TALLINNA TEHNIKAÜLIKOOL
INFOTEHNOLOOGIA TEADUSKOND
INFORMAATIKAINSTITUUT
INFOSÜSTEEMIDE ÕPPETOOL

Pärandsüsteemi moderniseerimine E-Notari veebiteenuse põhjal

Magistritöö

Üliõpilane: Andrei Senkiv
Üliõpilaskood: 132463IAPM
Juhendaja: Deniss Kumlander
Infosüsteemide
õppetool, vanemteadur

Tallinn
2015

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.

---------- ----------

(*kuupäev*) (*allkiri*)

Annotatsioon

Tänapäeva kiire infotehnoloogia areng tekitab vajadust kohandada pärandsüsteeme uute väljakutsetega, mis on seotud tehnoloogiate muudatustega, skaleerimisega, hooldatavusega ja erinevate probleemidega, mis raskendavad adopteerida olemasolevatele süsteemidele uusi muudatusi. Pärandsüsteemina võib käsitleda E-Notari infosüsteemi, mis on võetud uurimustöö aluseks.

Antud töö eesmärgiks oli valida arengustrateegia pärandsüsteemi moderniserimiseks, valides analüüsimiseks olemasoleva veebiteenuse kihi, mis on realiseeritud ASMX Web Service abil ja käsitleda üle viimist WCF tehnoloogiale. Töös on esitatud testide tulemused kahe teenuse kohta ja võimalikud riskid seotud üleminekuga.

Töös käsitletakse Dual-Spiral Reengineering Model tehnoloogiat teenuse funktsionaalsuse ülekandmiseks, mis on kaetud testidega. Veebiteenuste operatsiooni täitmise teoreerilise kiiruse mõõtmiseks esitakse test-projekt, mille tulemuste alusel on teostatud võrdlus teenuste vahel.

Töö olulisemateks tulemusteks võib nimetada effektiivsust WCF teenuse kasutamisel võrreldes ASMX teenusega. Testimise käigus toimub WCF tehnoloogia puhul andmete üleandmine kaks korda kiiremini. Töös on ka näidatud puudused WCF teenusega DataSet objekti edastamisel ning ka eelised tava objektide kasutamisel.

Lõpptulemuseks on loodud WCF lahenduse realisatsioon, mis võib olla kasutatud edaspidiselt E-Notari süsteemi juurutamiseks.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 69 leheküljel, 6 peatükki, 8 joonist, 3 tabelit.

Abstract

In today's world of rapidly developing Information Technology, there is a need for adapting Legacy Systems to new challenges that have to do with changing technology, scalability, maintenance and other issues that could hinder the existing system from accommodating to new requirements. E-Notar System, which this thesis centers on, is taken as an example of such a Legacy System.

The purpose of the thesis in hand is to modernize Legacy System while focusing on the chosen Web Service layer, that was implemented using ASMX Web Service, and considering its migration to WCF technology. This paper contains test results for both services as well as assessment of possible migration risks.

In order to transfer the functionality of the Service, Dual-Spiral Reengineering Model with code coverage was used. A test project was run to assess the theoretical speed of a Web Service operation execution. Based on the results of the test project, a comparison of both services is drawn.

Among the most significant conclusions of the thesis one can mention the apparent efficiency advantage of WCF service over its ASMX rival. During the testing, the speed of data fransfer while using the WCF service is two times higher than the speed of data transfer using ASMX. Moreover, the thesis features the WCF Service's shortcomings in DataSet Object transfer as well as the advantages of using Custon Objects.

In the end, the WCF solution was implemented that can subsequently be applied to E-Notar System.

The thesis is in estonian and contains 69 pages of text, 6 chapters, 8 figures, 3 tables.

Lühendite ja mõistete sõnastik

EN	***E-Noraty system*** E-Notar infosüsteem
RIK	***Center of Registers and Information Systems*** Registrite ja Infosüsteemide Keskus
X-Tee	***X-Road*** Riigi infosüsteemide andmevahetuskiht
SOAP	***Simple Object Access Protocol*** Lihtne objektipöördusprotokoll
XML	***Extensible Markup Language*** Laiendatav märgistuskeel
SQL	***Structured Query Language*** Struktuurpäringukeel
CRUD	***Create, Reade, Update, Delete*** Looma, Lugema, Uuendama, Kustutama
DBMS	***Database Management System*** Andmebaasihaldur süsteem
API	***Application Programming Interface*** Rakendusliides
IDE	***Integrated Development Environment*** Integreeritud programmeerimisekeskkond
URL	***Uniform Resource Locator*** Internetiaadress
HTTP	***HyperText Transfer Protocol***

	hüperteksti edastusprotokoll
ORM	***Object-relational mapping***
	Objektidevahelise mapping
DAL	***Data Access Layer***
	Andmebaasi liidese kiht
COM	***Component Object Model***
	Komponentobjektide mudel
ASMX	***Active Server Method Extended***
	Veebiteenus
WCF	***Windows Communication Foundation***
	Programmidevahelise infovahetuse korraldamine
ITIL	***Information Technology Infrastructure Library***
	IT-teenuste halduse parima praktika kogum
ITSM	***IT service management***
	IT teenuse haldus
MOF	***Microsoft Operations Framework***
	Microsofti teenuse haldus raamistik
SOA	***Service-Oriented Architecture***
	Teenustepõhine arhitektuur
TDD	***Test-Driven Development***
	Testidel põhinev arendus
GUI	***Graphical User Interface***
	Graafiline kasutajaliides
SVN	***Subversion***
	Versioonihaldustarkvara

Jooniste nimekiri

Tabelite nimekiri

Sisukord

1. Sissejuhatus

Paljud organisatsioonid tänapäeval omavad mitmeid aegunud süsteeme ja peavad hoidma neid töökorras, et täita oma igapäevast äritegevust. Vanemad süsteemid ei saa alati rahuldada arenevate ettevõtete vajadusi, seetõttu võivad ka mõjutada negatiivselt näiteks organisatsiooni aktsiaid turul. Tuleb tunnistada, et peame piirama aegunud süsteemide kasutamist ja kaaluma uusi võimalusi arvestades käepäraseid vahendeid. Üks võimalik lahendus on asendada kogu süsteem täiesti uue valmis tootega. Teine võimalus oleks olemasoleva aegunud süsteemi arhitektuuri muutmine, parandamine või täiendamine. Kolmas variant võib olla näiteks olemasoleva süsteemi kapseldamine vahevara abil. Õige valiku tegemiseks oleks vaja uurida erinevate lahenduste sobivust, siis saame langetada lõpliku otsuse aegunud süsteemide parendamiseks.

1.1 Taust ja probleem

E-notar on keskkond, mis aitab notareid nende igapäevatöös ning võimaldab elektroonilist suhtlust notarite ja riigi vahel. Notaritele loodud keskkonnas saavad kasutajad teha kõik oma tööks vajaliku ning lisaks on süsteemi kaudu võimalik teha päringuid 16-nesse erinevasse registrisse (näiteks Abieluvararegister, Ametlikud teadaanded, Eesti Väärtpaberi Keskregister, Ehitisregister, Kinnistusraamat, Liiklusregister, Maakataster, Pärimisregister, Rahvastikuregister, Väikelaevade register ja Äriregister). Notarite Koda on süsteemi omanik ning Registrite ja Infosüsteemide Keskus haldab servereid, pakub kasutajatuge, koolitab kasutajaid ja arendab süsteemi [1].

E-notari infosüsteem on Notarite Kojale kuuluv infosüsteem, mis on aastal 2007 võetud kasutusele ja tänasepäeval areneb edukalt edasi. Aastal 2010 oli loodud E-notari andmekogu kasutades X-tee teenuseid. Lõpptulemusena on Riigiportaalis igal kodanikul võimalus näha teenust nimega „Minuga seotud notariaalsed tehingud”, mis sisaldab kõiki antud isiku poolt ükskõik millise Eesti notari juures tehtud tehinguid digitaalsel kujul [2].

E-notar on notarite infosüsteem, mis on loodud notarite, notaribüroo töötajate ning ka erinevate registriosakondade töö kergendamiseks. Tänu E-notarile on notarite töö jaoks kõik vajalik (lepingud, notariaaltoimingute raamat, kalender, tehingud, arved, apostillid, hoiused

jne) võimalik ära teha ühest kohast – E-notari infosüsteemist. E-notari infosüsteemi üheks ideeks on kiirendada notarite tööd päringute tegemisel teistesse registritesse. E-notaris toimub erinevatesse andmekogudesse registripäringute sooritamine ja/või ka registrikannete edastamine X-tee kaudu lihtsalt ja mugavalt.

Kasutajad

E-notar programmi saavad kasutada ainult notarid ja notaribüroo töötajad (notari asendajad, juristid, sekretärid, registraatorid ja arhivaarid).

E-notar võimaldab:

- pidada notari päevakava ja kasutajate isiklikku kalendrit
- teha usaldusväärseid päringuid riiklikest registritest
- koostada lepinguid ja neid digitaalselt allkirjastada
- registreerida ametitegevusi (ametitoiminguid ja –teenuseid)
- kanda sisse hoiuseid (deposiite)
- väljastada apostille
- koostada arveid ja riigilõivude maksekorraldusi
- edastada kandeid riiklikesse registritesse

Eesmärgid:

- olla vahendiks, mis võimaldab sooritada kõiki vajalikke toiminguid notaribüroos
- luua ühtses kõrge turvalisusega keskkonnas notariaalaktide register
- minimeerida andmete topeltsisestust, paberitööd ja printimist nii notaritel kui ka registri pidajatel
- panustada teiste e-süsteemide arendusse ja täiendamisse
- muuta kliendile notariga asjaajamine lihtsamaks - erinevalt varasemast ei pea klient notarile eelnevalt esitama enam suurt hulka paberdokumente
- E-notari kaudu saab notar ise tutvuda vajalike andmetega [1].

Selleks, et E-Notari süsteem võimaldaks täita notarite ja notaribüroo töötajate kõiki vajadusi ja saavutada püstitatud eesmärke, on vaja tagada süsteemi töökindlus ja hoida süsteemi töökorras.

Põhiprobleemideks E-Notari süsteemis on aegunud tehnoloogia, arhitektuurilised lahendused, koodi kvaliteet, automaattestide puudumine ning süsteemi on raske sisse viia muudatusi. Järgnevalt uuritakse detailsemalt erineva taseme probleeme.

1.2 Ülesande püstitus

Suuri süsteeme, nagu on E-Notar, tuleb käsitleda terviklikuna. Muudatused süsteemis võivad mõjutada kogu süsteemi töötamist. Selleks, et teha muudatusi on vaja kinni pidada konkreetsest arendustrateegiast, kasutades vastavalt üldtunnustatud praktikat või metoodikat. Uurimistöö aitab vastata kahele kriitilisele küsimusele E-Notar süsteemi arendamisel:

Mis on kõige parem arengustrateegia veebiteenuste moderniseerimiseks?

Kuidas rakendada valitud arengustrateegiat veebiteenuste moderniseerimiseks?

Samas töös esitakse võimaliku lahendusena üle viimist ASMX Web service tehnoloogiast WCF peale, võrreldes neid tehnoloogiaid omavahel ning uuritakse millised riskid kaasnevad tehnoloogia vahetusega.

1.3 Metoodika

Pärandsüsteemi edasi arendamiseks on välja pakkutud erinevad metoodikad, mis võivad olla kohandatud või osaliselt võetud aluseks E-Notari süsteemi moderniseerimiseks. Uurimustöös on aluseks võetud A Dual-Spiral Reengineering Model for Legacy System ja RENAISSANCE metodoloogia, kui ka testimiseks TDD.

1.4 Ülevaade tööst

Uurimustöö alguses esitakse süsteemi üldine kirjeldus: olemasoleva süsteemi arhitektuur, kasutatavad arendusvahendid ja kaasatud probleemid erinevatel tasemetel. Samuti lisatakse teoreetiline aspekt infosüsteemi arendamises, kus pannakse rõhk süsteemide muutmisele ja toetusele, sest töö kontekstis on aluseks võetud pärandsüsteem. Pärandsüsteemi muudatusi tuleb teostada ainult siis, kui on ehitatud ka korralik testkeskond. Pakutakse infosüsteemi hoolduse strateegia ja veebiteenuste tehnoloogiate võrdlust koos realisatsiooniga test projekti abil. Uurimustöö lõppeb näidisega kuidas juhtida riski pärandsüsteemi moderniseerimise käigus.

2. Pärandsüsteemi ülesehitus

Edukus pärandsüsteemis sõltub paljudest faktoritest – otsused, mis olid tehtud minevikus, valitud arendusvahendid, arhitektuurilised mustrid, tehnoloogia, kvaliteedi kontroll, hooldavus. Õigesti ülesehitatud süsteem võimaldab pidevat arengut olemasolevas süsteemis, kuid aja jooksul võib hooldamine muutuda keeruliseks, mis võib olla kvaliteedi languse põhjuseks. Kahjuks ei saa uusi tehnoloogiaid implementeerida pärandsüsteemi.

E-notari projekt on realiseeritud kasutades Microsoft *.NET* raamistikku.

E-notari infosüseem omab kolmekihilist arhitektuuri (three-tier client server architecture): esitlusloogika kiht, äriloogika kiht ja andmekiht. E-Notari iga kiht asub erinevates kohtades arvutivõrgus.

Esitusloogika kiht on realiseeritud Windows Forms Application abil – klient (*fat client*), mis vahendab kasutaja ja serveri vahel infot. Kasutajaliidese põhikomponendiks on Janus WinForms Controls Suite v3.5 (Windows Forms Controls for Microsoft .NET). Esitlusloogika kiht suhtleb rakenduse (äriloogika) kihiga https protokoli kaudu.

Äriloogika kiht on realiseeritud ASMX Web service abiga, mis töötab IIS(Internet Information Server) abil. Rakendusserver suhtleb andmekihiga, saates päringuid erinevate registrite vahel X-Tee kaudu ja saadab kliendile sõnumid tagasi.

Andmekiht on rakenduse liides, mis ühendab suhtlemise rakendusserveri ja andmebaasi vahel. Antud kontekstis E-Notar kasutab Microsoft SQL Server 2012 andmebaasi tarkvara. Data Access Layer on realiseeritud SQLHelper-i abil, mida pakub Microsoft Data Access Application Block for .NET.

2.1 Töö reziim

Infosüsteem võimaldab töötada kahes reziimis – online ja offline. See on tehtud selleks, et infosüsteemi töö ei sõltuks võrgu ühendusest ja võimaldaks teha tööd minimalse funktsionaalsusega.

2.1.1 Online mode

EN infosüsteemi võimalik sisse logida ID-kaardi , Mobiil-ID või Justiitsministeeriumi välja antud sertifikaadi abil. EN kliendirakendus loob ühenduse rakendusserveriga üle https protokolli. Rakendusserver vajab kliendi sertifikaati autentimiseks infosüsteemis. Uue kasutaja lisamine toimub administraatori liidestuse ja vastava administraarori rolli kaudu.

Pärast autentimist, tekivad kasutaja masinas krüpteeritud kaustas abistavad lisakaustad: tehingupõhjad ja dokumendipõhjad, kuhu laetakse alla (uuendakse iga sisse loogimisega) töötaja bürooga seotud tehingupõhjad ja dokumendipõhjad. Allalaadimise protsess, kui ka salvestamine toimub tänu *asünkroonülekandele* (selle kasutamist võimaldab class BackgroundWorker) ja ei takista üldist kasutaja tööprotsessi.

2.1.2 Offline mode

Oluline omadus EN infosüsteemis on võimalus notari töötajatele oma tööd teha võrguühenduseta (*autonoomne, offlain*) reziimis. See tähendab, et süsteemil ei ole ligipääsu rakendusserverile ega muudele võrguteenustele – näiteks olukord, kus arvutil puudub internetiühendus. Tänu sellele, et lokaalses masinas on salvestatud tehingupõhjad ja dokumendipõhjad, on notari töötajal võimalus ette valmistada tehing offline reziimis ning internetiühenduse taastusmisel luua uuesti ühendus rakendusserveriga ja salvestada tehing andmebaasi.

Tehingu salvestamiseks kasutakse *DataSet* andmetüüpe, mis võimaldavad salvestada andmeid XML formaadis kõvakettale kasutaja profiili krüpteeritud kausta.

2.2 Pärandsüsteemi kasutavad vahendid

E-Notar infosüsteemi arendus oli alustatud kasutades .NET raamistiku versiooni 1.1. IDE aluseks võeti Microsoft Visual Studio 2003. Relatsioonilise andmebaasi juhtimissüsteemiks valiti alguses Microsoft SQL Server 2005. Tänaseks on süsteem üle viitud juba .NET raamestikule versiooniga 4.5, mis on uurimustöö kirjutamise hetkel viimane versioon. Arendusvahend IDE on uuendatud Microsoft Visual Studio Professional 2012 versioonile ning koodi refaktoreerimise, navigeerimise ja süntaksi kontolli jaoks kasutakse ReSharper abiliidest. Dokumentide genereerimiseks ja „pdf“ formaati salvestamiseks kasutakse vaba tarkvara nimega „PDFCreater“. EN kliendis on ka juurutatud MS Wordi

funktsionaalsus kasutades API-t. Dokumendi allkirjastamiseks kasutakse „DigidocService“ teenust mida pakutakse Sertifitseerimiskeskuse poolt. Kuna EN liidestatud mitme erineva registriga, siis päringute serialiseerimiseks kasutakse „Xsd2Code“ klassi generaatorit „CSharp“, mis genereerib klassi antud registrite teenuste kasutamiseks XSD skeemast.

2.3 Pärandsüsteemi probleemid

Paljud ettevõtted on silmitsi ka olemasoleva süsteemi integratsiooni probleemiga. Sellised süsteemid on tavaliselt vanemad ja mitte paindlikud ning ei sulandu sujuvalt uute tehnoloogiatega, veebirakendustega või mobiilsete rakendustega. See raskendab kasvatada uut ettevõttet ning ohustab ettevõtte edasist kasvu ja laienemist. Kahjuks paljudes ettevõtetes olemasoleva süsteemi asendamine ei ole reaalne lahendus. See on lihtsalt liiga kallis, riskantne ja keeruline. See jätab palju kinnisi otsi ning ei suudeta leida ideaalseid lahendusi [12].

Süsteemi disain, mis ei talu muudatusi on halva disaini tulemus. Iga pädeva tarkvara arendaja eesmärgiks on ehitada ja disainida süsteeme, mis taluvad muudatusi. See nõuab vastavaid teadmisi ja kogemust tarkvara arenduses.

Tuleb tunnistada, et peaaegu iga süsteem, mis on kunagi toodetud ja mis on arenduses või hoolduses, kannatab ebapädevate muudatuste, valede arhitektuuriliste otsuste all. Tarkvara arendaja puutub tihti kokku koodiga, mis on kirjutatud mitme erineva tasemega arendajate poolt. Tulemuseks on spagetti-kood (väga pikk meetodi funktsionaalsus, koodi dubleerimine), või kood, mis ei ole kaetud testiga - seda võib nimetada ka kui Legacy Code [3].

Olemasolevates süsteemides ei ole nii suuri probleeme, et oleks vaja ümber ehitada või rakendada suuri arhitektuurilisi muudatusi, vähemalt tellija poolt ei ole seatud selliseid nõudeid. Samal ajal on vaja tähele panna koodi kvaliteeti erinevatel tasanditel – võimalus uuendada kliendi poolt registripäringu kuvamist, importimine ja registrikande saatmise loogika. Samuti on vaja analüüsida ja optimeerida ka olemasolevad aeglased SQL päringud.

Antud töö eesmärk on anda soovitusi kuidas võiks moderneseerida olemasolevaid süsteeme, vastavalt üldtunnustatud metoodikatele. Moderniseerimine peab toimuma kolmel tasemel: kliendi, serveri ja andmekihi tasemel.

2.3.1 Kliendi uuendused

Probleem on seotud suures osas koodi dubleerimisega ja vormidega mis ei ole kasutusel. See on seotud sellega, et E-Notar on liidestud paljude registitega, mis pidevalt muudavad oma teenuseid ja pakuvad uusi teenuseid. Süsteem peab kuvama nii vanu kui ka uusi registripäringu vastuseid. Tuleb välja selgitada kas on võimalik rakendada „User Interface“ ümberkujundamine põhinedes mudeli kirjeldamisele või rakendada iga registri liidestuseks eraldi mudel, mida võiks kasutada kliendis andmete kuvamiseks, importimiseks ja kande saatmiseks.

