TARTU ÜLIKOOL
Matemaatika-informaatikateaduskond
Arvutiteaduse instituut
Tarkvarasüsteemide õppetool

Kaspar Loog

TARKVARA ARENDUSPROTSESS ERP-TARKVARA JUURUTUSPROJEKTIDES

Dissertatsioon *magister scientiarum* kraadi taotlemiseks informaatika erialal

Juhendaja: prof. Jüri Kiho

Tartu 2005

SISUKORD

SISSEJUHATUS

ERP-tarkvara (*enterprise resource planning software*) arendus ja juurutamine moodustab märgatava osa tarkvaraalastest projektidest. Paraku saadavad majandustarkvara juurutusprojekte sarnased probleemid, mida loetakse omaseks tarkvara arendusprojektidele. Tähtaegade venimine, eelarvete paisumine ja puudulik funktsionaalsus on ühtmoodi iseloomulik nii tarkvara arendusprojektidele kui ka ERP-tarkvara juurutusprojektidele.

Tarkvara arendusega seotud probleeme on saatnud suur tähelepanu ja selle tulemusena on välja töötatud mitmeid metoodikaid, mis peaksid aitama projekte paremini ja tulemuslikumalt juhtida. Viimasel ajal on valdavaks muutunud erinevate iteratiivsete ja inkrementaalsete tarkvara arendusmetoodikate levik. ERP-tarkvara puhul ei ole üldiste metoodikate valik nii suur – heal juhul on koos konkreetse paketiga juurutuspartnerile kättesaadav ka paketi arendaja poolt välja töötatud metoodika.

Käesoleva töö eesmärgiks on välja töötada sobiv tarkvaraarendusprotsess ERP-tarkvara juurutus- ja arendustöödeks Microsoft Navisioni näitel. Metoodika lähtepunktidena kasutatakse Navision OnTarget-it ja Rational Unified Process-i. Väljatöötamise aluseks on praktiline juhtumanalüüs, kus kasutati elemente mõlemast metoodikast.

Esimeses peatükis antakse ülevaade ERP-tarkvarast ja selle arengust. Lühidalt käsitletakse tarkvara arendusmetoodikaid, nende ajalugu ja tänapäeva. Metoodika rolli määratlemiseks vaadeldakse nii arendusprojektide kui ka ERP-tarkvara juurutusprojektide edutegureid ning võrreldakse neid, et leida erinevate metoodikate lähtepunktid.

Teises peatükis võetakse vaatluse alla tarkvaraarendusmetoodika unifitseeritud protsess. Unifitseeritud protsess on maailmas kõige enam kasutatav „raskekaaluline” iteratiivne ja inkrementaalne tarkvara arendusmetoodika. Peatükis antakse ülevaade tarkvara arendamise põhiprintsiipidest ja parimatest praktikatest unifitseeritud protsessi kohaselt.

Kolmandas peatükis käsitletakse ERP-tarkvara juurutusmetoodikat Navision OnTarget, mis on loodud ERP-paketti Microsoft Navision silmas pidades. Peatükis antakse metoodikast ülevaade koos mõningate praktiliste kommentaaridega. Täpsemalt võetakse vaatluse alla metoodika faasid ja ning käsitletakse praktilise rakendamisega seotud aspekte. Tegemist on esimese eestikeelse teadusliku käsitlusega Navision OnTarget metoodikast.

Neljandas peatükis vaadeldakse üht Microsoft Navision juurutusprojekti Eesti mobiiltelefonide distributsioonifirmas ja analüüsitakse, kuidas erinevate metoodikate komponentide rakendamine projekti käigule mõjus. Projekt seisneb ERP-tarkvara Navision ja kassamooduli Infostore kohendamises ja juurutamises. Järeldusena tehakse ettepanekud unifitseeritud protsessi ja Navision OnTarget-i metoodikate kombineeritud rakendamiseks ERP-tarkvara juurutusprojektides. Tehtud järeldusi saab rakendada lähtepunktina ERP-tarkvara juurutusmetoodika väljatöötamiseks projektile või juurutusorganisatsioonile.

1. TARKVARAARENDUSPROTSESSI VALIK ERP-TARVARA JUURUTAMISEL

1.1. Ülevaade ERP-tarkvara olemusest ja ajaloost

ERP-tarkvara (*enterprise resource planning*) on tarkvara, mis valmiskujul sisaldab funktsionaalsust enamlevinud ärifunktsioonide kajastamiseks ja juhtimiseks. Tüüpiliselt hõlmab funktsionaalsus raamatupidamist ja arveldamist, tootmise juhtimist, laofunktsionaalsust, CRM-funktsionaalsust (*customer relationship management*), inimressursside haldust ja palju muud. Tavapärane ERP-pakett on jagatud mooduliteks, mis võimaldavad ettevõtetel kasutada vaid vajalikku funktsionaalsuse osa. [Siringiniti 2000]

ERP-tarkvara juured on 1970-ndates aastates, kui suurtes ettevõtetes võeti kasutusele materjali- ja ressursijuhtimise süsteemid (MRP, *material requirement planning*). MRP-tarkvara oli oma sisult tootmistellimuste haldamise süsteem, mis võimaldas töökeskustele õigel ajal materjali ja tööd ette anda. MRP-tarkvara põhiline kasu peitus laovarude vähendamises, paranenud klienditeeninduses ja suuremas juhtimisest tulenevas efektiivsuses. [Siringiniti 2000]

1980-ndatel aastatel tuli kasutusele mõiste MRP II. Oma olemuselt tähendas see MRP laiendamist tootmise planeerimise ja töökeskustest saadava tagasiside arvestamisel planeerimisel. Näiteks, kui mõni töö venis või sai kiiremini valmis, sai MRP II süsteemi abil tootmisplaani jooksvalt ümber arvutada. 1980-ndatel hakati MRP II süsteeme suurarvutitelt (*mainframe*) üle viima personaalarvutitele. [Siringiniti 2000]

ERP on MRP II järjekordne edasiarendus, mis lisab paketile juurde finantsarvestuse, distributsiooni ja inimressursside halduse. Kõik need moodulid peavad olema integreeritud, et „kvalifitseeruda" ERP-süsteemina. Viimasel ajal on ERP-süsteemidele lisatud hulgaliselt mooduleid müügitoetuseks, e-tellimuste ning tarneahela halduseks. Tüüpiline juurutatud ERP-rakendus hõlmab endas kogu ettevõtte tööd alates kliendiga ühenduse võtmisest kuni tellimuse täitmise ja kauba kohaleviimiseni. [Siringiniti 2000]

Nõuded moodsa ERP-paketi jaoks on küllaltki laialdased, kuna rahuldada tuleb laia tarbijaskonna soove (mis on üldjuhul küll sarnased, kuid siiski väikeste variatsioonidega). Järgnevalt loetleme mõned tüüpilised nõuded, mis ei haaku konkreetse mooduli või valdkonnaga:

- tugi globaalsete korporatsioonidele – sama tarkvara peab saama kasutada erinevates maades integreerituna;
- tugi erinevatele platvormidele (seda nii serverite kui ka klientide osas);
- tugi erinevates valuutades finantsarvestusele;
- tugi mitmekeelsele kasutajaliidesele samas rakenduses;
- võimalus laiendada süsteemi kohendatud funktsionaalsusega.

Nimetatud nõuded kanduvad üle igale funktsionaalsele moodulile ning muudavad selle suurusjärgu võrra keerukamaks. Arvestades asjaolu, et funktsionaalsed moodulid omakorda peavad omavahel integreeritud olema, võib järeldada, et ERP-tarkvara on oma olemuselt väga keeruline infosüsteem, mille arendamiseks kulub märkimisväärne aja- ja rahakulu. Antud põhjusel ei ole keskmisele ettevõttele enamasti ka jõukohane endale sobivat ERP-tarkvara „rätsepatööna" arendada – see on liiga kallis ja riskantne.

Kuigi enamusel ettevõtetest toimivad tüüpfunktsioonid väga sarnaselt ja paketttarkvaraga on võimalik tarkvaralisi nõudeid rahuldada, leidub ettevõtteid, kes seavad tarkvarale lisanõudeid. Enamasti moodustavad need lisanõuded põhinõuetest väga väikese osa. Sellisel juhul on võimalik kasutusele võtta sobiv ERP-pakett ja vajalik osa kohandada või juurde arendada. Nii maandatakse suure arenduse kulukus ja risk ning saavutatakse väikese arendusprojektiga soovitud tulemus.

Kohandamisvajadus on tinginud samuti selle, et enamus ERP-pakette sisaldavad endas eraldi arenduskeskkonda või võimalust teha juurdearendusi. Näiteks käesolevas töös edaspidi vaadeldav tarkvara Microsoft Navision on osaliselt avatud lähtekoodiga – kogu äriloogikat puudutav funktsionaalsus on juurutuspartneritele avatud, muudetav ja täiendatav.

Maailmas hetkel enamlevinud ERP-tarkvara tarnijad on SAP, Oracle, Sage Group, Microsoft ja SSA Security. Maailma mastaabis levinud ERP-tarkvara paketid on Eesti taolise väikeriigi jaoks tihti liiga suured. Eestis on kasutusel väikestele ja keskmistele ettevõtetele (*small and medium-size business, SME*) mõeldud paketid nagu Microsoft Navision ja Microsoft Axapta, Epicor iScala ja Baan ning mõned kohalikud tooted. Kõik loetletud paketid vastavad suuremal või vähemal määral ERP-pakettidele esitatud nõuetele.

1.2. Tarkvara arendusprotsessid

Tarkvaratehnika on noor tööstus- ja teadusharu. Seetõttu on sellele omased ka teatud kasvuraskused, mida on nimetatud ka „tarkvara kriisiks". Selle kriisi tuntumaid indikaatoreid on 1994. aastast saadik Standish Group poolt välja antav CHAOS Report, mis uurib tarkvaraprojektide elluviimise edukust.

2003. aasta andmete põhjal tehtud CHAOS Report väidab, et 34% tarkvara arendusprojektidest on edukad. Edukaks loetakse projekti, mis valmib tähtajaks, kokkulepitud funktsionaalsusega ja ei ületa eelarvet. 15% projektidest katkestati enne valmimist ja 51% projektidest oli probleeme kas eelarve, ajakava või funktsionaalsusega. [CHAOS 2003]

Uuring toob esile ka põhilised projekti edu allikad:

1. juhtkonna toetus projektile;
2. lõppkasutaja seotus arendusprojektiga;
3. kogenud projektijuht;
4. selged ärieesmärgid, milleks tarkvara arendatakse;
5. vähendatud projekti maht;
6. standardse infrastruktuuri kasutamine;
7. kindlad põhinõudmised tarkvarale;

8. formaalne, range arendusmetoodika;
9. usaldusväärsed hinnangud;
10. kogenud meeskond.

Kui ülalloetletud edukriteeriumitest on mõni täitmata või puudu, siis on tõenäoliselt vaadeldaval projektil probleeme. Nende probleemidega tegelemiseks ja vältimiseks on välja töötatud mitmesuguseid tarkvara arendusmetoodikaid ehk protsesse.

Metoodika on süstemaatiline viis millegi tegemiseks [Hidding 1997]. Tarkvara loomise protsessi esimene süstematiseeritud käsitlus oli üldine kaskaad- ehk koskmudel. Dokumentidele orienteeritud kaskaadmudel oli domineerivaks metoodikaks paljudes valitsusagentuurides ja suurtes tarkvarakorporatsioonides. Kaskaadmudel pälvis palju kriitikat eelkõige selle pärast, et ei näinud ette eriti võimalusi prototüüpimiseks ja taaskasutamiseks. [Royce 1998]

Uue lähenemise tõi 1988. aastal Boehm spiraalmudeliga. [Boehm 1988] See on riskide analüüsimisele keskenduv lähenemine tarkvara arendamisele ning ühendab kaskaadmudeli ja prototüüpimise parimad omadused. Spiraalmudelil on iseloomulik kaskaadmudelile omane läbipaistvus, hea riskide maandamine projekti varajastes faasides ning võimalus tarkvara järk-järgult kasutusele võtta. [Kruchten 2000]

Spiraalse arendusmetoodika alusel on loodud mitmeid protsesse, seal hulgas ka käesolevas töös vaadeldav unifitseeritud protsess. Spiraalseid protsesse on kritiseeritud nende mahukuse ja väikestele projektidele-meeskondadele sobimatuse tõttu. Seetõttu on viimasel leidnud suuremat populaarsust nn väledad (*agile*) metoodikad. Väledad metoodikad eelistavad isikuid ja suhtlust protsessile ning vahenditele, töötavat tarkvara kõikehõlmavale dokumentatsioonile, kliendiga suhtlust lepinguläbirääkimistele ja muutustele vastamist plaani järgimisele. [Agile Manifesto 2001]

1.3. ERP-tarkvara juurutusmetoodika

Eelnevas peatükis mainitud metoodikad on kõik orienteeritud tarkvara arendamisele. Tarkvara arendusmetoodika ei pruugi aga tingimata sobida valmispaketi kohendamiseks

ja juurutamiseks. Tarkvara arendamine lähtub asjaolust, et tarkvara veel pole olemas ja see tuleb peaaegu nullist arendada. ERP-tarkvara juurutusprojekt aga lähtub asjaolust, et suurem osa nõuetest peaks olema täidetud juba standardses paketis, mida tuleb vajadusel kohendada ning täiendada.

Lähtepunktide erinevust arvestades on enamus ERP-tarkvara tootjatest koos tarkvaraga teinud partneritele kättesaadavaks oma juurutusmetoodika (*implementation methodology*). Tavapäraselt on juurutusmetoodika kättesaadavus isegi piiratud konkreetset tarkvara juurutavate partneritega. Näiteks jagab Microsoft Navision OnTarget metoodikat vaid sertifitseeritud ja potentsiaalsetele partneritele. Seetõttu pole kirjanduses ERP-tarkvara juurutusmetoodikaid eriti käsitletud või on käsitlus baseerunud sarnastel alustel tarkvara arendusprojektidega.

Põhiraskused ERP-tarkvara juurutamise õnnestumiseks on olulised järgmised tegurid: [Nah, Lau 2001]

- juurutusmeeskonna koosseis ja aja planeerimine – kliendipoolselt peab olema saadaval piisavalt otsustusjõuga inimesi ja neid tuleb piisavalt motiveerida, et nad projektiga igapäevatöö kõrvalt tegeleksid;
- tippjuhtkonna toetus;
- äriplaan ja visioon;
- efektiivne kommunikatsioon – kogu kliendi organisatsiooni tuleb projektist pidevalt informeerida;
- projektijuhtimine;
- projekti „staar” (*project champion)* – keegi kliendi organisatsiooni sees peab olema entusiast uue tarkvara juurutamisel;
- olemasolevad süsteemid – projektile on kasulik, kui organisatsioonis on eelnevalt juurutatud tarkvarasüsteemid, mis aitavad inimestel sõnastada nõudeid uuele tarkvarale;

- muudatuste halduse protsess ja selle järgimine;
- äriprotsesside muutmine ja minimaalne kohandamine – kui on valida tarkvara ja äriprotsesside muutmise vahel, siis suurema edu tagab äriprotsesside muutmine;
- tarkvara arendamine, testimine, vigade parandamine;
- monitooring ja tulemuslikkuse hindamine.

Tarkvara arendusprojektidega võrreldes on ERP-tarkvara edukuse kriteeriumid rohkem sõltuvad kliendi organisatsioonist ja inimestest. Osad tegurid nagu juhtkonna tugi ja muudatuste haldus langevad samas kokku CHAOS Report-i järeldustega. Seega võib öelda, et ERP-tarkvara juurutamise edutegurid mõnevõrra erinevad tarkvara arendusprojekti eduteguritest.

ERP-tarkvara edutegurites on huvitaval moel sees ka tarkvaraarendusega seotud teema – tarkvara arendamine, testimine ja vigade lahendamine. See näitab, et enamuses ERP-tarkvara juurutusprojektides leidub „tavapärast" arendustegevust. Arendustegevuse efektiivsus mõjutab ka juurutusprojekti kui terviku edukust. See tähendab, et ERP-tarkvara juurutusmetoodikas peab sisalduma ka arendusmetoodika elemente. ERP-projektide puhul, mis hõlmavad suuremal hulgal arendustöid, on tarvilik leida kompromiss arendusmetoodika ja juurutusmetoodika vahel. Käesolevas töös vaadeldakse kahte metoodikat. Esiteks käsitletakse tarkvara arendusmetoodikat unifitseeritud protsess ning teiseks ERP-tarkvara juurutusmetoodikat Navision OnTarget. Töö lõpus antakse hinnang, milliseid elemente kummastki metoodikast on kasulik rakendada arendustöid sisaldava ERP-projekti puhul.

2. UNIFITSEERITUD PROTSESS

2.1. Unifitseeritud protsessi olemus

Unifitseeritud tarkvaraarendusprotsess on maailmas hetkel *de facto* üks enamlevinud ja -kasutatav protsessikirjeldus. Selle algsed autorid on Grady Booch, Ivar Jacobson ja James Rumbaugh. Unifitseeritud protsess (edaspidi kasutatakse lühendit UP) hõlmab endas peaaegu kõiki eduka tarkvaraprojekti läbiviimiseks vajalikke distsipliine, töökirjeldusi ja juhiseid.

Tarkvaraprojekti teostamine on väga mitmetahuline protsess. Ühelt poolt on tegemist turundusliku tegevusega – kliendisuhtlus, intervjuud, uuringud. Teisalt on kaasatud juhtimisteadus – meeskonnajuhtimine, aja ja ressursside planeerimine. Kolmandaks on tarkvaraprojektis alati ka tehnika elemente nagu tehniline analüüs ja programmeerimine. Unifitseeritud protsess kirjeldab kõiki neid tahke ka ning annab projektiosalistele juhiseid protsessi edukaks rakendamiseks. [Rational Unified Process 2001]

Unifitseeritud protsessi väljatöötamine toimus tarkvarafirma Rational Software Corp. (edaspidi Rational) finantseerimisel ning sellesse kaasati ka palju tarkvara arendamise ja müügiga seotud firmasid ning eraisikuid. Kuna Rational soovis, et protsessikirjeldusest saaks rahvusvaheline standard, siis anti see arendamiseks sõltumatule organisatsioonile Object Management Group. Nii on põhimõtteliselt samal protsessikirjeldusel kaks nime – unifitseeritud protsess ja *Rational Unified Process* (edaspidi ka RUP). Neist viimast müüakse eraldi tarkvaratootena.

