2 Tarkvaratoode

Principles of Marketing [1] kirjeldatakse toodet kui lahendust mistahes soovile või vajadusele mida on võimalik turul pakkuda ning turult soetada. Toodete puhul ei ole tegemist ainult füüsiliste objektidega nagu mobiiltelefonid, autod või arvutid. „Laiemalt kirjeldades, saab toodeteks nimetada ka teenuseid, üritusi, inimesi, asukohti, organisatsioone, ideid või miskit, mis on segu neist eelnevaist.“ [1, lk. 248]

Tarkvaratoodet võib defineerida kui toodet, mille peamine komponent on tarkvara. [2] See tähendab, et tarkvaratoote puhul on tegu turul pakutava tarkvara abil välja töötatud lahendusega mingisugusele konkreetsele probleemile või soovile.

Tarkvaratoode võib põhineda nii rätsep kui ka universaalsele tarkvaralisele lahendusele. Rätseplahendus kujutab endast ühes kindlas keskkonnas oleva spetsiifilise probleemi tarkvaralist lahendust. Näiteks võib rätseplahenduseks nimetada mingis tootmisettevõttes A loodud lao haldamise programmi, mis arvestab antud ettevõtte tootmisliini omapäradega ning ei sobiks seetõttu kasutamiseks teises sarnases tootmisettevõttes B. Universaallahendus kujutab endast mõne soovi või probleemi ning selle tarkvaralise lahenduse taandamist piisavalt abstraktsele tasemele, et sobitada see rohkem kui ühele selle konkreetse probleemiga silmitsi seisvale osapoolele. Universaallahenduste näidetena võib välja tuua sotsiaalmeedia suhtlusvahendid, operatsioonisüsteemid ja raamatupidamistarkvara.

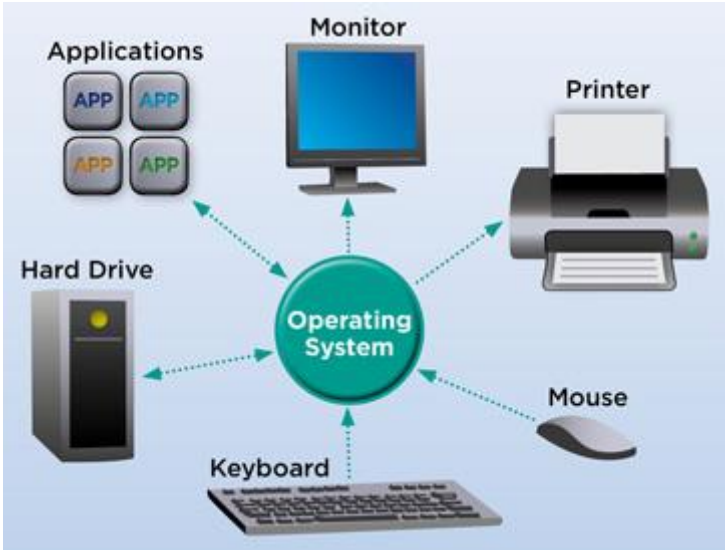
2.1 Tarkvaratoote tüübid

Tarkvaratooteid on võimalik liigitada üldise funktsionaalsuse alusel kolmeks: [2]

- *Operating system* ehk operatsioonisüsteem
- *Middleware* ehk vahevara süsteem
- *Application* ehk rakendus

2.1.1 Operatsioonisüsteemid

Operatsioonisüsteemid kujutavad endast arvutisüsteemidel töötavat tarkvara, mille eesmärgiks on hallata arvuti riistvara kontrollides selle ressursse ning pakkuda vajalikke teenuseid arvutisüsteemidel jooksva tarkvara toimimiseks. [3] Operatsioonisüsteemid tegelevad protsessi-, mälu- ja failihaldamisega. Lisaks on operatsioonisüsteemide ülesandeks hallata protsesside ja riistvaraseadmete vahelist I/O suhtlust ning olla tarkvaraplatvorm, mis võimaldab enda peal tarkvara jooksutada. Operatsioonisüsteemi roll on kujutatud joonisel 1.

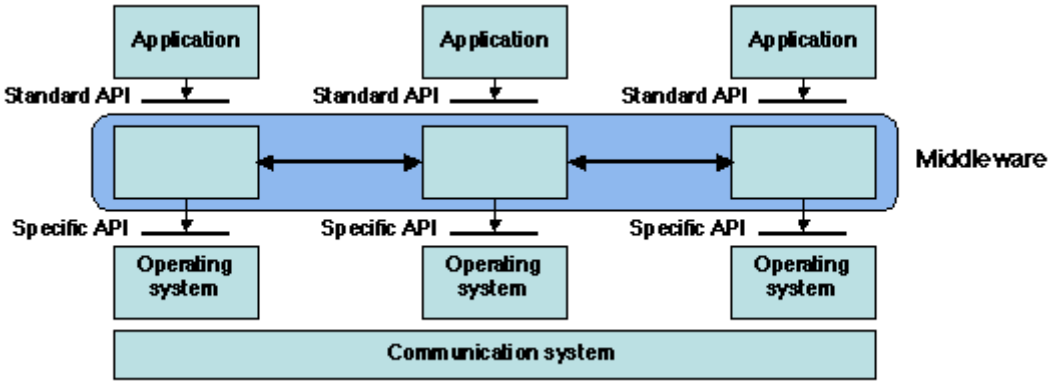


Joonis 1. Operatsioonisüsteemi diagramm [4]

Statcounter.com [5] määratleb levinumateks operatsioonisüsteemideks Microsoft Windowsi, Apple OSX-i ning Linuxi. Mobiilseadmete ning tahvelarvutite puhul on levinuimad operatsioonisüsteemid Android ning iOS. [6]

2.1.2 Vahevara süsteemid

Vahevara alla kuuluvad tarkvarakomponente ühendavad süsteemid mis asetsevad operatsioonisüsteemide ning rakenduste vahel. [2] Diagramm, mis kujutab vahevara seost rakenduste ning operatsioonisüsteemidega, on esitatud joonisel 2.



Joonis 2. Vahevara diagramm [7]

Vahevara on abstraheeriv kiht mille kasutamine lihtsustab erinevate tarkvarakomponentide omavahelist integreerimist. Suurt kasutegurit toob vahevara kasutamine olukordades, kus integreeritav rakendus koosneb mitmest eraldiseisvast kuid omavahel ühendatud alamsüsteemist. Vahevara kasutamine suurendab sõltumatust teistest süsteemidest ning vähendab keerukust olukordades, kus integratsioon toimub komplekse killustatud rakendusega. [7]

Mõningad näited vahevara tarkvaratoodetest:

- Apache Tomcat
- Microsoft Internet Information Services
- Oracle Fusion Middleware

2.1.3 Rakendused

Rakendused on lõpp-kasutajale suunatud tarkvaralahendused eesmärgiga lahendada mingit konkreetset probleemi või vajadust. [2] Rakendusi esineb paljudes erinevates vormides ning seega on nende üheselt määramine võimatu. Igal rakendusel on oma eesmärk mis võib olla seotud näiteks loometegevusega, meelelahutusega või äritegevusega. Rakenduste alla kuuluvad nii lihtsamad tekstitöötlusprogrammid kui ka keerukad majandustarkvaralised lahendused.

Rakendused jagunevad peamiselt kolme kategooriasse:

- Arvutitel jooksvad tarkvaraprogrammid (*Desktop applications*) – rakendused mis jooksevad lokaalselt lõppkasutaja arvutis. Näiteks MS Word ning Firefox.

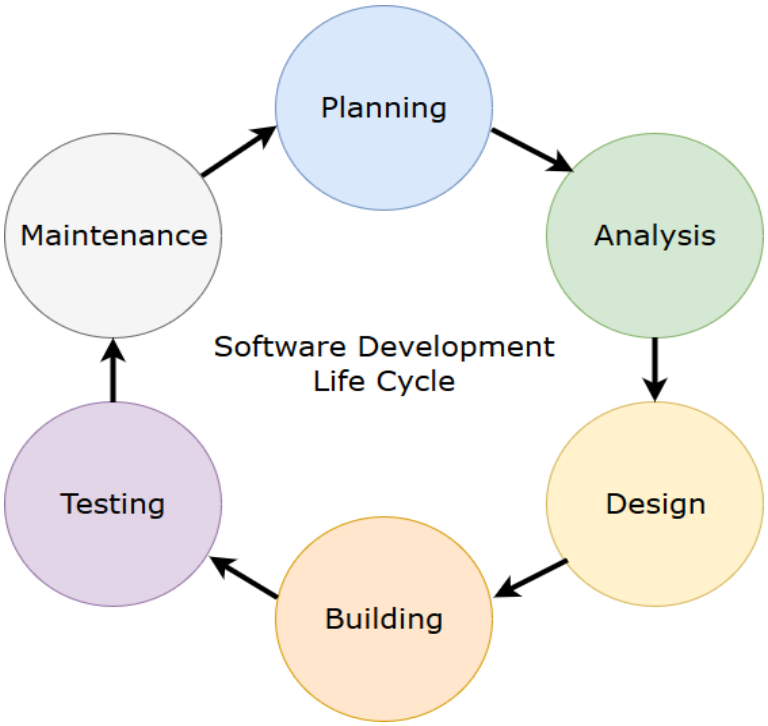
- Mobiilirakendused (*Mobile applications*) – Mobiiliseadmetel jooksvad rakendused. Näiteks mobiilimäng Angry Birds ning sotsiaalmeediarakendus Instagram.

- Veebirakendused (*Web applications*) – Siia alla kuuluvad rakendused, mis jooksevad täielikult veebibrauseris. Näiteks sotsiaalmeediarakendus Facebook.

2.2 Tarkvaratoote elutsükkel

Tarkavaratoote elutsükkel on kirjeldatav läbi tarkvaraarenduse elutsükli (inglise keeles *Software Development Life Cycle*). Tegemist on tarkvaraarenduses kasutatava metoodikaga mis kirjeldab tarkvara arenduse protsessi etappe ning nende etappide seost tervikuna kogu arendusprotsessi vältel. Protsessi eesmärk on tagada parem tarkvarakvaliteet ning vähendada tarkvaraarenduse ajalist ning rahalist ülekulu. Organisatsiooni International Organization for Standardization-i poolt on välja antud ka standard nimetusega ISO/IEC/IEEE 12207, mis tarkvara arenduse elutsüklit kirjeldab. [8]

Tarkvara arenduse elustsükkel koosneb etappidest, milleks on planeerimine, analüüs, disain, laheduse loomine, testimine ning hooldus. Etapid koos seoste ja järjestusega on välja toodud joonisel 3.



Joonis 3. Tarkvara arenduse elutsükli diagramm.

2.2.1 Planeerimine

Planeerimine on tarkvaratoote loomise esimene etapp, mille eesmärgiks on läbi viia strateegiline analüüs. Planeerimise käigus antakse ülevaade terviksüsteemist ning defineeritakse plaanitava tarkvara eesmärgid. Siin etapis pakutakse välja esmane lahendus ning vajadusel tuuakse välja alternatiivid. Samuti toimub selles etapis esmane kuluanalüüs ning lahenduse kasumlikkuse hindamine.

2.2.2 Analüüs

Analüüsi etapi eesmärk on viia läbi detailanalüüs mille käigus luuakse funktsionaalsed nõuded tulevasele süsteemile. Toimub faktide kogumine, lõppkasutajate intervjueerimine ning seotud süsteemide analüüs ning kirjeldamine, juhul kui need on olemas.

2.2.3 Disain

Disaini faasis luuakse disainidokument mis kirjeldab tulevast lahendust läbi protsessidiagrammide, ärireeglite, pseudokoodi ning andmemudelite.

2.2.4 Lahenduse loomine

Selles etapis toimub arendusvahendeid kasutades tarkvaralise lahenduse loomine.

2.2.5 Testimine

Testimise eesmärgiks on veenduda, et loodud rakendus töötaks ilma vigadeta ning täidaks analüüsi etapis kirjeldatud funktsionaalseid nõudeid.

2.2.6 Hooldus

Hoolduse etapp algab ajahetkest mil valmis ning testitud lahendus on jõudnud lõppkasutaja kätte ametlikuks kasutamiseks. Hoolduse käigus pakutakse tuge hoidmaks loodud lahendust töökorras, parandatakse vigu mis testimisel välja ei tulnud ning viiakse läbi rakenduse versiooniuuendusi, et juuruta uut või muuta olemasolevat funktsionaalsust vastavalt tellija tagasisidele.

3 Tarkvarauuendused

Tarkvarauuendused kujutavad endast muudatusi tarkvaraprogrammis, mille eesmärk on rakenduse täiustamine ning seal leiduvate vigade parandamine. Tarkvarauuenduste läbiviimine on tarkvara elutsükli üks osa, mis toimub peamiselt peatükis 2.2.6 kirjeldatud hoolduse faasis. Teatud juhtudel on tarkvarauuenduste läbiviimist vaja teha ka rakenduse installatsioonifaasis, kasutades tehnoloogiat nimega *slipstreaming* millest on täpsemalt juttu peatükis 3.1.3.

Kuigi tarkvarauuendused on väga rakendusespetsiifilised ja seetõttu sisult ning vormilt erinevad, on tarkvaraarenduse valdkonnas välja kujunenud metoodika uuenduste kategoriseerimiseks. Mõningad näited kategooriatest:

- *Hotfix* ehk kuumlapp – Kiireloomuline uuendus eesmärgiga parandada mingit konkreetset rakenduses esinevad funktsionaalset viga.

- *Product temporary fix* ehk ajutine parandus – Ajutine parandus tarkvaras esinevale veale.

- *Security update* ehk turvauuendus – Uuendus eesmärgiga parandada rakenduses leiduvaid vigu, mille olemasolu võib ohtu seada süsteemi turvalisuse.