Hetkel ei ole teada, milliseid kasutajaliidese komponente võiks kasutada WPF puhul. Täielikust UI uuendamisest ei saa rääkida, kuna Janus WinForm Controls Suite v3.5 versioon ei toeta WPF tehnoloogiat. Samas on võimalik WPF komponente rakendada Windows Forms-i arendamisel. Seda hakati juba kasutama ühe projekti raames ning seda võiks ka edasi arendada. WPF kasutab XAML (eXtensible Application Markup Language) tehnoloogiat kasutajaliidese kirjeldamiseks. Tuleb uurida ka seda, kas tulevikus on vajadust EN süsteemis võrguühenduseta töö järele. Kui mitte, siis ei ole vajadust arendada EN süsteemi Windows Forms tehnoloogiat vaid on võimalik teha EN veebipõhiseks süsteemiks.

Nagu oli mainitud, on E-Notar tihedalt seotud Janus WinForm Controls Suite v3.5 komponendiga ja hetkel on kättesaadaval ka uuendatud versioon. Tuleb välja selgitada kas Notari Koda on valmis uuendama GUI komponenti Janus WinForms Controls Suite v4.0 peale? See nõuab uue litsensi soetamist ja kõikide notarite kliendirakenduse uuendamist. Tegemist ei ole suure muudatusega, kuid siin võivad olla riskid, mis on seotud kasutaja kontrolli käitumisega või kliendi sündmuste töötlemistega.

2.3.2 Serveri uuendused

Nii nagu kliendi puhul, tekkis vajadus ka serveri teenuste refaktoreerimiesele. Osa teenuse meetoditest ei ole kasutusel ja osa koodi on dubleeritud, mis on kindlasti üks oluline mälulekke probleem.

E-Notari süsteem kasutab palju erinevaid allsüsteeme: notari kalender, dokumendi register, aruande register, e-maili register, tehingu register ja teisi. Tänaseks päevaks on see kõik ehitatud ühe monoliit-veebi teenuses - ASMX Web service. Võiks uurida tulevikus SOA

arhitektuuri, mikro teenuste [10] kasutamist või WCF (Windows Communication Foundation on uus meetod programmidevahelise infovahetuse korraldamiseks).

Viimased E-Notari laiendused, nagu näiteks T-Notar (tahvliarvuti peale dokumendi jagamine) ja K-Notar (notari kalender mobiil app-i) kasutavad WCF-i oma teenuste pakkumisel.

Kas on mõistlik ASMX Web service asemel kasutada WCF tehnoloogia, või Open Source ServiceStack? Tänapäevaks Notarite Koda ei käsitle pilvetehnoloogia teenuste kasutamist ebapiisava turvalisuse tõttu ja piirangute tõttu, mis tulevevad seaduseandlusest.

2.3.3 Andmekiht uuendused

Arvatavasti kõige suurem ja raskesti teostav muudatus on seotud andmekihi uuendustega (E-Notar DBMS on Microsoft SQL Server 2012). Microsoft Data Access Application Block on aegunud tehnoloogia – kõik baasi muudatused kaasatakse suurte muudatustega protseduurides. Kirjutakse iga tabeli jaoks eraldi insert, update, delete, select protseduurid. Esimene sammuna võiks võta kasutusele Microsoft Enterprise Library 6, mis sisaldab uuendatud versiooni Data Access Application Block-ist. Samas, tulevikus oleks mõistlik kaaluda ORM (Object-Relational Mapping) vahendite nt. NHibernate, Entity Framework, Linq to SQL, Telerik Data Access kasutamise võimalusi. See toob kaasa mahukama töö, mis on otseselt seotud klasside ja tabelite *mapping*-utega.

Tänapäevaks on kõik eelmainitud tehnoloogiad juba aegunud. Iga tehnoloogia eesmärk on lihtsustada, ühtlustada ja automatiseerida protsesse oma vastavas ärivaldkonnas. Analüüsida on vaja korraga kõiki kihte ja valida soobiv strateegia infosüsteemi muutmiseks. Vastastikune seos on väga tugev ja eraldi vaadatud muudatused võivad põhjustada kasutaja töös tõrkeid. Kõige tipuks on vaja teha detailne analüüsi iga kihi kohta, tuvastada kriitilised kohad ja riskid, mis on seotud muudatustega.

2.3.4 Koodi maht

Kasutades Visual Studio 2012 sisse ehitatud ’Calculate Code Metrics’, tulemuseks on saadud 234038 rida koodi. Kuna süsteemi arendakse pidevalt edasi ja lisatakse uut funktsionaalsust, siis koodi maht pidevalt ka suureneb. Suurem osa koodi ei ole kaetud testidega ja muudatuste sisse viimine kaasaks alati riski.

3. Teoreetilised aspektid

Legacy system (pärandsüsteem) on rakendus, mis töötati välja vanemate tehnoloogiate abil ja jääb kasutusele, kuid on liiga kriitiline äri jaoks.

Pärandsüsteem on arvutisüsteem, mis on kasutusel olnud juba pikka aega. Sageli on firmas mitu pärandsüsteemi, mis on loodud eri aegadel, erinevate inimeste poolt ja erinevaks otstarbeks. Andmed neis süsteemides on enamasti omavahel ühildamatud ja vahel ka ebatäpsed. Andmeaidanduse üks põhiülesandeid on kõigi pärandsüsteemides asuvate andmete kokkukogumine, nende korrastamine ja kättesaadavaks muutmine [5].

Pärandandmed on ühe organisatsiooni kogutud ja teisele siirdunud andmed [5].

Pärandrakendus – pärandrakendusteks ja –andmeteks peetakse infotehnoloogias neid rakendusi ja andmeid, mis on pärit vanematest keeltest, platvormidest ja riistvarast. Enamik ettevõtteid moderniseerib järk-järgult oma arvutiparki ja tarkvara, kusjuures on vaja, et pärandrakendused ja -andmed oleksid kasutatavad nii uues kui vanas keskkonnas (enamasti ei moderniseerita kõiki arvuteid korraga) [5].

Need süsteemid ei ole tavaliselt paindlikud muudatuste suhtes ja muutes sügavalt juurdunud äri funktsionaalsust on palju olulisemad ettevõtte andmed [6].

Mõiste "olemasolev süsteem" selles uurimis töös viitab töötavale süsteemile, mis vastab endiselt märkimisväärselt suurel protsendil organisatsiooni või ettevõtte vajadustele, lähtudes funktsionaalsetest aspektidest, kuid ei vasta arenevate arhitektuurilitsele standarditele.

Tuginedes arhitektuurilise stiili vastukorraldusele, saame liigitada süsteemi – kas see on olemasolev süsteem või mitte, olenemata funktsionaalsetest aspektidest, sest me eeldame, et kõik süsteemid on organisatsioonis kasutusel, vastasel juhul oleksid nad suletud. Näiteks töölaua rakendusi (*desktop application*), mis põhinevad klient-server arhitektuuri mustril võib pidada olemasolevateks süsteemideks, kui need on veel kasutusel suurettevõtes või organisatsioonides nagu Notari Koda. Põhjuseks on see, et klient-server lahendustel on palju piiranguid seoses uute tekkivate notari nõuete ulatuslike kohandamistega. Seega süsteemi võib pidada olemasolevaks süsteemiks ühes kontekstis, kuid teises mitte.

Summerville [7] on kindlaks määranud kaks peamist mõõdupuud hindamaks olemasolevaid süsteeme: keskkond ja rakendus. Keskkonna hindamine on seotud teatud vendor- ja tehnoloogiaga seotud aspektidega. Rakenduse hindamise, teisalt, uurib sisemist funktsionaalsust ja mittefunktsionaalseid nõudeid.

Seacord pakub kolm peamist mõõdupuud hindamaks olemasolevaid süsteeme: tehniline, programmiline ja organisatsioonilised kaalutlused. Neid tuleb põhjalikult uurida enne seda kui kaasajastame (*modernizing*) mis tahes olemasoleva süsteemi [8].

Bennett keskendub kahe etapilisele otsuse mudelile. Organisatsioonide abistamine teha teadlikke otsuseid oma olemasolevates süsteemides. Nende mudel on seotud äristrateegia ja tehniliste aspektidega olemasolevas süsteemis [9].

Iseloomustakse olemasolevat süsteemi tavaliselt kontekstis:

Süsteemi konfiguratsioon (*System configuration*) – viitab tarkvara/riistvara konfiguratsiooni keskkonnale, kus on paigaldatud olemasolev süsteem;

Süsteemi arhitektuur (*System architecture*) – lagundame olemasoleva süsteemi komponentideks ja tuvastame integratsiooni taseme välise tarkvara komponentidega;

Süsteemi disain (*System design*) – teeme kindlaks tehnilise dokumentatsiooni olemasolu, selle dokumentatsiooni usaldusväärsuse, täielikkuse ja nõudlikkuse viitega praeguses versioonisüsteemis;

Kasutajaliides (*User interface*) – kirjeldab milline on süsteemi kasutajaliidese tüüp ja seisund;

Süsteem liidesed (*System interfaces*) - kaaluda liidese tüüpe olemasolevate süsteemide vahel, iga sisemise pärandkomponendi paari vahel ja kättesaadavust dokumentatsiooni ja lähtekoodi osas;

Toimimine (*Performance*) – kaaluda tõhusust seotud olemasoleva süsteemi kasutamisel tööprotsessis, kuhu see süsteem areneb;

Lähtekood (*Source Code*) – osutame lähtekoodi, kas on kättesaadav, mis on koodi seisund, kas olemasolev kood on seotud dokumentatsiooniga, või kood ongi dokumentatsioon;

Eelmine hooldus (*Previous Maintenance*) – tuvastame kas eelmisse töösse sekkumine on

toimunud ja kas uuendused on dokumenteeritud.

Eespool kirjeldatud töö keskendub peamiselt äri aspektidele olemasolevates süsteemides, et toetada Notari Koda tegemaks teadlikke otsuseid selle kohta, kas vahetada välja kogu süsteem uue vastu või hoida olemasoleva süsteemi funktsionaalsust ja säilitada seda pikema aja jooksul.

3.1 Tarkvara evolutsiooni seadused

Lehman ja Belady Software Evolution (Tarkvara Evolutsiooni, evolution type (E-tüüp) systems) kaheksa seadust on:

1. Muudatuste jätkamine (*Continuing Change*) – E-tüüpi süsteemid peavad jätkuvalt (pikema aja perioodi jooksul) kasutusse tulema. Vastasel juhul muutuvad nad järjest vähem kasulikuks.

2. Kasvav keerukus (*Increasing Complexity*) – E-tüüpi süsteemid arenevad, nende keerukus suureneb, kui ei võeta ennetavaid meetmeid ette nende keerukuse vähendamiseks või tasakaalustamiseks.

3. Eneseregulatsioon (Enesereguleerimine) (*Self-Regulation*) – E-tüüpi süsteemi muutumise kiirus on isereguleeruv väikeste negatiivsete ja positiivsete kohandustega, suunaga normaaljaotuse poole süsteemi eluea jooksul (st E-tüüpi süsteemide kasv paratamatult aeglustub selle vananemisega).

4. Ülesehitusliku stabiilsuse kaitse (invariantne/muutumatu töökiirus) (*Conservation of Organizational Stability (invariant work rate)*) – Areneva/kasvava E-tüüpi süsteemi muudatuste keskmine kiirus kipub jääma samaks (invariant) süsteemi eluea jooksul.

5. Äratundmise (tutvusastme) kaitse (*Conservation of Familiarity*) – süsteemi kasvu järjepidevuse määr kipub jääma muutumatuks/samaks (statistiliselt invariantseks) või kahanema, sest arendajad peavad mõistma programmi lähtekoodi või käitumist selleks, et seda muuta. Liigne kasv vähendab süsteemi meisterlikku oskamist. Seega, keskmine kasv jääb invariantseks või kahaneb süsteemide arengu jooksul.

6. Jätkuv kasv (*Continuing Growth*) – E-tüüpi süsteemid tavaliselt arenevad aja jooksul muutuste vajaduste ning kasvava nõudmiste hulga surve all. Seega kulud hooldusele kipuvad paratamatult tõusma süsteemi eluae jooksul selleks, et säilitada kasutajate rahulolu.

7. Langev kvaliteet (*Declining Quality*) – E-tüüpi süsteemi kvaliteet näib kahanev olema, kui mitte rangelt rakendada ning arendada süsteemi arvestades operatiivse töökeskkonna muutusi.

8. Tagasiside süsteem (*Feedback System*) – protsess, mille kaudu E-tüüpi süsteemid arenevad on piiratud reaalse maailma töö dünaamika poolt (st need on mitmetasandilised, multiagent tagasiside süsteemid). Protsess, mis reguleerib nende muutumise kiirust paratamatult tõstab ka inimeste arengutaset, kelle otsuseid, taju, kalduvusi, kogemusi ja kalduvusi tuleb mõista, haarata ja reguleerida selleks, et saavutada olulist paranemist, kui tegelikult suurim osa tagasisidet on planeerimata, alateadvuslikku laati ja raskesti kontrollitav [6].

3.2 Infosüsteemi muudatused

Põhjused miks toimuvad muudatused tarkvara arenduses on:

1. Funktsionaalsuse lisamine
2. Vigade parandus
3. Disaini parandus/täiendus
4. Ressursikasutuse optimeerimine

Uue funktsionaalsuse lisamine on süsteemis kõige olulisem muudatus. Töökindel talitlus on kõige tähtsam tarkvara omadus. See on see, millest sõltuvad kasutajad. Kasutajad peavad oluliseks, kui lisatud uus funktsionaalsus töötab ja vastab nende vajadustele ning lihtsustab töö protsesse (eeldusel, et see on see, mida nad tegelikult tahtsid). Samal ajal muutes või eemaldades olemasoleva süsteemi käitumist, millest nende tööptotsess sõltub, tekitades vigu, viib selleni, et kasutajal lõppeb usaldus süsteemi vastu ja ta hakkab otsima alternatiive.

Vigade parandus on protsess mille käigu üritakse kõrvaldada vigu, rikkeid või tõrkeid tarkvaras või süsteemis sees. Veaks loetakse kõike, mis põhjustab toode ebaõiget käitumist või oodatust erinevate tulemuste saamist. Suurems osas tekivad vead ikkagi kirjutatud koodi või vale disaini tõttu. Samuti võib põhjuseks olla ka valitud arendusraamistikud või operatsioonisüsteem, kus rakendus töötab ehk hoopis kolmanda osapoole – tootja poolne viga.

Disaini parandamine on eri liiki tarkvara muutus millele kaasneb ka tarkvara struktuuri muudatus, et muuta selle hooldatavust. Kuigi üldiselt püütakse hoida süsteemi käitumist puutumata. Üks põhipõhjustest miks paljud arendajad ei hakka parandama või muutma disaini, on see, et on oht kaotada protsessi käitumine või oht sisse viia uusi vigu olemas olevasse protsessi.

Protsessi, mis on seotud koodi disaini parandamisega kahjustamata rakenduse käitumist nimetatakse – refaktoreerimiseks (*refactoring*). Refaktoreerimise eesmärk on lisada tarkvarale rohkem hooldatavust, loetavust ja arusaadavust. Refaktoreerimise aluseks peab olema korralikult ülesehitatud testimise süsteem, kus on kirjutatud testid (E-Notar projekti kasutakse testide kirjutamiseks Nunit raamistikku). Selleks, et tagada muutumatut käitumist, kirjutakse testid olemasoleva koodi muutiseks ja liigutakse väikeste sammudega, et kontrollida seda kogu protsessi jooksul. Refaktoreerimine aitab tagada puhta koodi (*clean up*).

Süsteemis *clean up* koodi teostakse tavalislt jooksvalt, vähemalt seda eeldatakse igalt koodi kirjutajalt. Üsna tihti refaktoreerimine jätakse viimaseks etapiks arenduses, või ei teostata üldse. Refaktoreerimine erineb tava clean up koodist väikeste struktuuri muudatuste ja toetatud testide kirjutamise poolest vastavalt muudetud koodis.

Optimeerimine on nagu refaktoreerimine, aga sellel on hoopis teine eesmärk. Mõlemal juhul teostatakse muudatusi säilides funktsionaalsust. Erinevus on selles, et refaktoreerimise puhul muudatused annavad parema hooldatavuse, lihtsustavad koodi lugemist ning kõrvaldatakse spaghetti code. Optimiseerimisega püüakse saavutada rakenduse väiksemat ressursikasutust nagu näiteks mälu, protsessori või ka käsu teostamise aja võitu.

Ühisosa refaktoreerimise ja optimiseerimise vahel on säilitada funktsionaalsuse invariantsus, kui sisse viiakse muudatusi. Muudatused võivad olla eristatud kolmeks liigiks: strukturaalsed, funktsionaalsed ja ressursside kasutamised.

Uue funktsionaalsuse lisamine, vigade parandamine, refaktoreerimine ja optimeerimine peab säilitama olemasoleva funktsionaalsuse invariandi. Uue funksionaalsuse lisamine ja vigade parandamine on väga sarnane refaktooringuga ja optimiseerimisega, selles mõttes, et muudatused otseselt või kaugselt mõjutavad olemasolevat koodi. Muutes funksionaalsust või käitumist, on eesmärk ikkagi säilitada käesolev käitumine.

Üldtunnustatud lähenemise viis infosüsteemi arendamiseks ja muutmiseks on TDD (*Test-driven development*).

3.3 Testidel põhinev arendus

Test - Driven Development (TDD) on tarkvaraarenduse praktika, kus arendajad käsitlevad väikse osa nõuedeid ning kirjutavad ja käivitavad ühikteste, mis algul ebaõnnestuvad, sest nõuded on realiseerimata. Nõuete realiseerimisel ja uuesti ühiktestide käivitamisel veendutakse, et testid õnnestuvad [13] [14].

Ühiktestid (*unit test*) on üldiselt kirjutatud detailsuses väikseima lahutatava moodulina, mis on tavaline funktsioon enamikel juhtudel. Hiljuti empiirilised tõendid on näidanud, et TDD võib vähendada *pre-release* vigade tihedust kuni 90 %, võrreldes teiste sarnaste projektidega, mis ei rakenda TDD [15].

Lisaks muud uuringud näitasid, et TDD aitab toota parema kvaliteediga koodi, parandada programmeerija tootlikkust ja tugevdab arendaja kindlust oma koodis. Enamik varasemad uuringud on praeguseks kaalunud TDD kasutamist uue tarkvara arendamiseks. Kuid varasemad uuringud on näidanud, et enam kui 90% tarkvaraarenduse kuludest on kulutanud hoolduse ja arengu tegevustele [15].

Teised uuringud on näidanud, et keskmiselt Fortune 100 firma hooldab 35000000 rida koodi ja see hooldatud koodi hulk eeldatavasti kahekordistub iga seitsme aasta tagant. TDD-i on väga kasulik kohandada projektis juba rakendatud koodi säilitamiseks ja testimiseks ja eriti pärandsüsteemides. Selles töös nimetakse seda Test Driven Maintenance (TDM) [15].

TDM kasutamine on väga oluline vanades süsteemides, sest toimub pidev arendus olemasolevas süsteemis ja vana koodi suuruse maht ainult kasvab hooldatavas koodis. Lisaks nende vanaduse tõttu, pärandisüsteemides ei ole piisavalt dokumentatsiooni ja nad muutuvad hapraks ja on vigadealdis. Seetõttu tuleks kasutada TDM-i nendes pärandsüsteemides, et nõuded oleksid täidetud kvaliteedi tagamiseks ja et vähendada rikete võimalust evolutsiooniliste muudatuste käigus. Pärandsüsteemid on tavaliselt suured ja kogu päransüsteemile korraga ühiktestide kirjutamine võtab kaua aega ja on praktiliselt võimatu [15].

Et leevendada seda küsimust, TDM kasutab sama - jaga ja valitse ideed TDD. Kahjuks

selle asemel, et keskenduda mõnedele ülesannetele nõuetest, arendajad, kes rakendavad TDM, isoleerivad pärandsüsteemi funktsioone ja testivad neid eraldi. Fuktsioonidele kirjutatakse järk-järgult ühikteste kuni saavutavatakse soovitud kvaliteedi eesmärk. Astmeliselt kirjutatud ühiktestide idee on väga praktiline ja sel on kolm peamist eelist.