Autori arvates protsessi Rationali ja nn sõltumatu versiooni vahel põhimõttelist vahet pole. UP on välja antud raamatuna, RUP-i saab soetada tarkvaratootena. UP-is on palju lehekülgi pühendatud protsessi olemusele, RUP-is on lisaks sellele palju praktilisi näpunäiteid ning ka valmis dokumendistruktuure. Kokkuvõtteks – kui UP on põhiliselt protsessikirjeldus, siis RUP keskendub protsessi reaalsele rakendamisele ning annab selleks ka praktilisi juhiseid ning vahendeid.

Unifitseeritud protsessiga paralleelselt loodi ka diagrammkeel – unifitseeritud modelleerimiskeel (UML, *unified modelling language*). UML võimaldab üheselt kirjeldada loodava tarkvara funktsionaalsust nii, et sellest saaksid aru nii süsteemi arendajad kui ka süsteemi kasutajad. Hetkel on UML üks enamlevinud modelleerimiskeeli maailmas. UML-i eesmärk on, et nii spetsialistid, juhid kui ka kasutajad võiksid loodavast tarkvarast võimalikult ühtemoodi aru saada. Diagrammid on siin universaalseim ja kiireim moodus eesmärgi saavutamiseks. [Fowler 1997]

Unifitseeritud protsessis on kasutusel termin 4P [Jacobson *et al*. 1999]:

- projekt (*project*) – tarkvaraprojekt,
- toode (*product*) – tarkvaratoode,
- protsess (*process*) - tarkvaraarendusprotsess,
- inimesed (*people)* – tarkvara arendavad inimesed.

Selle loetelu abil üritatakse näidata tarkvaraarenduse raskuskohti. Neli elementi ei ole üksteisest eraldiseisvad – nad mõjutavad üksteist ning on väga suurel määral ka üksteisest sõltuvad.[Jacobson *et al.* 1999]

Tarkvaratoote mõiste all ei mõelda vaid töötavat programmi. Lisaks programmkoodile käivad tarkvaraga kaasas kavandamise ajal loodud dokumendid, arhitektuuri kirjeldus, testikirjeldused ja veel palju muud. UP autorid on kõigi taoliste informatsioonikildude kirjeldamiseks loonud mõiste artefakt (*artifact*) [Littover 2000][1]. Kõik artefaktid kokku moodustavadki tarkvaratoote. Näiteks paralleelina majaehitusest võiks siin valmis tootena käsitleda maja koos projekti, algjooniste ja ehitusloaga.

Autori arvates räägitakse tarkvaraprojektide puhul liiga palju tehnilistest elementidest. Projekti õnnestumiseks on need küll olulised, kuid mitte peamised. Tarkvara tehakse ikkagi inimeste poolt ja inimeste jaoks. Seega on vaja, et tarkvaraprojektil oleks piisavalt lihtsustatud väljund, et asjaga seotud inimesed asjasse liialt süvenemata aru saaks, millega on tegu. Põhilised riskid tarkvaraarenduses ongi seotud eelkõige

[1] Mõned eesti autorid kasutavad mõistet “tehis“ inglisekeelse termini *artifact* tähistamiseks. Käesolevas töös kasutatakse sünonüümina ka terminit „tulem”.

inimestega ja nende omavahelise suhtlemisega (õigemini selle puudumisega). Õigesti koostatud väljundid aitavad inimestel omavahel projekti raames suhelda.

Jacobson, Booch ja Rumaugh on toonud inimeste ja projektidega seoses välja järgmised punktid, mida peaks iga projekti puhul jälgima [Jacobson *et al*. 1999]:

- Projekt peab olema kasutoov – mõttetute asjade tegemine ei meeldi kellelegi.
- Riskid peavad olema välja toodud – inimestele ei meeldi ebameeldivad üllatused.
- Väike meeskond – suures meeskonnas muutub suhtlemine keeruliseks ning töö kannatab seetõttu. “Parajaks“ meeskonna suuruseks loetakse 6-8 inimest.
- Realistlik ajakava – kui algusest peale on teada, et tähtaega pole mingil juhul võimalik saavutada, langevad inimeste moraal ja töövõime. See omakorda aga lükkab valmimise veelgi kaugemasse tulevikku.
- Arusaadavus – inimesed peavad teadma ja mõistma, miks projekti tehakse. Selleks peab neil olema ülevaade kogu projektist, isegi kui töö käib teatud väikese alamosa kallal.
- Saavutusvajadus – spiraalse tarkvaraarenduse puhul saavad inimesed tihedamini tagasisidet. See aga tekitab positiivse saavutusega seotud emotsiooni. Lühikesed tähtajad tõstavad ka töötempot.

Tarkvaraprojektide võtmekomponentidest on unifitseeritud protsessi autorite poolt välja jäetud projektimeeskonna väline roll – kasutaja. Kasutaja all mõistetakse inimest, kes tarkvara kasutama hakkab. Ehkki protsessikirjelduses tuuakse tihti välja kasutajate kaasamise olulisust, ei ole sellele rohkem tähelepanu pööratud. Samuti pole kasutatavusele (*usability*) kuigivõrd tähelepanu pööratud, ehkki see omab kasutajahinnangu vaatepunktist kriitilist tähtsust. Kasutajaliides on ainus asi, millega kasutaja kokku puutub ning kasutajaliidese kasutatavus määrab ära ka kasutaja hinnangu tarkvarale. Teisalt tuleb silmas pidada, et unifitseeritud protsess on universaalne ning tegeleb ka tarkvaraprojektidega, millel ei ole kasutajaliidest.

Unifitseeritud protsessi võtmekomponente vaadeldes tuleb muidugi arvestada ka seda, et protsessikirjeldus ei ole loodud tarkvara tellivate organisatsioonide arendamiseks vaid tarkvara loovate organisatsioonide arendamiseks. Seetõttu on ka mõistetav, et inimeste käsitlemisel on kasutaja roll vaatluse alt välja jäänud ning inimeste all mõeldakse vaid

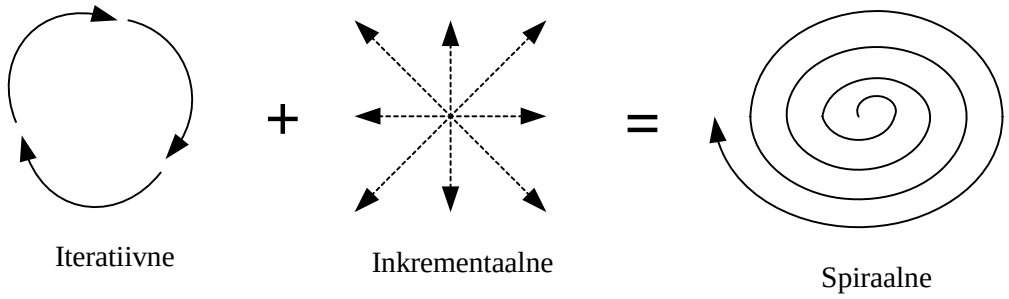
arendusmeeskonda. Autori arvates tuleks unifitseeritud protsessi veidi edasi arendada, et ka kasutajate roll oleks tegevustesse paremini integreeritud. Hetkel käsitleb unifitseeritud protsess kasutajaid vaid tarkvara sisendi ja väljundi staatuses.

Unifitseeritud protsessile pakuvad viimasel ajal suurt konkurentsi nn väledad protsessid. Põhjus peitub selles, et väiksemad arendusfirmad ja -meeskonnad ei suuda enda jaoks unifitseeritud protsessi kohandada ja rakendada. Samuti tajuvad väga paljud autorid unifitseeritud protsessi liigset keskendumist tulemitele, eemaldudes inimestest. [Otsason 2003]

2.2. Iteratiivse ja inkrementaalse arenduse olemus unifitseeritud protsessis

Unifitseeritud protsessi suurim muudatus võrreldes tavapärase tarkvaraarendusega on teistsugune lähenemine projektijuhtimisele. Tavapärane koskmudelil põhinev lähenemine nägi eelkõige ette projekti täies mahus valmis tegemist. Unifitseeritud protsess aga näeb projekti eduka läbiviimise alusena selle tükeldamist väiksemateks realiseeritavateks tükkideks. [Kruchten 2000]

Iteratiivne arendus tähendab, et arendus käib tsükliliselt – projekt on jagatud hästi defineeritud alamprojektideks. Inkrementaalne arendus tähendab, et arendus käib laienevalt – projekt ei valmi korraga, vaid tükk-tüki haaval laienedes. Mõlemad mõisted kokku võttes saame spiraalse arenduse (vt. joonis 2.1). Iteratiivsuse ja inkrementaalsuse mõiste on sisse toodud Rationali poolt. Põhjuseks oli ilmselt soov eristada RUP-i spiraalsest arendusest. Autori arvates on aga tegemist turundusvõttega RUP-i paremaks müümiseks. Seetõttu kasutataksegi kirjanduses üha enam spiraalse arenduse mõiste asemel iteratiivse ja inkrementaalse arenduse mõistet.



Joonis 2.1. Iteratiivsuse ja inkrementaalsuse mõiste (autori joonis).

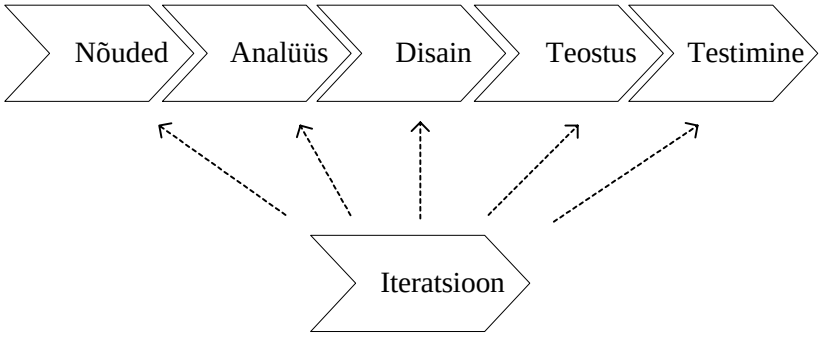
Tarkvaral on elutsükkel, mis algab projekti algatamisega ning lõppeb projekti "surmaga". UP autorid jagavad selle tinglikult neljaks – algatus (*inception*), kavandamine (*elaboration*), realiseerimine (*implementation*) ja siire (*transition*). Faasid jagatakse üheks või rohkemaks iteratsiooniks (*iteration*) ehk miniprojektiks. Igas iteratsioonis läbitakse töövood (*workflows*). [Rational Unified Process 2001]

Tarkvara elutsükkel on mõnes mõttes sarnane turunduses kasutatava toote elutsükli teooriaga, kuid mõlema elutsükli käsitlemisel on täheldatav oluline erinevus. Turunduses kasutatav toote elutsükli teooria lähtub eelkõige olemasoleva toote ostetavusest teatud turgudel. Tarkvara elutsükli puhul nähakse siin nii tarkvara kasutamist kui ka arendamist seni, kuni seda enam tarvis ei lähe. Toote ja tarkvara elutsüklil on küll niipalju ühist, et elutsükli pikendamiseks võib toodet ja ka tarkvara edasi arendada. [Dempsey *et al.* 1998]

Antud lähenemisel on kaks eesmärki. Iteratsioonideks jagamine ehk itereerimine võimaldab projekti paremini juhtida ning tähtaegu reaalselt tajutavamaks teha. Iteratsioon on põhimõtteliselt miniprojekt. Igas iteratsioonis läbitakse viis erinevat töövoogu ehk distsipliini [Rational Unified Process 2001]:

- nõuded,
- analüüs,
- disain,
- teostus,
- testimine.

Viis töövoogu on toodud abstraktsioonina, et protsessi olemust paremini edasi anda. Tegemist ei ole viie erineva etapiga, vaid küllaltki läbipõimunud protsessiga, mida oleks võinud ka näiteks neljaks töövooks jaotada [Jacobson *et al*. 1999]. Iteratsiooni ja töövoogude omavahelist suhet illustreerib joonis 2.2.



Joonis 2.2. Töövood iteratsioonis. [Jacobson *et al*. 1999]

Töövoogude järgnevus meenutab struktuurilt suhteliselt unifitseeritud protsessile eelnenud koskmudeli metoodikat. Koskmudelist erineb UP siiski kahe olulise põhimõtte poolest. Esiteks tegeletakse igas iteratsiooni töövoos projekti mingi konkreetse alamosaga, mitte kogu projektiga. Teiseks on kõik töövood omavahel läbipõimunud, mitte kindlas ajalises järjestuses. Töövoo mõiste ongi sisse toodud iteratsioonis toimuvate tegevuste paremaks liigendamiseks, need ei ole järjest toimuvad tegevused. [Jacobson *et al.* 1999]

Iga iteratsiooni läbimise tulemusena kas luuakse uusi või arendatakse olemasolevaid artefakte. Artefaktide abil saab mõõta projekti edenemist ning ajagraafikus püsimist. Seetõttu võib ka välja tuua iga töövoo eesmärgid. Töövoogude eesmärgid on ära toodud tabelis 2.1.

Iteratsioonide mõtte paremaks edasiandmiseks illustreerime seda näitega ehitusvaldkonnast. Maja ehitamine on üks suur projekt. Projekti valmimiseks on vaja luua ja kinnitada arhitektuuriline lahendus, valada vundament, ehitada majakarp ja teha siseviimistlus. Iga sellise alamprojekti võime nimetada iteratsiooniks. Iga iteratsiooni lõppedes saab öelda, et võrreldes iteratsiooni algusega on maja mingi osa võrra valmimisele lähenenud. Käsitletud iteratsioone on graafiliselt kujutatud joonisel 2.3.

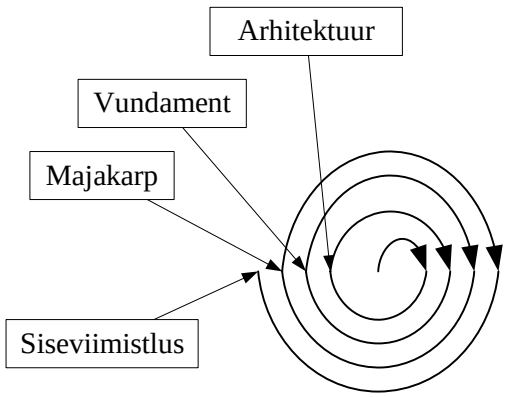
T a b e l 2.1

Töövood ja nende eesmärgid

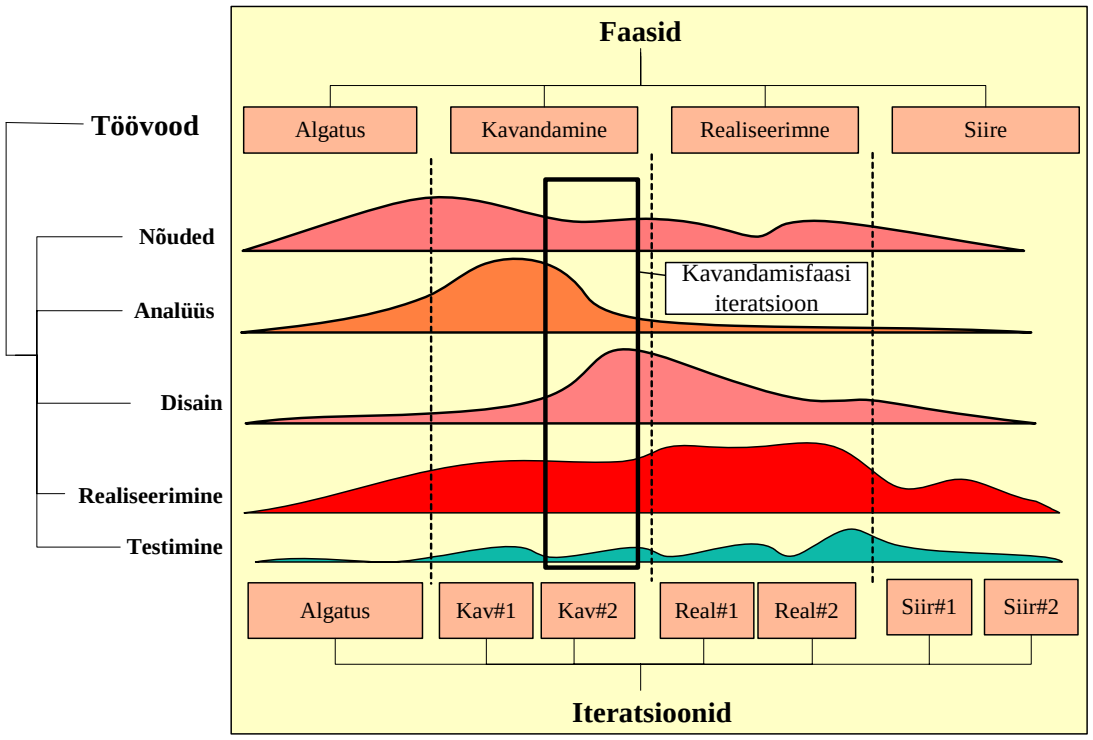
Töövoog	Eesmärgid
Nõuded	Nõusolek süsteemi arendajate ja tellijate vahel, mida süsteem tegema peaks Määratleda süsteemi piirid Aja- ja rahakulu hindamine
Analüüs	Nõuete täpsustamine Süsteemi üldine ülesehitus
Disain	Süsteemi detailne ülesehitus ja arhitektuur, mis täidaks kõik esitatud nõuded
Teostus	Lähtekood, käivitatav(ad) programm(id)
Testimine	Kinnitus, et ehitatud süsteem vastab nõuetele

Allikas: Autori kokkuvõte Rational Unified Process-i põhjal.