- *Point release* ehk väiksemahuline uuendus – Väiksemahuline rakenduse versiooniuuendus, mille peamiseks eesmärgiks on rakenduses leiduvate vigade parandamine. Ei sisalda tavaliselt väga palju uut funktsionaalsust ega suuremahulisi muudatusi olemasolevas tarkvaras.

- *Service pack* ehk remondipakett – Kumulatiivne tarkvarauuenduste paigalduspakett, kuhu on kokku kogutud pikema aja vältel rakendusse uuenduste näol tehtud muudatused, lisafunktsionaalsused, *hotfix*-id, turvauuendused ja veaparandused.

- *Major release* ehk suuremahuline uuendus – Tegemist on suuremahulise tarkvara versiooniuuendusega. Sisaldab tavaliselt suures mahus uut funktsionaalsust ja veaparandusi.

- *Unofficial release* ehk mitteametlik uuendus – Tarkvarauuendused, mille autoriks ei ole konkreetse tarkvara ametlik omanikfirma vaid mingi kolmas osapool. Enamasti eesmärgiga rakenduses leiduvate vigade parandamiseks juhtudel, kui omanikfirma või isik on lõpetanud tarkvara edasise arenduse ning toe pakkumise.

3.1 Tarkvarauuenduste paigaldusmeetodid

Järgnevalt on välja toodud mõned viisid tarkvarauuenduste paigaldamiseks.

3.1.1 Update managers

Update manager on programm rakenduste uuenduste haldamiseks. Programm teavitab kasutajat uuenduste ilmnemisel ning kui programm on vastavalt seadistatud, viib läbi ka uuenduse automaatse paigaldamise. Seda kasutatakse näiteks Windows operatsioonisüsteemis uuenduste läbiviimiseks.

3.1.2 Hot patching

Hot patching on metoodika, mis võimaldab uuendusi rakenduses läbi viia ilma süsteemi seiskamata. See muudab uuenduste paigaldamise mugavamaks ning vähendab süsteemi maasolekuaega. [9]

3.1.3 Slipstreaming

Slipstreaming kujutab endast protsessi mille käigus integreeritakse tarkvarale ilmunud uuendused tarkvara paigaldusprogrammi. See võimaldab tarkvara esmakordsel paigaldamisel installeerida rakendus koos sellele juba välja tulnud uuendustega. [10] *Slipstreaming* metoodikat on võimalik kasutada näiteks Microsoft Dynamics AX 2012 paigaldamisel. [11]

3.1.4 Manuaalne paigaldus

Manuaalne paigaldus kujutab endast tarkvarauuenduse käsitsi paigaldust süsteemiadministraatori või kasutaja poolt. Firmaomase ja suletud koodiga rakenduste käsitsi uuendamine käib peamiselt läbi käitusprogrammide, mida väljastab firma või isik

kellele programm kuulub. Käitusprogrammide abil rakendatakse programmile muudatusi ning parandusi läbi rakenduse binaarfailide muutmise. Avatud koodiga tarkvara puhul võivad uuendused esineda koodiparandustena otse rakenduse lähtekoodis, mida on võimalik alla laadida. Sellisel juhul rakenduvad muudatused parandatud lähtekoodiga programmi kompileerimisel. [12]

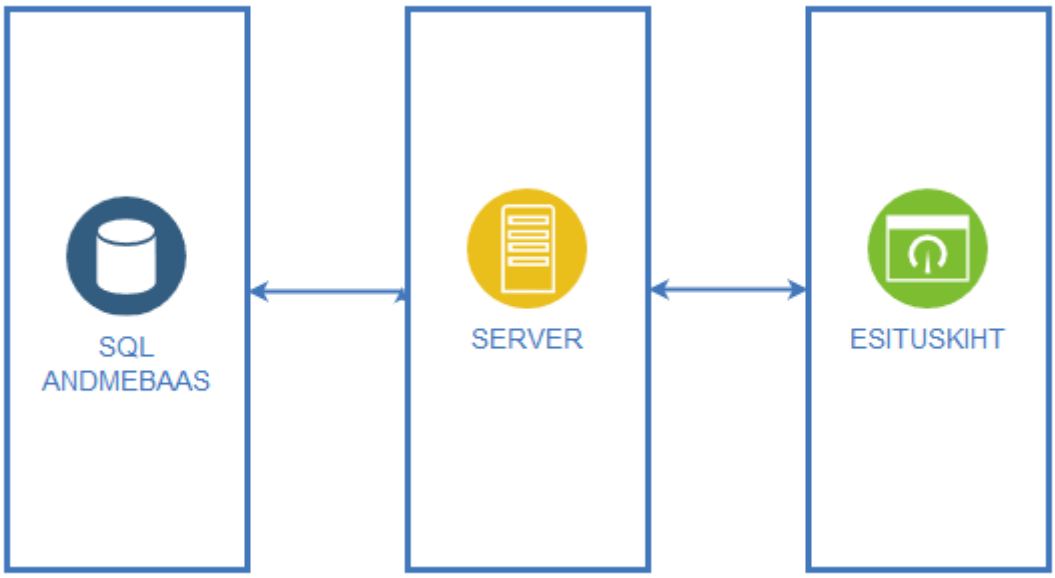
4 Ülevaade rakendusest Microsoft Dynamics AX 2012

Microsoft Dynamics Axapta 2012, lühendatult AX2012, on Microsofti poolt välja töötatud tarkvaratoode. Tegemist on keskmistele ja suurtele ettevõtetele suunatud majandus- ja juhtimistarkvaraga. Süsteem pakub moodulitena täisfunktsionaalset tarkvara, mis katab pea kõik ettevõtte äriprotsessid. Näiteks sisaldab rakendus mooduleid raamatupidamise, projektihaldamise, laologistika ja tootmise haldamise kohta.

Järgnevates alampeatükkides antakse ülevaade rakenduse arhitektuurist ning sellega seotud tehnilistest omadustest, mille tundmine on vajalik süsteemi versiooniuuenduse läbiviimiseks.

4.1 Microsoft Dynamics AX 2012 rakenduse arhitektuur

Dynamics AX 2012 rakendus koosneb arhitektuurselt kolmest kihist, milleks on andmebaasikiht, rakendusserver ning esituskiht. Rakenduse arhitektuuridiagramm on esitatud joonisel 4.



Joonis 4. Rakenduse arhitektuuridiagramm.

Põhjalikum joonis rakenduse arhitektuurist on välja toodud töö lisas 1.

4.1.1 Andmebaasikiht

Andmebaasikihi ainus komponent on SQL andmebaas. Rakenduse süsteeminõuete kohaselt toetab rakendus andmebaasiservereid Microsoft SQL Server 2014 ning Microsoft SQL server 2016. [13]

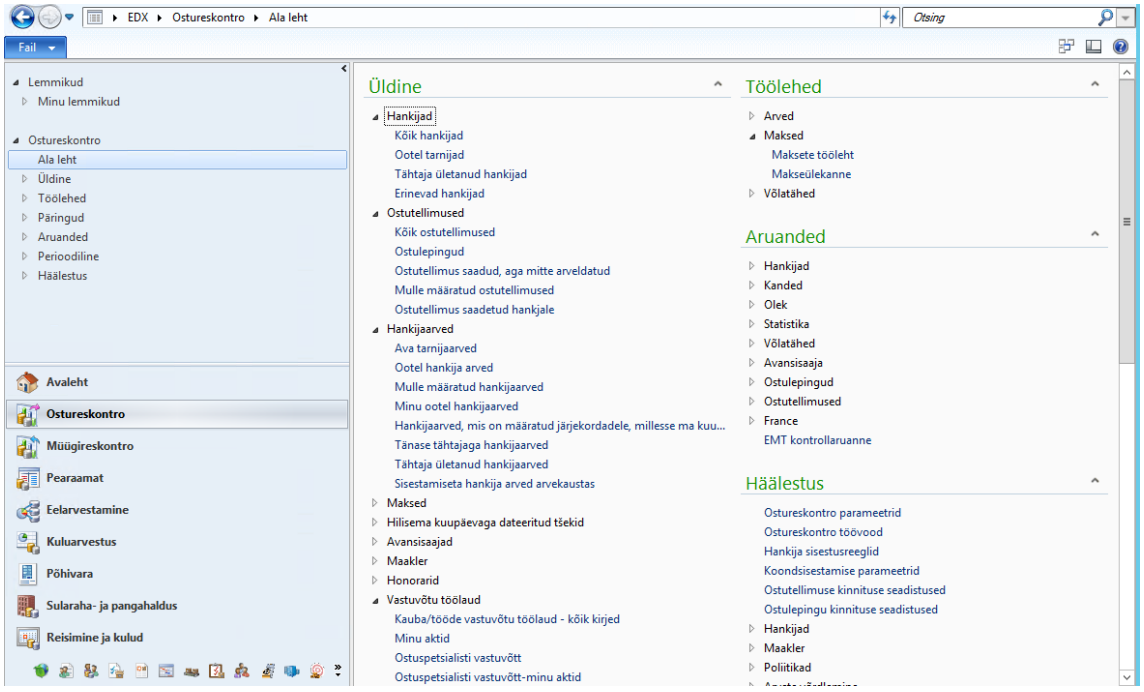
Tegemist on relatsioonandmebaasiga kus hoiustatakse kogu rakenduses kasutatavaid ning rakenduse enda metaandmeid.

4.1.2 Rakendusserver

Rakendusserveri peamiseks ülesandeks on süsteemi äriloogika jooksutamine ja suhtlus andmebaasiga, sealhulgas andmete töötlemine, salvestamine ja esituskihti andmete edastamine ning sealt nende lugemine. Lisaks haldab rakendusserver ka süsteemi kasutajate sessioone ning tegeleb kasutajate autentimise ja kasutajaõigustega.

4.1.3 Esituskiht

Esituskiht koosneb lõppkasutaja arvutis paiknevast rakenduse kliendist ja veebileheküljel asuvast portaalist, kus on kasutajal võimalik erinevate vormide abil endale äriprotsessis olulisi andmeid ja dokumente hallata ning neid rakenduse poolt pakutava äriloogika alusel töödelda. Rakenduse klient on kujutatud joonisel 5.



Joonis 5. Rakenduse kliendis olev ostureskontro vorm.

4.2 Rakenduse kihid

Dynamics AX 2012-sse on juurutatud kontseptsioon rakenduse elementide haldamiseks eesmärgiga lihtsustada süsteemi modifitseerimist ning muudatuste haldamist. [14]

AX 2012 rakendus koosneb elementidest nagu klassid, tabelid, andmetüübid, vormid jne. Süsteemis on võimalik igat rakenduse elementi, näiteks klassi *MyClass* meetodit *myMethod*, defineerida mitu korda. Sellist käitumist võimaldab kihtide kasutamine, mis kujutab endast süsteemi rakenduse elementide haldamiseks. Kihti võib defineerida kui hierarhias asetsevat asukohta, kus hoiustatakse mudeleid (vt. peatükk 4.3), mis sisaldavad rakenduse objektide definitsioone. Tabelis 1 on hierarhilises järjestuses välja toodud Dynamics AX-s olevad kihid koos kirjeldusega. [14] [15]

Tabel 1. Dynamics AX 2012 kihid.

Lühendatud nimetus	**Kirjeldus**
USP	*Patch layer*. ehk kiht, mis on mõeldus USR kihti paigaldavate uuenduste jaoks.
USR	*User Layer*. Kiht, mis on mõeldud kasutajapõhiste muudatuste jaoks.
CUP	*Patch layer*. ehk kiht, mis on mõeldus USR kihti paigaldavate uuenduste jaoks.
CUS	*Customer Layer*. Kiht mis on mõeldud kliendipõhiste lahenduste jaoks.
VAP	*Patch layer*. ehk kiht, mis on mõeldus VAR kihti paigaldavate uuenduste jaoks.
VAR	*Value Added Reseller Layer*. Kiht Microsofti partnerite poolt loodud rohkem kui ühele kliendile mõeldud lahenduste jaoks.
ISP	*Patch layer*. ehk kiht, mis on mõeldus ISV kihti paigaldavate uuenduste jaoks.

ISV	*Independant Software Vendor Layer*. Kiht, mis on Mõeldud Microsofti partnerite poolt loodud lahenduste jaoks.
SLP	*Patch layer.* ehk kiht, mis on mõeldus SLN kihti paigaldavate uuenduste jaoks.
SLN	*Solution Layer*. See kiht on Microsofti partnerite poolt loodud vertikaallahenduste jaoks.
FFP	*Patch layer.* ehk kiht, mis on mõeldus FPK kihti paigaldavate uuenduste jaoks.
FPK	*Feature pack layer*. Muudetav ainult Microsofti poolt.
SYP	*Patch layer.* ehk kiht, mis on mõeldus SYS kihti paigaldavate uuenduste jaoks.
SYS	*System layer*. Selles kihis asetsevad rakenduse standardobjektid.

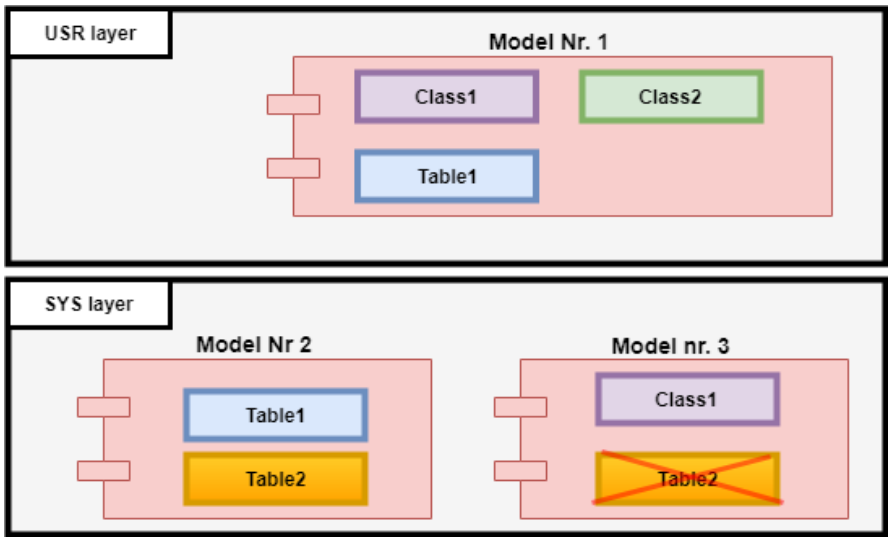
Kihid ei ole omavahel seotud. Süsteemi jooksutamisel kasutatakse kõige kõrgemas kihis asuvat objekti definitsiooni. Näiteks kui klassi *MyClass* meetod *myMethod* on defineeritud kihis SYS ning kihis CUS, kasutab rakendus käitusfaasis hierarhiliselt kõrgemal CUS kihis defineeritud klassiobjekti.