- Esiteks see annab piiratud resserssidega juhatajatele natuke hingamisruumi ja toimub ressursside jaotus (st see leevendab vajadust pikaajalise ressursi kohustuste järele).
- Teiseks, arendajad saavad rohkem tutvuda pärandkoodiga läbi ühiktestide kirjutamise.
- Kolmandaks, ühikteste saab hõlpsasti säilitada ja ajakohastada tulevikus, et tagada pärandsüsteemi kõrge kvaliteet.

Isegi pärast suurte pärandsüsteemide funktsioonide isoleerimist jääb küsimus, et kuidas tähtsustada ühiktestide kirjutamist, et saavutada parima investeeringu tasuvuse püsivust.

- Kas me juhuslikult kirjutame ühiktestide ülesandeid?
- Kas me kirjutame ühikteste funktsioonidele, mida viimati lõime?

Kasutades õiget prioriteetide strateegiat võib päästa arendajate aega, päästa organisatsiooni raha ja suurendada üldist toodete kvaliteeti.

Kasutades TDM, peab arendusmeskkond isoleerima funktsionalsust *legacy code* ja kirjutama *unit testid* nende jaoks. Otsus mis teste kirjutada on väljakutset pakkuv probleem ja see peab meskkonnale andma vastuse millised testi on vajalikud.

Ühiktestide kirjutamine kogu koodile on peaaegu võimatu. Näiteks, kui meeskonnal on piisavalt ressursse, et kirjutada ühikteste kvaliteedi hindamiseks – 100 rida süsteemi koodi päevas, siis ühiktestide kirjutamiseks 1000000 koodirealisele (LOC – line of code) süsteemile võtaks üle 27 aasta.

Primitiivne lähenemine – on juhuslikult valida funktsioone ja kirjutada ühikteste või kirjutada testide funktsioone, mis on hiljuti töötanud, kuid selline lähenemine ei ole väga tõhus. Mõndasid hiljuti töötanud funktsioone hiljem kasutatakse harva, samas kui teised on nii lihtsad, et ühiktestide kirjutamine ei ole prioriteediks.

Ühiktestide kirjutamise tähtsustamine pärandsüsteemis. Et aidata meeskondi arendamisega ja testimisega, esitakse lähenemine, mis kasutab projekti ajalugu, tähtsustades

pärandtarkvara süsteemile ühiktestide kirjutamist. Kasutades väljavõetud projekti ajaloost heuristikate lähenemist soovitatakse prioriteetsete funktsioonide nimekiri ühiktestide kirjutamiseks. Nimekirja suurust saab kohandada ressursside kogusega ja kindla ajaga. Lähenemine seisen selles, et tuleb uuendada kasutatud projekti ajalugu ja jätkata soovitatud funktsioone, et kirjutada ühikteste projekti edenemise jaoks.

Oma olemuses E-Notari süsteem kasutab testidel põhinevat arenduse metoodikat projektides NotarWS.Tests ja NotarWS.Util.XTee.Tests. Mõlemad test-projektid kasutavad NUnit raamistikku. NotarWS.Tests projekt on mõeldud DAL (Data Access Layer) üldfunktsionaalsuse testimiseks, mis on seotud CRUD loogikaga. Projekt on loogiliselt jaotud all-testideks, mis vastutavad konkreetselt ühe komponendi funktsionaalsuse testimise eest (nt. TehingDALTest, ArveDALTest, KalenderDALTest). Testidega on kaetud praktiliselt kogu DAL funktsionaalsus.

NotarWS.Util.XTee.Tests projekt on mõeldud registripäringute testimiseks. Testitakse sõnumi koostamist, saatmist ja päringu vastuse valideerumist ning serialiseerimist objektideks. Testitakse sõnumi saatmist nii X-Tee protokolli kaudu, kui ka HTTP kaudu. Testides, erandkorras, kasutakse ka vanade päringute vastuseid (XML failid) serialiseerimise testimiseks, kui vanad registriteenused on kinni ja päring vastust ei anna.

E-Notar projekti edasiseks arengu taastamiseks on soovitatud testidel põhenev disain, ehk järjestada tähtsuse järgi testitav funktsionaalus ja koostata plaan testide parandamiseks. Korralikult disainitud testkeskkond aitab tulevikumuudatused teostada ohutult ja samuti võimaldab refaktoreerida vana koodi. Antud uurimistöös on juurde kirjutatud vajalikud testid, mis on seotud teenuste meetodite üle viimisega ja osad vigased testid on parandatud.

3.4 Teenustaseme halduse protsess

On olemas palju erinevad raamistikke ja standarteid, mis võivad aidata paremini rakendada teenuse halduse protsessi. Näiteks kõige levinmad nendest on:

BS 15000: Service level management, Service reporting
ISO/IEC 20000:2011: Service level management, Service reporting
ITIL v2: Service level management
ITIL v3: Service Catalogue management (SDesign), Service Level management (SDesign), Service Portfolio Management (SStrategy)

MOF v3: Service level management (Optimizing Quadrant)
ISM Method: Service level management

Protsessi sisuks on aru saada ja kokku leppida IT teenustele sobiv tase. Kui nõuded on kõrgemad maksab ka teenus rohkem ning see ei pruugi äripoolele sobida. Seega on tegemist tasakaalu leidmisega soovide ja võimaluste vahel. Antud eesmärgi saavutamiseks on vaja esmalt aga kokku leppida teenuste kataloog ja portfell (ning seda ka edaspidi hallata). Samuti toimub tasakaalu otsimine/hoidmine peale kokkuleppe saavutamist, seetõttu on vajalik pidev töö teenustega, sh. Teenuste kvaliteedi/tervise jälgimine ja äripoolele raporteerimine ning operatiivne reageerimine kas nende mittevastavuse korral kokkulepitule või siis äripoole nõuete/ootuste muutumisel. Olulised mõisted on äripoole poolt seatav teenustaseme nõue (SLR – Service Level Requirement), leping kus äripool leping IT juhiga kokku teenustaseme ehk teenustaseme lepe (SLA – Service Level Agreement) ning selle alamlepped IT üksustega eelkõige vastutuse selgepiiriliseks määramiseks – st. käitluslepped (OLA – Operational Level Agreement) või lepingud väliste teenuspakkujatega – st. partnerleping (UC – Underpinning Contract). Siit selgub, et tellija organisatsiooni lepingut välise teenuse ostaja/IT firmaga pole õige SLA-ks nimetada – erinevalt laialdasest praktikast.

Tuleb tähele panna, et SLA-d sõlmitakse äripoole ja IT vahel ning mõistel „teenus“ peab seega olema mõlemale osapoolele arusaadav sisu ja skoop. See seab piirangu nii teenuste kataloogi koostamise meetodile kui ka kirjeldatavatele objektidele - mida teenuseks tuleks nimetada. Probleemist aitab üle saada ka täpsustav terminoloogia, kus äriteenused on need, mille osas äripoolega kokku lepitakse (notari rakenduse teenus), aga tehnilised teenused on IT sisesed, äriteenuste osutamiseks vajalikud IT teenused (nime serveri teenus). Viimased sisalduvad äripoole vaates äriteenuste hinnas ning nende teenustaseme nõuded määratakse kõike karmima äriteenuse poolt kehtestatud nõuetega. Loomulikult jagatakse ka nende kulud erinevate äriteenuste vahel, mis antud tehnilist teenust tarbivad [11].

E-Notari süsteemil on väga oluline pakkuda õiget ja usaldusväärset teenust oma klientidele. See tähendab, et peavad olema kirjeldatud ja dokumenteeritud kõik teenuse operatsioonid mida EN pakub ja millised välisteenuseid kasutab. Selles töö käigus on analüüsitud veebiteenuse kirjeldust ning järgmises peatükis antakse võimalik lähenemine teenuse täiustamisele.

4. Kasutatavad metoodikat

Üldiselt on olemas neli kontseptuaalset strateegiat. Tuleb välja selgitada kõige sobivam lahendus süsteemi arengule:

1. **Asendamine** (*replacement*) olemasoleva süsteemi uue lahendusega.
2. **Hooldus** (*maintenance*) ja/või **edasi arendamine**, koos limiteeritud juurutamisega uue tehnoloogia abil.
3. **Uue arhetiktuuri** (*re-architect/modernization*) rakendamine, säilitades vana funktsionaalsuse.
4. **Laiendada olemasoleva** süsteemi pakkimine (*wrapping*) musta kasti ja pakkudes oma tüüpliideseid suhelda uute süsteemidega, kasutades vahevara.

Igaüks neist strateegiatest omab piiranguid ja võib tuua palju kasu, kui rakendatakse õigesti. Otsus ühe strateegia jätkamiseks on keeruline ja nõuab põhjalikku äri väärtuste uurimist ning olemasoleva süsteemi kvaliteedi uurimist. Selle hindamiseks on mitmeid kontekste:

Hooldus (*support*): sellega on seotud riist- ja tarkvara tugi olemasolevas süsteemis. Lisaks, sellega seoses uuritakse ka lähtekoodi kättesaadavust ja arendusmeeskonda, et toetada täielikult koodiga tutvumist, muutmist või evolutsiooni.

Äri (*business*): äri nõue on oluline aspekt, kus organisatsioon saab otsustada, kas süsteem tuleb hoida töökorras, asendada uue süsteemiga, mis vastab nende vajadustele, või lihtsalt sulgeda täielikult. Seega selles kontekst hõlmab endas nõudeid lisaks organisatsioonile ka modelleerimistehnika kasutamisest ja olemasoleva süsteemi äri dokumenteerimisest.

Arhitektuur (*architecture*): käsitletakse süsteemi ülesehitust, arhitektuurilist stiili, komponentide omavahelisi sidemeid, kuidas on süsteem integreeritud välis süsteemidega ja millist funktsionaalsust saab klient kasutada samas süsteemis.

Tehnoloogia (*technology*): selles kontekstis käsitletakse tehnoloogia tüüpi tuginedes olemasolevale süsteemile ja veendutakse kas tehnoloogia on ikka vajalik võrreldes seda areneva ettevõtte vajadustega.

Tabel 1 kirjeldab üldist raamistikku, mis toob esile mitmeid arvamusi, mis on seotud hindamiste kriteeriumitega nendes kontekstides. See raamistik põhineb Summerville [7] tööle ja hindamise raames laiendatakse arhitektuurilisi aspekte tarkvara süsteemides. Kavandatav raamistik on üldistav, ning seetõttu ei ole ammendav.

Tabel 1 Legacy süsteemi hindamise raamistik [7]

Konteksi vaade(*Context View*)	
Hooldus (*Support*)	Hardware Application server Source code Database
Äri (*Business*)	Validity Modeling Documentation
Arhitektuur (*Architecture*)	Application integration Consumption Extensibility Interoperability Data integration Style
Tehnoloogia (*Technology*)	Vendor Version License Data center

Kui teha üldine ülevaade iga vaate järgi, siis antud töö kontekstis kõige huvitavamad on hooldus ja arhitektuur. Kuna E-Notar sisuliselt ongi oma olemuses hooldust vajav projekt, siis sellega seoses tahaks esile tõsta ka aspekti, mis on tänapäeval aktuaalne – lähtekoodi (*Source code*) ja rakendus serveri (*Application server*) hooldus.

Kui rääkida äri vaadest siis see protsess ei muutu nii palju, kuna notari tegevus on seadustes sätestatud. Kuigi büroo enda töö tegemiseks on vaja lisada juurde funktsionaalseid lahendusi.

E-Notar süsteem töötab vananenud koodil (*procedure based*), mis põhineb Windows vormidel ning arhitektuuri stiiliks võik pidada täielikult klient-server mudelit. Samuti tahaks märkida E-Notari aruande koostamise probleemi, kus kasutakse Crystal Reporti.

Crystal Reports Visual Studio pakub suure jõudlusega võimalusi luua ja integreerida interaktiivseid esitlusi Web aruannete ja Windows põhiste rakenduste ning Web Services rakenduste XML vahel. Integreeritud graafika Designer-i abil on lihtne luua IDE aruandeid ja vähendada jõupingutusi, et kirjutada koodi, mis on tavaliselt vajalik arenguaruannete jaoks.

Kaheksa aastat hiljem, EN süsteem on hakanud võitlema uue keskkonnaga, kuna suurenenud on kasutajate arv ja e-riigi rõhk toetada erinevaid registreid ning lisaks on väljakutse digitaalallkirjastamise uue formaadi – bdoc[1] kasutusele võtmisel. Täna, süsteem pakub kodanikuportaalis eesti.ee notariaalset toiminguid ja dokumente, mis seotud tehingu osalistega ning enam kui 800 töötajaga ja on tehtud palju suuri muudatusi esialgsetes põhifunktsioonides.

Kuna E-Notar süsteemis on palju vana koodi, siis ei tähenda see, et tegemist on halva koodiga, vaid pigem see, et osa koodi ei ole kasutusel ning palju on dubleeritud. Lõpuks, suurem osa koodist ei ole kaetud testidega. Suuremad muudatused või moderniseerimine ikkagi pidi tuginema heale testkeskkonnale.

Lisaks pärandtarkvara süsteemidel puuduvad paljud võimalused, mida peetakse praegu põhinõueteks. See on ühilduvus erinevates keskkondades (mobiiltelefon, K-Notar – notari kalender; T-Notar – tahvelarvuti peale jagatakse dokumendid notari poolt tehinguga tutvumiseks) ja muude x-tee teenuste kasutamine. Lisaks, kuna arenevad e-riigi teenused, peavad notarid taotlema suuri muudatusi oma põhitegevustes, et mahutada süsteemi uusi nõudeid, millest üks on protsessi automatiseerimine.

4.1 A Dual-Spiral Reengineering Model

On kaks võimalust migreerida vanemad süsteemid uutele platvormidele või raamistikele. Üks võimalus on täielik ümberkujundamine olemasoleva süsteemi uute funktsionaalsete kirjeldustega ja kasutamise juhtumitega (*use cases*). Teine on

[1] BDOC - uus digitaalallkirjastatud dokumendi formaat, mis hakkab asendama seni kasutusel olnud formaati DDOC.

evolutsiooniline migreerimine praeguse olemasolevas süsteemis, mida nimetatakse efektiivsemaks muutmiseks (*reengineering*). Kahjuks esimese meetodil on suur risk ja kõrge hind, mis on pikkade arengu ja ümberkorraldamise ajakava. Lisaks tuleb mõista olemas oleva tarkvara kodeeritud äriloogikat ja vastavat dokumentatsiooni, mis on päris keeruline. Kuigi evolutsiooniline migreerimine võib kesta kauem, see soodustab vähendada riski ja tuua olulist kasu.

Paljud uuringud on tehtud, et muuta efektiivsemaks muutmise protsessi – muuta tõhusamaks ja tulemuslikumaks. Popular sisaldab ümberkorralduse mudeleid Byrne Model [18], Evolutional Model [19], mis annavad lähenemisviise, kus on kirjeldatud mis kasu toob efektiivsemaks muutmise protsess, kuid on samuti ka puudused – näiteks on väga raske lisada uusi nõudeid Byrne Model-sse.

Esitatud uus mudel – Dual-Spiral Reengineering Model [23], põhineb spiraal mudelil tarkvara arendamiseks ja tõhustamiseks [20], ning püüab lahendada Byrne ja evolutsioonilise mudelite puuduseid.

Selle mudeli abil efektiivsemaks muutmine algab lähtekoodist olemasolevas süsteemis, ning jälgib reverse engineeringut olemasoleva süsteemi tootasüsteemi võtmiseks. Engineering sisaldab nelja taset: kontseptuaalne, nõuded, kujundamine ja rakendamisel, järeldab kogu efektiivsemaks muutmise protsessi.

Tuginedes erinevate projektide omadustele, on ka teisi erinevaid ümberkorraldusrühmade mudeleid, nagu RENAISSANCE [21], Hybrid Reengineering [22], Pattern-based Software Reengineering ja Iterative Reengineering of Legacy System. Üks ümberkorraldusrühma mudel, mis väärib mainimist on evolutsiooniline Mudel [19]. Seda evolutsioonile põhineva mudeli täielikult ära kasutamist spiraali mudeli tarkvara arendamiseks ja tõhustamiseks pakub Boehm [20], kus süsteem areneb edasi lisanduvate mustrite näol, kuni see on täielikult ümber kavandanud [19].

Näiteks, RENAISSANCE pakub terviklikku kogumit meetodeid kulude/tulude analüüsi komponentide hindamiseks, samuti kulu/riski analüüsi, kus kulude ja riski tegurite erinevaid alternatiive uuritakse selleks, et valida parim kombination – näiteks rohkem kulukas strateegia võib osutuda valituks, sest on vähem riskantne kui teine, soodsam strateegia [21]. Samas evolutsioonilise mudeli põhimõtte alusel moodulite hulka jagatakse moodulite kaupa. Nii algab spiraal iga mooduliga ning lõpetatakse kui kõik moodulid on muudetud.

Peamine töövoog Dual-Spiral Reengineering Mudel eeldab, et kahe süsteemi (*legacy* ja uus) koostöös liigutatakse funktsionaalsus (ei ole moodulid) olemasolevast süsteemist uue süsteem peale samm-sammult, nagu spiraali mudelis. Ümberkorraldus protsess Dual-Spiral Model on jagatud kolme peamisse etappi: jagada funktsionaalsust, alustada ja lõpetada spiraal protseduur.

Jaga funktsionaalsus (*Divide the Functionality*). Üks suurimaid erinevusi evolutsioonilise mudeli ja Dual-Spiral Model vahel on see, et Dual-Spiral Model põhineb funktsionaalsuse jagunemisele, selle asemel, et moodulite kaupa jagada. Alguses olemasolev süsteem analüüsitakse ja jagatakse funktsioonide nimekiri.

Spiraal protseduur alustamine (*Start the Spiral Procedure*). Tegelikult alam ümberkorralduste protseduur Dual-Spiral Model on väga sobiv kasutades Byrne Mudelit, nagu reverse engineeringu ja siis edasi reengineeringu puhul.

Funktsionaalsuse üleminekut, võib jaotada kolmeks muutmise protsessiks igas spiraali korras.

Esiteks on *üks-ühele*, mis tähendab, et implementeeritakse üks funktsionaalsus sihtsüsteemist üheks funktsionaalsuseks legacy süsteemi.

Teiseks on *üks-mitmele*. See tähendab, et me rakendame ühe funktsionaalsuse sihtsüsteemi, mis moodustab enam kui ühe funktsionaalsuse olemasolevast süsteemist.

Viimane on olukord *palju-ühele*. See juhtub siis, kui meil on vaja rakendada rohkem kui üks funktsioon samaväärseks ainult ühest funktsionaalsusest olemasolevast süsteemist.

Peale üleminekuperioodi funktsionaalsuse legacy system sihtsüsteemi peale, Dual-Spiral Model võimaldab lisada uut funktsionaalsust sihtsüsteemi vabalt ja lihtsalt.

Pärast arengut igas alam protseduuris, olemasolev süsteem (mis on suletud osa reengineered funktsionaalsus) ja uus süsteem (mis on rakendanud uusi funktsioone) peavad töötama mõlemad ja mõlema jaoks tehakse regressioontestid. Lisaks testitakse süsteemi ka kasutajhate poolt, et kinnitada äri funktsionaalsuse töökorras olekut.

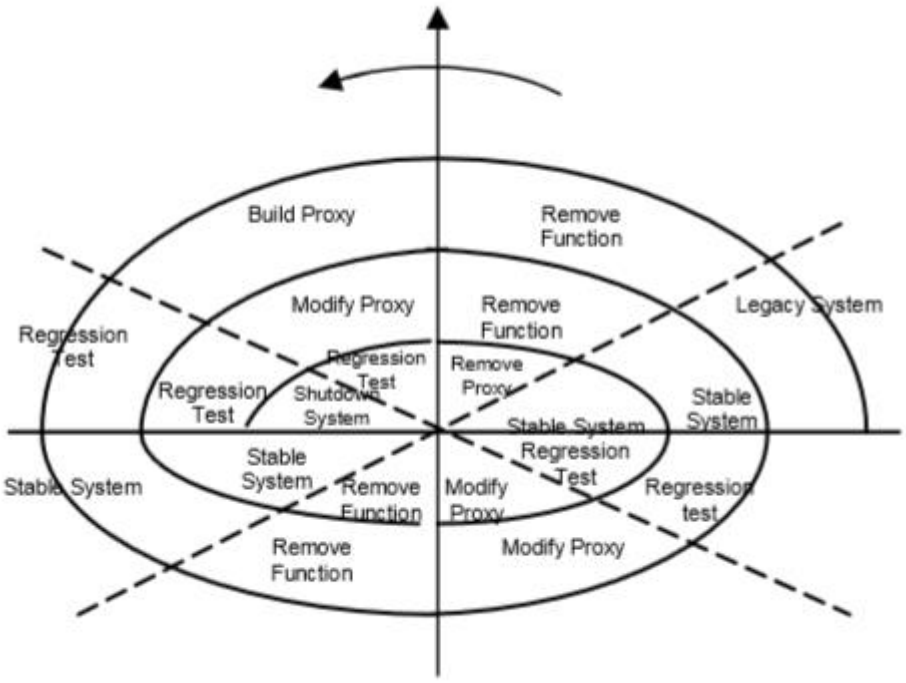
Lõpetada Spiral protseduur (*Terminate the Spiral Procedure*), kui kogu olemasolev süsteemi funktsionaalsus on kolinud uuele sihtsüsteemile ja kasutajad on selle heaks kiitnud, siis võib kogu spiraal protseduur lõppeda. Sel hetkel tuleks olemasolev süsteem sulgeda ja kommunikatsioonide proxy tuleks eemaldada.