Tarkvara arendamise ja majaehituse suurimaks erinevuseks on käesoleva töö autori arvates muutuv keskkond. Maja projekteerimisest kuni esimeste sissekolimisteni muutuvad nõudmised majale suhteliselt vähe. Maja projekteerimisel eeldatakse, et maa ei hakka kohe vajuma ning kliima ei muutu. Eeldus kehtib isegi juhul, kui majaehituse alustamise ja lõpetamise vahel on mitu aastat. Tarkvara keskkond seevastu muutub praktiliselt iga aastaga. Nii pole ka kuigi otstarbekas tarkvara arendada ajaliselt pika projektina, mille lõpus saavad inimesed programmi kasutusele võtta. Pigem on kasulikum tee “vähem aga kiiremini“. See võimaldab keskkonna muutustega kaasas käia ning vältida ümbertegemisi projekti lõpus.



Joonis 2.3. Maja iteratiivne ja inkrementaalne arendamine (autori joonis).



Joonis 2.4. Faasid, iteratsioonid ja töövood projektis. [Rational Unified Process 2001]

Faasid, iteratsioonid ja töövood moodustavad kokku ühe tervikliku projekti aja kulgu kirjeldava protsessi. Iteratsioonid erinevates faasides hõlmavad endas küll kõik töövood, kuid erinevates hulkades. Näiteks algatusfaasis on programmeerimisel tunduvalt väiksem roll kui kasutajanõuete analüüsil. [Rational Unified Process 2001] Töövoogude võimalikku jaotumist projektis kirjeldab joonis 2.4. Joonisel on tarkvaraprojekt jaotatud faasideks. Iga faas jaguneb üheks või enamaks iteratsiooniks. Iteratsioonis läbitakse üldiselt kõik töövood. Töövoo intensiivsust konkreetses iteratsioonis näitab selle nivoo. Joonisel 2.4 välja toodud kavandamisfaasi iteratsioonis on põhirõhk disaini töövool ja tegeletakse suhteliselt rohkelt ka tarkvara realiseerimisega. Joonis 2.4 näitab, et erinevates faasides on töövoogudel erinev osatähtsus.

Iteratsioonidel on oluline osa ka kasutajatega suhtlemisel. Kõige suurem koormus langeb kasutajatele tavaliselt algatusfaasis ja siirdefaasis. Algatusfaasis on suurem rõhk nõuete töövool ning siis peaks läbi viima võimalikult palju kasutajaintervjuusid. Siirdefaasis antakse tarkvara juba käiku ja siis on oluline kuulata tarkvara kasutajate poolt saadavat tagasisidet. See tähendab, et algatusfaas on oluline kasutajate ootuste loomisel ja siirdefaas ootuste täitmisel.

Unifitseeritud protsess soovitab projektide jagamist väiksemateks alamprojektideks. ehk iteratsioonideks. Igas iteratsioonis läbitakse töövood ehk tegevused alamprojekti eesmärkide saavutamiseks. Iteratiivne ja inkrementaalne arendus võimaldab vanemate metoodikatega võrreldes saada varem kasutatavat tarkvara ning muudab tarkvaraprojektid paremini ennustatavaks.

2.3. Parimad praktikad unifitseeritud protsessis

2.3.1. Nõuete haldus

Nõuete halduse all mõistetakse unifitseeritud protsessi kirjelduses süstemaatilist lähenemist nõuete leidmiseks, dokumenteerimiseks, korrastamiseks ja jälgimiseks kogu tarkvara arendamise protsessi vältel. Nõude mõiste seletatakse lahti kui „seisund või tingimus, millele süsteem peab vastama”. [Leffingwell, Widrig 2003]

Nõuete kogumine on keeruline töö, mille käigus tavaliselt tekivad järgmised probleemid [RUP 2001]:

- Nõuded pole alati iseenesestmõistetavad ja pärinevad erinevatest allikatest.
- Nõudeid on mõnikord raske sõnades selgitada.
- On väga palju erinevaid nõuete tüüpe erinevatel detailsuse astmetel.
- Nõuete arvukus võib kontrolli alt väljuda, kui neid ei hallata süstemaatiliselt.
- Nõuded on omavahel sõltuvuses.
- Nõuetest on huvitatud eri huvigrupid, mis tähendab, et nõuete haldamiseks on vaja eri gruppide vahelist koostööd.
- Nõuded muutuvad.

Asjaolu, miks nõuete haldamisele unifitseeritud protsessis palju ruumi pühendatakse, peitub statistikas. Nõuetes peituv viga on kõige tavalisem ja kõige kulukam parandada. 25% - 40% tarkvaraprojektide eelarvest kulub vigadele, mis tehakse nõuete halduses. [Leffingwell, Widrig 2003]

Unifitseeritud protsessis soovitatakse kogu tarkvara arendamist teostada kasutuslugude põhiselt. Kasutuslugu kujutab endast mingit funktsionaalsete nõuete komplekti ning selle realiseerimine peaks võimaldama kasutajatel rohkem tarkvara abil ära teha. Vastandiks sellele lähenemisele oleks näiteks tehniline lähenemine (realiseeritakse näiteks mõni süsteemi aluskihtidest), kus lisanduv arendustöö ei pruugi kasutajale käegakatsutavat tulemust pakkuda. Kasutuslood organiseeritakse unifitseeritud protsessis kasutuslugude mudelisse, mis tervikuna peaks sisaldama kogu tarkvara nõudeid kasutuslugude ja nende kirjeldustena.

Kasutuslood on unifitseeritud protsessis käibel kogu projekti vältel. Kasutuslugude meetodit saab kasutada äriprotsesside modelleerimisel ning äriprotsesside baasil saab luua kasutusloomudeli, milles kajastuvad nõuded loodavale tarkvarale. Analüüsi- ja disaini töövoos realiseeritakse kasutuslood disainimudelis. Sidumaks kasutuslugusid ja disaini komponente, kasutatakse unifitseeritud protsessis kasutusloo realisatsiooni (*use case realization*). Realiseerimisel lähtutakse disainimudelist (mis eelneva jutu põhjal lähtus kasutusloomudelist). Testimise töövoos on kasutusloomudel vajalik testjuhtumite kirjeldamiseks (kasutuslood kirjeldavad vajaduste tasemel seda, kuidas süsteemi kasutatakse). Projektijuhtimise töövoos on kasutuslood aluseks projektiplaani koostamisel – unifitseeritud protsessi järgi tuleks funktsionaalsus kasutajateni viia samm-sammu haaval. Evitusfaasis on kasutuslood aluseks lõppkasutaja juhendile ja koolitusmaterjalidele. [RUP 2001]

Nõuete määratlemiseks ja haldamiseks on unifitseeritud protsessis defineeritud põhilised oskused, meetodid ja soovitav tööde järjekord (mis on ka tarkvaraarendusprotsessi ülesanne). Nõuete halduses nähakse võtmetähtsusega töödena järgmiseid (neid käsitleme veidi pikemalt ka järgmistes lõikudes): [RUP 2001]

- Probleemanalüüs;
- Osanike vajadustest arusaamine;
- Süsteemi määratlemine ja defineerimine;
- Projekti skoobi haldus;

- Süsteemi täpse määratluse esitamine;
- Muutuvate nõuete haldus.

Probleemanalüüsi eesmärgiks on leida probleemide algpõhjused. Kui on tuvastatud probleemide algpõhjused saab analüüsima hakata, kuidas nende põhjustega toime tulla – tarkvaraliselt või ilma. Probleemanalüüsi oskus hõlmab endas ka osapoolte tuvastamist ning piiride tõmbamist – kõikide probleemide ja algpõhjustega tegeleda ei saa. Probleemanalüüsi üks tulemeid on tasuvushinnang, mille põhjal saab vastu võtta otsuse arendusprojektiga edasimineku kohta. [Leffingwell, Widrig 2003]

Osanike (*stakeholder*) vajadustest arusaamine hõlmab kõikide süsteemi osanike määratlemist, suhtlemisstiili ja info saamise mooduste valikut. Isikud, kellele on süsteem kasulik või vajalik ongi osanikud. Osanikud ja kasutajad ei ole alati üks-üheselt samad. Näiteks võib osanikuks olla firmaomanik, kes soovib saada kiiret finantsaruandlust firma kohta, samal ajal, kui selleks ei pea ta süsteemi kasutama, kuna kasutaja rolli täidab tema raamatupidaja. Osanike vajaduste selgitamiseks on mitmeid meetodeid ja võtteid alates ajurünnakutest lõpetades prototüüpimisega.

Süsteemi määratlemine ja defineerimine kujutab endast osanike vajadustest lähtuvalt täpse nõuete spetsifikatsiooni kirjutamist. Süsteemi arendamise algjärgus tuleb vastu võtta otsused nõude püstitamise vormi, dokumenteerimise mooduse, nõuete kirjeldamise detailsuse, prioriteetide, töömahuhinnangute, riskide ja projekt algse skoobi osas. Süsteemi määratlemise tulemus on kirjeldus loodavast tarkvarast nii loomulikus keeles kui ka diagrammide abil.

Projekti efektiivseks haldamiseks on tarvilik nõuete prioriteete jälgida ning projekti skoopi hoida. Skoobi hoidmine on projekti eduks hädavajalik. Paljudel projektidel on probleemiks asjaolu, et arendajad tegelevad funktsionaalsusega, mis tundub arendajale huvitav, kuid ei oma projekti edukale lõpetamisele kuigi palju mõju. Samuti panevad projekti skoobi löögi alla tellija esindajad, kel „süües kasvab isu”. Projekti skoobi jälgimine tähendabki süstemaatilist lisanõuete haldamist ja otsustusprotsessi, mis määratleb, kas mingi nõue mahub projekti skoopi või mitte. Kui otsustatakse, et nõue mahub skoopi, siis peab selle tulemusena muutuma ka projekti ajaplaan ja eelarve.

Süsteemi täpse määratluse esitamise all mõeldakse meetodit korraldada süsteemi kirjeldamine sellisel moel, et see oleks ühelt poolt piisavalt täpne ja teiselt poolt piisavalt arusaadav, et ka mittetehnilise taustaga projekti osanikud oleksid nõus vastavale dokumendile alla kirjutama. Nõuete kirjeldus peaks katma peale funktsionaalsete nõuete ka nõudeid seadusandluslike piirangute, kasutatavuse, usaldusväärsuse, kiiruse, toe pakkumise ja hilisema hooldamise osas. Unifitseeritud protsessis soovitatakse kasutada kasutuslugusid koos lihtsa prototüüpimisega, et saavutada parimaid tulemusi.

Muutuvate nõuete haldus on tihedalt seotud projekti skoobi haldamisega. Kui projekti skoobi haldus tegeleb nõuete lisandumisega, siis nõuete muutumine tähendab olemasolevate nõuete seas olevaid uuendusi ja täiendusi. Nõuete muutumine on keeruline, kuna nõuded on omavahel seotud. Nõuete muutmise protsess peab olema samamoodi reglementeeritud nagu skoobi muutmine. Autori arvates on suures osas tegu sama asjaga, mida vaadeldakse eri külgedelt.

Kokkuvõtteks võib öelda, et unifitseeritud protsess on kasutuslugude põhine ning pöörab väga palju tähelepanu nõuete määratlemisele ja haldusele. Nõuete kirjeldamiseks soovitatakse kasutada UML-keelt ja kasutusloomudelit. Nõuete haldamiseks ja kogumiseks on unifitseeritud protsessis kirjeldatud mitmeid meetodeid ja vajalikke oskuseid. Metoodikaga on kaasa pandud siiski vaid piiratud hulgal konkreetseid ning kohe kasutatavaid materjale.

2.3.2. Komponentarhitektuur

Komponentarhitektuuri osas mõeldakse unifitseeritud protsessis kogu tarkvara disainimist sellisel moel, et tarkvara oleks võimalik arendada ja laiendada tükk-haaval. Samuti tähendab komponentarhitektuur seda, et mõnda komponenti ei peagi arendama – selle võib sisse osta või mõne teise tarkvaraprojekti komponenti ära kasutada. Tarkvara, mis on komponendipõhiselt arendatud, on lihtsamini mõistetav ja muudatustele vastupidavam.

Unifitseeritud protsess tervikuna käsitleb kasutuslugusid, kui põhilisi alustalasid tarkvara arendamisel. Disaini ja kodeerimise tegevuse juures omab suuremat tähtsust aga süsteemi arhitektuuri mõiste. Tarkvara arenduse varajases faasis on oluline luua

vastupidav arhitektuur, mis projekti hilisemates faasides muutub valmis süsteemiks. [RUP 2001]

Unifitseeritud protsess näeb projekti varajases faasis ette arhitektuuri prototüübi valmistamist. See on väike funktsionaalne osa, mis aitab veenduda, et loodav tarkvara hakkab tulevikus täitma ka seatud mittefunktsionaalseid nõudeid (näiteks kiirus, laiendatavus jne.). Selline lähenemine haakub ka unifitseeritud protsessis laialt käsitletava riskide varajase maandamise strateegiaga.

Põhjused, miks unifitseeritud protsess arhitektuurile suurt tähelepanu pöörab, on võimalik jagada kolmeks üldiseks teemaks. Esiteks võimaldab arhitektuur ka suuri projekte tehniliselt hallata ning inimese jaoks haaratavana hoida. Teiseks on arhitektuuri kasutamine aluseks koodi ja komponentide taaskasutusele. Kolmandaks loetakse arhitektuuri aluseks projektijuhtimisele. Allpool käsitletakse ülesloetud teemasid veidi pikemalt.

Suurte ja tehniliselt keerukate projektide jaoks on arhitektuur ülioluline, kuna see on sisuliselt ainus osa, mis on kõikide süsteemi osade jaoks ühine. Ühisosa võib luua kihtideks jaotumine, kasutajaliides ja selle käitumisloogika ning paljud teised parameetrid. Sarnane struktuur erinevatel komponentidel võimaldab keerulist süsteemi hoida lihtsamana ja arusaadavamana. Lisaks tekitab selline lähenemine arendajate jaoks äratundmisrõõmu, kui asutakse uut asja tegema – alustada ei tule nullist.

Koodi ja komponentide taaskasutus on hetkel tarkvaraarenduse kiirendamise ja efektiivsemaks muutmise võtmevaldkondi. Taaskasutamisel tehakse vahet sisemisel ja välisel taaskasutusel. Sisemise taaskasutuse all mõeldakse asjaolu, et sama süsteemi piires ei programmeerita ühte asja kaks korda. Välise taaskasutuse näol peetakse silmas sama komponendi kasutamist erinevates süsteemides. Viimasel juhul moodustab taaskasutatav komponent juba omaette paketi, standardse osa, mida on võimalik järgmistes projektides lihtsalt välja kutsuda. Taaskasutuse ideoloogia on läinud ka kaugemale – kirjanduses on rohkelt käsitletud arhitektuuri kui objekti taaskasutamist eri süsteemides. Kokkuvõttes on arhitektuur taaskasutuse eelduseks – kui süsteemil pole selget arhitektuuri, on raskendatud ka efektiivne taaskasutus.

Unifitseeritud protsess loeb arhitektuuri ka projektijuhtimise aluseks. Nimelt aitavad arhitektuuris defineeritud komponendid ja funktsionaalsed osad jagada tööd inimeste ning meeskondade vahel. Samuti saab eri komponendid jagada eri meeskondade vahele – teadmised kapseldatakse meeskonna piiresse, tõstes sellega arendamise efektiivsust.

Kokkuvõtteks saab öelda, et disaini ja analüüsidistsipliinide jaoks on unifitseeritud protsessis arhitektuuril väga oluline koht. Arhitektuur on aluseks eelkõige efektiivsele arendustööle, mis kasutab maksimaalselt ära inimeste ja meeskondade kompetentsi.

2.3.3. Visuaalne modelleerimine ja UML

Visuaalne modelleerimine on moodus tarkvara kirjeldamiseks diagrammide abil. Diagrammide kogumist teeb mudeli diagrammide korrapärane organiseerimine, kirjeldamine ja standardse notatsiooni kasutamine. Unifitseeritud protsess soovitab notatsiooni valikul kasutada unifitseeritud modelleerimiskeelt (*unified modelling language, UML*). Visuaalse modelleerimise primaarne eesmärk on suhtlemise lihtsustamine, kiirus ja arusaadavus.

Modelleerimine ja diagrammid aitavad inimestel mõista tarkvara abstraktsemal ning üldistatud tasemel. Olulise näitamine ja mitteolulise peitmine võimaldab väga keeruliste kontseptsioonide mõistmist. Mudelid aitavad: [RUP 2001]

- mõista keerulisi süsteeme;
- väikese investeeringuga disainivõimalusi uurida ja võrrelda;
- realisatsiooni kirjelduse alustalasid määratleda;
- nõudeid täpselt määratleda;
- otsuseid selgelt edasi anda.

Tänapäeval ei kujutata tarkvarasüsteemide disainimist ilma modelleerimiseta hästi ettegi. Tarkvara on keeruline ja selle mõistmiseks tuleb lihtsustada ning üldistada. Diagrammide ja mudelite kasutamine on seejuures mugav vahend, et ennast arusaadavaks teha. Tarkvara arenduses on nüüdseks jõutud samade järeldusteni, kus ehitus ja tööstus on olnud juba aastakümneid. Tarkvarasüsteemides kasutatakse

kasutusloodiagramme selgitamaks süsteemi käitumist, klassi ja andmemudeleid süsteemi ülesehituse kirjeldamiseks ning olekudiagramme süsteemi tehnilise käitumise kirjeldamiseks.

Tarkvara arendajad ei saa endale lubada luksust „proovimiseks" arendada valmis kaks-kolm eri arhitektuuri ning siis võrrelda, milline neist konkreetse projekti jaoks parim on. Visuaalne modelleerimine koos iteratiivse arendusega võimaldab varakult teha selgeid valikuid ning vastu võtta otsuseid. Mudel on kordi odavam tegelikust asjast – seega on siin tegu puhta majandusliku kokkuhoiuga.