4.3 Rakenduse mudelid

Dynamics AX 2012 kontekstis nimetatakse mudeliks rakenduse objektide metaandmeid sisaldavat loogilist kogumit, kus hoitakse rakenduse elementide (klassid, vormid, tabelid jne.) definitsioone. Mudeleid hoitakse rakenduse andmebaasiserveri *modelstore* nimelises andmebaasis.

Mudelite hulk ei ole piiratud. Mudeli loomisel paigutatakse iga mudel peatükis 4.2 kirjeldatud rakenduses olevasse kihti. Tulenevalt rakenduse kihtide kohta kehtivast piirangust mis keelab rakenduse elemendi defineerimist kihis rohkem kui üks kord, ei tohi ühes kihis korraga paikneda mudelid, mis sisaldavad sama rakenduse elementi. [14]

Näiteks kui luua kaks mudelit M1 ning M2, mis sisaldavad rakenduse *myClass* elemendi definitsiooni, peavad need mudelid asuma erinevates rakenduse kihtides. Seos rakenduse elementide, mudelite ja kihtide vahel on kujutatud joonisel 6.



Joonis 6. Kihtide ning mudelite seos.

Mudelite kasutamine võimaldab lihtsalt rakenduses tehtud arendusi erinevate keskkondade vahel liigutada. Dynamics AX 2012 sisaldab funktsionaalsust, mis võimaldab kõik *modelstore* andmebaasis olevad mudelid binaarfailina eksportida ning importida. Kui teha arendus keskkonnas A ning sealne *modelstore* binaarfaili eksportida, siis pärast selle faili importimisel keskkonda B on tehtud arendus olemas ka seal.

4.4 Dynamics AX 2012 rakenduse arendamine

Microsoft Dynamics AX 2012 sisaldab moodulitena täisfunktsionaalset äriprotsesse katvat tarkvara. Lisaks olemasolevale funktsionaalsusele on võimalik rakendust vastavalt ettevõtte kohaste soovide järgi modifitseerida. Rakenduse klient ning seal taga paiknev äriloogika on kirjutatud X++ programmeerimiskeeles. Tegemist on Microsofti poolt välja töötatud Dynamics Axapta 2012 rakenduse arendamiseks mõeldud keelega.

X++ on objektorienteeritud keel, mis toetab objektide abstraheerimist, polümorfismi ning kapseldamist. X++ keeles Dynamics AX programmeerimisel on võimalik kasutada selliseid võtmesõnu nagu *client*, *server* ja *changecompany*, mis lihtsustavad klient/server ERP süsteemi funktsionaalsuse programmeerimist. [14] Lisaks sisaldab antud keel

raamistikku, mis mugavdab rakenduse koodis SQL päringute tegemist. Näiteks on võtmesõna *while select* abil võimalik pärida andmebaasi tabeliobjektist kirjeid ükshaaval ning kasutades võtmesõna *forUpdate* neid muuta ilma standardset SQL päringut kirjutamata. Koodinäide antud funktsionaalsuse kasutamisest on esitatud joonisel 7.

```
static void Job86(Args _args)
{
    SalesTable  salesTable;
    utcDateTime dateTimeFrom    = DateTimeUtil::addDays(DateTimeUtil::getSystemDateTime(), -1);
    utcDateTime dateTimeTo      = DateTimeUtil::getSystemDateTime();

    ttsBegin;

    while select forUpdate salesTable
        where salesTable.createdDateTime > dateTimeFrom
        && salesTable.createdDateTime < dateTimeTo
    {
        salesTable.SalesStatus = SalesStatus::Invoiced;
        salesTable.update();
    }

    ttsCommit;
}
```

Joonis 7. X++ koodinäide

Dynamics AX rakendus sisaldab endas MorphX nimelist keskkonda, kus toimub rakenduse arendamine. Samuti on võimalik süsteemi arendada Visual Studio arenduskeskkonnas.

4.4.1 MorphX

MorphX on peamine Dynamics AX 2012 arendamiseks mõeldud arenduskeskkond, mis võimaldab muuta ning luua uusi rakenduse objekte ning programmikoodi. MorphX arenduskeskkonna peamised komponendid ning nende kirjeldused on välja toodud tabelis 2.

Tabel 2. MorphX arenduskeskkonna komponentide loetelu.

Arenduskeskkonna komponent	**Komponendi kirjeldus**
Rakenduse objektipuu	Rakenduse objektipuu sisaldab kõiki rakenduses leiduvaid objekte.
Projektide haldustöövahend	Võimaldab grupeerida objekte ühe projekti alla.

Elementide omaduste haldustöövahend	Võimaldab muuta elementide omadusi. Omadused on esindatud võti-väärtus paaridena.
X++ koodiredaktor	Koodiredaktor X++ koodi kirjutamiseks.
Sildiredaktor	Rakenduse elementide sildihalduse töövahend.
Kompilaator	Kompileerib X++ koodi.
Silur	Võimaldab testida ning jälgida rakenduses kasutatavat koodi.
Hea tava kontrolltööriist	Võimaldab tuvastada X++ koodis ning rakenduse elementides esinevaid heade tavade rikkumisi.
Tabelibrauser	Tööriist mis võimaldab otse rakenduse tabelielemendis vaadelda tabelis leiduvad andmeid ning neid modifitseerida.
Versioonikontroll	Rakenduse elementide versioonihalduse töövahend.

4.4.2 Visual Studio

Visual Studio on Microsofti poolt pakutav programmeerimiskeskkond. Visual Studios on võimalik arendada .NET raamistikul põhinevaid liideseid AX 2012 rakenduse kliendi, serveri ning teenustega. Vistual Studios toimub ka rakenduse veebiportaali ning SSRS raamistikul põhinevate raportite arendus. [16]

5 AX 2012 Cumulative Update 13 versiooniuuendus

Cumulative Upgrade 13 (edaspidi CU13) on septembris 2017 välja antud ning töö kirjutamise hetkel uusim Dynamics AX 2012 rakenduse versiooniuuendus. [17] Tegemist on peatükis 3 kirjeldatud *service pack* tüüpi, ehk kumulatiivse uuenduste paketiga mis sisaldab endas kõiki rakendusele saadaolevaid uuendusi.

Versiooniuuenduse paigaldamiseks on kaks meetodit: käsitsi paigaldus ning *slipstreaming*.

Slipstreaming on võimalik vaid rakenduse esmasel paigaldamisel. Sellisel juhul integreeritakse kõik CU13-s sisalduvad uuendused rakenduse paigaldamise käitusprogrammi, mis koos baasrakendusega paigaldatakse. Tulemuseks on rakenduse värske installatsioon, mis on viidud uusimale versioonile.

Eelnevalt paigaldatud rakenduse korral tuleb CU13 versiooniuuendus süsteemile peale panna käsitsi. Sellisel juhul toimub paigaldus läbi Microsofti poolt pakutava rakenduse versiooniuuenduse käitusprogrammi. Programm sisaldab endas kahte teineteisest sõltumatut paigalduskomponenti:

- *Binary* – Komponent, mille kaudu toimub rakenduse *kernel*-i uuendamine. Antud komponendi paigaldamisel muudetakse rakenduse kompileeritud osa, mille programmikood on suletud lähtekoodiga.

- *Application* – Komponent, mille kaudu toimub *modelstore* andmebaasis olevate rakenduse elementide uuendamine. Uuendatakse süsteemi osa, mis sisaldab ärilist funktsionaalsust (tabelid, klassid, vormid jne) ning mis on rakenduse arendaja jaoks ligipääsetav.

Tehniliselt on võimalik mõlemad komponendid korraga paigaldada, kuid riskide maandamiseks on soovitavad komponentide paigaldus läbi viia eraldi. Selline tegevuskava on küll ajakulukam, kuid vähendab võimalike veaolukordi paigaldusprotseduuri ajal.

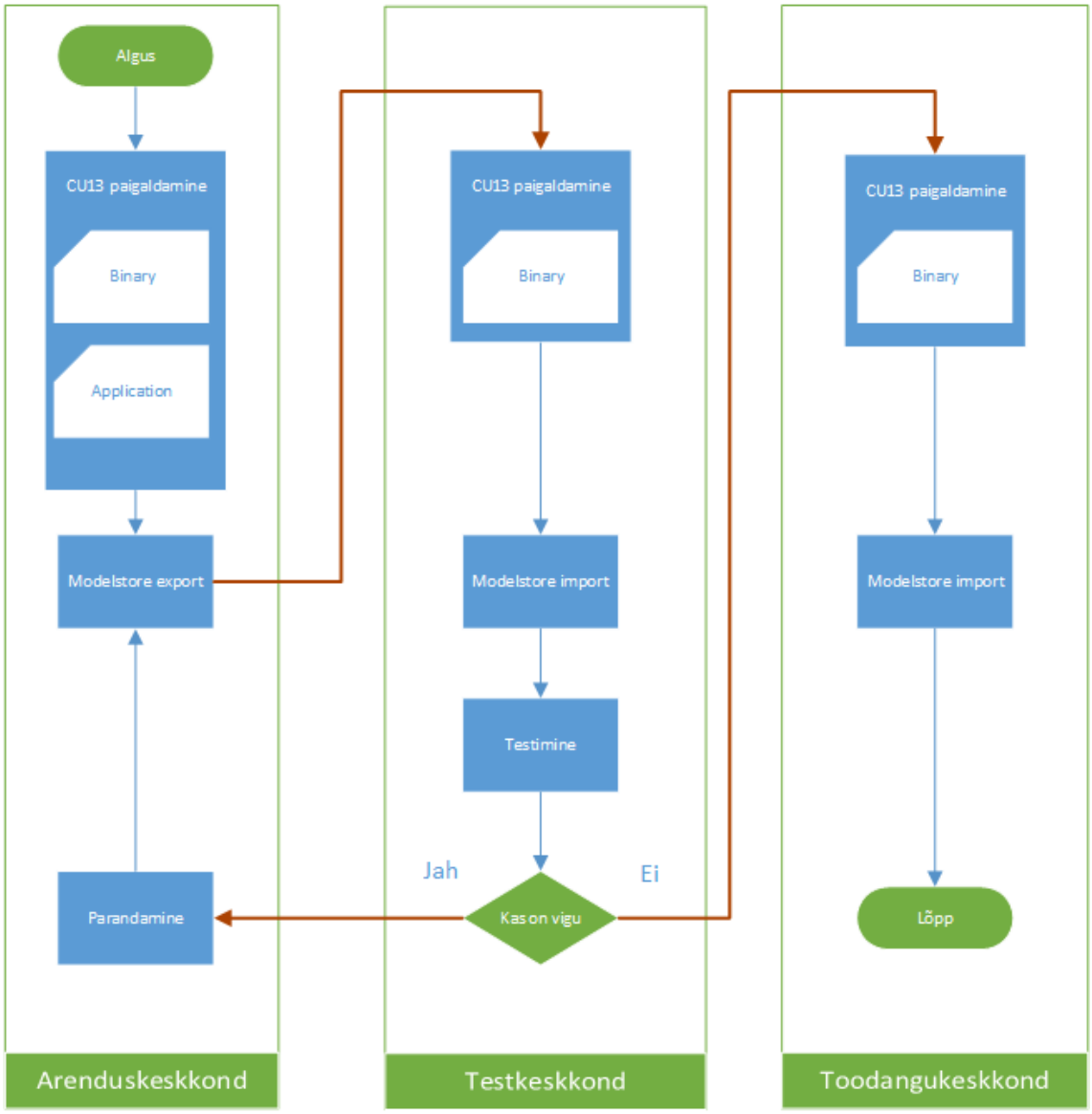
6 Versiooniuuenduse paigaldusprotsess

Käesoleva bakalaureusetöö käigus viidi läbi käsitsi Dynamics AX 2012 rakenduse versiooniuuenduse CU13 paigaldus. Rakenduse uuendus tuli läbi viia kolmes keskkonnas: arenduskeskkonnas, testkeskkonnas ning toodangukeskkonnas. Üldine paigaldusprotsess keskkondades on esitatud joonisel 8.

Esmane uuenduse paigaldus viidi läbi rakenduse arenduskeskkonnas eesmärgiga luua ning valideerida uuenduse paigaldusprotsessi, tuvastada paigalduse käigus tekkivaid vigu ning viia läbi leitud vigade parandamine. Arenduskeskkonnas teostati versiooniuuenduse *binary* ja *application* komponentide paigaldus. Täpsemalt räägib arenduskeskkonda uuenduse paigaldamisest töö peatükk 7.

Arenduskeskkonnas uuenduse teostamise järel paigaldati versiooniuuendus testkeskkonda. Erinevalt arenduskeskkonnas toimunud paigaldusest, toimus testkeskkonnas vaid uuenduse *binary* komponendi installatsioon. *Application* komponendi paigaldus toimus läbi *modelstore* andmebaasi importimise. Testimiskeskkonnas toimus ka rakenduse versiooniuuenduse järgne testimine. Vigade leidmisel toimus paranduste tegemine arenduskeskkonnas, kust need *modelstore* eksportimise ning importimise kaudu testkeskkonda jõudsid.