Eelised Dual-Spiral Reengineering Mudel. Peamine eelis Dual-Spiral muudeli ees on järjepidavus ülekandmisel. Samm-sammult spiraal protseduur lihtsustab ülekanda uusi funktsioone uude sihtsüsteemi. Lisaks mudel täidab tsüklilist arendamise lähenemise viisi järk-järgult süsteemi kasvu rakendamisel, vähendades samal ajal riskiastet [23].

Joonis 1 ja Joonis 2 näitavad antud mudeli spiraal protseduure.

- Lisa funktsioon
- Kustutada funktsioon
- Ehitada/Uuendada vahendaja(proxy)
- Regressiooni testimine
- Stabiliseerida süsteem
- Spiraal protsessi lõpp

Joonisel 1 on näha põhiprotsessi *legacy* süsteemi funktsionaalsuse vähendamisel. See tähendab, et üks haaval võetakse funktsioonid ja teostakse regressioontestimine, et veenduda sellest, et kustutatud funktsionaalsus ei tee katki üldist funktsionaalsust. Samal ajal paralleelselt toimub vahendaja realisatsioon.



Joonis 1 Decremental Legacy system Pattern in Dual-Spiral Model [23]

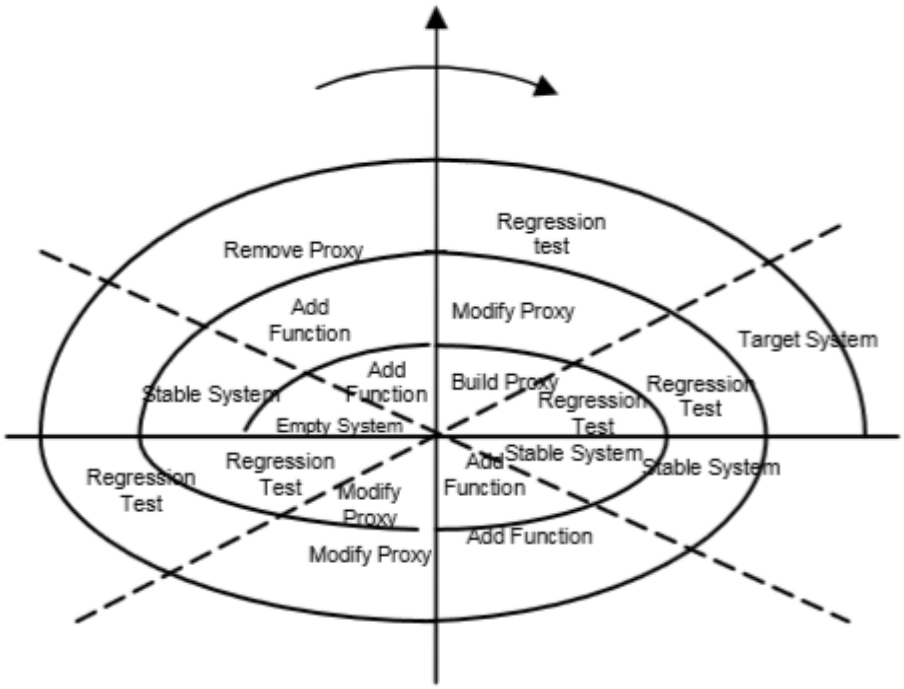
Vahendaja (Proxy) eesmärk on hallata ja suunata tööprotsess uue funktsionaalsuse peale.

Kustutatud funktsionaalsuse jaoks ehitakse vahendaja ja tesotakse regressioonitestimine, et veenduda selles, et uus funktsioonalsus on kantud korrektselt uude süsteemi.

Samal ajal vana süsteemi funktsionaalsus lisatakse uude süsteemi Joonis 2. Testimine protsess kordub ka uues süsteemis, saavutades stabiilse töökord. Protsess lõppeb kui kõik *legacy* funktsionaalsus ülekandud ja testitud. Ainult peale seda tehakse *Shutdown System* ehk lülitakse välja.

Dual - Spiral Model sobib hästi just projektides, kus toimub funktsionaalsuse üle kandmine aegunud süsteemist uuemale. Uuringud näitavad, et selle mudeli effektiivus on 95 % [23], seega võiks teha järelduse, et viis protsendi ei ole kantud ja vajab lisa arendust.

Funktsionaalsuse üle viimine tähendab seda, et me võime üle viia ka vead mis on seotud vana koodiga. Sel põhjusel ongi vaja pidevalt läbi viia regressioonteste. Tegelikult paljud muutmise protsessid tuginevad sellisel tehnikat nagu testimine. Nagu oli mainitud, et TDD on kõige parem lähenemise viis tarkvara arenduses ja funktsionaalsuse üle viimiseks otstarbekas praktika.



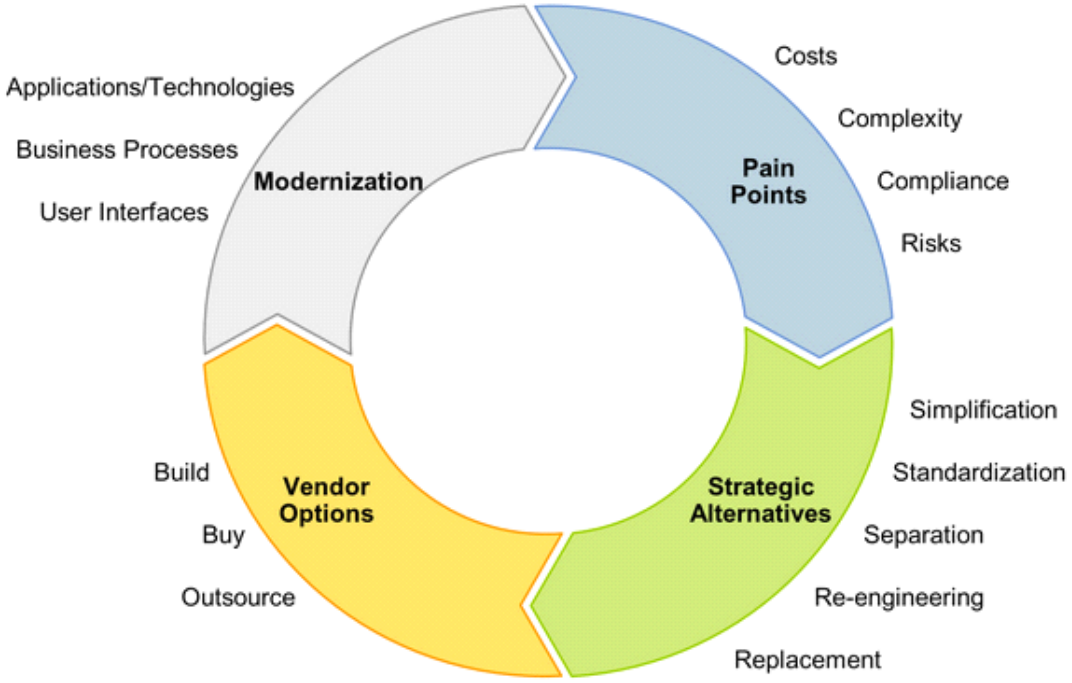
Joonis 2 Incremental Target system Pattern in Dual-Spiral Model [23]

Joonisel 2 on näha kuidas saavutada *Target System* – uus süsteem. Alustades tühjast süsteemist ja samm sammult lisades ja testides uut funktsionaalsust ning stabiliseetides süsteemi vigade parandustega. Protsess lõppeb siis, kui rohkem ei ole midagi vaja kanda vanast süsteemist uude. Peale seda uus süsteem võib eksisteerida iseseisvalt, mis tähendab, et uue funktsionaalsuse lisamine ei sõltu vanast süsteemi.

Dual-Spiral Model väldib probleeme, mis olnud Evolutional Model ning sobib paljude erinevate tehnoloogia efektiivsemaks muutmise [23]. Antud mudel vastab E-Notari üld arengustrateegiat. Kuna uue süsteemi ehitamist võtab palju aega ja toob palju riske. Samas võib tekkida olukord kus uue süsteem ka ei vasta uue tehnoloogiat ja nõuded.

Sama muudeli põhjal võib teostada muudatusi olemasolevas süsteemis. Sest EN süsteem on paindlikum lubades sisse viia muudatusi, väike moderniseerimise osaga, kandes olemas oleva funktsionaalsust üle üks haaval. Analüüsi käigus tuvastati, et kõige täitsamaks ülesandeks on rakenduse serveri moderniseerimine. EN veebiteenust ei saa kasutada veebi kaudu – tänasepäeval ei ole üldse võimalik veebilehitseja kaudu liigipääseda. Tulevikus on plaanis Notari Kojal kasutada veebi klienti. Selleks oleks vajalik hoopis teine lähenamine veebiteenuste arendamiseks.

Joonisel 3 on illustreeritud põhiprotsessid, mis on seotud vana süsteemi uuendamisega.



Joonis 3 Legacy System edasi arendamine (Gartner, Juuni 2013)[24]

Moderniseerimist (uuendamist) võib teostakse kasutaja liidese (*User Interface*) tasemel, äriprotsessi (*Business Processes*) ja rakenduse/tehnoloodia (*Application/Tehnologies*) tasemel.

Kuna praeguses olukorras kasutajaliides ehk E-Notar klient rakendus (*desktop application*) rahuldab kasutaja nõudeid ja vajadusi, siis samas kontekstis seda teemat ei käsitle. Kuigi tulevikus võib arutleda arendust veebikliendile.

Äriprotsess eriti ei muutu, muudatused tulevad kas seaduste muutmisega või need muudatused initsialiseeritakse kolmanda osapoole poolt – nt. uue digital dokumendi allkirjastamise üleminek ddoc formaadist uue bdoc formaadi peale.

Kui rääkida rakendusest või tehnoloogiast, siis võib siinkohal arutleda mitme erineva teema üle – näiteks registripäringu optimeerimine, andmebaasi protseduuri optimeerimine, veebiteenuse moderniseerimine.

Iga muudatus toob kaasas oma hinna ja riski. Nagu nimetatud *Pain Points* võib olla kõrge

hind, keerukus, nõustumine, risk. EN on väga keeruline infosüsteem, mis on tihedalt seotud Pärimisregistriga, ja kogu süsteemi korralikkust tööst sõltuvad kõik notari bürood. Ehk moderniseerides mingit osa suurest süsteemist, kaasatakse ka omad riskid. Selleks, et maandada riske on vaja teostada vastavalt uurimistöö, et veenduda selles, et moderniseerimine on põhjendatud ja toob reaalset kasu, mite ainult lühiajaks vaid ka pikemas prespektiivis.

Strateegilised alternatiivid (*Strategic Alternatives*) aitavad aru saada milleks on vajalik üks või teine moderniseerimine. Kas see on lihtsustus (*Simplification*), püüakse optimeerida protsess, protseduuri, taaskasutavus või muu. Või muudatused mis on seotud standardiseerimisega (*Standardization*), püüakse muudatustega sisse viia kehtivaid standarteid, et tagada kõrge kvaliteet. Samuti võimalik liikuda eraldumise (*Separation*) suunas, tõsta välja osa süsteemist eraldi üld protsessist. *Reengineering* on toote parandamine, muutmine või osaliselt asendamine, et muuta selle funktsiooni, kohandades sellega uusi vajadusi. *Reengineeringut* võiks vaadelda kui süstemaatilist ümberkujundamist olemas olevas süsteemis uude vormi, selleks, et tagada kvaliteetsed paranenud operatsioonid, süsteemi võime, funktsionaalsus, jõudlus, või edasi areng teha madala hinnaga, ajakava, või maandada kliendi riski. Eripäraseks võiks pidada muutmisi, mis on seotud süsteemi asendusega (*Replacement*), asendatud süsteem peab töötama vähemalt võrdselt kui vana või paremini. Kui seda ei ole tagatud siis võiks lugeda, et projekt ei ole õnnestunud.

EN kontekstis on võimalik valida kõik üleval mainitud strateegiad. Tegelikult võiks valida kaks neist – reengineering ja asendamine. Tegemist ei ole kogu süsteemiga vaid alguses võib see olla veebiteenuse kiht.

Järgmiseks on toodud küsimused, mis on seotud muutmisega, kust võtta ressursse muutmise teostamiseks. Kas võib olla valmis lahendusi, mida võimalik soetada ja integreerida enda süsteemiga? Kas tellida muudatused või teostada muudatused enda vahendidtega? EN muutmine ja hooldus on hetkel RIK-i töö ja edasi arengu strateegia osaliselt langeb hooldavale asutusele, kuigi üldine strateegia süstemi muutmisel pidi olema kooskõlastatud süsteemi omanikuga.

Viimase kahe aasta jooksul on heaks kiidetud EN laendused nagu T-Notar ja K-Notar, mis oli realiseeritud välise arendaja poolt kasutades WCF teenuseid. Selle tehnoloogia valiku põhjus seisnes selles, et tehnoloogia kasutamisel olid puudused olemasoleva teenuse protokolli kasutamisel (http). Uue süsteemi vajadused nõudsid tcp protokolli kasutamist.

Sellega seoses üritakse töös arusaada, mis on põhierinevus ASMX ja WCF teenuste vahel.

Kui rääkida EN edasi arengust tekib küsimus kas on mõistlik migreerida ASMX teiste veebiteemuste tehnoloogia peale ja kas on märkimisväärset kasu selles või ikkagi edasi tegeleda hooldusega ja täiendustega.

Edasi esitakse ülevaade olemasolevast veebiteenusest ja kasutatud WCF teenusest EN laenduse projektides.

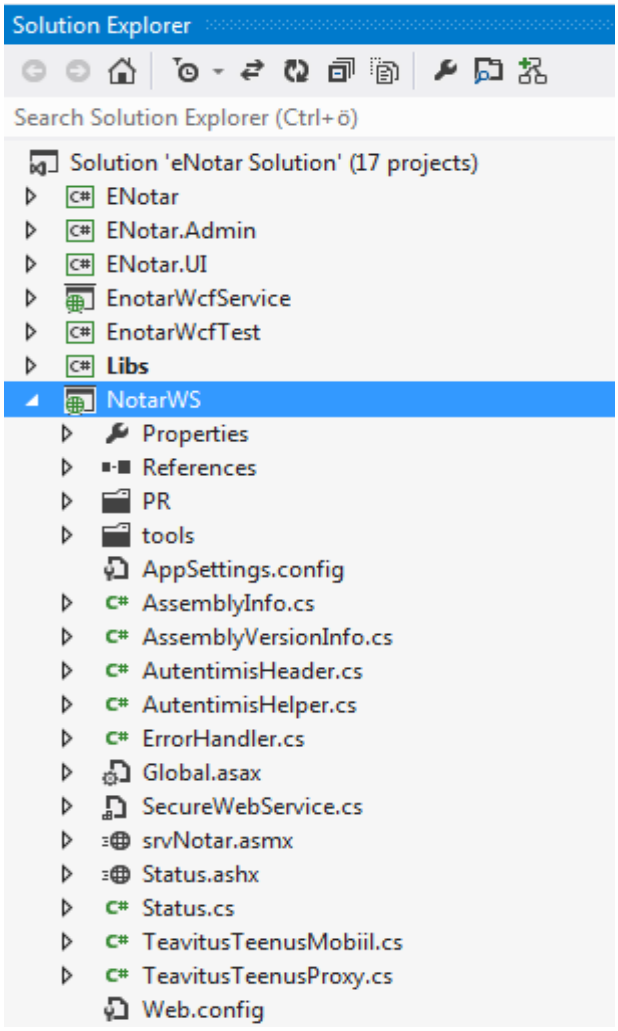
Veebiteenus (Web Service)

Veebiteenus on komponent, mis on majutatud kaugel veebi serveris [26]. Veebiteenused võimaldavad luua hajutatud rakendusi, mille komponendid suhtlevad omavahel, kasutades ühiseid standarteid ja protokolle. Sisuliselt on tegemist klassiga, mis võib olla realiseeritud erinevas programmeerimiskeeles ja pakub oma meetodeid kasutamiseks võrgu kaudu. Tähtis on ära märkida, et meetodid on veebist kasutatavad meetodid. Teenuse kasutamine on kirjeldatud WSDL-s. See on XML vormingus kirjeldus, mis sisaldab endas operaatsioone, sõnumeid, andmetüüpe ja võrguprotokolle, mida veebiteenus pakub.

Teenust kasutatakse koos toimivate tüübide modelleerimiseks, mis on vajalikud äriprotsessides, et täita äri vajadusi ja võimaldavad eraldada äriloogika kliendi koodist. See mehhanism lubab kasutada ja üldistada teenuse rakendamist ning avaldada ja teha kättesaadavaks erinevatele klientidele. Veebiteenused põhinevad kolmel omavahel seotud tehnoloogial: Web Service Description Language (WSDL, veebiteenuse kirjelduse keel), ühendusprotokolli (HTTP-GET, HTTP-POST ja SOAP) ja Discovery Service (avastus teenus).

Veebiteenused võimaldavad lihtsalt korraldada rakenduste omavahelist andmevahetust, kasutades sõnumi saatmiseks ühist sõnumite edastuse protokolli SOAP (Simple Object Access Protocol). Andmevahetuseks kasutakse XML kujul koostatud sõnumeid ja saadetud Hypertext Transfer Protocol (HTTP), mis võimaldab programmil töödelda samu sõnumeid sõltumata operatsioonisüsteemidest [26].

E-Notari veebiteenus on realiseeritud kasutades ASP.NET Web Service – srvNotar.asmx. ASMX tähendab Active Server Methods (Microsoft filename extension), mis tänapäevaks on aegunud tehnoloogia. ASP.NET veebiteenus kasutab sõnumi edastamiseks teenuse ja kliendi vahel standartses vormingus XML-i. Sõnumi saatmiseks kasutakse protokolli HTTP, E-Notar süsteem kasutab andmete edastamiseks – SOAP, sest sama protokoll võimaldab edastada väga keerulist tüüpi andmeid (kasutaja määratletud klassi objektid andmestik, massiivid objektide jt.).



Joonis 4 E-Notari veebiteenuse projekt

ASP.NET Web Service-t on võimalik majutada ainult IIS (Internet Information Service) peale – komplekt serverid mitmetele Interneti teenustele, mida pakub Microsoft. IIS laieneb erinevatele Windows operatsioonisüsteemidele.
Põhikomponendiks IIS-is on veebiserver, mis võimaldab postitada näiteks veebilehte või veebiteenused. IIS veebiserver annab mitmeid võimalusi piirata juurdepääsu veebilehtedele ja veebirakendustele. Vaikimisi on IIS peale loodud rakenduse sõlm E-Notari juurkataloogi kohas.

Joonisel 4 on esitatud olemasoleva süsteemi ASMX Web Service projekt, mis implementeerib E-Notari teenuse. Projekt sisaldab autentimise ja autoriseerimise loogikat. Autentimine toimub Eesti ID-kaardiga, Mobiil-IDga, või RIKe väljaantud sertifikaadiga. Autoriseerimine toimub vastavalt kasutaja õiguste abil, mis määratakse e-notari administraatori poolt.

Teenuse kasutamine toimub HTTPS (side protokoll turvalisuse suhtlemiseks üle arvutivõrgu) kaudu ja serveri poolne IIS on konfigureeritud nii, et klindi sertifikaat on

kohustuslik. Ehk peale seda kui toimub pöördumine teenuse poole server küsib kasutaja autentimiseks PIN-koodi (pin1).
E-Notari klient (*desktop application*) otseselt ei pöördu serveri poole, vaid teeb seda läbi proksi klassi. Projektis Notar.UI on lisatud Web Reference teenuse srvNotar.asmx peale. Madalal tasemel Web Reference loob kliendile proksi klassi, mis võimaldab kliendi koodis kasutada veebiteenuse meetodeid mis on kirjeldatud teenuse WSDL failis. Väga tähtis on see, et teenuse kirjeldus oleks korrektne ehk kasutatavate meetodite signatuur, tagastatavad parameetrid ja meetodi väljakutse oleks õige.