Arenduspõhimõtete paikapanemisel on visuaalne modelleerimine asendamatu, eriti objekt-orienteeritud keskkonna puhul. Tavapäraselt on modelleerimine suunatud lõpptulemuse ehk tarkvara koodi genereerimisele (see võib toimuda nii käsitsi kui ka automaatselt). On juhtumeid, kus on vaja mõista juba olemasolevat tarkvara ning teostada tagasikonverteerimist (*reverse-engineering*), kus olemasolevast süsteemist luuakse mudel. Unifitseeritud protsess näeb seda ühtse tervikuna – mudel ja kood peaksid olema alati vastavuses, ükskõik kummalt poolt muudatus algatatakse. [RUP 2001]

Nõuete määratlemisel on modelleerimine käigus eelkõige kasutusloodiagrammide näol. Unifitseeritud protsessis väidetakse, et kasutusloomudel on nõuete haldamiseks ja kokkulepete saamiseks kõige parem moodus. Kasutusloomudel eraldab süsteemi käitumise nõuded selle realisatsioonist, luues abstraktsiooni.

Kokkuvõtteks on visuaalne modelleerimine unifitseeritud protsessi üks alustalasid. Modelleerimine lihtsustab suhtlemist ja arusaamist, mis on tarkvaraprojekt õnnestumise jaoks elementaarne. Unifitseeritud protsess soovitab kasutada kokkulepitud konkreetset notatsiooni UML-i näol. Seega on teoreetiliselt võimalik kokku hoida aega ja raha arusaamatuste ja suhtlemise arvelt.

2.3.4. Pidev kvaliteedikontroll

Pideva kvaliteedikontrolli all mõeldakse unifitseeritud protsessis asjaolu, et iga dokumenti, koodiplokki, diagrammi kontrollitakse pärast muudatusi. Kontroll ei ole pika tegevuse viimane lüli, vaid läbiv osa protsessist. Pidev kvaliteedikontroll on ka iga

iteratsiooni lõpus välja lastav tarkvara versioon, kuna sellega kontrollitakse tarkvara elujõulisust ja töökindlust. Taolise kontrolli eesmärk on vähendada riske, et projekti lõpuks valmiv tarkvara ei vasta kliendi soovidele. [RUP 2001] Maailma enam levinud kvaliteedijuhtimisstandard ISO 9001:2000 väidab, et „Kvaliteet on vastamine kliendi ootustele”. [ISO 9001:2000]

Unifitseeritud protsessis jagatakse kvaliteet kaheks: a) tootekvaliteet – kõigi projekti käigus valminud tulemite vastavus ootustele; b) protsessi kvaliteet – kuidas järgitakse nende tulemite valmimise protsessi. Kvaliteet on kõigi vastutusel, kuid lõppvastutajaks kvaliteedi osas on unifitseeritud protsessi kohaselt projektijuht.

Kvaliteedikontrolli elluviimiseks on vaja järgmist: [RUP 2001]

- määratleda kvaliteedi meetrika või indikaatorid;
- määratleda kvaliteedi mõõdikud, mis vastavad meetrikale;
- tegutseda kvaliteeti mõjutavate juhtumite korral võimalikult varakult ja efektiivselt.

Püüame antud lähenemist tarkvarale üle kanda. Kõigepealt on vaja otsustada, mis määratleb tarkvara kvaliteedi (kiirus, mugavus, kliendirahulolu vms). Teiseks tuleb meetrikale leida vastavad mõõdikud iga indikaatori jaoks. Kolmandaks on vaja iga kõrvalekalde puhul midagi ette võtta, et kvaliteet oleks pidevalt tagatud.

Unifitseeritud protsess annab ka indikatsiooni, kuidas kvaliteeti erinevates tarkvara arendamisega seotud distsipliinides jälgida. Sisulise poole pealt taanduvad kõik soovitused ühele lihtsale valemile – pärast mingi tulemi valmimist on see vaja teise inimese poolt üle vaadata ning protsessi korrata iga muudatuse järel ja iteratsioonide lõpus.

2.3.5. Muudatuste haldus

Muudatuste haldus on teema, mis haakub eelkõige inimeste koostööga. Kui on rohkelt inimesi – on vaja protsessi, mis võimaldaks inimestel koos efektiivselt töötada. Unifitseeritud protsessis taandub muudatuste haldus neljale põhipunktile:

- töökeskkonna haldus;
- integratsioon;
- paralleelne arendus;
- väljalaskeversioonide haldus.

Töökeskkonna halduse all mõeldakse eelkõige sellise töökeskkonna tagamist arendusmeeskonnale, mis võimaldab ja soodustab koostööd. Töökeskkond tähendab siin füüsilist kontoripinda, arvutivõrke, servereid, tööjaamu ja kõike, millega inimesed töötama peavad. Kogu infrastruktuur peab võimaldama inimestel efektiivselt töötada ja suhelda. Praktikas on siinkohal loomulikult probleeme ja lahendusi – erinevates asukohtades olevad meeskonnad, kellele tuleb võimaldada mõni elektroonilise suhtluse vorm jne.

Kui tarkvarasüsteemis on komponente, mida arendatakse erinevate tiimide poolt, kuid mis peavad omavahel koos töötama, võib komponentide vahel tekkida integratsiooniprobleem. Nimelt, kui pannakse kokku süsteemi tervikuna, siis üks (sõltuv) komponent ei pruugi töötada koos teise komponendiga (näiteks mõni baaskomponent). Siinjuures on vaja integratsioonihaldust ja sõltuvusahelate jälgimist, et süsteemi ühes kohas tehtud muudatus ei teeks kogu süsteemi mittekasutatavaks. Selle jaoks on erinevaid strateegiaid – nendest üks kuulsamaid on nn *nightly build*, mida kasutab ka Microsoft Corporation. [Kudrjavets 2004]

Paralleelne arendus muudatuste halduses tähendab tavalist versioonihaldust koos organisatoorse protsessiga. Tegu on lihtsa süsteemiga, mis peaks tagama, et inimesed üksteise tööd üle ei saaks kirjutada ja eksikombel kustutada. Samuti on paralleelse arenduse võimaldamiseks vaja vahendeid ühe ja sama dokumendi või lähtekoodi kallal samaaegseks töötamiseks ja hiljem tehtud töö ühildamiseks ühte faili. Antud teemaga tegeleb versioonihaldus ja versioonihaldustarkvara.

Väljalaskeversioonide halduse all peab unifitseeritud protsess silmas vajadust planeerida ja teostada regulaarseid väljalaskeid ning veenduda nende töökindluses. Kõige problemaatilisem koht praktikas on siinjuures eri versioonides väljalastud

tarkvara järelhoole ja modifitseerimine. Vajalik on, et firmas oleks võimalused jälgida versioonipuus erinevaid harusid ja versioone. Tavapraktikas tähendab see, et kasutajate käes olevat versiooni (näiteks versioon 1.1) parandatakse aeg-ajalt ning samal ajal arendatakse uut versiooni (versioon 2.0). Tarkvaraorganisatsioon peab tagama, et on olemas oskused ja vahendid selliste olukordade juhtimiseks.

Kokkuvõtteks saab öelda, et muudatuste haldus on tarkvara arendamise jaoks hädavajalik ning võrreldav elementaarse infrastruktuuriga. Muudatuste all peetakse siinkohal silmas tarkvaraorganisatsiooni sees tulemitele teostatavaid muudatusi. Eduka projekti elluviimiseks on vaja versioonihaldussüsteemi, plaani väljalaskeversioonide tegemiseks ja hooldamiseks, pidevat integreerimist ja infovahetust ning võimalusi paralleelselt tööd teha.

3. NAVISION ONTARGET

Navision OnTarget (edaspidi OnTarget) näol on tegemist tootepõhise tarkvara juurutusprotsessiga. Protsessi algne väljatooja on Navision AS, kes praeguseks hetkeks on omandatud Microsofti poolt. OnTarget on mõeldud eelkõige Microsoft Navision ja Microsoft Axapta ERP-süsteemide kiireks juurutamiseks ilma lisafunktsionaalsuseta.

OnTarget-il on mitmeid erinevusi võrreldes tarkvara arendusmetoodikatega (sh. ka RUP). Erinevused on järgmised:

- Metoodika jaguneb kahte ossa: 1) ärimetoodika, mis kirjeldab tarkvaraorganisatsiooni tegevust; 2) projektimetoodika, mis kirjeldab tegevusi konkreetse juurutusprojekti korral.

- Metoodika on konkreetsest tootest tulenevalt väga rakendusliku iseloomuga ning sisaldab dokumente, kus on vaja täita ainult lüngad.

- Soovitatakse koskmudeli laadset lähenemist vastandudes iteratiivsetele metoodikatele.

OnTarget metoodika koosneb järgmistest komponentidest:

- Faasid – kirjeldavad projekti jagunemist etappideks lähtuvalt ajalistest ja tegevuselistest kriteeriumitest. Iga faas kirjeldatakse töövoodiagrammiga, kus on toodud sisendid, väljundid ning tehtavad tegevused.

- Tegevused – igal tegevusel võivad olla sisend- ja väljundtulemid. Tegevusel võib hõlmatud olla üks või rohkem rolli. Igal tegevusel peab olema vastutav roll.

- Tulemid (tehised) – dokumendid, presentatsioonid, kooditükid või programmimoodulid. Tulemid saavad olla sisendiks ja väljundiks tegevustele. Praktiliselt kõigile OnTarget-is defineeritud tulemitele on loodud ka eeltäidetud mallid, mis teeb metoodika praktilise kasutusele võtmise mugavamaks.

- Rollid (aktorid) – kirjeldavad, millised rollid on tarvilikud mingite tulemite saamiseks.

Sarnaselt RUP-ile soovitatakse OnTarget-is kasutada UML-i notatsiooni. Lähtuvalt Navisioni iseärasustest ei suruta aga peale objekt-orienteeritud paradigmat. Autorile on jäänud mulje, et UML-i on kasutatud eelkõige selle laia leviku pärast. OnTarget-iga käib kaasas ka eraldi modelleerimisvahend OnTarget Modeler. Siin on jällegi tegu Rationalile sarnase kombinatsiooniga – protsessikirjeldusega koos tuleb ka toetav tarkvara.

3.1. Faasid ja projektorganisatsioon

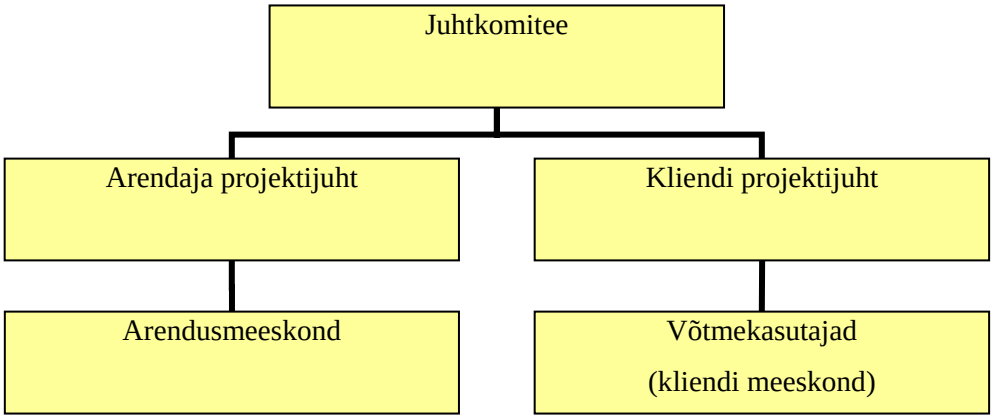
OnTarget metoodika jagab projekti elutsükli faasideks (*phase*). Faasid aitavad projekti staatust paremini defineerida. Igas faasis on projektiga seotud erinevad rollid ja tegevused. OnTarget-i faasid ja RUP-i faasid on sarnased mõisted, kuid ühe erinevusega. Nimelt tuleb silmas pidada, et RUP on iteratiivne metoodika ja OnTarget põhineb koskmudelil. OnTarget-is on defineeritud on kuus faasi:

- diagnoos,
- analüüs,
- disain,
- arendus ja testimine,
- juurutamine,
- jätkuv kasutajatugi.

OnTarget-i metoodika ülesehituse juures tuleb märkida, et see sisaldab ka müügiprotsessi ja kogu ettevõtte ärimetoodikat. Antud lähenemine on mõistetav, kuna OnTarget-i puhul tegu on tootepõhise lähenemisega. Praktikas on tavapärane situatsioon, kus tarkvarafirma esimeste kohtumiste eesmärk on selgitada, mida klient vajab ja alles seejärel asutakse pakkumist tegema ning projekti skoopi määrama. Autori

arvates tuleks ka teisi enamlevinud metoodikaid täiendada just sellest vaatepunktist lähtuvalt.

OnTarget-is pööratakse eraldi tähelepanu ka projektorganisatsiooni ülesehitusele. Nimelt nähakse metoodika kohaselt ette nn sümmeetrilist organisatsiooni, kus meeskonda kuuluvad inimesed nii arendaja kui ka kliendi poolelt. Sellega vähendatakse tarkvaraprojektide üht suurimat riski – vähest kaasatust. Projektorganisatsiooni näitlik ülesehitus on esitatud joonisel 3.1.



Joonis 3.1. Projektorganisatsiooni ülesehitus OnTarget metoodika järgi (autori joonis OnTarget-i põhjal).

Juhtkomiteesse kuuluvad projekti üle finants- ja otsustuskontrolli omavad inimesed. Juhtkomitee otsusel võib muuta projekti eelarvet, tähtaegu, ressursse ja skoopi. Juhtkomiteel peavad olema ka volitused vajadusel projekt tühistada. Arendaja projektijuht on arendajapoolne projektijuht, kes vastutab kogu planeerimis- ja arendustegevuse eest. Kliendipoolne projektijuht on tarkvara juurutamise ja nõuete kogumise eest vastutav isik klientorganisatsioonis. Kliendipoolse projektijuhi korraldada on infovahetus arendusmeeskonna ja kasutajate vahel. Samuti tegeleb kliendipoolne projektijuht vajadusel lisaressursi hankimisega klientorganisatsioonis (juhul kui vaja läheb lisatööjõudu andmete sisestamiseks jne.). Järgnevalt vaadeldakse OnTarget-is ülesloetud projekti faase lähemalt.

3.2. Diagnoos

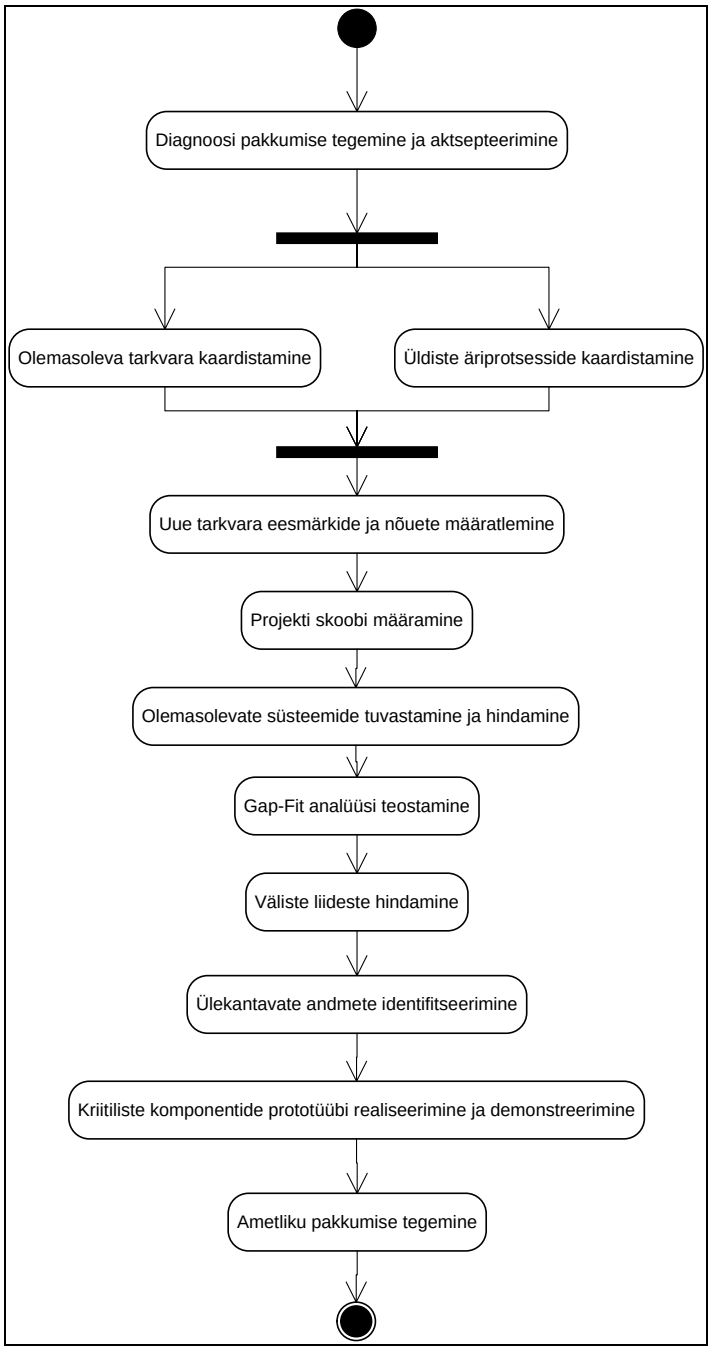
Diagnoosifaas on tarkvara juurutaja poolt pakutav teenus perioodil, mil tellija ei ole veel otsustanud tarkvara juurutama asuda. Oma olemuselt on see võrreldav RUP-is kirjeldatud algatusfaasiga. Diagnoosifaas moodustab osa müügitsüklist ning sisaldab ka praktilisi nõuandeid, kuidas klienti „ära rääkida".