Viimase tegevusena toimus rakenduse uuendamine toodangukeskkonnas. Uuenduse paigaldamine toimus identselt testkeskkonnas läbi viidud paigaldusega: *binary* komponent installeeriti ning *application* komponendi paigaldus toimus *modelstore* importimise kaudu. Uuenduse läbiviimisele test- ning toodangukeskkonnas keskendutakse antud töö peatükis 8.



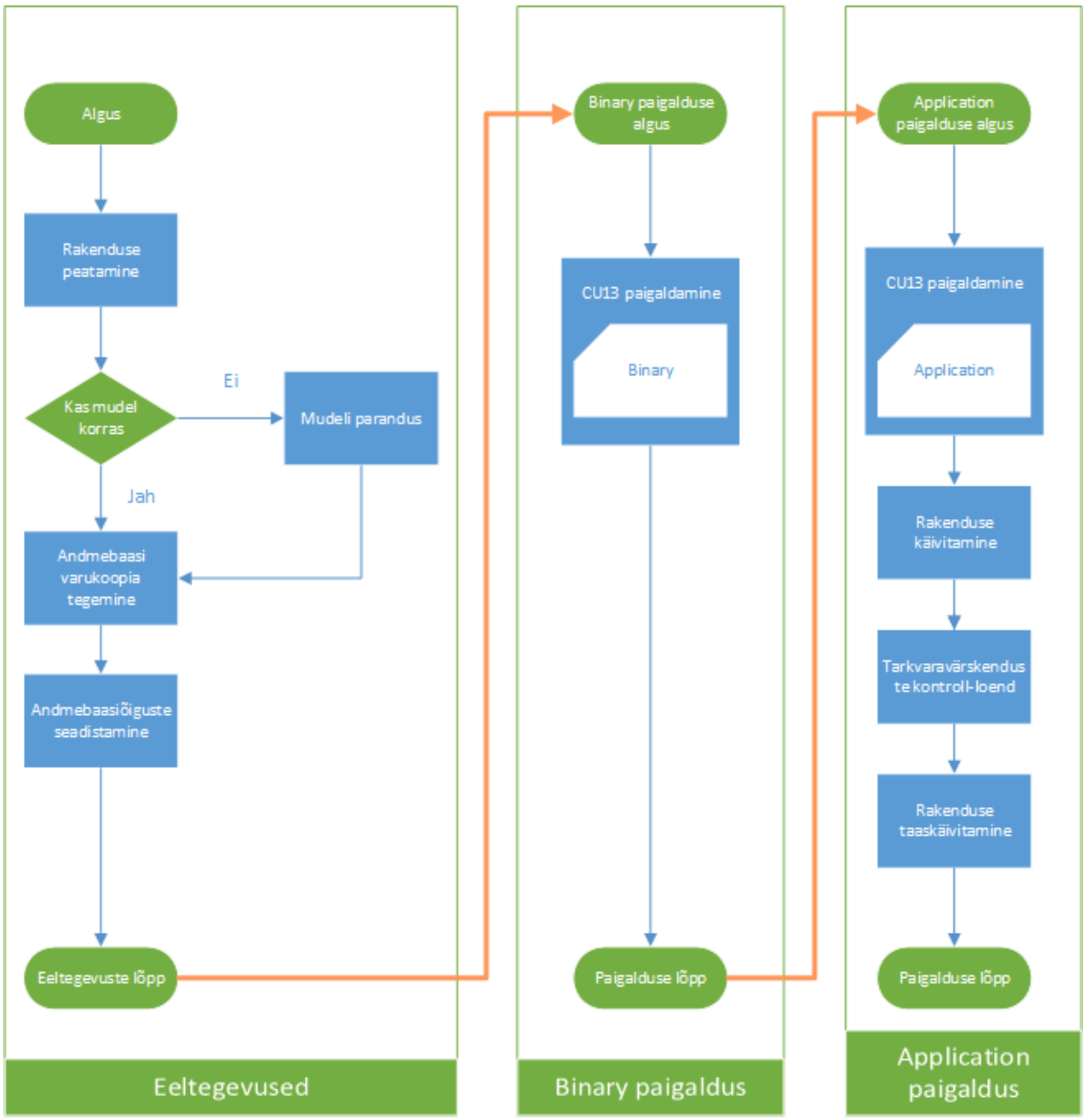
Joonis 8. Versiooniuuenduse protsess.

6.1 Code freeze

Kogu versiooniuuenduse läbiviimise protsessi vältel toimus *code freeze* ehk koodi külmutamine. Koodi külmutamine kujutab endast rakenduse tavapärase arendustsükliga seotud arendustegevuse peatamist. [18] *Code freeze* eesmärgiks oli tagada, et ei esineks olukordi kus versiooniuuenduse järgselt *modelstore* importimisel test-ja toodangukeskkonda kirjutatakse üle mõni paralleelselt uuendusprotsessiga tehtud versiooniuuendust mitte tegeva arendaja poolt tehtud arendus.

7 CU13 paigaldus arenduskeskkonnas

Cumulative Update 13 esmane paigaldus viiakse läbi arenduskeskkonnas. Paigaldusprotsess koosneb kolmest suuremast etapist: eeltegevuste läbiviimine, uuenduse *binary* komponendi paigaldamine ning uuenduse *application* komponendi paigaldamine. On oluline, et kõik paigaldusprotsessis olulised tegevused saaksid täidetud õiges järjekorras. Versiooniuuenduse paigaldusprotsess on täpsemalt esitatud joonisel 9.



Joonis 9. Versiooniuuenduse paigaldusprotsessi etapid arenduskeskkonnas.

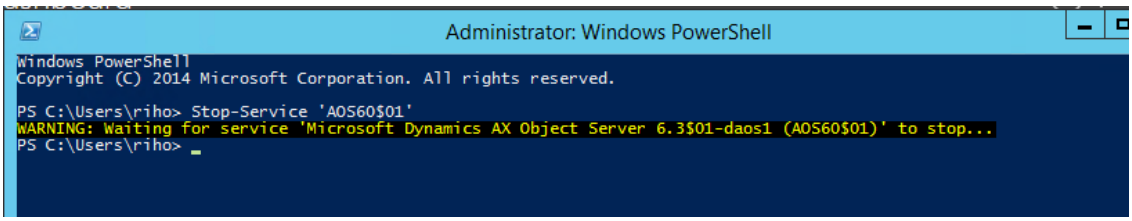
Järgnevalt kirjeldatakse joonisel 9 kujutatud paigalduse etappe koos neis sisalduvate tegevustega täpsemalt.

7.1 Paigalduse eeltegevused

Eeltegevuste läbiviimine on versiooniuuenduse paigaldamisel esimene faas, mille eesmärgiks on uuenduse läbimiseks vajalike ettevalmistuste tegemine. Etapp koosneb neljast tegevusest, mille läbimine on eelduseks järgmisesse etappi liikumisele.

7.1.1 Rakenduse peatamine

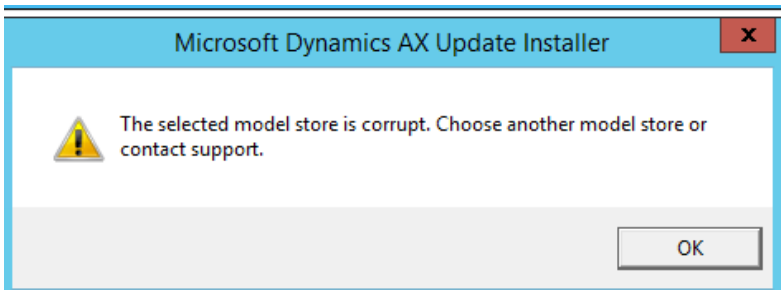
Esmase tegevusena tuleb rakenduse töö peatada. Dynamics AX 2012 kontekstis tähendab see rakendusserveri peatamist, mida on võimalik teha kasutades Powershell käsurealiideses käsklust „*Stop-Service* **<Name>**“. Kohatäide <**Name**> tuleb asendada rakendusserveri nimega. Powershell käsurea abiga rakenduse peatamist kujutab joonis 10.



Joonis 10. Dynamics AX 2012 aplikatsiooniserveri peatamine.

7.1.2 Mudeli parandamine

Mudeli parandamine on valikuline tegevus, mille täitmise vajadus sõltub sellest, kas rakenduse *modelstore* andmebaas sisaldab ebakõlasid või mitte. Nimelt kontrollib versiooniuuenduse paigaldusprogramm rakenduses olevaid mudeleid, et välja selgitada milliseid uuendusi on vaja installeerida ning millised on juba rakendusele peale pandud. Kui programm leiab mudelis ebakõla, teavitab paigaldusprogramm kasutajat veateatega ning ei võimalda versiooniuuendusega jätkata. Kasutajale kuvatav veateade on esitatud joonisel 11.



Joonis 11. Rikutud *modelstore* puhul kuvatav veateade.

Mis täpsemalt kvalifitseerub ebakõlaks ning mis on selle tekkepõhjus ei ole teada. Microsoft on sellise paigaldusprogrammi käitumise kvalifitseerinud *bug*-iks, millele hetkel lahendust ei ole. [19] Eeldatavasti on kirjeldatud olukord harva esinev ning seetõttu ei oma Microsofti silmis suurt prioriteeti. Antud olukord esines aga selle töö raames läbi viidud CU13 versiooniuuendus käigus.

Rikutud andmebaasi korral on versiooniuuendusega jätkamiseks vajalik *modelstore* andmebaasis leiduvate mudelite parandus. Sisuliselt kujutab see endast uue *modelstore* andmebaasi loomist ning antud tegevus koosneb järgnevatest sammudest:

1. Teha varukoopia rakenduse mudelilao (*modelstore*) ning äriandmebaasist. Seejärel tuleb mõlemad andmebaasid kustutada.

2. Kasutades Dynamics AX2012 paigaldusprogrammi installeerida uus *modelstore* andmebaas, uus äriandmebaas ning seadistada rakendusserver neid andmebaase kasutama. Tulemuseks tühja andmebaasiga ning modifitseerimata standardrakendus.

3. Importida loendi punktis 2 paigaldatud uude *modelstore* andmebaasi kõik lisaarendusi ning muudatusi sisaldavad mudelid. Tulemuseks on rakendus, mis on funktsionaalsuselt samaväärne enne mudeli parandusprotsessi alustatud rakendusega, kuid tühja äriandmebaasiga.

4. Kompileerida rakendus ning sünkroniseerida rakenduse andmebaas.

5. Kustutada loendi punktis 2 paigaldatud tühi äriandmebaas ning taastada see punktis 1 tehtud äriandmebaasi varukoopiast. Tulemuseks on uue *modelstore* andmebaasi kuid vana äriandmebaasiga rakendus.

6. Taastada loendi punktis 1 tehtud *modelstore* andmebaas. Olgu see taastatud andmebaas M1 ning punktis 2 loodud uus *modelstore* andmebaas M2. Seejärel käivitada SQL skript (vt. Joonis 17. Elementide SQL parandusskript.), mis uuendab andmebaasis M2 olevate rakenduste elementide id-d võrdseks andmebaasis M1 olevate elementide identifikaatoritega.

7. Kompileerida rakendus ning sünkroniseerida rakenduse andmebaas.

Modelstore andmebaasi uuesti loomise tulemusel versiooniuuenduse paigaldusprogramm enam veateadet ei kuva ning uuendusprotsessiga on võimalik edasi minna.

7.1.3 Andmebaasi varukoopiate tegemine

Riskide maandamiseks tuleb enne versiooniuuenduse paigaldust teha varukoopia rakenduse äriandmebaasist ning *modelstore* andmebaasist. Juhul kui rakenduse uuenduse käigus või selle tulemusel ilmnevad kriitilised vead, on varukoopiate abil võimalik rakenduse versiooniuuenduse-eelne seis taastada.

7.1.4 Andmebaasi õiguste ning rollide määramine

Versiooniuuendust paigaldav kasutaja peab omama teatud rolle ning õigusi andmebaasiserveris, millega rakendus on ühendatud. Uuenduse installeerimiseks on vajalik, et: [11]

- Kasutajale peab rakenduse SQL Serveris olema seadistatud **securityadmin** roll.
- Kasutajale peab SQL Serveri *modelstore* andmebaasis olema seadistatud **db_owner** roll.

7.2 Binaarkomponendi paigaldus

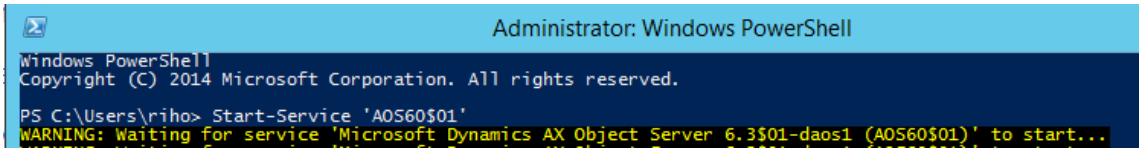
Eeltegevustele järgneb uuenduse binaarkomponendi paigaldamine. Selleks tuleb käivitada Microsofti poolt pakutab CU13 paigalduse käitusprogramm ning valida installeeritavaks komponendiks *binary*. Selle tulemusel käivitub tavaline Windows Installeri paigaldusprotseduur, mille lõppedes on *binary* komponent paigaldatud.

7.3 Rakenduskomponendi paigaldus

Versiooniuuenduse rakenduskomponendi paigaldus käib sarnaselt peatükis 7.2 kirjeldatud binaarkomponendi paigaldamisele. Uuenduse käitusprogrammi käivitamisel tuleb paigaldatavaks komponendiks valida *application* ning installatsiooni lõppedes on rakenduse uuendused paigaldatud.

7.3.1 Rakenduse käivitamine ja *modelstore* andmebaasi lähtestamine

Paigaldusprotsessile järgneb rakenduse käivitamine. Rakendusserveri käivitamiseks tuleb Powershell käsurealiideses käivitada käsklus „*Stop-Service* <**Name**>“. Kohatäide <**Name**> tuleb asendada rakendusserveri nimega. Rakendusserveri käivitamist käsurea abil on kujutatud joonisel 12.