See on ongi üks suund teenuse muutmiseks. Mitu aastat teenuse hooldust ja erineva kliendi veersiooni toetamist viis süsteemi sellesse seisundisse, et tekis mitmeid teenuse meetodeid mis ei ole kasutuses ja tekitab asjata segadust. Mitte kasutatavad teenused omakorda viitavad rohkele *legacy* koodile. Seda probleemi võib lahenda osaliselt Visual Studio enda sisseehitatud abimeetodiga nt. *Clean up code, Refactoring*. Samas seda on vaja teha ettevaatlikult. Tuleb tegelikult analüüsida kliendi poolt viimati kasutatud veebiteenuse meetodit.

Kõik teenuse muudatused pidid olema toetatud testidega. Kuna vana koodiga, mis ei ole kasutuses võivad olla kirjutatud testid ja tegelikult antud kontekstis see on reaalne juhtum, siis sellest järeldub, et teenuse meetodi kustutamine teenuse kirjeldusest kaasab endas muudatusi testkeskkonnas ja testide korrastamisi. Selleks on vaja järgida EN projektis Testidel põhinev arendusmetoodikat.

WCF – Windows Communication Foundation

Windows Communication Foundation (WCF) on Microsofti kombineeritud programmeerimise mudel ehitada teenustele orienteeritud arhitektuur (SOA). WCF kasutamine võimaldab arendada turvalisi, usaldusväärseid ja kõikide .NET platvormide jaoks teostatatavaid lahendusi. WCF toetab kontsentreeritud arvutamist, kus teenused on isoleeritud tarbijatele. Kliendid saavad tarbida mitmeid teenuseid, samuti teenuseid saab tarbida palju kliente.

WCF teenuse kirjutamisel on alati kolme etappi:

- Määratakse teenuse leping (*Service Contract*) ja selle realiseerimine teenuses.

- valitakse või määratakse teenuse sidemed (*Binding*) kus on võimalik märkida teenuse kvaliteet, turvalisus, transpordi protokoll, kodeerimine, ja protokolli nõutud andmed klientide ja teenuste omavahelisel suhtlusel.

- määratakse otspunktid (*Endpoints*) lepingus, sidudes selle (määrates *binding* nimi järgi) konkreetselt võrguaadressiks. Kõik suhtlemine Windows Communication Foundation (WCF) teenustes toimub läbi otspunktide teenuste. *Endpoints* pakuvad klientidele juurdepääsu funktsioone, mida pakkub WCF.

Teenuse leping (*Service Contract*) võib toetada mitmeid sidemeid ja sidemed omakorda saavad kasutada mitut lepingut. Teenuses võib olla palju *Endpoints-e* (lepingutega seotud aadressid) samal ajal. Nii, et kui teha teenus kättesaadavaks üle HTTP kasutades SOAP maksimaalset ühilduvust, samuti TCP kasutab andmete kahendkujul maksimaalset jõudlust, siis kaks *Endpoints* võivad ka koos eksisteerida.

WCF teenuse loomist alustatakse teenuse lepingu (*Service Contract*) määramisest, mis kirjeldab, kuidas suhelda teenusega ja millesed operatsioone teenust toetab. Operatsiooni võib vaadelda kui veebiteenuse meetodit, nagu ASMX teenuses esitatud atribuudiga meetodit [WebMethod].

WCF tehnoloogia ei ole uus ja oli lisatud .NET 3.0 raamistiku versiooni ning on täiesti laialt kasutaud ja toetatud Microsofti poolt. Microsoft arendab WCF tehnoloogiat lisades uusi talitluslikke elemente. Tänapäevaks on toetatud versioon WCF 4.5.

RESTful service

Enamik SOA (Service Oriented Architecture) rakendusi ei ole täiesti uued ja tavaliselt arenenud legacy süsteemid. Olemasolevad süsteemid kasutavad äriprotsessi teostamiseks andmeid, mida päritakse vanadest süsteemidest. Enamik praeguseid veebiteenuseid on SOAP-RPC stiilis teenused [16].

Remote Procedure Call (RPC) – kaugprotseduurikutse protokoll, mis võimaldab ühel arvutil asuval programmil täita teisel ehk serverarvutil asuvat programmi. RPC kasutamise korral pole programmeerijal vaja välja töötada spetsiaalseid protseduure serveri jaoks. Klientprogramm saadab serverile vajalikke argumente sisaldava sõnumi ning server saadab tagasi programmi täitmise tulemused. RPC kasutab klient/server mudelit [16].

Tänu suurele effektiivsusele teenuse kasutamisel hajusarhitektuuris, hakatakse kasutama olemasoleva süsteemi SOA-d koos RESTful veebiteenustega, mitte ainult taaskasutavuse pärast, vaid ka tänu oma eripärasusele. REST (Representational State Transfer) on arhitektuuri stiil võrgusüsteemide [17].

Selles töös ei ole käsitletud RESTful veebiteenuse kasutamist ega võtmeküsimusi ümberkorralduse kohta pärandsüsteemis. Kosutavad veebiteenused on teema tulevikus arutamiseks, kuid näidis projekti kirjelduse on olemas võimalus WCF veebiteenust dekoreerida RESTful teenuseks. Ühise protsessi ümberkorraldusel vanemates süsteemides REST-stiil peale on üks paljudest alternatiivdest. Kandidaat veebiteenusest identifiseeritakse legacy süsteemide mõistmine alguses. Mitu teenust realiseeritakse, põhineb suhete üksuste vahel ja piiranguta kehtestatud reeglitel. Loodud URI on rafineeritud ja peab hoolikalt esindama RESTful veebiteenuseid [16].

Lisades WCF teenusele meetodi koos atribuutidega [WebGet] ja [WebInvoke] mis asuvad System.ServiceModel.Web nimiruum all, saab muuta rakenduse RESTful teenuseks. WebGet operatsiooni loogilise teabe saamist teenuse toimimisest võib nimetada REST programmeerimise mudeliks. WebGet ja OperationContract atribuudite kasutatakse teenuse meetodi kirjeldamiseks ja seostatakse koostöös UriTemplate parameetritega, mida kasutakse HTTP GET meetodiga. WebGet tegeleb andmete pärimisega.

WebInvoke operatsioon loogiliselt tõstab teenuste valikuvõimalusi ja seda võib nimetada REST programmeerimise muudeliks. WebInvoke ja OperationContract atribuute kasutakse teenuse meetodi kirjeldamiseks ja seostatakse koostöös UriTemplate parameetritega, mida kasutakse HTTP POST, PUT, või DELETE puhul. WebInvoke atribuudi omadus Method täpsustab HTTP meetodi (POST, PUT või DELETE) ja vaikimisi meetod on POST.

Tabel 2 Teenuste ASMX, WCF, WCF Rest vahetesitakse põhi erinevused teenuste vahel.

Tabel 2 Teenuste ASMX, WCF, WCF Rest vahet

ASMX Web Service	WCF	WCF REST
Majutatud (*hosted*) IIS	Majutatud (*hosted*) IIS, WAS, Console, WinNT Service, WCF Provided Host	Et kasutada WCF kui WCF Rest teenust, konfig failis on vaja võimaldada webHttpBindings.
Limiteeritud turvalisus	Terviklik turvalisuse programmeerimise mudelit, võimalus muuta konfig failis	Terviklik turvalisuse programmeerimise mudelit, võimalus muuta konfig failis
Toetus ainult HTTP protokolli	Toetab HTTP, TCP, MSMQ, NamedPipes	See toetab HTTP GET ja POST tegusõnad (verbs) poolt [WebGet] ja [WebInvoke] atribuudid võrra.
SOAP (XML)	SOAP (XML)	Toetab XML , JSON ja ATOM andmete esitusviisi.
Kasutab XmlSerializer	Kasutab DataContractSerializer	Kasutab DataContractSerializer

Nagu oli mainitud E-Notari süsteem pakub selliseid teenuseid nagu K-Notar ja T-Notar, mõlemad teenused kasutavad WCF RESTful stiili. Otsus miks valiti just sama tehnoloogia, ei ole otseselt põhjendatud välispartnerite poolt. Aga olulisemaks põhjuseks on TCP protokolli kasutamine Google Notification Service-ga suhtlemiseks. Google Notification Service on tasuta teenus Androididele teavituste edastamiseks, rakendust haldab Google Inc. T-Notar teenus vahendab notari dokumendid TCP protokolli kaudu kõikide kliendiga (Android rakendustele) kellele see dokument on määratud jagamise ajal.

Viimasel ajal on soovitav käsitleda ASP.NET Web API, selleks, et luua REST teenuseid. See osa MVC-st (Model–View–Controller) 6.0, mis võimaldab lihtsalt kasutada ja konfigureerida veebirakenduste tegemist (tänapäevaks arutletakse E-Notari veebirakenduse versiooni loomise üle).

Motivatsioon veebiteenuste kasutamise

Ettevõtted, kus on olemasolevad süsteemid, sisaldavad tavaliselt segu erinevaid tehnikaid ja protokolle, mida on raske säilitada ja ajakohastada. Arendades teenustele

orienteeritud tehnoloogiale, on tänapäeval on võimalik *wrap-da* olemasolevat funktsionaalsust veebiteenusteks ja ehitada *ready-to-use* teenuseid, mida saab üles ehitada uutele süsteemidele ilma täiendavate muudatusteta.

Peamine motivatsioon reengineerida vanemaid süsteeme veebiteenustele võib kokku võtta järgmiselt.

1. Funktsionaalsus taaskasutamisel (*functionality reuse*). Kordus kasutus põhineb teenuste ümber mässimise (*wrapping*) asemel olemasoleva koodi ümberkirjutamisel, mis viib alla kogu jõupingutusi.

2. Teenuste kaevandamise võimed (*services capabilities mining*). Uute teenustele-orienteeritud süsteemide ehitamisel kaasatakse lõppkasutajate nõudmised koos teenustega, võime ettevõttele kuuluvast osa võtta. See võib aidata vältida liigsed arendusi ja ennustada vajalikke teenuseid isegi ilma otseste nõudmisteta lõppkasutajatelt.

3. Tuleviku integratsioon teistes projektides (*future mash-up*). Kõik mis on seotud Web 2.0 (kirjeldab World Wide Web sites, mis rõhutab kasutaja loodud sisu, kasutatavust ja koostalitlusvõimet), nõuab uut arengu lähenemist komposiit-rakenduste arendamises, mis võimaldab lihtsamini ja kiiremini integreerida olemasolevaid veebirakendusi.

4. Süsteemi hooldamine ja uuendamine (*system maintain and update*). Arusaamise metoodika olemasolevate süsteemide ümberkorraldamiseks ja strateegiate ümber mässimiseks, mida saab rakendada ka süsteemi hooldamiseks ja ajakohastamiseks.

Kõige suurem motivatsioon teenuse edasi arendamises ongi suutlikus madala kuluga hooldamine, uuendamine ja olla valmis tuleviku muudatuste teostamiseks. Lähtudes sellest, et olemasolev süsteem kasutab aegunud ASMX tehnoloogiat teenuse ehitamiseks, käsitlekse uurimisel kasutusele võtta WCF teenus E-Notari süsteemis, ASMX asemel.

Järgmises peatükis kirjeldakse üldist arengustrateegiat teenuste moderniseerimise kohta, alustades teenuste analüüsist, reengineeringu mudeli valikust ja TDD kasutamisest uue veebiteenuse loomisel koos veebimeetodite implementeerimisega ja teenuste võrdlusega.

4.2 Töö käik

Tuginedes ITILi teenustaseme halduse protsessile, mis on lühidalt kirjeldatud peatükis 3.4, oleks vaja aru saada ja kokku leppida milliseid teenuseid E-Notari süsteem pakub.

Raskused protsessi muutmises seisnevad mitte piisavalt hästi dokumenteeritud nõuetes. Osaliselt on süsteemi nõuded ja teenused dokumenteeritud. On olemas erievaid metoodikad kuidas taastada päransüsteemi nõudeid [25].
Nõuete arusaamine pärandsüsteemis on eelkõige tähtis selleks, et:

- mõista sidusrühmasid, kasutajaid ja äri konteksti süsteemis;
- saada üldpilt süsteemi funktsionaalsusest ja keskkonnast;
- mõista kavatsust originaal disainis;
- aru saada protsessist kasutaja tasemel.

See teadmine on kasulik, selleks et suurendada olemasolevat süsteemi, integreerides seda teiste infosüsteemidega, või *reengineering* olemasolevas süsteemis [25].

Antud töö kontekstis, nõuete taastamine ja teadmine millised teenused on kasutusel, aitaks liigipääseda kogu süsteemi veahalduse ajaloole ja viimase viie aasta jooksul koodi muutmise ajaloole.

E-Notari veebiteenuse kirjeduse uurimisega ja sama veebiteenuse meetodite kasutamise kaardistamisega, klendi pool, tuli välja selline statistika, et kümme protsenti veebimeetoditest ei ole kasutusel. Sellest võib järeldada, et vana meetodi funktsionaalsust ei ole testitud ja see toob segadust teenuse kasutamisel, sest kliendi poolt need meetodid on nähtavad ja võivad olla eksiklikult kasutatud.

Mitte kasutatavate veebimeetodite kustutamisest teenuse kirjeldusest ei piisa, sest veebimeetodist andmete pärimine, salvestamine, kustutamine toimub läbi Service Fassade (Fassaad on objekt, mis annab suurema koodi lihtsustatud liidese) kihi. Tuleb uurida, mis loogika selle taga on. Antud süsteemis toimub liigipääs DAL-le läbi service fassaad kihi. Ehk kui kustutada üleliigsed veebimeetodid oleks vaja läbi vaadata ka fassaad teenus ja uurida kas on mingit loogikat, mis on mujal kasutusel.

Pärast veebiteenuse kirjelduse korrastamist ja kliendil proksi klassi (*web reference*) uuendamist järgmise sammuna tuleks välja valida kõige rohkem kasutatavad veebimeetodid

E-Notari süsteemis. Seejärel tuleks üritada neid üle viia uue WCF tehnoloogia peale. Samuti käsitletakse ülekantavate teenuste prioriteeti, muutmise ajalugu (SVN kood) ja vastavaid vigu, mis selle veebiteenusega seotud olid.

Koostatud veebiteenuste meetodi nimekirja põhjal (kandidaadid üleviimiseks), oleks vaja luua uus WCF projekt. Arenduseks valiti Visual Studio 2012, nagu teistelgi projektidel, mis on seotud E-Notari süsteemiga.

Tahaks pöörata tähelepanu asjaolule, et alates versioonist Visual Studio 2010 on eemaldatud võimalus lisada project ASP.NET Web Service Application ja on võimalik valida New Project/Templates WCF Service Application. Visual Studio 2012 säilitas sellise võimaluse WCF service loomiseks.

Võrdluseks ja vanade meetodite ülekandmiseks on loodud uus project EnotarWcfService. Üldine asmx service funktsionaalsus, mis oli seotud autentimisega, sai üle kantud. Võrdluseks on valitud kümme põhilist funktsionaalsust, mis on kriitilised süsteemi töötamiseks (kandidaadid):

- Ühe dokumendi laadimine (viis megabaiti)
- E-Notari klassifikaatori laadimine(Dataset),
- E-Notari klassifikaatori laadimine(bitijada),
- Notari kalenderi laadimine notari Id järgi,
- Notari tehingu laadimine,
- Notari dokumendi laadimine tehingu Id järgi,
- Registripäringu laadimine tehingu Id järgi,
- Koordus registripäringu saatmine,
- Dokumentide otsing Notari Id järgi,
- Dokumendi laadimine ja muutmine.

Nagu eelnevalt oli mainitud, kõik muudatused mis teostakse süsteemis ja eriti kui tegemist *legacy* süsteemiga, peavad olema testitud. Tuginedes TDD-le, enne veebimeetodi uuele teenusele üleminekut, kirjutati test.

E-Notaris on olemas testkeskkonnad, kus üks neist on NotarWS.Tests projekt. Osad testid ei ole ammu hooldatud, osad on katki ja osad testid üldse puuduvad. Testide kirjutamiseks oli valitud teenus mis vastutab registripäringute leidmise eest, mida on teinud

notaribüroo töötaja konkreetses tehingus. Kuna selline test puudus, oli hea võimalus näidata kuidas üldine protsess käib.

Testiga ikkagi püüame tõestada, et meie oodatud tulemus on korrektne ja test õnnestub. Kõige pealt üritame tõestada, et antud testis parameetri järgi on olemas DataSet, registripäringu hulk on rohkem kui null ja registripäring on seotud antud tehinguga, lisaks kontrollitakse kas on muudatusi DataSet Schemas.

Esitakse väike näidis loodud testklassist, koos testiga.

```
namespace NotarWS.Tests.DALTest
{
   /// <summary>
   /// Summary description for RegistriPäringDALTest.
   /// </summary>
   [TestFixture]
   public class RegistriPäringDalTest : AbstractDALTest{
      [Test]
      public void LaeRegistriPäringTehinguIdJärgiTest(){
         const int tehingId = 122516;
         const int notarId = 320;
         RegistriPäringDataset registriPäring = new RegistriPäringDAL()
            .LaeTehinguRegistripäringud(tehingId, notarId);
         Assert.IsNotNull(registriPäring);
         Assert.IsTrue(registriPäring.RegistriPäring.Count > 0);
         Assert.IsTrue(registriPäring.RegistriPäring[0].TehingId == tehingId);
         TestHelper.KontrolliStruktuuriSamasust(new RegistriPäringDataset(), registriPäring);
      }
   }
}
```

Kõikide kandidadite muutmiseks on kirjutatud või muudetud *Unit Tests*. Tulevikus on vaja täpselt sama moodi, enne muutmist, kirjutada test. Igal juhul testide korda tegemine vajab lisa töö ressurssi. Testide kirjutamine pidi olema prioritiseeritud [15]. Selle tööks on vajalik SVN ülevaatuse- ja veadehalduse süsteem. Mõlemas kohtas on võimalik tuvastada mis faili kõige on muudetud ja millised vead sellega kaasatud. Lähtudes korjatud informatsioonist

ja analüüsides koodi võib eraldada need koodi osad, mis ei ole kaetud testidega või seotud testide rikkega.

Tuginedes ’A Dual-Spiral Reengineering Model for Legacy System’ [23], mis on kirjeldatud üleval, alustatakse protsessi üle viimesega vanalt teenuselt uue teenuse peale. Loodud tühjas projektis EnotarWcfService on olemas kaks tähtsat faili INotarService.cs ja EnotarService.cs, mis genereeritakse projekti loomisel. INotarService liides on atribuudiga [ServiceContract] ja selle liikmed ning meetodid on atribuutiga [OperationContrat] (kasutades System.ServiceModel nimeruumi).

[ServiceContract] - teenindamise leping väljastab klassi või liidese kliendi rakendustele, mis võivad sisaldada ühte või mitut atribuutidega [OperationContract] meetodid.

[OperationContract] - näitab, et meetod on operatsioon, mis on osa teenindamise lepingu taotlusest, st ainult meetod, mis on atribuutiga [OperationContract] on nähtav WCF kliendi taotluse poolt ja selle meetodi kasutamiseks peab klassis olema atribuut [ServiceContract].

Tühja teenuse alguses lisatakse ainult üks OperationContract, mida klient võib kasutada, näiteks ’Registripäringu laadimine tehingu Id järgi’.

```
namespace EnotarWcfService
{
   [ServiceContract]
   [ServiceKnownType("GetKnownTypes", typeof (KnownTypesProvider))]
   public interface IEnotarService
   {
      [OperationContract]
      RegistriPäringDataset LoadNotaryRegistryRequestBy(int operationId);
   }
}
```

Peale seda kui OperationContract on määratud, on vaja sama kontrakti implementeerida veebiteenuse klassis EnotarService

```
namespace EnotarWcfService
{
   [AspNetCompatibilityRequirements(RequirementsMode
      = AspNetCompatibilityRequirementsMode.Allowed)]
```

```
  public class EnotarService : IEnotarService{
    public RegistriPäringDataset LoadNotaryRegistryRequestBy(int operationId){
      try{
        return new Fassaad().LaeTehinguRegistripäringud(operationId,
          TöötajaIdentity.LeiaThreadiTöötajaIdentity().Id);
      }
      catch (Exception ex){
        throw ErrorHandler.Handle(ex);
      }
    }
  }
}
```

Kuna mõlemad testitavad ASMX ja WCF kasutavad ühte ja sama teenuse fassaadi, ja fassaadi metoodi LaeTehinguRegistripäringud jaoks on juba kirjutatud test, siis ei ole vaja eraldi kirjutada test WCF jaoks.