OnTarget'i kohaselt on diagnoosifaasi eesmärgid:

1. Tekitada konsulteerimisega endale konkurentsieelis.
2. Vältida projekti hilisemat paisumist, aidates kliendil defineerida visiooni, eesmärgid ja tarkvara funktsionaalsuse.
3. Mõista ja dokumenteerida potentsiaalsed riskid.
4. Määratleda kogu projekti keerukus.
5. Koguda infot, mis aitaks ette valmistada võitvat pakkumist.

Ligi pooled diagnoosifaasi eesmärkidest on keskendunud müügitegevuse edukale läbiviimisele ning ülejäänud tellija vajaduste selgitamisele. Diagnoosifaas on metoodika kohaselt eraldi tasustatav „eelprojekt", mis kestab umbes ühe töönädala. Tuleb tähele panna, et nii Navision kui ka OnTarget metoodika lähtuvad teatud „keskmistest", mis väikeste hälvetega peaksid sobima enamusele juurutustest.

Diagnoosifaasi, nagu iga teise faasi, jaoks on koostatud väga detailne töövoodiagramm. Joonisel 3.2 on diagnoosifaasi töövoodiagrammist esitatud autori poolt üldistatud variant, mis sisaldab tegevuste kokkuvõtteid.



Joonis 3.2. Diagnoosivaate üldistatud töövoodiagramm (autori joonis OnTarget põhjal).

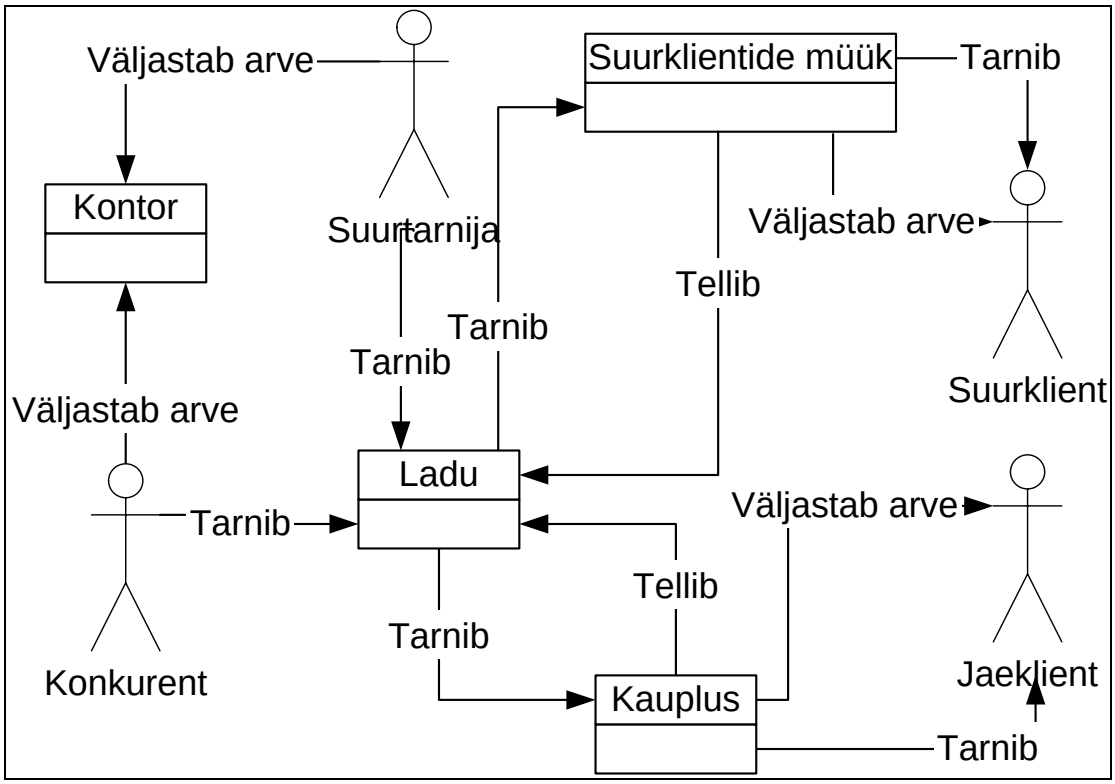
Diagnoosifaas algab pakkumise tegemisega. Pakkumise skoop hõlmab diagnoosifaasis vaid sisuliselt nõuete analüüsi tegemist juurutatava tarkvara kohta. Metoodikas on määratletud ka üsna detailne kirjeldus, milliste argumentidega klienti veenda, et diagnoosifaasiga üldse algust tehtaks. Autori hinnangul on üsna laialt levinud praktika, et klient laseb nõuete analüüsi kulud tarkvarafirmal müügikuludena katta. OnTarget rõhutab aga, et oluline on ka diagnoosifaasi eest tasu võtta, kuna see motiveerib klienti temaatikaga põhjalikumalt tegeleda.

Nõuete analüüsi võib oma olemuselt laias laastus jagada neljaks osaks: a) olemasoleva situatsiooni kaardistamine; b) kõikide soovide kaardistamine; c) soovide ja uue tarkvara funktsionaalsuse võrdlemine; d) projekti skoobi ja teostatava funktsionaalsuse määramine. Igas osas saab kasutada erinevaid meetodeid, töövõtteid ja ettevalmistatud dokumendimalle töö läbiviimiseks. Järgnevalt antakse ülevaade mõnedest olulisematest töövõtetest ja mallidest.

Olemasoleva situatsiooni kaardistamisel vaadeldakse riistvara, tarkvara ja äriprotsesse (mõnel pool kirjanduses kasutatakse ka mõistet talitlusprotsess). Igaühe kaardistamiseks on OnTarget-is välja pakutud juhis või töömall (*template*) kas Microsoft Excel või siis OnTarget Modeler baasil.

Diagnoosifaasis nähakse Navision OnTarget-is ette üsna rohke info sisestamist tarkvaraarendaja poolt hallatavasse kliendiprofiili (*Client Profile Workbook)*. Kliendiprofiil sisaldab infot alates müügisoovidest kuni tarkvara nõuete ja väliste liidesteni. Kliendiprofiili idee seisneb selles, et projektiga liituv töötaja saab väga kiiresti ülevaate kliendist, võtmeisikutest ja teostatavast funktsionaalsusest.

Füüsiline diagramm kajastab organisatsiooni struktuuri ja väga üldisel tasemel füüsilisi äriprotsesse (vt. joonis 3.3). Kasutatakse UML notatsioonist tuttavaid aspekte, kuid veidi teises tähenduses. UML-is objekti tähistav kujund on füüsilises diagrammis organisatsiooni allüksuse tähenduses. UML klassdiagrammi üldistusnool tähendab füüsilises diagrammis andme-, info- või kaubavoogu. UML kasutusloodiagrammi aktor tähendab välist suhtlusobjekti nagu näiteks tarnija, klient või partner.



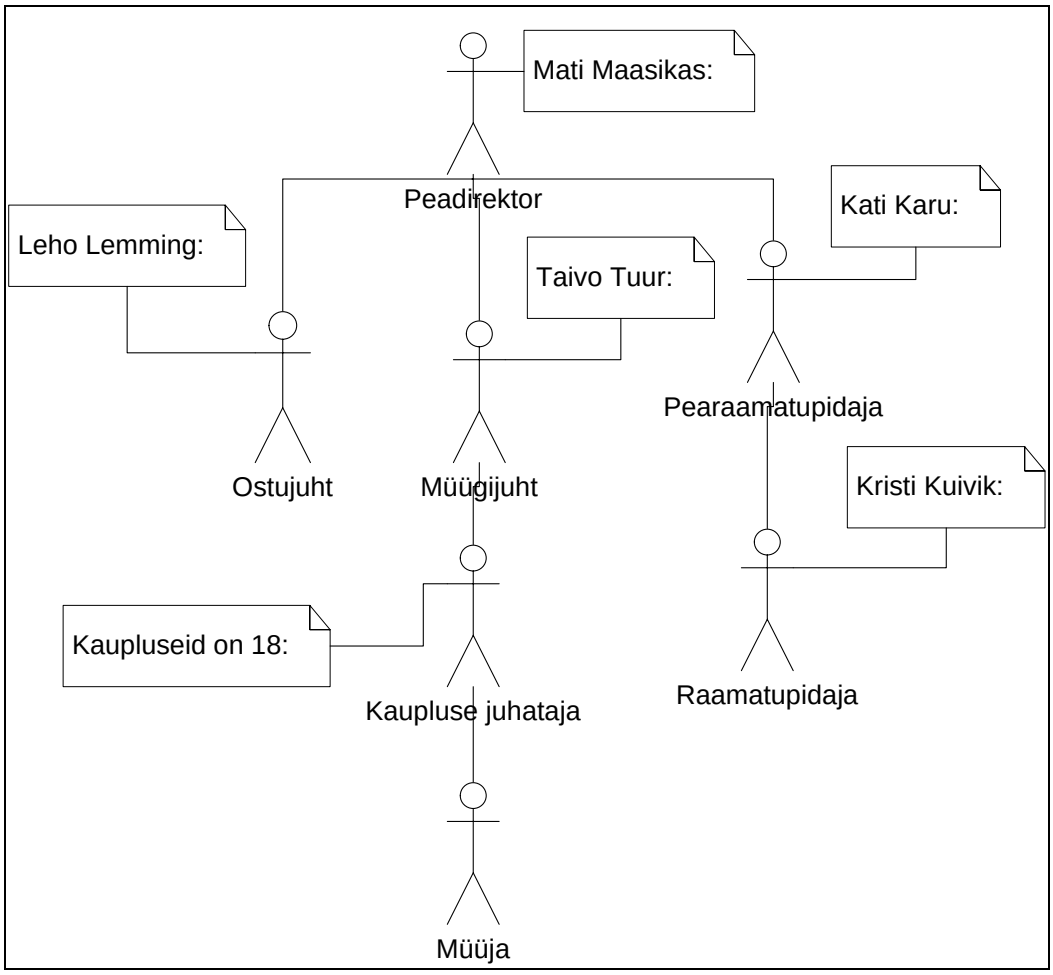
Joonis 3.3. Füüsilise diagrammi näide (autori joonis).

Riistvara diagramm ja tarkvara diagramm peavad andma ülevaate kliendi organisatsioonis kasutusel olevast riistvarast ja tarkvarast. Riistvara ja tarkvara diagrammid on illustratiivse olemusega ning aitavad aru saada kliendi organisatsiooni kaetusest riist- ja tarkvaraga. Notatsioon on sekundaarne ning arusaadavus tavainimesele on seatud esikohale. Käesolevas töös nende diagrammide näiteid ära ei tooda.

Organisatsioonidiagramm kajastab organisatsiooni staatilist struktuuri – põhielementideks on rollid. Kommentaaridena on juurde pandud neid rolle täitvad inimesed. Organisatsioonidiagrammi on vaja kahel põhjusel: a) müügiprotsessis on väga oluline teada milliste inimestega suheldakse ja kuidas on lahendatud alluvussuhted; b) see aitab suuremate ja keerulisemate asutuste puhul mõista selle ülesehitust, kuna üldjuhul alluvussuhted kajastavad ka osakondadeks jagunemist.

Organisatsiooni diagrammil kasutatakse UML notatsiooni ja vajadusel kirjutatakse rollide vahelistele seostele ka seose iseloom. Kommentaarimärgendit kasutatakse eelkõige konkreetsete nimede või lisainfo andmiseks. Kuna UML-i üks eesmärke oli

tavainimestele arusaadavus, siis antud joonise puhul tuleb see hästi välja, kui diagramme joonistada klassikalisel „ülevalt alla” meetodil. Keerulisemate struktuuride puhul võib vajalikuks osutuda rohkemate kommentaaride ja alluvussuhete selgitamine ning siis ei pruugi autori arvates organisatsioonidiagramm olla parim vahend antud info edastamiseks.



Joonis 3.4. Organisatsioonidiagrammi näide (autori joonis).

Tavaliselt on ERP-tarkvara soetada soovivad asutused endal juba mõne(d) tarkvarasüsteemi(d) kasutusele võtnud. Diagnostikafaasis on tarvilik kaardistada olemasolevad tarkvarasüsteemid. Kaardistamise eesmärgiks on teada saada, millised andmete ülekandmisnõuded on vanalt tarkvaralt uuele üle minnes. Samuti on vana tarkvara kaardistamine kasulik võte funktsionaalsete ja mittefunktsionaalsete nõuete kiiremaks kogumiseks.

Olemasolevate tarkvarasüsteemide kaardistamiseks nähakse OnTarget-is ette tarkvaramaatriksit. Selle eesmärkideks on:

- Luua nimekiri kõigist organisatsioonis kasutusel olevatest tarkvarapakettidest.
- Kaardistada tarkvarapakettide kasutus eri osakondades (sh. ka kasutuse eesmärgid).
- Kaardistada eri tarkvarapakettide omavahelised liidesed ja vahetatavad andmed.

Osakond/Pakett	Hansa Financials	Ladu 2.0
Raamatupidamine	*Kasutab*	*Kasutab*
Müügiosakond		*Kasutab*
Ladu		*Kasutab*
Liides (kust/kuhu)	Hansa Financials	Ladu 2.0
Hansa Financials		*Laekumised (automaatselt, iga päev*
Ladu	*Käsitsi (arved, laokuluarvestus) (iga päev, iga kuu)*	

Joonis 3.5. Näide lihtsast tarkvaramaatriksist kahte tarkvarapaketti kasutavas firmas (autori joonis).

OnTarget näeb ette alternatiivset võimalust tarkvaramaatriksi asemel kasutada tarkvaradiagrammi. Diagrammile on esitatud tabelile sarnased nõuded ning kasutatav notatsioon pärineb UML-i komponentdiagrammist. Diagrammi kasutamine on eelistatud, kui süsteeme ja nendevahelisi seoseid on vähe. Kui kasutatavate süsteemide arv kasvab, läheb ka olemasoleva tarkvara diagrammil loetavalt kajastamine raskeks.

Diagnoosifaasi oluline osa on ka kõrgtaseme nõuete kogumine, mida soovitatakse OnTarget-is teha kasutusloomudeli abil. Mõisted, kasutatavad meetodid ja notatsioon on väga suures osas sarnane RUP-is kasutatavale ja antud töös seda pikemalt ei käsitleta. Ainukese erinevusena on OnTarget-is välja toodud ka konkreetseid Microsoft Navisioni

põhiseid näiteid, mis kajastavad standardseid ärifunktsioone, mis on esindatud pea kõigis ettevõtetes.

Nõuete kirjeldamisel ja määratlemisel on OnTarget-is üles loetud hulk dokumente, mis oma olemuselt sarnanevad väga palju RUP-i nõuete halduses kirjeldatud dokumentidega. Põhilised dokumendid, mis kaasnevad nõuete haldusega OnTarget-i diagnoosifaasis, on kasutusloomudel, visioon, skoobi kokkulepe ja võtmetähtsusega nõuete nimekiri.

OnTarget-i üks eripära peitub väga tugevas projekti skoobi jälgimise rõhutamises. Kuna OnTarget on loodud eelkõige tarkvara juurutajate (mitte tellijate) vajadustest lähtuvalt, siis on metoodikas ka väga selgelt toodud juhised, kuidas tõmmata joon juurutatava ja mittejuurutatava funktsionaalsuse vahele.

Projekti skoobi määratlemine nõuab sisendiks enamust eelpool mainitud dokumente (eelkõige kasutusloomudelit) ja väljundina lihtsalt jagab dokumentides kajastatud info kahte kategooriasse – skoobis ja skoobist väljas. Skoopi mahtuv funktsionaalsus peaks saama vaadeldud projekti raames realiseeritud või juurutatud. Funktsionaalsus kirjeldatakse visioonidokumendi osana (märkusena võib siinkohal öelda, et RUP-is toimub see samamoodi). Autori arvates on diagnoosifaasi kõige olulisem tulemus just skoobi määratlemine, sest see annab töö teostajale mõistliku aluse teha finantspakkumine töö teostamiseks.

OnTarget on diagnoosifaasi ühe osana määratlenud ka meetodi, mis on rakendatav vaid tarkvara valmispakettide puhul, nn *gap-fit* analüüsi. *Gap-fit* analüüsi eesmärk on võrrelda olemasolevat tarkvara ja määratletud nõudeid tarkvarale ning iga nõude kohta vastu võtta otsus. Otsused jagunevad üldiselt kolme kategooriasse: a) funktsionaalsus on standardsena olemas; b) funktsionaalsuse realiseerimiseks on standardtarkvara vaja kohendada (siinkohal mõeldakse tavaliselt mõne andmebaasi välja lisamist vms.); c) funktsionaalsus on vaja täielikult juurde arendada.

Gap-fit analüüsi jaoks koostatakse tabel, mis sisaldab kõiki tarkvarale esitatavaid nõudeid. Iga nõude kohta märgitakse kaks aspekti: a) millisesse kategooriasse ta *gap-fit*

analüüsi mõistes kuulub; b) kas nõue realiseeritakse käesoleva projekti skoobis või mitte.

Gap-fit analüüs on oluline töövahend projektiplaani ja mahu hindamisel, kuna siis saab selgeks, milline osa tarkvarast on juba olemas ja milline tuleb juurde arendada. Kui on välja selgitatud tööde hinnatav maht, saab vastu võtta otsuseid ka skoobi ja eelarve kohta.

Võtmetähtsusega funktsionaalsuse ja/või juurdearenduste kohta soovitab OnTarget teha prototüübi. Standardfunktsionaalsuse osas tähendab prototüüp lihtsalt süsteemi seadistamist nii, et saaks kontrollida, kas funktsionaalsus töötab soovitud kujul. Juurdearenduste puhul on tegemist tavalise prototüüpimisega, mis hõlmab endas lõppkasutajate tagasiside saamist ekraanivormide kohta.

Kui diagnoosifaasis kirjeldatud tööd on tehtud, võib arendaja OnTarget metoodika kohaselt teha pakkumise tarkvara juurutamiseks. OnTarget-is on kaasa pandud ka eraldi dokumendimall pakkumise jaoks. Kui pakkumine kliendi poolt aktsepteeritakse, võib projektiga edasi liikuda järgmistesse faasidesse.