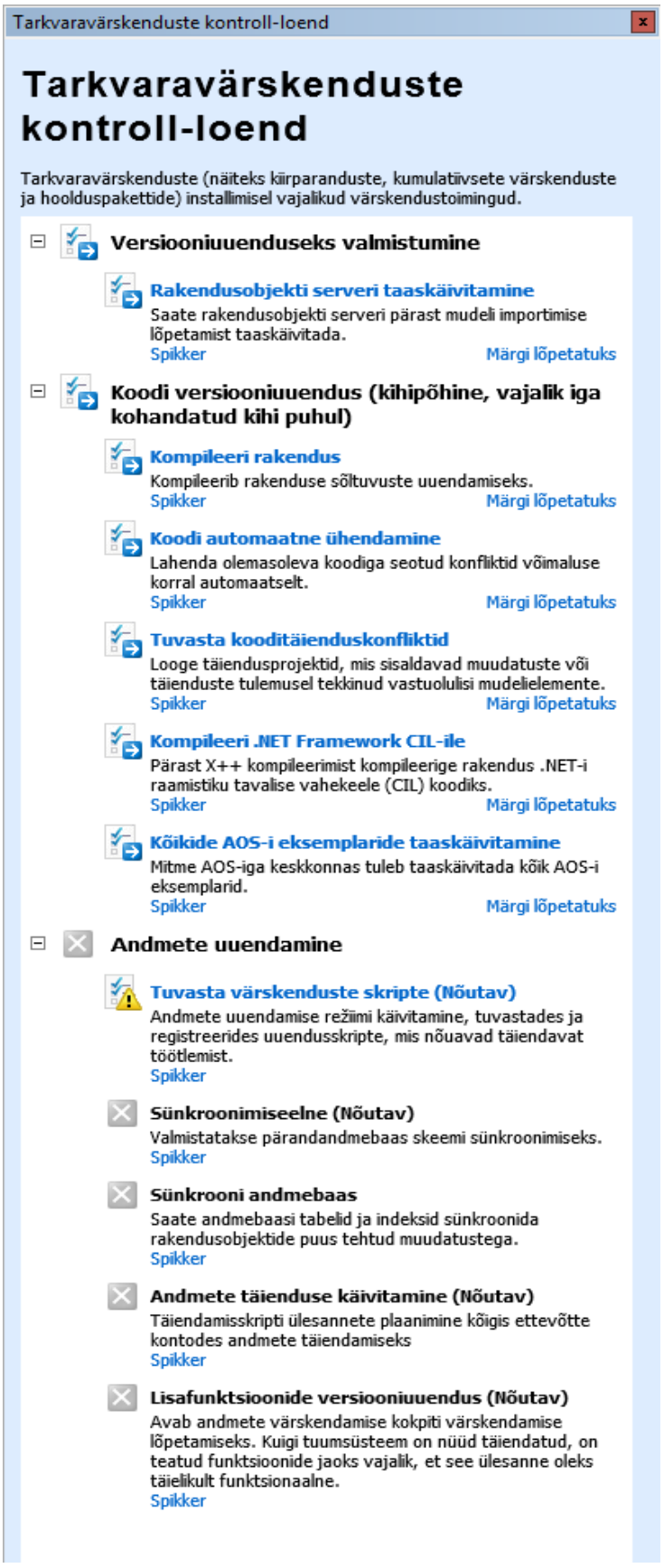


Joonis 12. Rakendusserveri käivitamine.

Rakendusserveri käivitamise järel tuleb läbi viia *modelstore* andmebaasi lähtestamine. Lähtestamine kujutab endast *modelstore* andmebaasi skeemi taasloomist, mille käigus luuakse andmebaasi uuenduse käigus lisandunud tabelid, indeksid, andmebaasiserveri talletatud protseduurid jne. [20] Skeemi taasloomiseks tuleb Powershell käsurealiideses käivitada käsklus „*Initialize-AXModelstore”*.

7.3.2 Tarkvaravärskenduste kontroll-loend

Tarkvaravärskenduste kontroll-loend kujutab endast versiooniuuenduse järgselt läbitavat tegevuste jada ning on ligipääsetav rakenduse süsteemihalduse vormilt. Tegemist on interaktiivse vormiga, kus on loetletud 11 tegevust, mis pärast uuenduse paigaldamist läbida tuleb. Tarkvaravärskenduste kontroll-loend on kujutatud joonisel 13.



Joonis 13. Tarkvarauuenduse kontroll-loend.

Enamus tegevusi on võimalik otse vormilt käivitada ning teostatakse automaatselt rakenduse poolt. Lisaks loetelus välja toodud tegevustele on vajalikud ka mõned lisategevused. Sellest tulenevalt on tegelik tegevuste jada järgnev:

1. Rakenduse kompileerimine - Eesmärgiga uuendada rakenduse objektide vahelisi sõltuvusi.

2. Kooditäienduskonfliktide tuvastamine – Tuvastab automaatselt rakenduse objektid, mille puhul esineb vastuolusid objekti erinevate kihtide vahel ning koondab kõik sellised objektid rakenduse objektipuus ühe projekti alla

3. Koodi automaatne ühendamine – *Auto-merge* funktsionaalsus, mis proovib lahendada uuenduse käigus tekkinud konflikte koodis ning kuvab sellekohase infologi kasutajale.

4. Koodikonfliktide lahendamine – Rakenduse toimimiseks tuleb uuenduse käigus tekkinud konfliktid lahendada. On oluline, et enne järgnevate sammude täitmist oleks kõik konfliktid rakenduse objektides lahendatud. Konfliktide lahendamist käsitletakse täpsemalt töö peatükis 7.3.3.

5. Rakenduse kompileerimine - Rakendus tuleb pärast koodikonfliktide lahendamist üle kompileerida.

6. .NET Framework CIL-ile kompileerimine – Kompileeritakse X++ kood .NET raamistiku CIL koodiks, mis võimaldab X++ koodi jooksutamist serveris.

7. Rakendusserver taaskäivitamine – Rakendusserver tuleb taaskäivitada, et uueneks serveris jooksev rakenduse kood.

8. Andmeuuendusskriptide loomine – Rakenduse poolt tehtav automaatne tegevus, mille käigus luuakse andmeuuendusskriptid. Andmeuuendusskriptid sisaldavad endas uute väljate väikeväärtustamist, andmetüübi muutmisel andmeparandust, tabeli indeksite loomist jne.

9. Eelsünkroniseerimine – Viiakse läbi vajalikud sünkroniseerimiseelsed tegevused.

10. Andmeuuendusskriptide käivitamine – Toimub punktis 8 loodud andmeuuendusskriptide käivitamine.

11. Andmebaasi sünkroniseerimine – Sünkroniseerimise käigus viiakse rakenduse andmebaasitabelis nimetusega SQLDictionary läbi andmete värskendamine. Tabelis hoitakse rakenduse elementide ID-sid, nimesid ning nende vastavusi süsteemi SQL andmebaasis. [21]

Jada läbimisel on oluline, et kõik tegevused saaksid täidetud sellises järjekorras nagu nad välja on toodud. Olles kõik tegevused läbinud võib lugeda versiooniuuenduse paigalduse arenduskeskkonnas lõpetatuks.

7.3.3 Koodikonfliktid ning nende lahendamine

Versiooniuuenduse käigus tekkivad koodikonfliktid rakenduse koodis on tingitud peatükis 4.2 kirjeldatud Dynamics AX 2012 kihilisest arhitektuurist. Nimelt sisaldab uuenduse *application* komponent rakenduse koodi, mis installeeritakse rakendusse Microsofti poolt hallatavasse SYP kihti. Sellest tulenevalt tekib konflikt juhul kui uuendatav rakenduse objekt on defineeritud kõrgemas kihis kui SYP – rakendus kasutab kõige kõrgemas kihis defineeritud objekti ning madalamas kihis defineeritud CU13 paigaldamisega tulnud koodiuuendus ei rakendu.

Peatükis 7.3.2 kirjeldatud rakenduse poolt tehtavad tegevused „Tuvasta kooditäienduskonfliktid“ ning „Koodi automaatne ühendamine“ suudavad sellised rakenduse objektid üles leida. Nende tegevuste käivitamisel läheb tööle programmikood, mis otsib rakendusest objekte millel on SYP kihis definitsioon olemas. Objekti leidmisel võrdleb programm SYP kihis olevat definitsiooni objekti kõige kõrgemas kihis oleva definitsiooniga ning üritab automaatselt kaks kihti omavahel ühendada. Kui võimalik, toimub *auto-merge* ning koodi lisatakse sellekohane kommentaar. Kui tekib konflikt ning koodi automaatselt ühendada ei õnnestu, lisatakse koodi kommentaar kujul mis antud objekti katki teeb. Sellises olukorras on vaja arendaja sekkumist, et konfliktid lahendada ning erinevates kihtides asetsev kood käsitsi ühendada. *Auto-merge* poolt õnnestunud ning ebaõnnestunud koodiparanduse näited on kujutatud joonisel 14. Lisaks koguvad antud töövahendid kõik nii konfliktis olevad kui ka automaatselt lahendatud objektid kokku ühte projekti faili, lihtsustades sellega arendaja tööd.

```
// TODO: The following conflict requires manual resolution.
>>>> Original (SYS) ++++++++++++++++++++++++++++++++++++++++++++++++++
    binData = new BinData();
    binData.setData(docuRef.docuValue().File);
    if (isRunningOnServer())
    {
        error(error::wrongUseOfFunction(funcName()));
        throw Exception::Error;
    }
    else
==== Yours (CUS) ++++++++++++++++++++++++++++++++++++++++++++++++++
    //binData = new BinData();
    //binData.setData(docuRef.docuValue().File);
    if (isRunningOnServer())
    {
        error(error::wrongUseOfFunction(funcName()));
        throw Exception::Error;
    }
    else
==== AX Update (SYP) ++++++++++++++++++++++++++++++++++++++++++++++++++
    binData = new BinData();
    binData.setData(docuRef.docuValue().File);

    filePath = endSlash(internetCache ? Docu::getTempPath() : xInfo::directory(DirectoryType::Temp));
    filename = docuRef.completeFilename(filePath);

    isPermRequired = isRunningOnServer();

    if (isPermRequired)
<<<< End
    {
// The following was merged successfully, but should be reviewed.
// Original (SYS) ++++++++++++++++++++++++++++++++++++++++++++++++++
//         filePath = endSlash(internetCache ? WinAPI::getFolderPath(#CSIDL_INTERNET_CACHE) : xInfo::directory(DirectoryType::Temp));
// Yours (CUS) ++++++++++++++++++++++++++++++++++++++++++++++++++
//         filePath = endSlash(internetCache ? WinAPI::getFolderPath(#CSIDL_INTERNET_CACHE) : xInfo::directory(DirectoryType::Temp));
// AX Update (SYP) ++++++++++++++++++++++++++++++++++++++++++++++++++
        perm = new FileIOPermission(filename, #IO_WRITE);
        perm.assert();
// End
    }
```

Joonis 14. *Auto-merge* poolt tehtud koodimuudatused.

Rakenduse korrektseks toimimiseks on vaja kõik *auto-merge* poolt parandamata jäänud koodikonfliktid paranda. Samuti on vaja üle kontrollida *auto-merge* poolt ühendatud kood. Seda seetõttu, et automaatne koodi ühendamise funktsionaalsus ei toimi alati õigesti ning ühendab koodi vigaselt kokku. Samuti ei leia programm alati kõiki konfliktis olevaid rakenduse elemente üles. Sellised olukorrad tulevad välja rakenduse täiskompileerimisel kompilaatori poolt väljastatud logist.

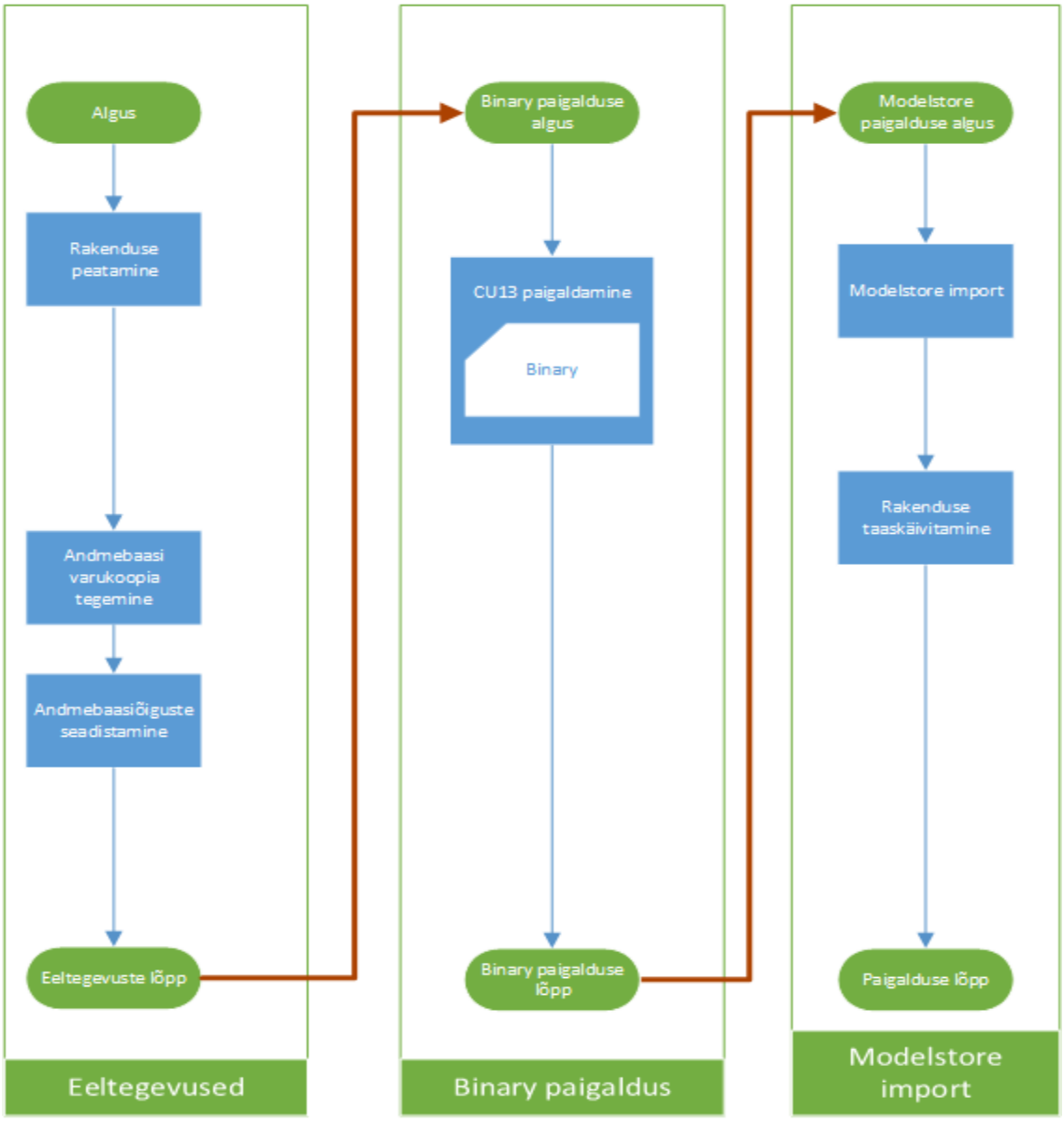
Rakenduses leiduvate vigaste objektide parandamine ning *auto-merge* poolt parandatud koodi kontrollimine toimub AX 2012 MorphX arenduskeskkonnas. Läbi tuleb käia kõik uuenduse käigus muudetud rakenduse objektid. Konfliktid tuleb parandada koodi kontekstist lähtuvalt selliselt, et:

- Kliendi jaoks kirjutatud funktsionaalsus ei muutuks.
- Muudatuste käigus ei läheks katki integratsioon väliste liidestega.
- Rakenduksid versiooniuuenduses sisalduvad muudatused (kui ei lähe vastuollu kliendi jaoks muudetud rakenduse funktsionaalsusega).
- Muudetud kood ei teeks katki mõnda rakenduse teist osa.
- Rakenduse standardfunktsionaalsus jääks korda.

Näiteid koodikonfliktidest ning parandustest koos kirjeldustega leidub töö lisades 3-5.

8 CU13 paigaldus test- ja toodangukeskkonnas

CU13 versiooniuuenduse paigaldus on test- ning toodangukeskkonnas identne. Paigaldusprotsess koosneb kolmest etapist: eeltegevuste läbiviimine, uuenduse *binary* komponendi paigaldamine ning *modelstore* andmebaasi importimine. Versiooniuuenduse paigaldusprotsess testimis- ja toodangukeskkonnas on esitatud joonisel 15.



Joonis 15. Paigaldusprotsess test-ja toodangukeskkonnas.

8.1 Paigalduse eeltegevused

Test- ning toodangukeskkonnas läbi viidavad paigalduse eeltegevused ei erine sisult samas etapis arenduskeskkonnas läbi viidud tegevustest ning on juba kirjeldatud töö varasemates peatükkides:

- Rakenduse peatamine – kirjeldatud töö peatükis 7.1.1.
- Andmebaasi varukoopia tegemine – kirjeldatud töö peatükis 7.1.3.
- Andmebaasiõiguste seadistamine – kirjeldatud töö peatükis 7.1.4.

8.2 Binaarkomponendi paigaldus

Uuenduse binaarkomponendi paigaldus toimub samuti test- ning toodangukeskkonnas identselt arenduskeskkonnaga ning on kirjeldatud varasemas töö peatükis:

- Binaarkomponendi paigaldus – kirjeldatud töö peatükis 7.2

8.3 *Modelstore* andmebaasi importimine

CU13 uuenduse *application* komponendi paigaldamist pole test-ja toodangukeskkonnas vaja teha. Selleks, et antud komponendis sisalduvad muudatused jõuaksid test- ja toodangukeskkonda, piisab arenduskeskkonnast võetud *modelstore* andmebaasi importimisest vastavate keskkondade rakendustesse. Arenduskeskkonna *modelstore* andmebaasi importimisel paigaldub test- ja toodangukeskkonna rakendusse CU13 juba parandatud kujul.

Modelstore andmebaasi eksportimine-importimine on võimalik kasutades Microsoft Dynamics AX 2012 Management Shell käsurea töövahendit. Mudeli eksportimine toimub kasutades käsku „*Export-AXModelStore -File* **<Filename>**“. Mudeli importimiseks tuleb käsurea töövahendis käivitada käsk „*Import-AXModelStore -File* **<Filename>**“. [22] Kohatäide **<Filename>** tähistab eksporditava/imporditava failinime.

8.4 Rakendusserveri taaskäivitamine

Viimane samm paigaldusprotsessi läbiviimisel on rakendusserveri taaskäivitamine. Serveri peatamist ning käivitamist on käsitletud töö varasemates peatükkides:

- Rakendusserveri peatamine – kirjeldatud töö peatükis 7.1.1.
- Rakendusserveri käivitamine – kirjeldatud töö peatükis 8.4.

9 CU13 testimismahu hindamine

CU13 versiooniuuenduse protsessi kuulub ka rakenduse testimine. Uuenduse käigus toimus suurel hulgal programmikoodi muutmist ning sellest tulenevalt vajab rakendus testimist. On vaja veenduda, et süsteem töötab vigadeta ning ettevõtte jaoks kohandatud rakenduse funktsionaalsus toimib samamoodi kui enne uuenduse läbi viimist.

Testimise muudab keerukaks uuenduse käigus muutunud programmikoodi hulk ning selle kaootilisus. Sellest tulenevalt on rakenduse testimiseks vaja hinnangut, mis annaks ülevaate, et kui suures mahus tuleks süsteemi testida. Antud hinnang oleks aluseks testijatele testplaani koostamisel. Testimismahu hinnangu eesmärk on:

- Anda üldine hinnang sellele, et kui suures osas vajab rakendus uuenduse järgselt testimist.
- Analüüsida, kas on funktsionaalsust mida on vaja põhjalikumat testida.

Testimismahu hinnang kujundatakse võttes arvesse:

- Konfliktide arvu rakenduse objektide lõikes – Eesmärk on tekitada ülevaade konfliktidest ning tuua välja objektid mille puhul esines suures mahus konflikte.
- Rakenduse koodi ühik-testidega kattuvuse ulatus – Eesmärgiks on määrata versiooniuuendusega muudetud koodi hulka mis on ühiktestidega kaetud.
- Testija subjektiivne hinnang – Testija subjektiivne hinnang rakenduse testimise mahule.
- Arendaja subjektiivne hinnang – Versiooniuuendust läbi viinud arendaja subjektiivne hinnang rakenduse testimise mahule.

9.1 Konfliktide arv rakenduse objektide lõikes

Konfliktide arvu allikaks on peatükis 7.3.2 kirjeldatud rakenduse koodi ning uuenduste *auto-merge* tulemusena kuvatav logi, mis sisaldab infot rakenduse objektides olevate koodikonfliktide kohta.

Uuenduse järgselt esines konflikte 272-s rakenduse objektis ning konfliktide arv oli 2387, mis teeb keskmiselt 8 konflikti objekti kohta.

Tabelis 3 on välja toodud 10 kõrgeima konfliktide arvuga rakenduse objekti:

Tabel 3. Koodikonfliktid objektide lõikes.

Objekti nimi	**Objekti tüüp**	**Konfliktide arv**	**Objekti kirjeldus**
AssetProposalDeprecation	Klass	131	Klass sisaldab endas põhivara väärtuse vähenemisega seotud funktsionaalsust
ProjBudgetManager	Klass	100	Klass sisaldab projekti eelarve majandamisega seotud funktsionaalsust
CustTrans	Tabel	77	Kliendikannete tabel
LedgerJournalCheckPost	Klass	63	Pearaamatu kande sisestamisega seotud klass
PurchAutoCreate_PurchReq	Klass	57	Ostutaotluste loomisega seotud klass

PurchReqLine	Tabel	54	Ostutaotluste ridade tabel
PurchLine	Tabel	53	Ostutellimuste ridade tabel
PurchTable	Tabel	37	Ostutellimuste päise tabel
SysWorkflowWorkItem	Klass	35	Töövoogudega seotud klass
SalesLine	Tabel	35	Müügitellimuste ridade tabel

Tabeli põhjal saab järeldada, et kindlasti vajavad põhjalikku testimist rakenduse funktsionaalsused mis on seotud järgnevate tegevustega:

- Ostuprotsess
- Müügiprotsess
- Töövood
- Projekti eelarvestamine
- Põhivarade haldamine

9.2 Rakenduse koodi ühik-testidega kattuvuse ulatus

Ettevõttes, kus antud töö raames versiooniuuendus läbi viidi, ei ole Dynamics AX 2012 rakenduse arendamisel ühik-teste kasutatud. Kulude kokkuhoiuks ei soovitud ühik-testide loomist ka versioonuuenduse käigus läbi viidud programmikoodi muudatustele.

9.3 Testija subjektiivne hinnang

Testija hinnang põhineb varasemal kogemusel rakenduses Dynamics AX 2009 läbi viidud versiooniuuenduste testimisel. Võttes arvesse uuenduse käigus tehtud muudatuste mahtu, tuleb testija sõnul rakenduses läbi viia täieulatuslik testimine. Erilist tähelepanu tuleb pöörata sellele, et:

- Töötaksid kõik tähtsamad ettevõtte protsessidega seotud funktsionaalsused.
- Rakenduse liidestatus teiste süsteemidega jääks töökorda.

Testimine viiakse läbi lõpp-kasutaja vaates rakenduse kasutajaliideses.

9.4 Arendaja subjektiivne hinnang

Mida rohkem on tehtud rakenduse standardobjektides muudatusi seda suurem on konfliktide arv. Samuti mõjutab konfliktide arvu rakenduse uuenduseelne seis: tihedam uuenduste paigaldamine tagab väiksema konfliktide arvu järgnevate uuenduste puhul.

Rakenduses, kus selle bakalaureusetöö käigus versiooniuuendus läbi viidi, oli uuenduse paigaldusjärgseid konflikte rakenduse peale kokku 2387. Dynamics AX 2012 uuendati versioonilt CU7 versioonile CU13 ning rakendust on suures mahus muudetud rahuldamaks ettevõtte ärilisi vajadusi.

Töö autor nõustub peatükis 9.3 kirjeldatud testija poolse hinnangug, et rakenduses tuleb läbi viia täieulatuslik lõpp-kasutaja test.

Testimist vajab rakenduse standardfunktsionaalsus ning standardfunktsionaalsusele tehtud ettevõtte kohased muudatused. Versiooniuuendus ei mõjuta rakendusse juurde lisatud uut funktsionaalsust mis süsteemi standardfunktsionaalsusega kokku ei puutu. Testimise mahtu aitaks vähendada ühik-testide kasutusele võtmine.

9.5 Hinnang testimismahule

CU13 versiooniuuenduse järgselt vajab süsteem täieulatuslikku testimist, kuna uuenduse käigus muudeti suures mahus programmikoodi ning selle koodi automaatseks testimiseks ei soovitud ühik-teste luua.

Testimine peaks keskenduma rakenduse standardfunktsionaalsuse ning sellele tehtud muudatuste testimisele. Rakenduse osa mis standardiga kokku ei puutu, ei ole vaja testida.

Testimisel tuleb veenduda, et töötaksid ettevõtte protsessidega seotud rakenduse funktsionaalsused. Erilist tähelepanu tasuks pöörata ostuprotsessi, müügiprotsessi, töövoogude, projekti eelarvestamise ning põhivarade haldamisega seotud funktsionaalsuse testimisele, kuna antud protsessidega seotud objektides esines kõige enam koodikonflikte. Samuti tuleb veenduda, et liidestatus teiste süsteemidega ei oleks katki läinud.

10 Kokkuvõte

Käesoleva bakalaureusetöö eesmärgiks oli luua Dynamics AX 2012 versiooniuuenduse läbiviimise taaskasutatav metoodika ettevõttes, kus antud rakendus kasutusel on. Metoodika pidi sisaldama vastuseid küsimustele:

- Kuidas toimub versiooniuuenduse paigaldamine ning milliseid probleeme võib selle käigus esineda?
- Kui mahukat testimist versiooniuuendus vajab?

Toetudes rakenduse tehnilisele kirjandusele viis autor esmalt läbi süsteemi versiooniuuenduse arenduskeskkonnas ning kaardistas kogu protsessi. Arenduskeskkonnas uuendust läbi viies ilmnesid probleemid nii paigalduse eelselt kui ka järgselt:

- Rakenduse mudelandmebaasis olevad ebakõlad takistasid paigaldusprotsessi alustamist.
- Versiooniuuenduse järgselt sisaldas rakendus suures mahus koodikonflikte.

Töö autor lahendas probleemid ning kirjeldas nende lahendused versiooniuuenduse paigaldamise metoodika osana.

Järgnevalt viis autor läbi versioonuuenduse paigalduse testimiskeskkonnas ning kaardistas kogu protsessi. Seejärel toimus uuenduse paigaldamine toodangukeskkonnas, kus läbiviidud paigaldusprotsess oli identne testkeskkonnas läbitud paigaldusprotsessiga.

Versiooniuuenduse järgselt rakenduse testimismahu hindamise aluseks võeti koodikonfliktide arv rakenduse objektide lõikes, rakenduse koodi ühik-testidega kattuvuse ulatus, ühe testmeeskonnaliikme ning arendaja rollis olnud töö autori subjektiivsed hinnangud.