Teenuse meetodi üle kandmise ja uue OperationContract loomise protsess kordub, kuni kõik valitud veebimeetodid vanast teenusest ei ole tõstetud uude teenusesse.

Uued keerulisemad andmetüübid peavad olema määratletud atribuutiga DataContract, et neid saaks serialiseerida. Vaikimisi DataContractSerializer järeldab andmeid lepingu kohta ja järjestab kõik tüübid avalikult nähtavaks. Kõik avalikud get/set omadused ja antud tüüpi väljad on serialiseeritavad. Selleks, et loobuda liigsest serialiseerimisest tuleb määratleda omadus või väli atribuudiga IgnoreDataMemberAttribute. Võimalikult selgesõnaliselt tuleb luua lepingu andmete abil DataContractAttribute ja DataMemberAttribute atribuudid. Tavaliselt toimub see kohandades DataContractAttribute atribuute tüübiks. Seda omadust võib kohandada klassidele, struktuuridele ja loenditele. DataMemberAttribute atribuut tuleb kohandada iga liikme jaoks, mille andmed on lepingu liiki, mis näitab, et see andmete liige on serialiseeritud (*serialized*).

Antud projektis on realiseeritud klass Document.cs, mis on määratletud atribuutiga DataContract. See on selleks, et loobuda DokumetDataseti saatmist kasutusele võetud Document-le, mis kirjeldab kõiki väljasid, mis on seotud dokument objektiga. WCF-is on võimalik kasutada ADO.NET tüüpe. ADO.NET on kogum arvuti tarkvara komponente mis

arendajad saavad kasutada juurdepääsuks andmetele ja teenustele, mis põhineb lahti ühendatud DataSet ja XML.

XML ja ADO.NET Types kasutamine Data Contracti sees.

WCF lepingu andmete (*data contract*) mudel toetab teatud andme tüüpe, mida XML esindab otse. Kui need tüübid serialiseeritakse XML-ks, serializer kirjutab XML sisu tüübi ilma täiendava töötlemiseta. Toetatud tüübid on XmlElement, massiiv XmlNode (kuid mitte XmlNode tüüp ise), samuti liigid, mis rakendavad IXmlSerializable. Samuti DataSet ja DataTable tüüpe, ehk nii nimetatud andmekogusid, mida tavaliselt kasutatakse andmebaasi programmeeriminel. Seda tüüpi implementeerivad IXmlSerializable liides ja on seega Serializable andmete lepingu mudelis.

DataSet tüüpi kasutatakse kliendi poolel, siis kui andmekogu, mis talletab andmeid on sarnane andmebaasi struktuurile. Objekt DataSet klass saab opereerida andmetega, salvestades need tabelid, mis on esitatud DataTable-is ja kasutada võtmeväljasid ja välisvõtmeid, et määratleda tabelite vahelised seosed, mida nimetatakse DataRelations.

DataSeti objekt on laialt kasutusel EN kliendi ja serveri andmete vahetusel. Data Access Layer on realiseeritud SQLHelper-i klassi abil, kus on implementeeritud kõik vajalikud meetodid, mida DataAdapter kasutab selleks, et täita DataSet objekt ja salvestada sama objekt baasi.

E-Notari kliendi poolt DataSet objekti kasutakse näiteks selleks, et salvestada *offline* reziimis tehinguid või tehingupõhjasid XML failidesse lokaalsesse masinasse, selleks, et hiljem salvestada baasi. DataSet objektis on konverteerimiseks XML-iks ja tagasi DataSet-iks, olemas kõik vajalikud meetodid. Lisaks on olemas kasulik funktsioonalsus DataSet objektil - see on võimalus ühendada (*merge*) kaks sarnast DataSeti üheks.

Andmekogud (*Datasets*) koos andmetega (*Datatables*) on liiga rasked objektid serialiseerimiseks. On tarvis kasutada tava objekte ja kogusid nagu List <T> või saata tava objekti tagasi ühe objektina.

Antud töös esitakse Dokument klassi realisatsioon DokumentDataSet asemel ja konverteri klass, mis teisendab dataset objekti tava objektiks. Nagu järgnevalt on näidatud – tava objekti saatmine on efektiivsem kui dataset objekti saatmine.

[DataContract]

```
public class Document : BaseKnownType, IDocument{
    [DataMember]
    public int Id { get; set; }
      // teised väljad ei näita
}
```

Lepingu andmed (*DataContract*) on formaalne kokkulepe teenuste ja kliendi vahel, et abstraktselt kirjeldada milliseid andmeid tuleb vahetada. Sisuliselt kujutab see ühte klassi, mis võib olla DataContractSerializable tüüpi. See tähendab, et suhtlemiseks klient ja teenus ei pea jagama samu tüüpe, vaid samu lepingu andmeid. Lepingu andmetes kirjeldakse elemendid, nende tüübid ja millises järjekorras nad on (kujuneb XML).

Vastuvõetavaks lahendusena võiks WCF teada, milliseid tüüpe kavatsetakse kasutada, sest see peaks olema kirjeldatud WSDL dokumendis. Selle eesmärgiks on vaja võtta kasutusele kas ServiceKnownType atribuut ServiceContract, KnownType lepingu andmed või lisada tuntud tüüp configuratsiooni faili.

```
<system.runtime.serialization>
    <dataContractSerializer>
        <declaredTypes>
            <add type="EnotarWcfService.Helper.BaseKnownType,
            EnotarWcfService ">
            <knownType type=" EnotarWcfService.DataContract.Document,
            EnotarWcfService "/>
             </add>
        </declaredTypes>
    </dataContractSerializer>
</system.runtime.serialization>
```

Windows Communication Foundation (WCF) kasutab serialiseerimise mootorit, mida nimetakse Data Contract Serializer, mis vaikimisi serialiseerib ja deserialiseerib andmeid (teisendab XML-st objektideks ja vastupidi). Kõik .NET Framework primitiivtüübid, nagu täisarvud ja stringid, samuti teatud tüüpe käsitletakse primitiividena, nagu DateTime ja XmlElement, võivad serialiseeritud olla ilma muude ettevalmistamisteta ja peetakse vaikimisi lepingu andmeteks. Paljusid .NET Framework tüüpe kasutavad ka olemasolevad lipingute andmed.

Näidis ServiceContract WCF teenusest projektis EnotarWcfService, pärast migreerimist

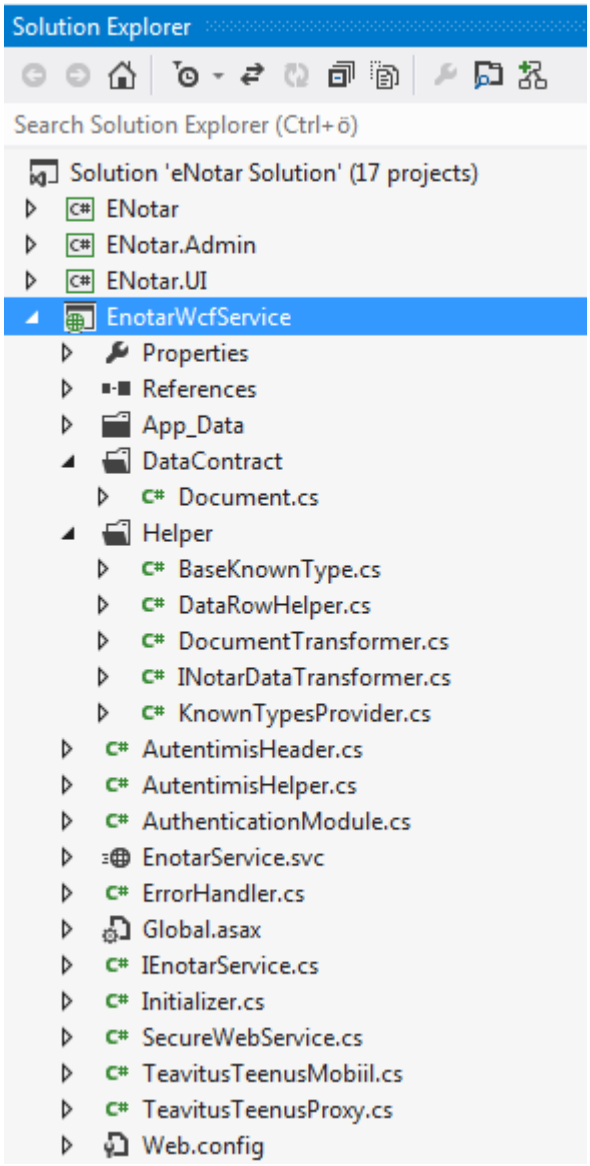
```
namespace EnotarWcfService
{
   [ServiceContract]
   [ServiceKnownType("GetKnownTypes", typeof (KnownTypesProvider))]
   public interface IEnotarService
   {
      [OperationContract]
      LookupDataset LoadLookup();

      [OperationContract]
      byte[] LoadLookupZip();

      [OperationContract]
      byte[] LoadNotarCalenderZip(DateTime begin, DateTime end,
         int operationId, int[] employeeIdList);

      [OperationContract]
      DokumentDataset LoadDocument(int documentId);

      [OperationContract]
      Document GetDocument(int documentId);

      [OperationContract]
      void SaveDocument(byte[] docByteArr);

      [OperationContract]
      TehingDataset LoadNotaryOperationWithShortView(int operationId);

      [OperationContract]
      RegistriPäringDataset LoadNotaryRegistryRequestBy(int operationId);

      [OperationContract]
      DokumentDataset LoadNotaryDocumentsBy(int operationId);

      [OperationContract]
      DokumendiOtsingDataset     SearchDocuments(DokumentDAL.DokumendiOtsingParam
param);

      [OperationContract]
      RegistriPäring RepeatXteeRegistryRequestBy(int primRequestId);
   }
}
```

Lõplik projekti struktuur peale kõik muudatusi on esitatud Joonusel 5.

DataContract kaustas peavad olema kõik klassid, mis peegeldavad andmebaasi tabeleid. Antud kontekstis on loodud Dokument DataContract, mis peegeldab üks ühele andmebaasi tabelit Dokument schema võrra.

Helper kaustas asuvad kõik abiklassid, mida kasutakse serialiseerimiseks ja erinevate tüüpide konvereerimiseks. Liides INotarDataTransformer pidi realiseerima kõiki klasse, mida tahab konverteeerida DataSet. Kuna DataSeti kasutamine on ebaeffektiivne WCF projektides, on vaja käsitleda võimalust konverteerida DataSet objektidest teist tüüpi objektideks, mis on märgitud attribuudiga DataContract.

Kogu funktsionaalsus, mis on seotud autentimisega ja autoriseerimisega on samuti üle kantud antud projekti vana teenuse implementatsioonist. See on koht, mida on vaja tulevikus uurida, kuna WCF pakub oma turvalisuse mudelit ja võimaldab seda Web.config failis määrata.



Joonis 5 E-Notari WCF projekt

WCF programmeerimise turvalisus põhineb kolmel sammul, mis on järgmised: turva režiimis, kliendi *credential* tüüp ja *credential* väärtus. Seda kõike on võimalik kas koodi või konfiguratsiooni kaudu realiseerida.

Praegune lahendus kuidas hoida ja identefitseerida kasutajd E-Notaris toimub läbi Thread.CurrentPrincipal.Identity. WCF puhul tuleb kasutada HttpContext.Current.User. Identity.

Oluline koht WCF konfigureerimisel on sõnumi suurus, mida teenus suudab vastu võtta. Seda suurust on võimalik muuta Web.config või app.config failis. Testimiseks see on

number seatud maksimaalseks kuid toodangu keskkonnas pidi see olema konfitud vastavalt vajadusele.

```
<bindings>
  <basicHttpBinding>
    <binding name="basicHttp" allowCookies="true"
             maxReceivedMessageSize="100000000"
             maxBufferSize="65536"
             maxBufferPoolSize="2147483647">
    </binding>
  </basicHttpBinding>
</bindings>
```

WCF kasutab palju puhvreid. Puhvrite loomine ja hävitamine iga kord on mahukas ja kallis tegevus, kui ressursside tühistamine, mis on seotud puhveriga, on ka kallis. Puhvri konfigureerimiseks kasutakse MaxBufferPoolSize ja MaxBufferSize muutujaid, mis on defineeritud web.config või app.config failis.

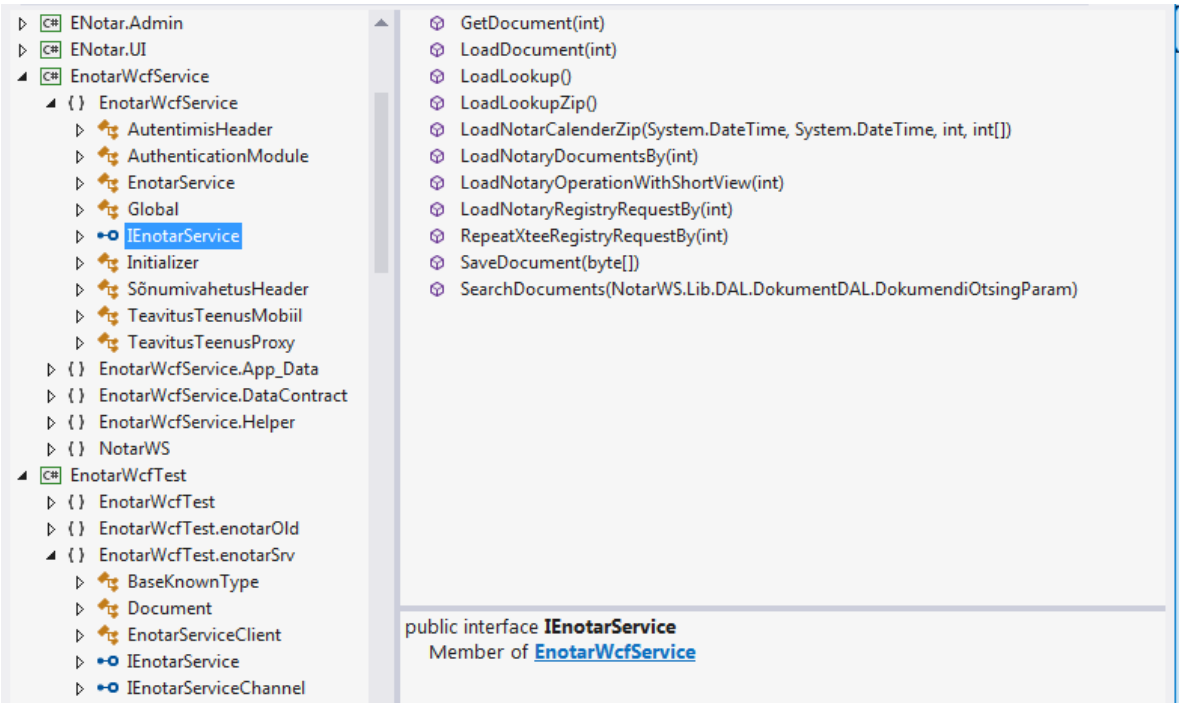
MaxBufferPoolSize näitab maksimaalset suurust baitides, mida kasutab puhvri kogum (pool) andmete töötlemiseks ja edastamiseks.

MaxBufferSize näitab maksimaalset suurust baitides, mis talletab sõnumeid, enne kui need töödeldakse konfigureeritud otspunktis koos *binding-ga*.

MaxReceivedMessageSize näitab maksimaalset sõnumi suurust baitides, mida on võiamlik töödelda.

Antud töös kasutakse põhitranspordi protokolli HTTP, kuigi tulevikus dokumendi sisu edastamiseks võib käsitleda TCP-d. WCF võimaldab seda konfigureerida ühes failis kirjeldades mittu *binding-ut.*

Joonis 6 Uue E-Notari teenuse Service Contract on esitatud lõplik interface ServiceContract, mis on nähtav välis klindile. Tahaks ära märkida, et tulevikus ikkagi on vaja hoida inglise keelne teenuse meetodi signatuur. Aga see on soovituslik, mitte kohustuslik.

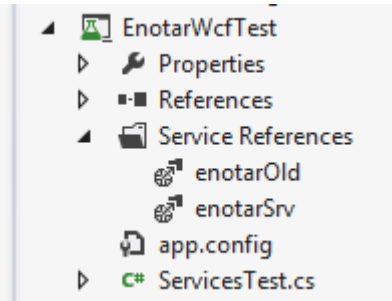


Joonis 6 Uue E-Notari teenuse Service Contract

Kui Service Contract on realiseeritud ja kõik testid on korras, siis järgmine samm on paigaldamine (*deploy*). Nagu olemasolev teenus paigaldatud IIS peale, on sama uue WCF implementatsioon paigaldatud samuti IIS-le, peale seda teenus on kättesaadav URL-i http://localhost/EnotarService.svc - Joonis 7.

Joonis 7 E-Notar WCF Service

Tugines TDD-le, kõik rakendatud meetodid on kaetud testidega. Testimiseks ja võrdlemiseks oli tehtud testprojekt EnotarWcfTest, kus oli lisatud kas viide vanale ASMX service (enotarOld) või WCF service (enotarSrv). Test projekti struktuur on näha Joonisel 8.



Joonis 8 Uue E-Notari teenuse test project

Kokku on kirjutatud kakskümmend üks testi, iga veebiteenuse meetodi jaoks kaks testi, üks mõõdab kui palju aega võtab vastuse saamine ASMX teenusest, teine mõõdab kui palju võtab vastuse samine aega WCF teenusest.

Lisaks on tehtud testimisel katse, mis mõõdab kui palju aega võtab andmete küsimine kui serialiseerimine toimub mitte DataSet objekti, vaid andmetüübi Document määratletud atribuutiga DataContract. Võrdluseks on võetud funktsionaalsus, kus kliendile saadetakse notaril dokument, kuna dokumendi suurus varieerub ja keskmine suurus on viis megabaiti faili kohta ning samas tehingus dokumentide hulk ei ole määratud. Sellega seoses oleks hea selgeks teha kui suur see vahe on. Vaata Lisa 1.

Teenuse meetodi täitmise aeg, mis tagastab DataSet objekt on:

DokumentDataset LoadDocument(int documentId); - 1,034 sec.

Teenuse meetodi täitmise aeg, mis tagastab Dokument objekt on:

Document GetDocument(int documentId); - 0,722 sec.

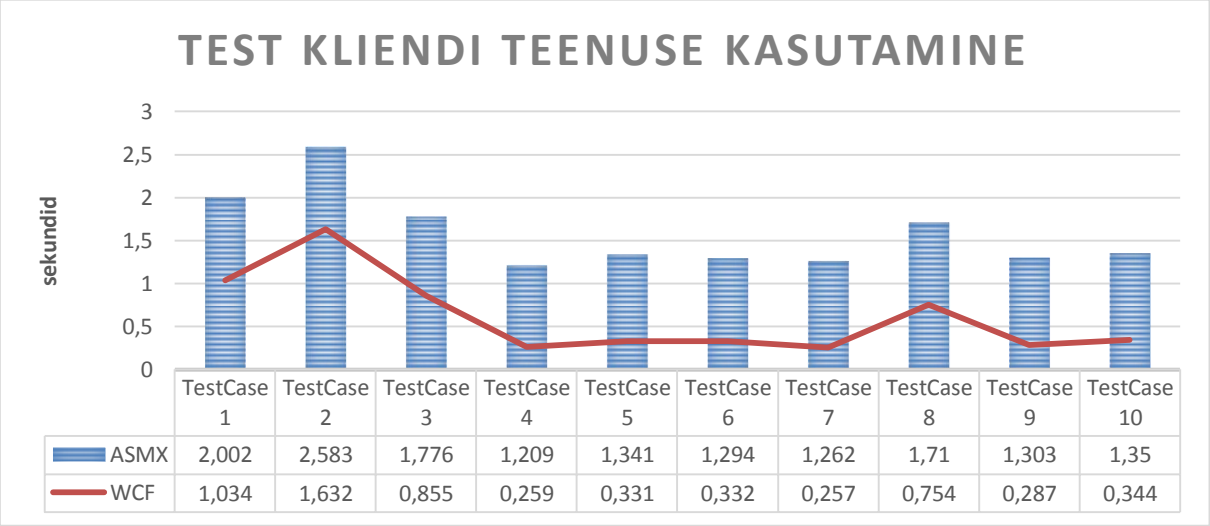
Sellest järeldub, et viie megabaidine fail DataSet objektis üle andmine WCF-le toimub ~ 0,3 sec aeglasemalt kui sama dokument, mida kasutatakse DataContracti objektis. Konfiguratiooni failis (app.config) kirjeldakse teenuste seadete parameetrid, mida kasutakse testimiseks, sidemed (*bindings*) ja andmete edastamise protokoll. Klient seksioonis määratakse kahe teenuse aadressid, kuhu saadetakse päringud ja millised sidemed neid kasutavad.