Diagnoosifaas võib mõnede projektide puhul tähendada ka projekti lõppu, kui võetakse vastu otsus, et tarkvara juurutamine ei ole otstarbekas. Antud põhjusel soovitab OnTarget metoodika diagnoosifaasi käsitleda eraldi projektina, mis tasustatakse eraldi.

3.3. Analüüs

Analüüsfaasi eesmärgiks on diagnoosifaasi dokumente aluseks võttes viia asutuse äriprotsesside kaardistus lõpuni, kaardistada uued või muutuvad äriprotsessid ja luua funktsionaalsete nõuete dokument. Oma olemuselt ei ole analüüsifaas midagi muud, kui diagnoosifaasis loodud dokumentide täiendamine. Eesmärgiks on seekord aga nõuete sajaprotsendiline kirjeldamine ning detailse projektiplaani väljatöötamine.

Projektiplaani koostamine OnTarget metoodika järgi on kasutajale lihtsaks tehtud tingimusel, et tegu on standardse juurutustööga, kus pole vaja erilisi lisaarendusi teostada. Nimelt on metoodikas olemas valmis projektiplaan, kus tuleb vaid kuupäevad

täita ning lisatööd sisse panna. Projektiplaani peab OnTarget-i kohaselt kinnitama ka juhtkomitee.

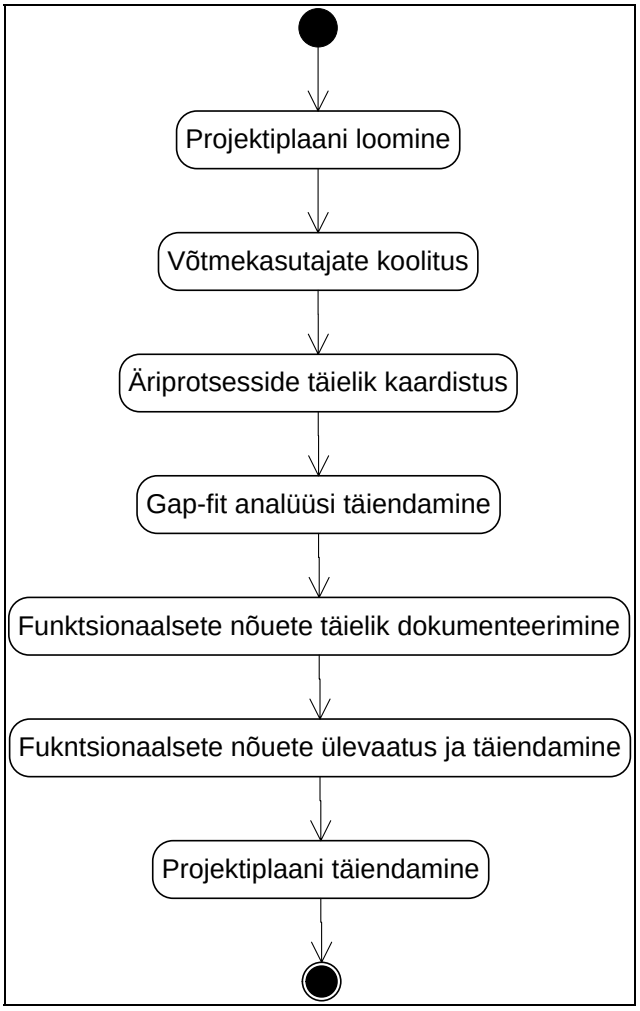
Järgmine samm analüüsifaasis on metoodika kohaselt võtmekasutajate koolitus. Võtmekasutajate koolitus peaks tutvustama eelkõige tarkvara standardfunktsionaalsust ja selle kohandamise võimalusi lähtudes juurutatavatest moodulitest. Tarkvara juurutusprojektide puhul lihtsustab see projekti meeskonna omavahelist suhtlemist. Projekti meeskond saab suhelda samades terminites (mis võib autori kogemuse põhjal osutuda päris suureks probleemiks) ning lahendusi välja töötades osatakse paremini hinnata võimalusi äriprotsesside või tarkvara kohandamiseks. Samuti saavad võtmeisikud uut tarkvara juba „oma käega katsuda", tekitades kohe positiivse tagasiside efekti. Arusaadavatel põhjustel ei ole tarkvara arendusprojektides koolitamine enne tarkvara valmimist võimalik.

Ülejäänud analüüsifaas keskendub uute äriprotsesside sajaprotsendilisele kaardistamisele, mille käigus täiendatakse kasutusloomudelit ja *gap-fit* analüüsi. Siinkohal tuleb tähele panna, et kasutusloomudel ja *gap-fit* analüüs on oma olemuselt vaheetapid täpsete nõuete seadmiseks hilisemas etapis. Mõlemad dokumendid on piisavalt kõrge abstraktsioonitasemega, et vältida liigset detailidesse süvenemist varases projekti faasis.

Analüüsifaasi kõige olulisem tulem on funktsionaalsete nõuete dokument (*functional requirements document*). Funktsionaalsete nõuete dokument koondab endas kõik kasutusloomudelis ja *gap-fit* analüüsis sätestatud (skoobis olevad) nõuded ning kirjeldab iga nõude kohta järgmised elemendid:

- hetkeseis (kajastab kas kasutatavaid süsteeme või käsitsi teostatavaid äriprotsesse);
- soovitatav lahendus (kuidas uus tarkvara antud teema lahendab, lahendus võib olla ka hetkeseisu säilitamine);
- mõju äriprotsessidele (näiteks: lahendus vähendab andmete sisestustööd kaks korda) – täidetakse vaid nõuete puhul, mis on väga olulised;

- edukuse mõõdupuu (näiteks: kokkuhoid andmete sisestuskuludelt 30%) – täidetakse vaid nõuete puhul, mis on väga olulised.



Joonis 3.6. Analüüsifaasi üldistatud töövoodiagramm (autori joonis Navision OnTarget-i põhjal).

ERP-süsteemide puhul saab funktsionaalsete nõuete kirjeldamisel kasutada võimalust kirja panna vaid need nõuded, mis ei sisaldu standardses paketis. Autori kogemus on näidanud, et selline lähenemine eeldab kliendilt tarkvarapaketi väga head tundmist ja/või piiritut kliendipooset usaldust. OnTarget-is mainitakse samuti taolise lähenemise ohtlikkust ning soovitab seda kasutada vaid juhul, kui võtmekasutajatele tehakse eelnevalt väga põhjalik koolitus.

Funktsionaalsuse kirjeldamisel soovitab OnTarget siiski hoiduda elementaarsete nõuete kirjeldamisest (näiteks: kliendi andmeid peab saama muuta ja kustutada). Piiri tõmbamine, kus algab mitteelementaarne nõue sõltub siiski rohkem konkreetsest kliendist ja projektist ning konkreetseid soovitusi siinkohal ei anta. Autori kogemus näitab, et elementaarsed nõuded on tavakasutajatele väikese koolitusega selgitatavad ning funktsionaalsete nõuete dokumendis selleks eraldi ruumi reserveerima ei pea.

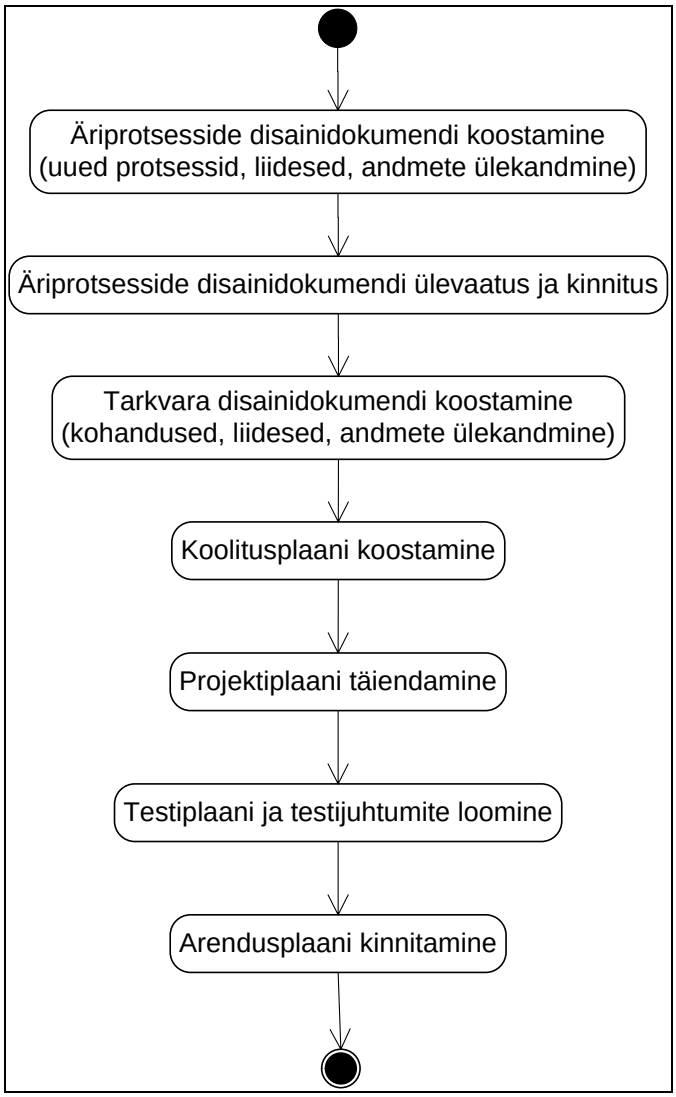
Funktsionaalsete nõuete dokument läheb OnTarget metoodika kohaselt kõigepealt kliendi juurde ülevaatusele ja täiendamisele. Lõpliku kinnitamise eel tuleb täiendamisettepanekud dokumenti sisse viia ja seejärel funktsionaalsed nõuded kahepoolselt kinnitada. Funktsionaalsete nõuete kinnitamise ja projektiplaani täiendamisega loetakse analüüsifaas ka lõppenuks.

3.4. Disain

Disainifaasi ülesanne on analüüsifaasis sätestatud nõuded viia sellisele kujule, et neid saaks arendajatele töökäskudena anda. Disainifaasi eesmärgid jagunevad neljaks:

- Saada detailne kirjeldus kõikidest funktsionaalsete nõuete dokumendis kirjeldatud kohandustest tarkvaras.
- Saada detailne kirjeldus uutest või muudetus äriprotsessidest.
- Saada detailne kirjeldus kõigist liidestest.
- Saada detailne kirjeldus uude süsteemi ülekantavate andmete struktuurist ja mahust.

Disainifaasis tehtavad tegevused viivad ka täpsemate töömahu hinnanguteni ning selle tulemusena tuleb uuendada ka projektiplaani ja teisi projekti ajagraafikut puudutavaid dokumente. Disainifaasis tehtavad tegevused on toodud üldistatud kujul joonisel 3.7.



Joonis 3.7. Disainifaasi üldistatud töövoodiagramm (autori joonis OnTarget põhjal).

Disainifaasis luuakse kaks uut dokumenti, mis kirjeldavad väga detailselt, kuidas ettevõte uut tarkvara kasutades tööle hakkab. Nendeks on:

- äriprotsesside disainidokument (*enterprise design document*);
- tarkvara disainidokument (*software detailed design document*).

Mõlemad dokumendid käsitlevad tarkvara ja äriprotsesside detailset disaini, kuid erinevate rakursside alt. Äriprotsesside disainidokument on mõeldud konsultandi ja võtmekasutajate vaheliseks suhtluseks, samal ajal kui tarkvara disainidokument keskendub tarkvara kohendamise realisatsiooni osale. Näide: äriprotsesside

disainidokumendis on kirjeldatud nõue, et kliendi andmete kaardil peab olema näha kliendi jooksva aasta ostusumma. Tarkvara disainidokumendis on sel juhul kirjeldatud järgmised aspektid: kus vormil mingi väli peab asuma ja milliste andmete põhjal antud summat tuleb arvutada. Järgnevalt vaatleme mõlemaid dokumente eraldi.

Äriprotsesside disainidokument on oma struktuurilt sarnane funktsionaalsete nõuete dokumendiga. Erinevus seisneb lihtsalt detailsuse astmes ja detailide käsitlemises. Kõik kohendused originaaltarkvaras tuleb kirjeldada detailselt ekraanipiltide tasemel. Kui mõni nõue ei vaja täpsustamist, saab äriprotsesside disainidokumendis piirduda viitega nõuete dokumendile. Lugeja jaoks võib äriprotsesside disainidokument kohati sarnaneda ka tarkvara kasutusjuhendiga, kuna lisaks ekraanipiltidele kirjeldatakse ka tarkvara kasutusprotsess. See on aga normaalne, kuna dokument peab olema arusaadav ka tavakasutajale.

Väliste andmevahetusliideste kirjeldamisel lähenetakse äriprotsesside disaini-dokumendis samuti kasutaja vaatepunktist. Käsitletakse vahetatavate andmete olemust ja ülekandmise eesmärki ning seejärel ülekandmise protsess kasutaja seisukohast. Protsess võib olla ka automaatselt käivituv – siis kirjeldatakse, kui sageli andmevahetus toimub. Kokkuvõtteks peaks äriprotsesside disainidokument olema piisav alus lõppkasutajate koolitusmaterjalide loomiseks. Väiksematel projektidel võib piisata dokumendi väljavõtete tegemisest lõppkasutajatele.

Tarkvara disainidokument on oma ülesehituselt ja sisult suunatud tarkvara arendusmeeskonnale. Tegu on standardse disainidokumendiga, kus tuleb dokumen-teerida muudatused ja täiendused andmemudelis, vormidel ja uus funktsionaalsus. Kuna OnTarget on suunatud konkreetse platvormile ja tootele, siis on kaasa pandud ka mitmeid näiteid, kuidas disainidokumendis asjad välja peaksid nägema. Näited ja valmis dokumendimallid on autori arvates OnTarget-i oluline eelis mittespetsialiseerunud metoodikate ees.

Disainidokumentide valmimise järel tuleb OnTarget-i kohaselt uuendada projektiplaani ja mahuhinnanguid ning teha testiplaan ja testjuhtumid. Testjuhtumite käsitlemine on OnTarget-i kohaselt standardne – iga funktsionaalse nõude kohta peaks olema vähemalt

üks testjuhtum. Testjuhtumite kirjeldamiseks on antud ka standardne mall, mis on sarnane teistes metoodikates kasutatavatele mallidele.

Testimisele ja testjuhtumitele on OnTarget-is pühendatud proportsionaalselt vähem mahtu kui näiteks RUP-is. Autori arvates peitub põhjus standardse tarkvara kasutamises, mis on kõigi eelduste kohaselt juba põhjalikult läbi testitud. Sellegipoolest peaks OnTarget veidi põhjalikumalt käsitlema juurdearenduste ja kohanduste testimise teemat, kuna seal peitub potentsiaalseid ohte. Levinumaid eksimusi Navisioni kohendamisel on standardfunktsionaalsuse või protsesside kergekäeline muutmine. Kuna standardpakett on sisemiselt väga tihedalt integreeritud võib standardi muutmine ühes moodulis rikkuda mõne teise mooduli funktsionaalsuse.

3.5. Arendus ja testimine

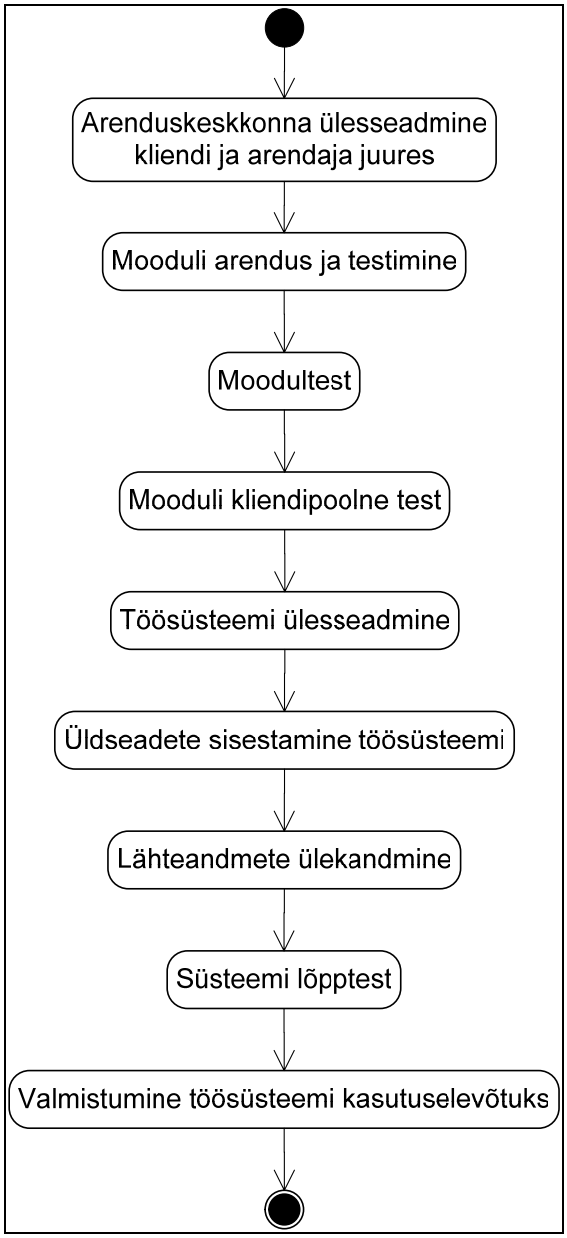
Arendus- ja testimisfaasi eesmärgiks on ette valmistada kliendi ja arendaja arenduskeskkond, teostada arendustööd, koguda kokku algandmed süsteemi seadistuseks (algsaldod jne.) ning lõpuks läbi viia aktsepteerimistestid. Arendus- ja testimisfaas realiseerib süsteemi nagu see oli ette nähtud disainifaasis.