Testimismahu hinnanguks kujunes vajadus rakenduse täieulatuslikule testimisele, kuna rakenduses toimus suures mahus programmikoodi muutmist ning rakenduse koodis ei ole kasutatud ühik-testimist. Koodikonflikte esines kõige enam rakenduse ostuprotsessi, müügiprotsessi, töövoogude, projekti eelarvestamise ning põhivarade haldamisega seotud objektides, mis tõttu tuleks rakenduse testimisel nendele süsteemi osadele suuremat tähelepanu osutada.

Kasutatud kirjandus

[1] P. Kotler ja G. Armstrong, Principles of Marketing, 15th Edition, Edinburgh: Pearson Education Limited , 2014.

[2] P. N. Clough ja H.-B. Kittlaus, Software Product Management and Pricing, Berlin: Springer-Verlag, 2009.

[3] „Wikipedia," Wikimedia Foundation, 2003. [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Operating_system. [Kasutatud 5 mai 2018].

[4] V. Beal, „webopedia," [Võrgumaterjal]. Available: https://www.webopedia.com/TERM/O/operating_system.html. [Kasutatud 8 Mai 2018].

[5] „statcounter," [Võrgumaterjal]. Available: http://gs.statcounter.com/os-market-share/. [Kasutatud 5 Mai 2018].

[6] „statcounter," [Võrgumaterjal]. Available: http://gs.statcounter.com/os-market-share/mobile-tablet/worldwide/#quarterly-201301-201703. [Kasutatud 5 Mai 2018].

[7] S. Krakowiak, „Middleware Architecture with Patterns and Frameworks," 27 Veebruar 2009. [Võrgumaterjal]. Available: http://lig-membres.imag.fr/krakowia/Files/MW-Book/Chapters/Intro/intro-body.html. [Kasutatud 5 Mai 2018].

[8] „ISO," ISO, November 2017. [Võrgumaterjal]. Available: https://www.iso.org/standard/63712.html. [Kasutatud 6 Mai 2018].

[9] A. Kaplan, „Oracle," Oracle Corporation, September 2007. [Võrgumaterjal]. Available: https://web.archive.org/web/20080514023816/http://www.oracle.com/technology/oramag/oracle/07-sep/o57field.html. [Kasutatud 9 Mai 2018].

[10] „IT Pro Today," IT Pro Today, 07 Mai 2008. [Võrgumaterjal]. Available: http://www.itprotoday.com/windows-server/slipstreaming-windows-xp-service-pack-3-sp3. [Kasutatud 9 Mai 2018].

[11] Microsoft corporation, „Microsoft Support," Microsoft, 2 Veebruar 2018. [Võrgumaterjal]. Available: https://support.microsoft.com/en-us/help/4032175/cumulative-update-13-cu13-for-microsoft-dynamics-ax-2012-r3. [Kasutatud 9 Mai 2018].

[12] „Wikipedia," Wikimedia Foundation, Inc, 3 Mai 2018. [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Patch_(computing). [Kasutatud 7 Mai 2018].

[13] The Microsoft Dynamics AX Team, Inside Microsoft Dynamics AX 2012 R3, Redmond, Washington: Microsoft Press, 2014.

[14] Microsoft Corporation , „microsoft," Microsoft, 1 September 2017. [Võrgumaterjal]. Available: https://www.microsoft.com/en-us/download/confirmation.aspx?id=11094. [Kasutatud 6 Mai 2018].

[15] Microsoft corporation, „Developer Network,“ Microsoft, 19 Oktoober 2011. [Võrgumaterjal]. Available: https://msdn.microsoft.com/en-us/library/aa891248.aspx. [Kasutatud 15 Mai 2018].

[16] Microsoft corporation, „Developer Network,“ Microsoft, 25 Aprill 2011. [Võrgumaterjal]. Available: https://msdn.microsoft.com/en-us/library/gg889157.aspx. [Kasutatud 11 Mai 2018].

[17] R. C. Carlson, „Microsoft Technet,“ Microsoft, 13 September 2017. [Võrgumaterjal]. Available: https://blogs.technet.microsoft.com/dynamicsaxse/2017/09/13/announcing-cumulative-update-13-for-microsoft-dynamics-ax2012-r3/. [Kasutatud 10 Mai 2018].

[18] „Professional QA,“ Professional QA, 16 August 2016. [Võrgumaterjal]. Available: http://www.professionalqa.com/code-freeze. [Kasutatud 14 Mai 2018].

[19] Microsoft corporation, „Lifecycle Services,“ Microsoft, [Võrgumaterjal]. Available: https://fix.lcs.dynamics.com/Issue/Details?bugId=3886595. [Kasutatud 11 Mai 2018].

[20] Microsoft corporation, „Technet,“ Microsoft, 13 Jaanuar 2014. [Võrgumaterjal]. Available: https://technet.microsoft.com/en-us/library/hh433540.aspx. [Kasutatud 13 Mai 2018].

[21] Microsoft corporation, „Developer Network,“ Microsoft, [Võrgumaterjal]. Available: https://msdn.microsoft.com/en-us/library/sqldictionary.aspx. [Kasutatud 15 Mai 2018].

[22] Microsoft corporation, „Technet,“ Microsoft, 1 Veebruar 2013. [Võrgumaterjal]. Available: https://technet.microsoft.com/en-us/library/hh433530.aspx. [Kasutatud 14 Mai 2018].

[23] L. S. Sterling, The Art of Agent-Oriented Modeling, London: The MIT Press, 2009.

[24] M. Pocius, Microsoft Dynamics AX 2012 Development Cookbook, 35 Livery Street: Packt Publishing Ltd., 2012.

[25] „statcounter,“ [Võrgumaterjal]. Available: http://gs.statcounter.com/os-market-share/desktop/worldwide/#quarterly-201301-201703-bar. [Kasutatud 5 Mai 2018].

[26] „tutorialspoin,“ [Võrgumaterjal]. Available: https://www.tutorialspoint.com/sdlc/sdlc_overview.htm. [Kasutatud 6 Mai 2018].

[27] H. Vallaste, „e-teatmik,“ [Võrgumaterjal]. Available: http://www.vallaste.ee/. [Kasutatud 6 Mai 2018].

[28] Microsoft corporation, „Developer Network,“ Microsoft, 22 September 2011. [Võrgumaterjal]. Available: https://msdn.microsoft.com/en-us/library/aa660629.aspx. [Kasutatud 6 Mai 2018].

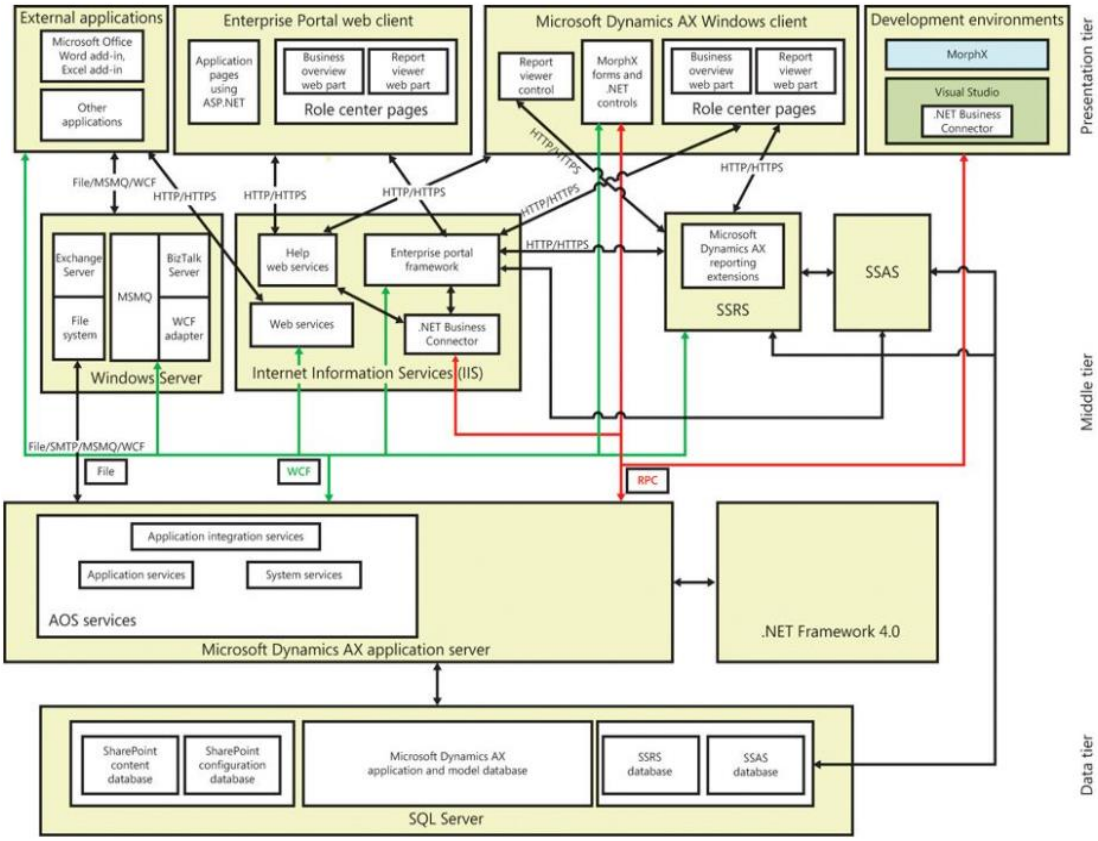
[29] Microsoft corporation, „Developer Network,“ Microsoft, 2 August 2011. [Võrgumaterjal]. Available: https://msdn.microsoft.com/en-us/library/aa853691.aspx. [Kasutatud 7 Mai 2018].

[30] Microsoft corporation, „Technet,“ Microsoft, 15 Oktoober 2014. [Võrgumaterjal]. Available: https://technet.microsoft.com/en-us/library/dn313121.aspx. [Kasutatud 11 Mai 2018].

[31] Microsoft corporation, „Technet,“ Microsoft, 15 Oktoober 2014. [Võrgumaterjal]. Available: https://technet.microsoft.com/en-us/library/dn313121.aspx. [Kasutatud 11 Mai 2018].

[32] „wikipedia,“ Wikimedia Foundation, Inc, 8 Mai 2018. [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Systems_development_life_cycle. [Kasutatud 9 Mai 2018].

Lisa 1 – Rakenduse arhitektuurijoonis



Joonis 16. Dynamics AX 2012 arhitektuur. [14]