<system.serviceModel>

```
        <bindings>
          <basicHttpBinding>
            <binding name="BasicHttpBinding_IEnotarService" allowCookies="true"
              maxBufferPoolSize="2147483647" maxBufferSize=" 65536"
maxReceivedMessageSize="100000000" receiveTimeout="00:30:00"
sendTimeout="00:30:00" >
            </binding>
            <binding name="srvNotarSoap" allowCookies="true"
              maxBufferPoolSize="2147483647" maxBufferSize="65536"
maxReceivedMessageSize="100000000" receiveTimeout="00:30:00"
sendTimeout="00:30:00" >
            </binding>
          </basicHttpBinding>
        </bindings>

        <client>
          <endpoint address="http://localhost:8787/EnotarService.svc"
            binding="basicHttpBinding"
bindingConfiguration="BasicHttpBinding_IEnotarService"
            contract="enotarSrv.IEnotarService" name="BasicHttpBinding_IEnotarService" />
          <endpoint address="http://localhost:9000/srvNotar.asmx"
binding="basicHttpBinding"
            bindingConfiguration="srvNotarSoap" contract="enotarOld.srvNotarSoap"
            name="srvNotarSoap" />
        </client>
      </system.serviceModel>
```

Testide käivitamise tulemused esitatud Tabelis 3, testide kirjeldus esitakse Lisas 1. Antud kontekstis testitakse ainult SELECT lauseid, ehk küsitakse hulk andmeid andmebaasist. Andmete lisamine ja kustutamine ei ole teostatud ning seda võiks edasi uurida tulevikus. Näiteks on tehtud andmete saatmine – kliendi poolt uuendamiseks valitud dokumendi salvestamise näitel (TestCase 10). Testide tulemusest saame järeldada, et WCF abil andmete töötlemine toimub kaks korda kiiremini.

Tabel 3 Testimise tulemused



	TestCase 1	TestCase 2	TestCase 3	TestCase 4	TestCase 5	TestCase 6	TestCase 7	TestCase 8	TestCase 9	TestCase 10
ASMX	2,002	2,583	1,776	1,209	1,341	1,294	1,262	1,71	1,303	1,35
WCF	1,034	1,632	0,855	0,259	0,331	0,332	0,257	0,754	0,287	0,344

Tulevikus on vaja teostada põhalikum funktsionaalsuse testimise, mis on seotud tehingu andmete lisamisega ja uuendusega, kuna tehingu objekt on kõige kriitilisem ja mahukam.

Peale seda kui tehnoloogiat on võrreldud ja võimalik arendustraateegia on valitud, tuleb käsitleda riske, mis võida muudatuste protsess kaasta võib. Järgmises peatükis kirjeldakse riskijuhtimise lähenemist ja millised riskid kaasnevad teenuste vahetusega, pidadades silmas E-Notari süsteemi.

5. Riskijuhtimine

Riskijuhtimine on üks tarkvaratehnika tavasid, mis hindab pidevalt, mis saab valesti minna (ohud), määrab millised riskid on olulised ja rakendab strateegiaid, et tegeleda nende ohtudega. Riskijuhtimine ei ole uus idee, see on lahutamatu osa spiraal arendamise protsessist, mille on välja töötanud Barry Boehm ja on dokumenteerinud SEI (Software Engineering Institute) abil [4].

Eesmärk riski juhtimises antud töös on aidata jõudaa moderniseerimise kava koostamiseni, mis minimeerib moderniseerimise pingutust. Joonis 9 illustreerib UML *activity* diagrammil riski juhtimise meetmete lähenemist. Ovaalid diagrammil kujutavad tegevusi. Nooled tähistavad tegevuste üleminekut. Horisontaalne sünkroniseerimise baarid nõuavad lõpetamist senisele tegevusele enne uue alustamist. Protsess algab projekti moderniseerimisest, mis on valitud kasutades portfelli analüüsi. Protsess lõppeb integreeritud kaasajastamise kavaga [4].

Selle riski juhtimisemeetme alusel on võimalik rakendada E-Notaris edasi modernisatsioon. Näiteks antud töös valiti veebiteenuse kihi moderniseerimine (modernization candidate). Analüüsi käigus on valitud funktsionaalsed osad süsteemist, mis realiseeritakse uue teenuses.

Sihtgrupp (*Identify stakeholders*) kelle jaoks see muudatus teostakse on notaribüroo töötajad.

Nõuded (*Understand requirements*), jäävad samaks ehk tööprotsess ei muutu, nõuded mis kehtivad olemasoleva süsteemi teenuses pidid olema säilitatud ka uues teenuses. Kui tulevad mingid uued reeglid, siis tuleb luua ärimudeli (*Create the business case*).

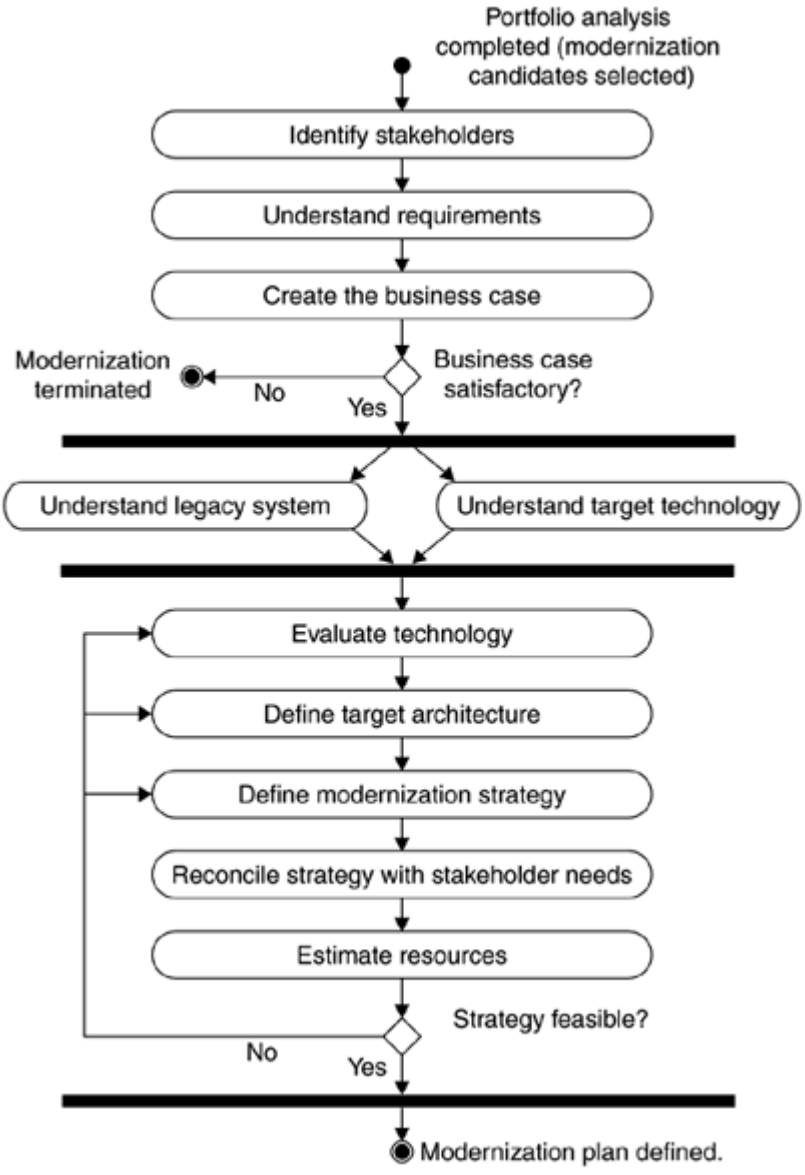
Kuna antud töös ei lisata uusi *business case* ja eeldatakse, et kõik kirjeldatud nõuded rahuldavad notari töötajaid, siis järgmise sammuna tuleb aru saada olemasolevast süsteemist ja aru saada sihtmärgiks seatud tehnoloogiast, mis antud kontekstis on WCF tehnoloogia.

Kui sihtmärgiks seatud tehnoloogia on valitud, on vaja hinnata kõiki eeliseid ja puudusi mis on antud. Selleks, et rääkida eelistest ja puudustest, on võrdlus teostatud olemasoleva süsteemi ASMX Web Service-ga ja suunatud rohkem päringute kiirusele

kasutades kahte erinevat tehnoloogiat.

Tänapäeval ja antud uurimistöös ei käsitle süsteemi arhitektuuri muutmist (Define target architecture), kuigi olemasoleva süsteemi arhitektuur võiks tulevikus liikuda SOA poole. Kuna E-Notar süsteem kasutab palju erinevad allsüsteeme, siis oleks mõistlik, et iga allsüsteem vastutab oma teenuste eest. See ongi praeguse EN teenuste puudus, sest kui teenus on maas, siis kogu büroo töö seiskub.

Strateegia kuidas moderniseerida (*Define modernization strategy*) EN teenust tugineb Dual-Spiral mudelile, mis on kirjeldatud peatükis 4.1.



Joonis 9 Riski juhitud moderniseerimine lähenemine[4]

Valitud tehnoloogia ja hindamise tulemused, koos moderniseerimise strateegiatega pidi olema kokku lepitud sihtgrupiga (*Reconcile strategy with stakeholder needs*). Resursside maht (*Estimate resources*) ei ole käsitletud antud töös. Riski juhtimise moderniseerimine lõpeb plaani koostamisega.

Edukas efektiivsemaks muutmine, nõuab otsust juhitud mudeli kindlaks tegemiseks ja erinevate riskide komponentide mõõtmiseks ning kõigi riski komponentide arvutamise üldist tulemuse mõju. E-Notari projektis käsitletakse järgmiseid riski komponente.

Deployment Risk Component

Deployment riski komponent – kahju tekkimise oht, mis on seotud praeguse organisatsiooni struktuuriga, toetamaks uue WCF teenuse kasutuselevõttu. Selle võib omakorda jaotada kolme tegevuspiirkonda: rakendus, infrastruktuur, hooldus.

E-Notari redaktsioonihalduse (release management) protsess on praegu seotud ASMX teenuse haldusega ja uue teenuse kasutamine nõuab uut korraldamist. Uue teenuse juurutamisel on vaja määrata rollid ja vastutused, kes tegeleb „release“ planeerimisega, kes vastutab arenduse ja disaini eest, konfiguratsiooni eest ning kes testimise eest? Peab olema ka isik, kes vastutab reliisi aktsepteerimise eest. Kui reliis ei ole õnnestunud, siis tuleb uurida kas teenuse taastamine on planeeritud.

E-Notari *deployment* puuduseks võiks pidada pädevate inimeste puudust, kes suudavad konfigureerida teenuse seadeid toodangukeskkonnas. See on ühtlasi ka risk, sest vale konfiguratsioon võib kahju tekitada.

Organizational Risk component

Organisatsioonilised riskid on seotud organisatsiooni struktuuriga, hoiakudega, kogemustega, eesmärkidega ja väärtustega (isiklik ja kultuuriline) organisatsioonis, kus olemasolev süsteem tegutseb, et toetada süsteemi arengut läbi efektiivsemaks muutmise. Olemasoleva süsteemi teenuse moderniseerimisel on vaja lisaressursse arendamiseks ja testimiseks. Notari Koja riskiks võiks pidada investeeringute ja eelarve riske. Uue teenuse kasutusele võtmine peab olema sama turvaline või turvalisem kui vana teenus. Suurimaks riskiks võiks pidada turvalisuse kvaliteedi langust.

Resource Risk component

Ressursside riskid on seotud kättesaadavusega ja kvaliteeti ressurssidega, mis sisaldavad riistvara, tarkvara, inimesi ja korduv kasutatavaid komponente, vastavalt olemasolevale eelarvele, ajakavale ning strateegiliste eesmärkide efektiivsemaks muutmisele arendamaks olemasolevat süsteemi.

E-Notari ressursside puuduseks võib pidada nii inimesi kui ka eelarvet. Teenuste kättesaadavus, peab olema tagatud vastavalt kokku lepitud teenuse kirjeldusele. Teenuse mitte kättesaadavuse aeg peab olema minimeeritud. Inimeste ressursid on piiratud ja ei ole pädevaid inimesi kellel on kogemus WCF teenuse arendamisel ja administreerimisel. Riistvara ja tarkvara ei ole arvesse võetud, kuna neid komponente pidevalt uuendakse.

Development process risk

Arendusprotsess seob erinevate osade väljatöötamise protsesse, et saavutada vastuvõetav otsus, mis on suunatud efektiivsemaks muutmise võimalustele. Käsitledes arendamistegevuse protsessi riske on vaja silmas pidada organisatsiooni eesmärke, mis ühendavad inimese optimaalsed füüsilised ja rahalised ressursid, et teha parem otsus arvestades olemasoleva süsteemi ümberkorralduste võimalusi.

E-Notar-i projektis puuduvad pädevad arendajad, kellel oleks kogemus WCF-teenuse loomisel. Seetõttu vajatakse lisaaega õppimiseks ning uute tehnoloogiate arendamiseks ja juurutamiseks. Olulisemaks võib pidada teenuse testimist, kuna EN suhtleb oma alamsüsteemiga - Pärimisregisteriga. Pärimisregister kasutab olemasolevat teenust EN-st andmete saamiseks. Arendusprotsessis võivad tulla riskid, mis mõjutavad pärimisregistri töötamist. Riskide hindamine ja nende mõju tööprotsessidele ei ole antud töös käsitletud.

6. Kokkuvõte

Töö põhieesmärgiks oli anda praktilisi nõuandeid ja kirjeldada arengustrateegiaid pärandsüsteemi veebiteenuse moderniseerimiseks, kasutades üldtunnustatud praktikaid ja metoodikaid. Uurimistöö käigus on kirjeldatud pärandsüsteemi olemasolev seisund E-Notari põhjal ja käsitletakse edasi arenemise vajadusi ja vajadust moderniseerida olemasolevat süsteemi. Moderniseerimine pidi toimima kolmel tasemel: klient, server ja andmebaasi liidese kiht.

Majanduslikke ja tehniliste piirangute tõttu ei saa praegu olemasolevat süsteemi nullist ümber kirjutada uuele süsteemile. See tähendab, et tuleb saavutada tasakaal ühelt poolt seatud piirangute osas olemasolevates süsteemides ja pakutavate võimaluste vahel notaribüroo töötajate tööprotsesside efektiivsemaks muutmisel. Seetõttu käsitletakse moderniseerimist ühes kihis ja edasist hooldust teistes kihtides, kuigi muudatusi ei saa teha ilma põhjaliku uurimiseta, mis mõju avaldavad muudatused teistele kihtidele. Antud töö kirjeldab veebiteenuse kihi moderniseerimist, kui ka edasi arengu starateegiat uue teenuse kasutamisel.

Olulisemaks tulemuseks võiks pidada Dual-Spiral mudeli kasutamist pärandsüsteemi reengineeringu puhul, nagu see on kasutatud antud töös vana veebiteenuse ASMX migreerimisel WCF teenuse peale ja efektiivsus sama teenuse kasutamisel. TDD kasutamine uue teenuse implementeerimisel on märkimisväärne tulemus, mis tuleb edasipidi kasutusele võtta igas kihis. Oluline aspekt päransüsteemi muudatuses on riski juhtimise lähenemine, mis kaasatakse teenuste vahetamisel.

Olulisemad järeldused pärandsüsteemi arengustrateegia analüüsis - võiks esile tõsta süsteemi muudatused läbi TDD. Ühtegi tegevust mis on seotud funktsionaalsuse üle viimisega uuele süsteemile või *refactoring* vana koodi ei saa teostada ilma eelneva testita. Samas, suured muudatused pidid alati olema analüüsitud, hinnatud ja planeeritud, tuginedes tunnistatud metoodikatele.

Antud uurimistöö raames püstitatud eesmärgid on täiesti saavutud ning saadud tulemused ja järeldused on kasulikud E-Notari süsteemis rakendamiseks ja edasi arendamiseks.

Summary

The main goal of this thesis is to offer practical advice and describe the Legacy Systems Modernization Strategies for web service using generally acknowledged practice and universally recognized methodology. The research consists of the description of the current state of an existing Legacy System using *E-Notary* example as well as of possible development and modernization perspectives. Modernization had to take place on three levels: Client, Server ja Data Access Layer (DAL).

It is impossible to overwrite the System from scratch due to the economic and technical. This means that a balance has to be struck between the boundaries of the existing system and the chances of optimizing the work process of a notary. Therefore, the modernization of one layer is analyzed separately from the maintenance of other layers. No change can be implemented without thorough analysis of its effects on other layers. The thesis describes the modernization of the Web Service as the preferred course of development strategy for a new Service.

Among the significant results of this paper, one can mention the use of the Dual-Spiral Model while reengineering the Legacy System: the migration of the old Service from ASMX to WCF, not forgetting the growth in efficiency of the WCF option. Another considerable outcome was applying TDD while implementing a new Service, which ought to be done on all three levels. An important aspect to consider is risk management approach in Legacy System changes.

System change through TDD is one of the most crucial consequences in analyzing Development strategy of Legacy Systems. No changes affecting System's functionality transfer should be implemented without extensive testing. In other words, big modifications must always be analyzed, assessed and carefully planned, relying on universally recognized methods.

All of the goals set for the research part of the thesis were reached. The gathered results and the consequences drawn are in favor of implementing and developing of the *E-Notary* System.

Kasutatud kirjandus

[1] E-Notar, Avaldatud veebilehel RIK. [WWW] http://www.rik.ee/et/muud-teenused/e-notar (15.03.2015)

[2] E-Notar, IT avalikus halduses. Aastaraamat 2008. [WWW] http://www.riso.ee/sites/default/files/2.2.3_eNotar_RIK_IT2008.pdf (15.03.2015)

[3] Michael C. Feathers. Working Effectively With Legacy Code. Pearson Education, 2005. 3-19 lk.

[4] Modernizing Legacy Systems. [WWW] http://www.sei.cmu.edu/library/abstracts/news-at-sei/cotsspot4q02.cfm (15.03.2015)

[5] Pärandsüsteem [WWW] http://www.vallaste.ee (15.03.2015)

[6] Dolly M. Neumann. Evolution Process for Legacy System Transformation (IEEE, 1996)

[7] Basem Y. Alkazemi, Mohammed K. NourI.Towards a Framework to Assess Legasy System, 2013 IEEE International Conference. 924-928 lk.

[8] R.C. Seacord, D. Plakosh, and G.A. Lewis. Modernizing Legacy Systems. Boston, MA. Addison-Wesley, 2003.

[9] K. H. Bennett, M. Ramage and M. Munro. Decision model for legacy systems. IEE Proceedings - Software 146(3): 1999. 153-159 lk.

[10] Martin Fowler, James Lewis. Microservices [WWW] http://martinfowler.com/articles/microservices.html (12.04.2015)

[11] Indrek Hiie. Infosüsteemide ülalhoid [WWW] http://www.hiie.com/ito/IS_ylalhoid_v1.00.pdf (12.04.2015)

[12] Disconnected Systems [WWW] http://deposco.com/challenges/disconnected-systems/ (12.04.2015)

[13] L. Williams, E. M. Maximilien, and M. Vouk, “Test-driven development as a defect-reduction practice,” in ISSRE ’03: Proceedings of the 14th International Symposium on

Software Reliability Engineering, 2003. 1-3 lk.

[14] K. Beck and M. Fowler, ”Planning extreme programming.” Addison-Wiley, 2001.

[15] Shihab, E. ”Prioritizing Unit Test Creation for Test-Driven Maintenance of Legacy Systems”, Quality Software (QSIC), 2010 10th International Conference. 132-140 lk.

[16] Yan Liu, Qingling Wang, Mingguang Zhuang and Yunyun Zhu “Reengineering Legacy Systems with RESTful Web Service“. 2008, Annual IEEE International Computer Software and Applications Conference. 785-790 lk.