Arendus- ja testimisfaas on eriline faas, kuna suurem osa tööst teostatakse kliendi juurest eemal. Seetõttu soovitab OnTarget eraldi tähelepanu pöörata kliendi informeerimisele jooksvate tööde osas. Informeerimise süsteemseks tagamiseks soovitatakse OnTarget-is läheneda arendamisele iteratiivselt – toimetada valmisarendatud funktsionaalsus kliendile testimiseks mitmes etapis. Tuleb tähele panna, et iteratiivsus OnTarget-i mõistes on erinev iteratiivsusest RUP-i mõistes. RUP näeb iteratiivsust ette kõigis projekti faasides, OnTarget vaid arendus- ja testimisfaasis.

Kuna arendus- ja testimisfaas on koht, kus klient näeb esimest korda juurdearendatud või kohandatud funktsionaalsust, siis tekib klientidel tavaliselt siinkohal lisasoove või muudatusettepanekuid. OnTarget julgustab rakendama formaalset protsessi muudatuste elluviimiseks, kuna liigsete muudatuste sisselubamine selles projekti staadiumis võib ohtu seada tähtajad ja eelarve.

Võtmetähtsusega edukriteeriumiteks arendus- ja testimisfaasis on OnTarget-i kohaselt:

- Võtmetähtsusega funktsionaalsus tuleb kõigepealt arendada ja testida.
- Alguses tuleb tegeleda kõige vähem arusaadud teemadega.
- Formaalne ja kõikehõlmav muudatuste ja konfiguratsiooni haldamine.



Joonis 3.8. Arendus- ja testifaasi üldistatud töövoodiagramm (autori joonis OnTarget-i põhjal).

Arendus- ja testimisfaasis ei looda uusi dokumente – pigem tegeletakse olemasolevate dokumentidesse vajadusel pisikohanduste sisseviimise ja jälgimisega. Oluline koht

arendus- ja testimisfaasi käigus on muudatusettepanekutel ja vearaportitel. Muudatusettepanekute kohta soovitatakse hoida lihtsalt formaalset joont. Vearaportite haldamiseks annab OnTarget soovituse rakendada spetsiaalne süsteem.

Analüüsifaasi oluline osa on koostöös kliendiga sisse kanda süsteemi põhiparameetrid ning kliendi poolt ettevalmistatud algandmed. ERP-süsteemide iseärasus on asjaolu, et nende käivitamisel on tarvilik teatava alginfo olemasolu (algsaldod, kliendid, hankijad, laoseisud jne.), et asutus saaks jätkata oma igapäevast tööd uues süsteemis. Praktikas esineb probleeme algandmete kvaliteedi ja õigeaegse ettevalmistuse osas – need on samuti projektijuhtimise seisukohalt riskid, mida varakult maandada. Kui funktsionaalsus on kliendi poolt aktsepteeritud ja algandmed sisse kantud, loetakse arendus- ja testimisfaas lõppenuks.

3.6. Juurutusfaas ja kasutajatugi

Evitusfaas tähistab arendustööde lõppu – tarkvara on analüüsitud, disainitud, arendatud ja testitud. Projekti skoobis on olnud ilmselt mõningaid muudatusi ja kriitilised vead on parandatud. Evitusfaasis viiakse läbi kasutajate koolitused ja süsteemi lõplik installeerimine tööserveris.

Esineb juhtumeid, kui Navisioni juurutusprojekt jõuab evitusfaasi nii, et ta ei ole läbinud kõiki ettenähtud testjuhtumeid. Taolised juhtumid on põhjustatud välistest asjaoludest. Näiteks on enamusel raamatupidamissüsteemidel nõue, et süsteemivahetus saab toimuda ainult aastavahetusel. Samuti võivad põhjuseks olla seadusandlusest tulenevad pealesurutud funktsioonid, mis on vanas süsteemis realiseerimata, kuid uues süsteemis juba toimivad (näiteks Eesti liitumisel Euroopa Liiduga 1. mail 2004 tekkis ettevõtetel kohustus esitada Intrastat aruandlus kaubavahetuse kohta teiste EU riikidega).

Kui tarkvara läheb kasutusele hoolimata sellest, et kõik aktsepteerimistestid on läbitud, tuleb jätkata arendus- ja testimistegevusega seni, kuni vead on kõrvaldatud. Otsuse, kas ja millal tarkvara kasutusse läheb, teeb juhtkomitee. OnTarget-i kohaselt peavad projektijuhid tegema juhtkomiteele tarkvara tutvustuse enne, kui käivitamise otsus vastu

võetakse. Vastav otsus tähendab ka vastutuse tarkvara eest minekut arendaja/juurutaja käest kliendi kätte.

Tarkvara käikulaskmise järel soovitatakse OnTarget-is teostada projekti tulemuste analüüs. Tulemuste analüüsis võetakse ette projekti visioon ja peamised valupunktid, mille pärast projekt üldse ellu kutsuti, ning võrreldakse neid teostatud tarkvaraga. Taoline analüüs peaks näitama võimalikud järgmised sammud tarkvara edasiarendusel ning kindlustama kliendile, et tema ootused said täidetud.

Kui süsteem on käiku lastud, muutub arendaja tegevus aktiivsest reaktiivseks. Kätte jõuab jooksva kasutajatoe pakkumise faas. Kasutajatoe all mõistetakse siinkohal kõiki tegevusi, mis teostatakse enne seda, kui jõutakse uuesti müügifaasi. Kasutajatoe faasiga seotud tegevused on seotud tarkvara toe, hooldustegevuse ning uuendamisega. Oluline roll hooldustegevusel on kliendi uute vajaduste tuvastamine ja neile lahenduste väljapakkumine. Igast sellisest soovist võib tulla uus arendus- või täiendusprojekt, mida tuleb juba eraldi käsitleda.

4. RAKENDUSANALÜÜS MAJANDUSTARKVARA ARENDUSPROJEKTI PÕHJAL

4.1. Ülevaade projekti olemusest ja ärivaldkonnast

Vaadeldav ettevõte (edaspidi ka klient) tegeleb mobiiltelefonide jaemüügi, hoolduse ja kõneoperaatorite kliendihaldusega. Tegu on mobiiloperaatori tütarfirmaga, kuhu on delegeeritud kõik vahetud klienditeenindustegevused. Klienditeeninduseks on ettevõttel üle Eesti üle kahekümne kaupluse. Tütarfirma on oma otsustes suures osas sõltuv emafirmast, kuna müügi- ning kampaaniate kohta käivad otsused tehakse tavaliselt emafirma tasemel ja need peavad kajastuma ettevõtte kauplustes.

Projekti alguses oli kliendil peakontor, 22 kauplust ja ladu. Firma struktuur on funktsionaalselt ülesehitatud – direktorile alluvad müügijuht, ostujuht, piirkonnajuhid. Finantsjuhtimine, raamatupidamine ja IT-teenused olid emafirma ülesandeks. Üks töötaja tegeleb olemasoleva tarkvaralahenduse hooldamise ja pakub kauplustele tehnilist tuge.

Mobiilside ja mobiiltelefonide turgu võib kirjeldada sõnaga „tormiline”. Sagedased kampaaniad, muudatused müügitingimustes ja moodustes, erinevad allahindluse moodused, järelmaksud ja kauba „tasuta kaasa” andmine on tunnistus pidevast konkurentsisurvest. Lisaks kliendi organisatsiooni pidevale muutusteks valmisolekule peab selleks valmis olema ka tarkvara.

Klient kasutas olemasolevat ERP-süsteemi Microsoft Navision, mida eelnev arendaja oli tugevalt kohendanud, lisades müügimoodulile kassadele vajalikku funktsionaalsust. Olemasoleva süsteemiga rahul ei oldud. Kõige olulisemate probleemidena nimetati:

- müügisüsteem (isegi kohandatuna) ei olnud sobilik kassatööks ning kohanduste tõttu tekkisid vead ka standardsüsteemis;
- müügitehingu sisestamine süsteemi võttis liiga aega ja nõudis palju käsitööd – tulemuseks näpuvead;

- kassad said teha kontrollimatuid allahindlusi ja muid muudatusi süsteemi seadistustes;
- puudus integratsioon kontserni teiste süsteemidega ning infovahetus toimus käsitööna;
- laoarvestus ei olnud usaldusväärne ning ei haakunud finantsarvestusega.

Ülalloetud probleeme oli püütud ka lahendada. Esiteks prooviti kohendatud müügisüsteemi veel täiendada, kuid see ei andnud soovitud tulemusi (kasvas vigade arv ja kasutatavus kannatas). Seejärel asuti aktiivselt otsima vertikaallahendust Navision tarkvarale. Lahendus leiti Islandil tehtud Navision lisamooduli Infostore näol. Pärast Infostore litsentside soetamist sõlmiti leping autori tööandjaga juurutusprojekti elluviimiseks.

Vaadeldava projekti eesmärgiks oli Infostore lisamooduli juurutamine, et lühendada kassas tehtavate toimingute ajakulu ja saavutada kassapidajate tegevuse üle tarkvaraline kontroll. Kõrvaleesmärgiks oli ka olemasoleva Navision süsteemi seadistuste korrigeerimine ja ühtlustamine kontserni nõuetega.

4.2. Metoodikate rakendamise analüüs

Vaadeldava projekti esimene juhtimispõhimõte oli, et „tehakse vaid seda, mis tundub osapooltele mõistlik ja vajalik”. Sellega oli vastu võetud ka otsus, et ühtegi metoodikat pimesilmi järgima ei hakata. Pigem vaadeldi metoodikaid kui parimate praktikate kirjeldusi ja nopiti neist sobivaid põhimõtteid, meetodeid või komponente.

4.2.1. Projektorganisatsioon

Projektorganisatsioon ehitati üles OnTarget põhimõttest lähtuvalt. Projekti ajagraafikute, eelarve ja ressursiprobleemidega tegeles juhtkomitee. Projekti igapäevast kulgu juhtisid kliendi ja teostaja poolsed projektijuhid. Teostajapoolset arendusmeeskonda juhtis teostaja projektijuht ja kliendipoolset meeskonda kliendipoolne projektijuht. Konkreetsete probleemide lahendamiseks kutsuti kokku gruppe, kuhu kuulusid vajalikud inimesed nii kliendi- kui teostaja juurest.

Projekti kõige kõrgem juhtorgan oli juhtkomitee, kuhu kuulusid kliendi ja tellijapoolse ettevõtte juhid ja projektijuhid. Juhtkomitee käis koos minimaalselt kord kuus või iga iteratsiooni üleandmisel. See pidi tagama projektist ülevaate juhtkonna tasemel ning andma projektijuhtidele võimalusi tõstatada teemasid, mille lahendamine ei olnud nende pädevuses.

Projektijuhid pidid juhtima projekti igapäevast tööd. Arendusmeeskonda juhtis teostaja projektijuht (ka käesoleva töö autor). Kliendipoolne projektijuht pidi hoolitsema kliendipoolse infrastruktuuri (serverid, ligipääsud ja õigused), info hankimise ja kokkusaamiste organiseerimise eest. Projekti planeerimist teostati koostöös ning projektiplaanides kirjeldati mõlema osapoole ressursid ja ajakava.

Arendusgrupp koosnes projektijuhist, analüütikust ja kahest arendajast. Seega oli tegemist väikese tarkvaraprojektiga, kus oli hõlmatud neli inimest. OnTarget loeb seda keskmise suurusega projektiks, kuid enamus tarkvara arendusprojekte käsitlevad autorid loeksid projekti väikseks.

Kliendipoolne meeskond koosnes iga funktsionaalse valdkonna võtmeisikutest (direktor, müügijuht, ostujuht, hooldusjuht, finantsjuht, raamatupidaja, IT-tugiisik). Nendele lisaks kaasati meeskonda ka kaks müüjat, kes pidid esindama lõppkasutajate huve. Taoline kliendi meeskonna ülesehitus haakub nii OnTarget-i kui ka unifitseeritud protsessi parimate praktikatega. Müüjate haaramine projekti varajases faasis oli vajalik, kuna nemad olid ainukesed, kes loodava tarkvara kassaliidest pidid ka kasutama hakkama.

Kliendipoolne projektijuht ei olnud kliendi organisatsioonist, vaid tuli väljastpoolt – emafirma IT-osakonnast. Ühelt poolt oli ta kursis emafirma tarkvaralise poliitikaga ja teadis kuidas kontserni IT-süsteemid tervikuna töötasid. Teisalt puudus tal siseinfo ja alluvussuhe kliendi organisatsiooni direktoriga. Nii oli kliendipoolse projektijuhi omadustes ühendatud kompetentsus ja sõltumatus.

Projektorganisatsiooni tagantjärele hinnates võib öelda, et selle ülesehitus oli teooriale täielikult vastav ja praktikasse rakendatuna töötas autori hinnangul hästi. Projektorganisatsiooni struktuuri tutvustati projekti alguses kõigile osalistele esimesel

koosolekul. Tutvustus tagas, et inimesed olid teadlikud oma rollist projektis ja sellest, kelle poole info saamiseks või probleemide lahendamiseks pöörduda.

Autori hinnangul oli suur abi juhtkomitee regulaarsel kooskäimisel. Juhatuse tasemel oli olemas info projekti staatuse kohta. Aktsepteerimistestid andsid juhtkomitee liikmetele ka selge käegakatsutava tõestuse projekti hetkeseisust. Projekti käigus tekkis probleeme, mille lahendamiseks polnud projektijuhtidel volitusi või tekkis erimeelsusi. Need lahendati juhtkomitees. Kui juhtkomitee poleks koos käinud, oleksid need probleemid projekti lõpus esile kerkinud ja edukat lõpetamist edasi lükanud.

Kliendipoolse projektijuhi tulek kliendi organisatsioonist väljastpoolt andis projekti kvalitatiivsele küljele palju juurde. Esiteks oli info hankimise vajadus olemas ka kliendi projektijuhil ning nõuete kirjeldamisel ei jäänud midagi märkamata või „iseenesest mõistetavaks". Inimlikul tasandil vältis väljastpoolt tulnud projektijuht kliendi ja teostaja vahelise konflikti tekkimise võimaluse, kuna suutis objektiivsemalt hinnata mõlema poole argumente ning leida konstruktiivne väljapääs. Oluline aspekt siinjuures on ka see, et kliendipoolsel projektijuhil polnud muid igapäevaseid kohustusi – kui projektijuhiks pannakse keegi organisatsiooni seest, siis võib tekkida probleeme konkreetse inimese ajaressursi jagamisega.

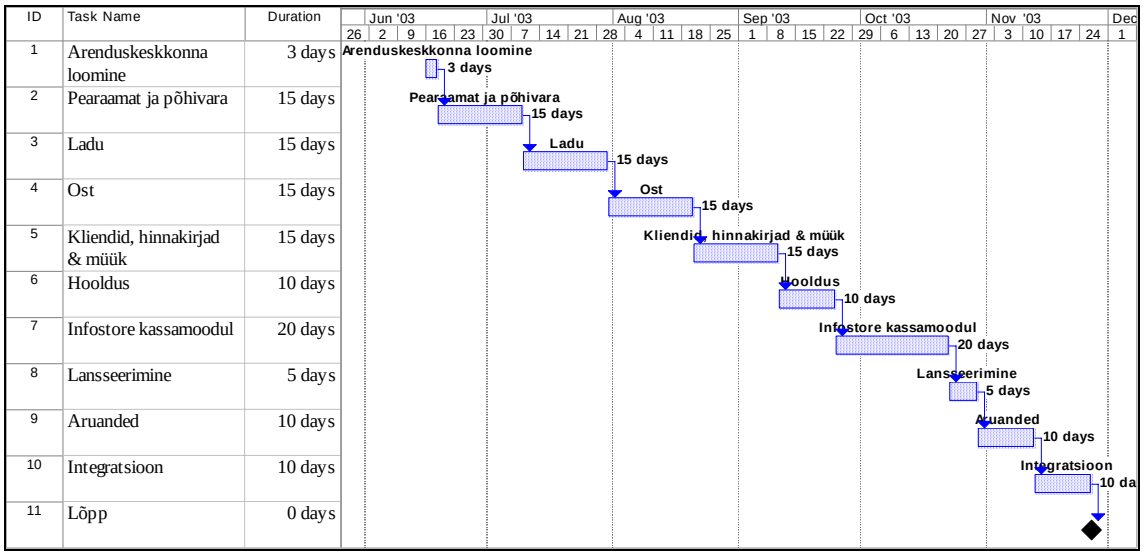
Kokkuvõtteks saab öelda, et OnTarget-is kirjeldatud projektorganisatsioon toimib reaalses elus väga hästi. Antud ülesehitus sobib ka unifitseeritud protsessi põhimõtetega (unifitseeritud protsessis on kirjeldatud vaid see, et projekti peavad olema kaasatud kõik osapooled). Kasuks tuleb juhtkomitee regulaarne kooskäimine ja klientorganisatsiooni väline täiskohaga töötav projektijuht, kes tegeleb kliendipoolse töökorralduse ja planeerimisega.

4.2.2. Projekti ülesehitus ja planeerimine

Projekt tervikuna oli üles ehitatud unifitseeritud protsessi põhimõtete järgi – iteratiivselt ja inkrementaalselt. Iga iteratsioon lõppes kliendipoolse aktsepteerimistestiga. OnTarget-iga aga haakus üldine põhimõte, et tarkvara võetakse kasutusele väliste tegurite poolt ettemääratud ajal – antud juhul kuu aega enne jõulumüügiperioodi algust. Tuleb silmas pidada, et unifitseeritud protsess näeb ette iga iteratsiooni lõpus tarkvara

kasutusele võtmise (kui see vähegi võimalik on), kuid antud projekti puhul polnud see võimalik.

Projektil oli kokku planeeritud kaheksa iteratsiooni. Iga iteratsioon pidi kestma kaks kuni neli nädalat – pikkus sõltus keerukuse astmest. Tarkvara pidi kasutusse minema pärast kuuenda iteratsiooni läbimist. Eesmärgiks oli töötava funktsionaalsuse võimalikult varajane kasutajateni viimine. Lisaaruandlus ja integratsioon väliste süsteemidega jäeti seetõttu lansseerimisjärgseteks tegevusteks. Iteratsioonide üldine plaan on toodud joonisel 4.1.