Lisa 2 – Rakenduse elementide SQL parandusskript

```
DECLARE @oldModelDBName nvarchar(500)
DECLARE @newModelDBName nvarchar(500)
/*
Usage:
Replace the value for @newModelDBName with the name of your Microsoft
Dynamics AX 2012 R2 model store database.
Replace the value for @oldModelDBName with the name of your Microsoft
Dynamics AX 2012 or Microsoft Dynamics AX 2012 Feature Pack database.
*/
SET @newModelDBName = 'MicrosoftDynamicsAX_model'
SET @oldModelDBName = 'MicrosoftDynamicsAX'


DECLARE @usingStmt [NVARCHAR](MAX)
DECLARE @dropModelPatchTableStmt [NVARCHAR](MAX)
DECLARE @dropCollisionsTableStmt [NVARCHAR](MAX)
DECLARE @selectIntoModelPatchTableStmt [NVARCHAR](MAX)
DECLARE @updateAxIdsStmt [NVARCHAR](MAX)
DECLARE @updateParentAxIdsStmt [NVARCHAR](MAX)
DECLARE @selectIntoAxIDCollisionsTableStmt [NVARCHAR](MAX)
DECLARE @axIdCollisionsStmt[NVARCHAR](MAX)

SELECT @usingStmt = 'USE '+ @oldModelDBName
EXEC(@usingStmt)

SELECT @dropModelPatchTableStmt = 'IF EXISTS (
SELECT * FROM ['+@oldModelDBName+'].[SYS].[TABLES] WHERE [NAME] =
''MODELSTOREAXIDPATCH''
)
DROP TABLE ['+@oldModelDBName+ ']..[MODELSTOREAXIDPATCH]'
EXEC (@dropModelPatchTableStmt)

SELECT @dropCollisionsTableStmt = 'IF EXISTS (
SELECT * FROM ['+@oldModelDBName+'].[SYS].[TABLES] WHERE [NAME] =
''MODELSTOREAXIDCOLLISIONS''
)
DROP TABLE ['+@oldModelDBName+ ']..[MODELSTOREAXIDCOLLISIONS]'
EXEC (@dropCollisionsTableStmt)

BEGIN TRAN

SELECT @selectIntoModelPatchTableStmt = 'SELECT * INTO
['+@oldModelDBName+']..[MODELSTOREAXIDPATCH] FROM (
```

```
SELECT [OLDMODELELEMENT].[AXID] AS OLDAXID, [NEWMODELELEMENT].[AXID] AS
NEWAXID, [NEWMODELELEMENT].[PARENTID] AS NEWPARENTID,
[NEWMODELELEMENT].[PARENTHANDLE] AS NEWPARENTHANDLE,
[NEWMODELELEMENT].[ELEMENTHANDLE] AS
NEWELEMENTHANDLE,[NEWMODELELEMENT].[NAME] AS NEWELEMENTNAME,
[NEWMODELELEMENT].[ELEMENTTYPE] AS NEWMODELELEMENTTYPE
FROM [' + @oldModelDBName + ']..[MODELELEMENT] OLDMODELELEMENT
INNER JOIN [' + @newModelDBName + ']..[MODELELEMENT] NEWMODELELEMENT
ON ([OLDMODELELEMENT].[ORIGIN] = [NEWMODELELEMENT].[ORIGIN])
WHERE [OLDMODELELEMENT].[ORIGIN] != ''00000000-0000-0000-0000-000000000000''
AND [OLDMODELELEMENT].[AXID] != [NEWMODELELEMENT].[AXID]) CHANGEDAXIDS'
EXEC(@selectIntoModelPatchTableStmt)

SELECT @selectIntoAxIDCollisionsTableStmt = 'SELECT * INTO
['+@oldModelDBName+']..[MODELSTOREAXIDCOLLISIONS] FROM (
SELECT [NEWMODELELEMENT].[AXID] AS NEWAXID, [NEWMODELELEMENT].[PARENTID] AS
NEWPARENTID, [NEWMODELELEMENT].[PARENTHANDLE] AS NEWPARENTHANDLE,
[NEWMODELELEMENT].[ELEMENTTYPE] AS NEWELEMENTTYPE, [NEWMODELELEMENT].[NAME]
AS NEWELEMENTNAME, [NEWMODELELEMENT].[ELEMENTHANDLE] AS NEWELEMENTHANDLE,
[NEWMODELELEMENT].[PARTOFINHERITANCE] AS PARTOFINHERITANCE
FROM ['+@newModelDBName+']..[MODELELEMENT] NEWMODELELEMENT
INNER JOIN ['+@oldModelDBName+']..[MODELSTOREAXIDPATCH] AXIDPATCH
ON ([NEWMODELELEMENT].[AXID] = [AXIDPATCH].[OLDAXID]
AND [NEWMODELELEMENT].[PARENTHANDLE] = [AXIDPATCH].[NEWPARENTHANDLE]
AND [NEWMODELELEMENT].[ELEMENTTYPE] = [AXIDPATCH].[NEWMODELELEMENTTYPE]
AND [AXIDPATCH].[NEWELEMENTHANDLE] != [NEWMODELELEMENT].[ELEMENTHANDLE])
WHERE
NOT EXISTS (SELECT [NEWELEMENTHANDLE] FROM
['+@oldModelDBName+']..[MODELSTOREAXIDPATCH] PATCHLIST
WHERE PATCHLIST.[NEWELEMENTHANDLE] = [NEWMODELELEMENT].[ELEMENTHANDLE]))
COLLISIONS'
EXEC(@selectIntoAxIDCollisionsTableStmt)


SELECT @AxIdCollisionsStmt =   'DECLARE collidingElementsCursor CURSOR FOR
SELECT [NEWELEMENTTYPE], [NEWPARENTID], [PARTOFINHERITANCE],
[NEWELEMENTHANDLE] from ['+@oldModelDBName+']..[MODELSTOREAXIDCOLLISIONS]
OPEN collidingElementsCursor

DECLARE @newElementType INT
DECLARE @newParentId INT
DECLARE @partOfInheritance INT
DECLARE @newElementHandle INT

FETCH NEXT FROM collidingElementsCursor
INTO @newElementType, @newParentId, @partOfInheritance, @newElementHandle

WHILE @@FETCH_STATUS = 0
BEGIN
DECLARE @newAxID INT;
```

```
EXEC ['+@newModelDBName+']..XU_GetNextAvailableAxId @newElementType,
@newParentId, @partOfInheritance, @newAxID OUTPUT

UPDATE ['+@newModelDBName+']..[MODELELEMENT] SET [AXID] = @newAxID
WHERE [MODELELEMENT].[ELEMENTHANDLE] = @newElementHandle

UPDATE ['+@newModelDBName+']..[MODELELEMENT] SET [PARENTID] = @newAxID
WHERE [MODELELEMENT].[PARENTHANDLE] = @newElementHandle

FETCH NEXT FROM collidingElementsCursor
INTO @newElementType, @newParentId, @partofInheritance, @newElementHandle
END

CLOSE collidingElementsCursor
DEALLOCATE collidingElementsCursor'
EXEC(@AxIdCollisionsStmt)

SELECT @updateAxIdsStmt = 'UPDATE [' + @newModelDBName + ']..[MODELELEMENT]
SET [AXID] = ['+@oldModelDBName+']..[MODELSTOREAXIDPATCH].[OLDAXID]
FROM ['+@oldModelDBName+']..[MODELSTOREAXIDPATCH ]
WHERE [' + @newModelDBName + ']..[MODELELEMENT].[ELEMENTHANDLE] =
['+@oldModelDBName+']..[MODELSTOREAXIDPATCH].[NEWELEMENTHANDLE]
AND [' + @newModelDBName + ']..[MODELELEMENT].[NAME] =
['+@oldModelDBName+']..[MODELSTOREAXIDPATCH].[NEWELEMENTNAME]
AND [' + @newModelDBName + ']..[MODELELEMENT].[ELEMENTTYPE] =
['+@oldModelDBName+']..[MODELSTOREAXIDPATCH].[NEWMODELELEMENTTYPE]'
EXEC(@updateAxIdsStmt)

SELECT @updateParentAxIdsStmt = 'UPDATE [' + @newModelDBName +
']..[MODELELEMENT] SET [PARENTID] =
['+@oldModelDBName+']..[MODELSTOREAXIDPATCH].[OLDAXID]
FROM ['+@oldModelDBName+']..[MODELSTOREAXIDPATCH]
WHERE [' + @newModelDBName + ']..[MODELELEMENT].[PARENTHANDLE] =
['+@oldModelDBName+']..[MODELSTOREAXIDPATCH].[NEWELEMENTHANDLE]'
EXEC(@updateParentAxIdsStmt)

COMMIT TRAN
```

Joonis 17. Elementide SQL parandusskript.

Lisa 3 – Koodikonflikti paranduse näide nr.1

```
/// </summary>
/// <param name="_docuValueRecId">
/// The RecId of the document to get in a binary object.
/// </param>
/// <returns>
/// The binary representation for the specified document.
/// </returns>
public static CLRObject getDocuValueAsBinary(RecId _docuValueRecId)
{
// TODO: The following conflict requires manual resolution.
>>>> Original (SYS) ++++++++++++++++++++++++++++++++++++++++++++++++++
    return Binary::constructFromContainer(DocuValue::find(_docuValueRecId).File).getMemoryStream();
==== Yours (CUS) ++++++++++++++++++++++++++++++++++++++++++++++++++++
    return Binary::constructFromContainer(SofDocuValueFileData::fileDB2Con(DocuValue::find(_docuValueRecId))).getMemoryStream();
    //return Binary::constructFromContainer(DocuValue::find(_docuValueRecId).File).getMemoryStream();
==== AX Update (SYP) ++++++++++++++++++++++++++++++++++++++++++++++++
    DocuValue docValue = DocuValue::find(_docuValueRecId, true);
    str fileName;
    Microsoft.Dynamics.AX.Framework.OfficeAddin.SharePoint.AuthenticationType docuAuthType;
    SharePointAuthenticationType authType = SharePointAuthenticationType::Windows;
    System.IO.MemoryStream memStream;
    System.Uri uri;

    // if docValue.File not empty, get data from DocuValue.File
    // otherwise, get data from the URL
    if (docValue.File != conNull())
    {
        memStream = Binary::constructFromContainer(docValue.File).getMemoryStream();
    }
    else
    {
        if(docValue.Type == DocuValueType::URL)
        {
            fileName = docValue.Path;
        }
        else
        {
            fileName = System.String::Concat(System.IO.Path::Combine(docValue.Path, docValue.FileName), '.', docValue.FileType);
        }

        try
        {
            uri = new System.Uri(fileName);

            authType = DocuSharePointParameters::GetAuthenticationType(fileName);
            docuAuthType = DocuSharePointParameters::ConvertAuthenticationType(authType);

            memStream = DocuLibrary::GetMemoryStreamFromURL(uri, docuAuthType);
        }
        catch
        {
            memStream = CLRInterop::Null('System.IO.MemoryStream');
        }
    }

    return memStream;
<<<< End
}
```

Joonis 18. *Auto-merge* poolt lahendamata programmikood.

```
public static CLRObject getDocuValueAsBinary(RecId _docuValueRecId)
{
    DocuValue docValue = DocuValue::find(_docuValueRecId, true);
    str fileName;
    Microsoft.Dynamics.AX.Framework.OfficeAddin.SharePoint.AuthenticationType docuAuthType;
    SharePointAuthenticationType authType = SharePointAuthenticationType::Windows;
    System.IO.MemoryStream memStream;
    System.Uri uri;

    // if docValue.File not empty, get data from DocuValue.File
    // otherwise, get data from the URL
    if (docValue.File != conNull())
    {
        memStream = Binary::constructFromContainer(docValue.File).getMemoryStream();
    }
    else
    {
        if(docValue.Type == DocuValueType::URL)
        {
            fileName = docValue.Path;
        }
        else
        {
            fileName = System.String::Concat(System.IO.Path::Combine(docValue.Path, docValue.FileName), '.', docValue.FileType);
        }

        try
        {
            uri = new System.Uri(fileName);

            authType = DocuSharePointParameters::GetAuthenticationType(fileName);
            docuAuthType = DocuSharePointParameters::ConvertAuthenticationType(authType);

            memStream = DocuLibrary::GetMemoryStreamFromURL(uri, docuAuthType);
        }
        catch
        {
            memStream = CLRInterop::Null('System.IO.MemoryStream');
        }
    }

    return memStream;
}
```

Joonis 19. Programmikood parandatud kujul.

Lisa 4 – Koodikonflikti paranduse näide nr.2

```
    if (TaxParameters::checkParameterForPosting_IN(TaxType_IN::Customs) &&
        PurchTable::find(this.PurchId).purchTable_W().CustomsImportOrder_IN == NoYes::Yes &&
        CustomsVendBOETrans_IN::findPurchLineRecId(PurchLine::findInventTransId(this.InventRefTransId).RecId).RecId != 0 &&
        this.PurchId == this.InventRefId)
    {
// The following was merged successfully, but should be reviewed.
// Original (SYS) ++++++++++++++++++++++++++++++++++++++++++++++++++
//         checkFailed("@GLS5764");
// Yours (CUS) ++++++++++++++++++++++++++++++++++++++++++++++++
//         checkFailed("@GLS5764");
// AX Update (SYP) ++++++++++++++++++++++++++++++++++++++++++++++++++
        ok = checkFailed("@GLS5764");
// End
    }
// The following was merged successfully, but should be reviewed.
// Original (SYS) ++++++++++++++++++++++++++++++++++++++++++++++++
//     else
// Yours (CUS) ++++++++++++++++++++++++++++++++++++++++++++++++
//     else
// AX Update (SYP) ++++++++++++++++++++++++++++++++++++++++++++++++

    if (ok && #PdsApprovedVendorListEnabled)
// End
    {
// TODO: The following conflict requires manual resolution.
>>>> Original (SYS) ++++++++++++++++++++++++++++++++++++++++++++++++++
        this.insert(dropInvent,_searchMarkup,true, _skipPurchTableUpdate,_skipInterCompanyCalcDisc);
==== Yours (CUS) ++++++++++++++++++++++++++++++++++++++++++++++++
        this.insert(dropInvent,_searchMarkup,true, _skipPurchTableUpdate,_skipInterCompanyCalcDisc, _skipUpdatePaymMode);
==== AX Update (SYP) ++++++++++++++++++++++++++++++++++++++++++++++++++
        ok = this.pdsCheckApprovedVendorList();
<<<< End
    }

// The following was merged successfully, but should be reviewed.
// Original (SYS) ++++++++++++++++++++++++++++++++++++++++++++++++++
// Yours (CUS) ++++++++++++++++++++++++++++++++++++++++++++++++
// AX Update (SYP) ++++++++++++++++++++++++++++++++++++++++++++++++++
    if (ok)
    {
        this.insert(dropInvent,_searchMarkup,true, _skipPurchTableUpdate,_skipInterCompanyCalcDisc);
    }
// End
}
```

Joonis 20. *Auto-merge* poolt lahendamata ning valesti ühendatud programmikood.

```
    if (_validation)
        if (!this.validateWrite())
        {
            throw error("@SYS18447");
        }

    if (TaxParameters::checkParameterForPosting_IN(TaxType_IN::Customs) &&
        PurchTable::find(this.PurchId).purchTable_W().CustomsImportOrder_IN == NoYes::Yes &&
        CustomsVendBOETrans_IN::findPurchLineRecId(PurchLine::findInventTransId(this.InventRefTransId).RecId).RecId != 0 &&
        this.PurchId == this.InventRefId)
    {
        ok = checkFailed("@GLS5764");
    }
    else
    {
        this.insert(dropInvent,_searchMarkup,true, _skipPurchTableUpdate,_skipInterCompanyCalcDisc, _skipUpdatePaymMode);
    }
}
```

Joonis 21. Programmikood parandatud kujul.

Lisa 5 – Koodikonflikti paranduse näide nr.3

```
// TODO: The following conflict requires manual resolution.
>>>> Original (SYS) ++++++++++++++++++++++++++++++++++++++++++++++++++

    ttscommit;
==== Yours (CUS) ++++++++++++++++++++++++++++++++++++++++++++++++++++
    VendTable::updatePaymMode(this);
    ttscommit;
==== AX Update (SYP) ++++++++++++++++++++++++++++++++++++++++++++++++

    // <PubSect>
    if (this.isActive())
    {
        this.updateVendInvoiceProjectFromBudgetRes(true, false);
    }
    // </PubSect>

    // <GTE>
    if (TaxSolutionScopeIntegrationUtil::isCompanyEnabled())
    {
        this.markCurrentTaxDocumentTaxStatusDirty();
    }
    // </GTE>
<<<< End

// The following was merged successfully, but should be reviewed.
// Original (SYS) ++++++++++++++++++++++++++++++++++++++++++++++++++++
// Yours (CUS) ++++++++++++++++++++++++++++++++++++++++++++++++++++++
// AX Update (SYP) ++++++++++++++++++++++++++++++++++++++++++++++++++
    ttscommit;
// End
}
```

Joonis 22. *Auto-merge* poolt lahendamata programmikood.

```
    VendTable::updatePaymMode(this);

    // <PubSect>
    if (this.isActive())
    {
        this.updateVendInvoiceProjectFromBudgetRes(true, false);
    }
    // </PubSect>

    // <GTE>
    if (TaxSolutionScopeIntegrationUtil::isCompanyEnabled())
    {
        this.markCurrentTaxDocumentTaxStatusDirty();
    }
    // </GTE>

    ttscommit;
}
```

Joonis 23. Programmikood parandatud kujul.