[17] Building Web Services the REST Way Contracts [WWW] http://www.xfront.com/REST-Web-Services.html (25.04.2015)

[18] E. J. Byrne. A conceptual foundation for software reengineering. Conference on Software Maintenance, Nov 1992. 226-235 lk.

[19] Lu Ping, and Xiaohu Yang. An evolutional model of legacy system reengineering. 9th Joint International Computer Conference, Zhuihai, China, Nov 2003. 160-164 lk.

[20] B. Boehm. A spiral model of software development and enhancement. IEEE Computer, Vol.21, May 1988. 61-72 lk.

[21] Marco Battaglia, Giancarlo Savoia, and John Favaro.RENAISSANCE: a method to migrate from legacy to immortal software systems. 2nd CSMR’98, March 1998. 197-200 lk.

[22] Veebiarendus [WWW] http://www.eneta.ee/oppimine/veebistuudium/Lehed/veebiarendus.aspx (25.04.2015).

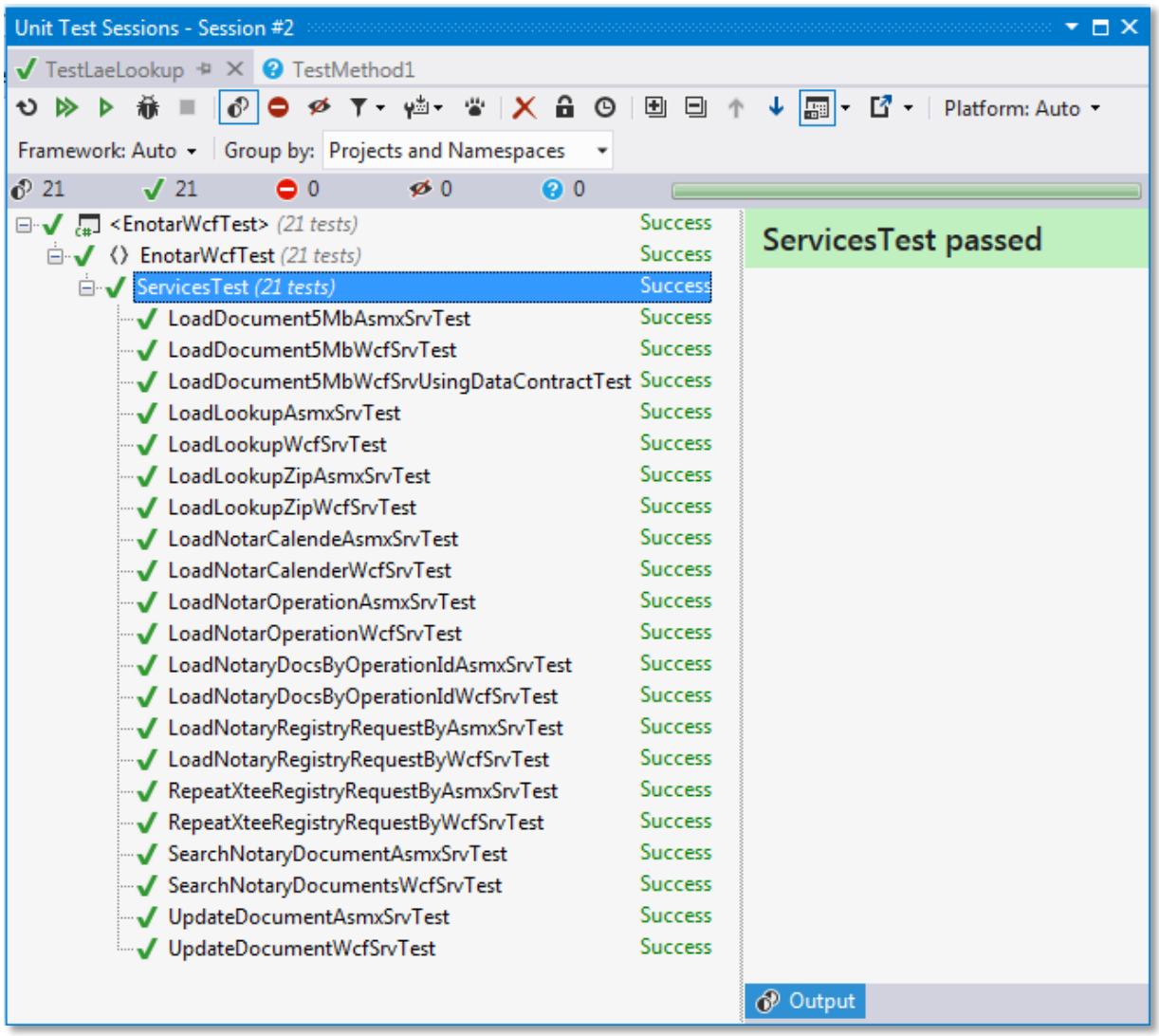
[23] Lu Chen, Xinyu Wang, Cristoforo, J. A Dual-Spiral Reengineering Model for Legacy System. TENCON 2005, IEEE Region 10, Nov 2005. 1-5 lk.

[24] Insurance Legacy Modernization Key Initiative Overview, Gartner (June 2013) [WWW] https://www.gartner.com/doc/2515317?ref=SiteSearch&sthkw=legacy&fnl=search&srcId=1-3478922254#784509177, (25.04.2015)

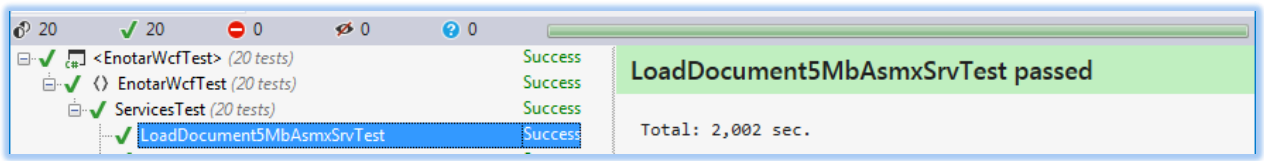
[25] K. Liu, A. Alderson, Z. Qureshi. Requirements Recovery from Legacy Systems by Analysing and Modelling Behaviour. Software Maintenance, 1999. Proceedings. IEEE International Conference. 2-3 lk.

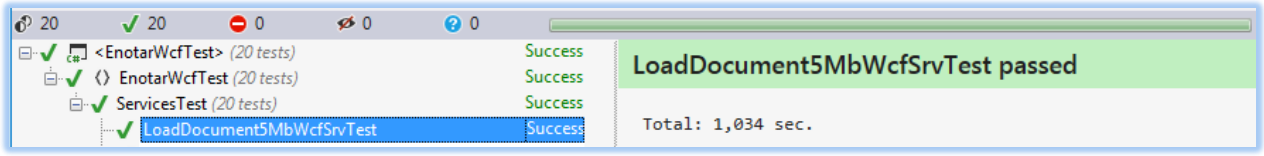
Lisa 1

NUnit testid asmx ja wcf teenuste kasutamisel.

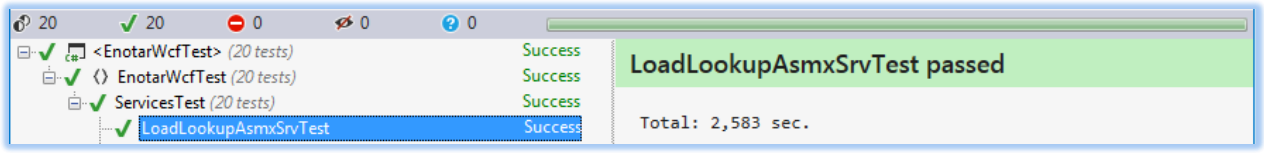


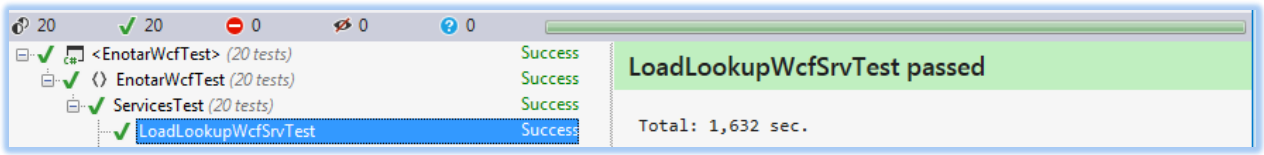
TestCase 1. Viie megabaiti dokumendi laadimine.



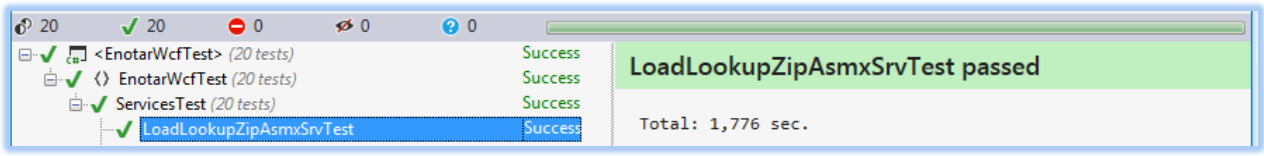


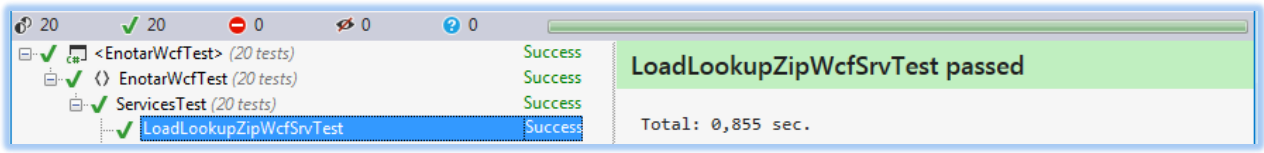
TestCase 2. E-Notari klassifikaatori laadimine(Dataset).



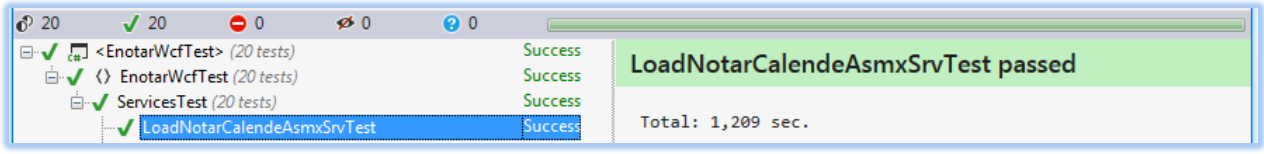


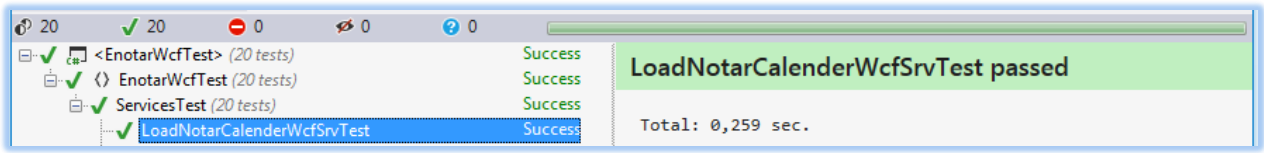
TestCase 3. E-Notari klassifikaatori laadimine(baiti jada).



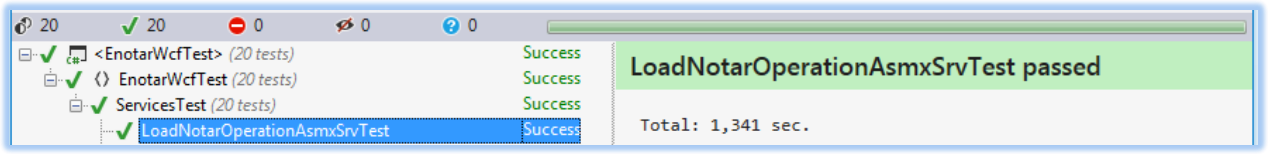


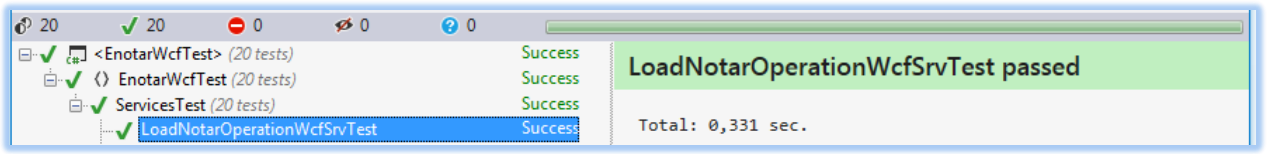
TestCase 4. Notari kalenderi laadimine.



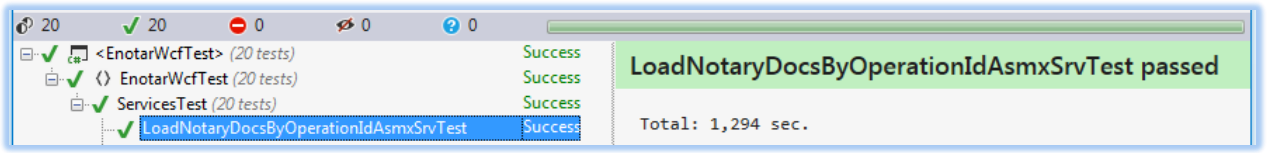


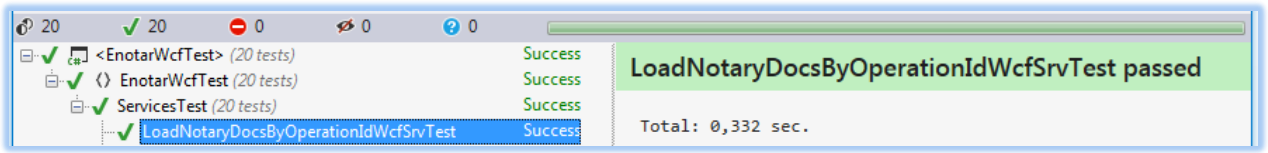
TestCase 5. Notari tehingu laadimine.



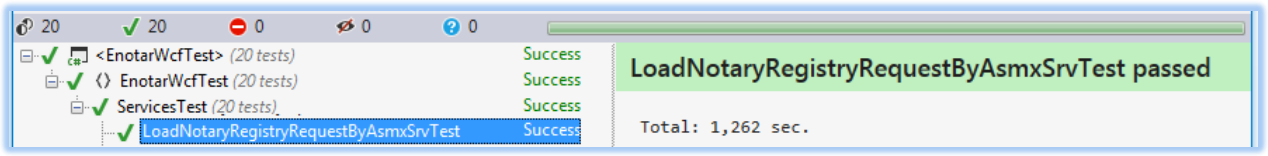


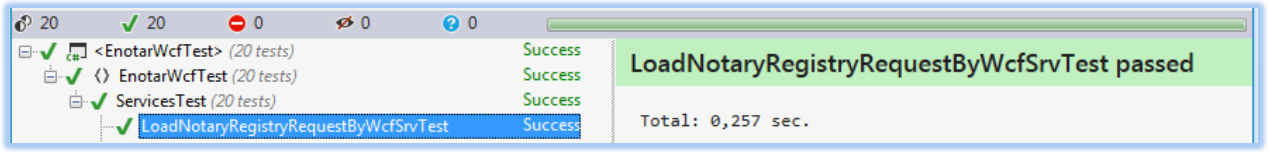
TestCase 6. Notari dokumendi laadimine tehingu Id järgi.



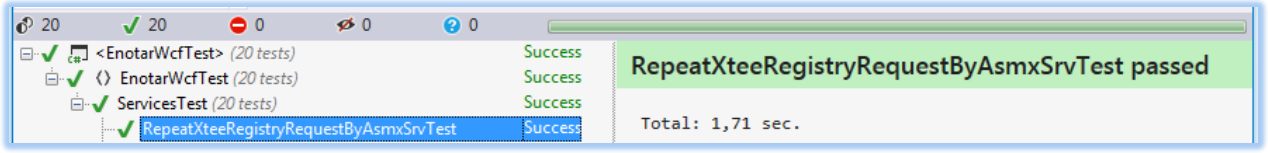


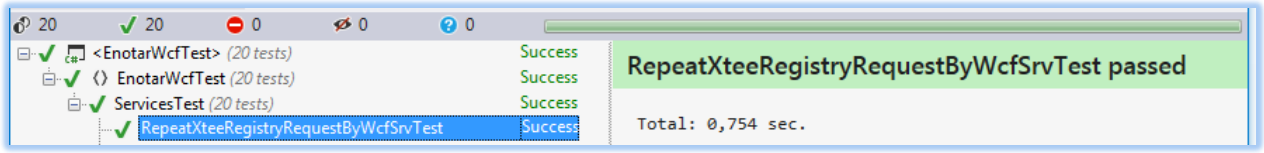
TestCase 7. Registripäringu laadimine tehingu Id järgi.



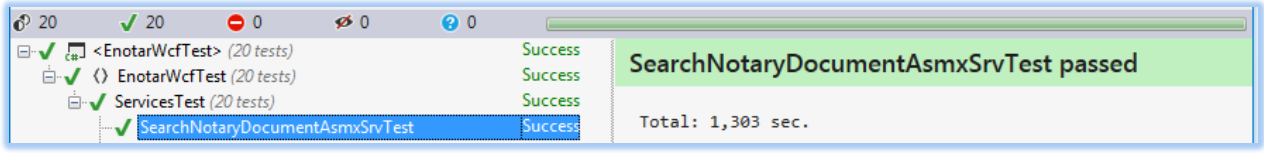


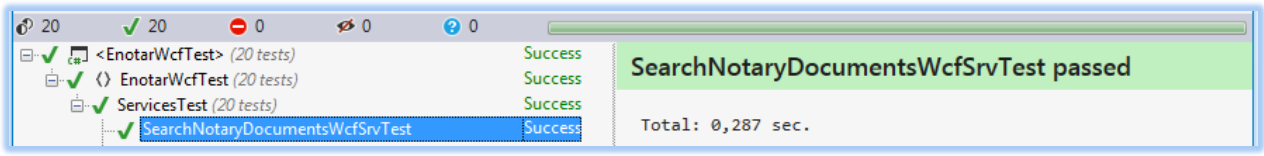
TestCase 8. Koordus registripäringu saatmine.



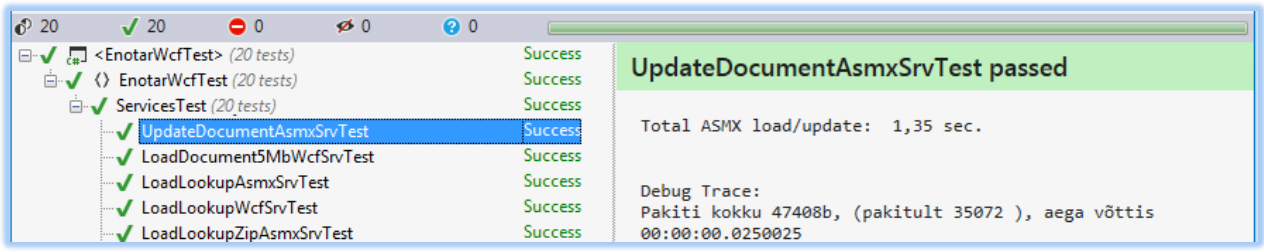


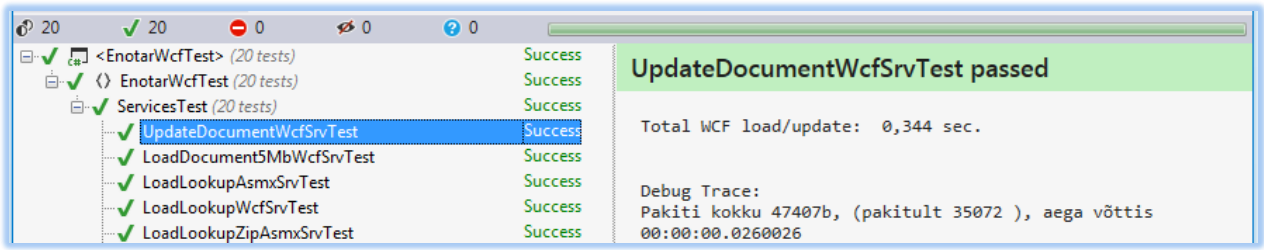
TestCase 9. Dokumentide otsing Notari Id järgi





TestCase 10. Dokumendi laadimine ja muutmine





TestCase 11. Viie megabaiti dokumendi laadimine (vaata TestCase1 võrdlemiseks), kasutades Dokument Data Contract