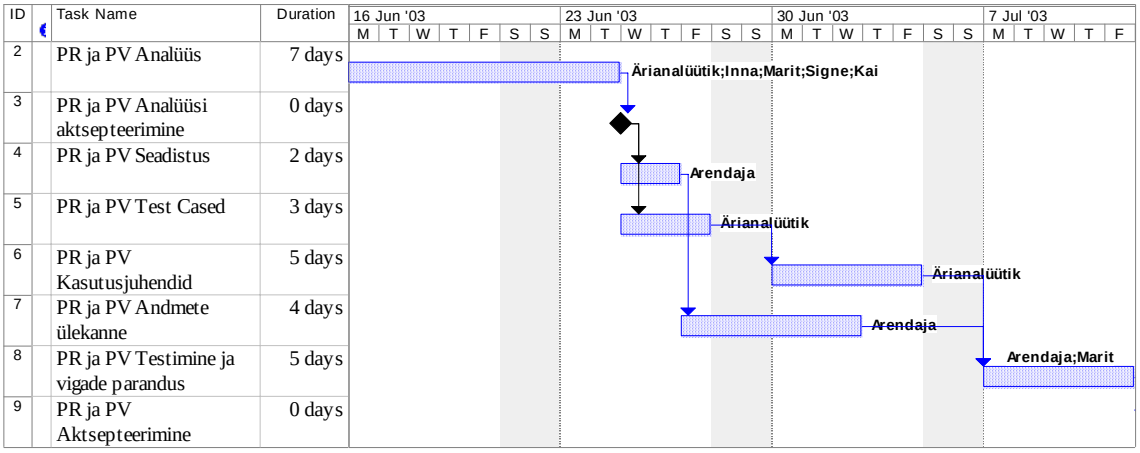


Joonis 4.1. Projekti iteratsioonide algne plaan (autori joonis).

Sisemiselt olid iteratsioonide plaanid äravahetamiseni sarnased – kasutati sarnaseid tegevusi ja vahepunkte (*milestone*). Iga iteratsiooni lõpus olid planeeritud aktsepteerimistestid ning iteratsiooni tulemite üleandmine juhtkomiteel. Oma olemuselt oli iga iteratsioon väike OnTarget-i põhine juurutusprojekt, mis algas analüüsist ja lõppes testimisega. Pearaamatu ja põhivara iteratsiooni plaan on toodud joonisel 4.2. Planeerimisel detailsemaks ei mindud, kuna sellel ei nähtud lisanduvat kasu – vaid lansseerimise jaoks tehti eraldi väga detailne lansseerimisplaan.

Igas iteratsioonis oli määratud üks vaheetapp – analüüsi aktsepteerimine, millega kontrolliti ja fikseeriti iteratsioonisisest progressi. Analüüsi aktsepteerimine toimis kui iteratsioonisisene nõuete dokumendi allakirjutamine – sellega kinnitati iteratsiooni

teostatava mooduli skoop ja funktsionaalsus. Iteratsiooni lõpetas kliendipoolne aktsepteerimistestide edukas läbiviimine.



Joonis 4.2. Iteratsiooni „Pearaamat ja põhivara" plaan (autori joonis).

Projektiplaani haldamiseks ja jälgimiseks kasutati Microsoft Project 2000 tarkvara. Autori hinnangul oli antud projekti jaoks tegu hea tööriistaga, kuna see võimaldas korraga jälgida projekti mitut aspekti – aja kestvust, töö mahtu ja hõivatud ressursse. Põhiline kasu tarkvarast tõusis ressursside planeerimisel, kuna tarkvara informeeris asjaolust, kui mõnda ressurssi (tavaliselt arendajat) planeeriti rohkem kui sajaprotsendilise koormusega.

Projektiplaani jooksvaks jälgimiseks kasutati ettevõtte A. Y. Goldratt Baltic abi. Esiteks töödeldi projektiplaani ajagraafikuid – kõikide tööde mahust jäeti alles kaks kolmandikku ja „võidetud aeg" pandi lõppu „puhvriks". Nii oli võimalik kriitilise ahela kaudu jälgida, kas puhver väheneb ettenähtust kiiremini või aeglasemalt. Antud meetodi üks eeldusi oli, et projekti täideviiv meeskond ei omaks infot, kui kaua käsiloleva ülesande jaoks aega on ette nähtud. Käsilolev ülesanne pidi alati valmis saama „niipea kui võimalik". Praktikas seda meetodit siiski ei rakendatud ja Goldratt-i meetodit kasutati eelkõige alternatiivina projekti edusammude mõõtmiseks.

Iteratiivne plaan tagas projekti läbipaistvuse juhtkomitee jaoks, kindlustades õigeaegse info ajakava venimise kohta või konfliktidest projekti skoobi käsitlemisel. Kahe- kuni neljanädalased iteratsioonid „sundisid" juhtkomiteed ka piisavalt sageli kogunema.

Projekti esimestel kuudel selgus näiteks asjaolu, et kliendipoolsete võtmeisikute puhkuste tõttu ei ole võimalik vahepeal töid teostada – see põhjustas projekti venimist kahe nädala võrra. Iteratiivse projektiplaani tõttu „paistis" see kohe ka juhtkomiteele välja ja projektiplaani tehti vajalikud korrektiivid.

Goldratti projektijuhtimismeetodi rakendamine ei andnud erilisi tulemusi. Kui arendusmeeskond ei saanud ülesannete tähtaegade kohta rohkem infot kui, et konkreetse ülesandega on „väga kiire", siis oli tulemuseks dokumentide pikk viimistlemine. Arendusmeeskond ei teadvustanud enda jaoks hetke, mil dokument oli „piisavalt hea", et aktsepteerimiseks esitada. Tulemuseks oli täielik tähtaegade ületamine. Kui meeskonnale anti infot kolmandiku võrra kärbitud tähtaegade kohta, tuli sellele reaktsiooniks „ebareaalne" ning ülesanne täideti üldjuhul kas esialgu planeeritud ajaga või veidi hiljem. Autori arvates väärib Goldratti projektijuhtimismeetod meetod eraldi uurimust ja käesolevas töös ei käsitleta seda pikemalt.

Projektiplaani ülesehitamisel ei lähtutud põhimõttest, et kõige suurema riskiga osa tuleb lahendada kõige varem. Nimelt oli antud projekti kõige olulisem ja kriitilisem osa Infostore kassa lisamoodul. Paraku eeldas Infostore-i juurutamine Navisioni põhimoodulite eelnevat seadistamist ja arendamist. Projekti käigus selgus aga, et Infostore standardlahenduse funktsionaalsus ei vastanud täielikult kliendi vajadustele ning sinna tuli teha olulisel määral kohandusi ja täiendusi. Kokkuvõttes paisus iteratsiooni „Infostore" töömaht kaks ja pool korda. Selle tulemusena nihkus projekti lansseerimise tähtaeg poolteist kuud edasi, mis kuuekuulise projekti puhul on oluline nihe. Sisuliselt realiseerus koskmudelile iseloomulik „suur pauk" projekti lõpus, kus lansseerimise eel tehakse palju täiendusi ja muudatusi ning lisatakse funktsionaalsust, mida algselt polnud planeeritud.

Infostore iteratsiooniga seoses selgub ka järgmine puudus planeerimisel – nõuete haldus. OnTarget näeb nõuete fikseerimist ette projekti alguses. Iteratiivse lähenemise korral on iteratsiooni alguses juba nõuded lukus ja iteratsiooni skoop fikseeritud. Vaadeldavas projektis eksiti selle põhimõtte vastu. Iteratsiooni nõuded fikseeriti iteratsiooni käigus. Tulemuseks oli olukord, kus iteratsiooni alustades polnudki tegelikult võimalik teada, kui kaua iteratsioon täpselt kestab. Funktsionaalsus polnud fikseeritud ja skoop selletõttu määramata. Autori arvates parandaks olukorda kogu

projekti nõuete analüüsi ja spetsifikatsiooni koostamine esimese iteratsiooni käigus või projekti alguses (nagu on sätestatud diagnoosifaasis metoodikas OnTarget). Selle põhjal saab koostada põhjaliku projektiplaani (on see siis iteratiivne või koskmudelil põhinev), kartmata, et projekti käigus lisandub realiseeritavat funktsionaalsust.

4.2.3. Tulemid

Vaadeldavas projektis tehti igas iteratsioonis eraldi komplekt dokumente ja vajadusel täiendati varasemates iteratsioonidest tehtut. Projekti alguses lepiti kokku järgmiste tulemite tegemises igas iteratsioonis:

- analüüsidokument;
- testjuhtumid;
- aktsepteerimistestide protokoll;
- kasutusjuhend.

Ülalloetud tulemitele lisanduvad ka projektijuhtimisega seonduvad üldised dokumendid – projektiplaan, aktid jne. Järgnevalt käsitleme tulemeid ja nende sisu täpsemalt. Nimelt ei vasta iga tulemi sisu täpselt OnTarget-i vastava dokumendi nimetusele.

Analüüsidokument on kompositsioon ärianalüüsist, *gap-fit* analüüsist, nõuete spetsifikatsioonist ja disainidokumendist. Kompositsiooni põhjuseks oli iteratiivne arendus – nii oli lihtne eristada erinevates iteratsioonides valminud dokumente ja moodulite projektidokumentatsiooni. Projekti lähtekohast (klient juba kasutas Navisioni ja suures osas sooviti kohendusi ja muudatusi) tulenevalt otsustati nõuded ja nende lahendamise kirjeldus igas iteratsioonis ühte dokumenti koondada.

Analüüsidokumendi üldine struktuur oli järgmine:

1. Dokumendi skoop ja eesmärk.
2. Hetkeolukorra kirjeldus – äriprotsessid, kasutatav (tarkvaraline) lahendus, probleemid ja nõuded uuele tarkvarale.

3. Lahendusettepanek – analüüs, disain ja uus tarkvaralahendus.

Hetkeolukorra kirjelduses vaadeldi käsitletava mooduli äriprotsesse. Iga äriprotsessi kohta anti üldine kirjeldus vajaduste tasemel. Seejärel kirjeldati olemasolevat tarkvaralahendust (kui see eksisteeris) ja sellega seonduvaid probleeme. Hetkeolukorra ja probleemide põhjal sai sõnastada nõuded uuele tarkvarale. Lahendusettepaneku peatükis käidi samad äriprotsessid läbi ning esitati lahendus uues tarkvaras. Vajadusel lisati lahendusele ka tehniline kirjeldus. Lahendusettepanek oli punkt, kus OnTarget-i mõistes teostati *gap-fit* analüüs – võrreldi nõudeid ja tarkvara võimalusi ning vajadusel sõnastati arendus. Vana ja uut lahendust käsitleti selguse huvides eraldi peatükkides äriprotsesside kaupa. Äriprotsesside kirjeldamisel lähtuti kasutuslugudest. Samas, kasutuslugusid ei joonistatud diagrammidena ega modelleeritud, kuna see ei olnud kummagi osapoole arvates vajalik.

Seega analüüsidokument on komplekteeritud OnTarget-ist lähtuvalt ja projekti vajadustele kohendatud. OnTarget-i lähenemist otsustati modifitseerida järgmistel põhjusel:

- igas iteratsioonis eraldi *gap-fit* analüüs, nõuete spetsifikatsioon, äriprotsesside disain ja tarkvara disain oleks dokumendid liigselt tükeldanud;
- ühe dokumendi pidev uuendamine ja täiendamine oleks tekitanud kliendis pideva kordamise tunde ning hägustanud iteratiivset protsessi;
- ärifunktsioonide kaupa grupeeritud dokumendid võimaldavad eri iteratsioonides kaasata erinevaid inimesi – võtmeisikud peavad läbi töötama vaid neid puudutavat materjali.

Testijuhtumite kirjelduse dokumendid olid oma ülesehituselt sarnased nii OnTarget-is kui ka unifitseeritud protsessis kirjeldatud dokumentidele. Iga iteratsiooni kohta tehti komplekt testjuhtumite kirjeldusi. Testjuhtumite kirjeldused kinnitati kliendi poolt pärast analüüsi kinnitamist. Testimise tulemused fikseeriti aktsepteerimistestide protokollis, mis märkis ka iteratsiooni edukat lõppu.

Järelhinnanguna võib öelda, et dokumentide ülesehitus üldiselt õigustas ennast. Kui kliendi poolt oli analüüsidokument kinnitatud, ei olnud realiseerimisel erilisi probleeme ega juurde tekkivaid nõudeid.

Nõuetega seonduvalt on alguses soovitatav teha üks üldine kogu projekti kohta käiv nõuete spetsifikatsiooni dokument. Nõuete spetsifikatsioonidokument aitab fikseerida kõikide iteratsioonide töömahud ja ajakava enne iteratsiooni algust. Nõuete haldusega seotud probleeme käsitleti pikemalt ka eelnevas peatükis.

Ärianalüüsi ja tarkvara disaini ühte dokumenti liitmine andis kasu ka tarkvara dokumenteerimise ja kommunikatsiooni seisukohalt. Kui äri- ja tarkvaradisain on üheskoos, saab arendaja ka parema ülevaate arendatavast funktsionaalsusest ning teeb suurema tõenäosusega nõuetele vastava funktsionaalsuse. Kui programmeerija ei tea disainis kirjeldatava funktsionaalsuse tagamaadest, on võimalik mitmeti mõistmine ja tulemus ei vasta ootustele (mis tähendab ümbertegemist testimise järel). Antud projektis juhtus seda harva, kuna arendajal oli parem ülevaade kliendi vajadustest.

Reaalne kasu tõusis ka funktsioonide kaupa dokumentatsiooni grupeerimisest. Võrreldes teiste analoogsete projektidega, oli kliendipoolne tagasiside kiirem ja kvaliteetsem. Kliendipoolsed võtmeisikud lugesid esitatud dokumente suurema motivatsiooniga, kuna seal sisaldus eelkõige neid endid puudutav info.

Kokkuvõtteks saab öelda, et majandustarkvara juurutamisel on vajalik teha nõuete spetsifikatsioon enne tavapärase iteratsioonidega arendustsükli alustamist. Igas iteratsioonis tuleks teha ärianalüüs, analüüs ja disain. Kui iteratsioonid on piisavalt lühikesed, on mõttekas need koondada ühte dokumenti. Tähtis on iga iteratsiooni lõpuks teha testjuhtumite põhjal aktsepteerimistestid, et klient saaks veenduda funktsionaalsuse olemasolus ja töökindluses.

4.2.4. Parimad praktikad tegelikus projektis

Muude praktikate all vaatleme unifitseeritud protsessis ja OnTarget-is välja toodud parimate praktikate rakendamist ERP-tarkvara juurutusprojektis. Vaadeldavas projektis käsitleme neist järgmiseid:

- Visuaalne modelleerimine;

- Muudatuste haldus;
- Komponentarhitektuur;
- Vigade haldus;
- Pidev kvaliteedikontroll.

Vaadeldavas projektis kasutati minimaalselt visuaalset modelleerimist. Äriprotsesside ja organisatsiooni kirjeldamiseks kasutati küll vabas vormis diagramme, kuid kindlat notatsiooni ja süstematiseeritud mudeleid ei kasutatud. Autori arvates see antud projektile erilist mõju ei avaldanud, kuna näiteks kasutusloodiagrammide joonistamine poleks kellelegi projekti osalistest lisaväärtust andnud. ERP-projektide puhul on modelleerimise põhieesmärk eelkõige äriprotsesside selgem kirjeldamine ja antud projektis kasutati sel eesmärgil vaba notatsiooniga diagramme teksti sees.

Visuaalne modelleerimine võib aga väga vajalikuks osutuda, kui olemasolevale ERP-lahendusele arendatakse juurde uus lisamoodul. Sellisel juhul on visuaalne modelleerimine vähemalt konkreetse mooduli puhul täiesti õigustatud ja kasulik. Lisamooduli arendamine aga tähendab juba arendusprojekti – seega on seal mõttekam kasutada rohkem elemente arendusmetoodikatest (nagu näiteks unifitseeritud protsess).

Muudatuste haldusega tegelesid vaadeldavas projektis põhiliselt projektijuhid. Projekti käigus üldiselt oli nõuete muutumist vähe– pigem oli tegemist nõuete paisumisega (mida käsitleti põhjalikumalt eelpool). Vähesed nõuete muudatused, mis esile kerkisid, olid seotud integratsiooniga – nimelt tekkis kliendi meeskonnal integratsiooni iteratsiooni poole peal uusi mõtteid ja muudatused aktsepteeriti projektijuhtide tasandil.

Metoodiliselt ei ole selline lähenemine õige – nii unifitseeritud protsess kui ka OnTarget näevad sellisel juhul ette „kõrgemate organite” poole pöördumist. Iga nõude muutmine tähendaks ka projektiplaanide jms. muudatust. Vaadeldavas projektis tuleb arvestada asjaolu, et nõuded fikseeriti iteratsiooni sees ja projektijuhtidel oli võimalus muutused „ära teha” ilma juhtkomiteed informeerimata. Isegi kui informeerimine oleks toimunud, siis oleks seda nõuete muutumist käsitletud kui nõuete täpsustamist ja tulemus oleks olnud sama.

Lahenduseks on nõuete fikseerimine enne iteratsiooni algust. Sellisel juhul pole projektijuhtidel volitusi nõudeid (ja sellega seonduvalt ajaplaani, ressursse ja eelarvet) muuta. Nõude muutmise peab aktsepteerima juhtkomitee ja kinnitama sellega koos ka uue aja- ja ressursigraafiku. Autori arvates läheb muudatuste haldus tööle siis, kui seda toetab nii projektiplaani ülesehitus kui ka projektorganisatsioon.

Muudatuste halduse all mõeldakse ka versioonihaldustarkvara kasutamist, mis võimaldaks meeskonnal koos elektroonilisi dokumente muuta ilma, et töö kaotsi läheks. Vaadeldavas projektis kasutati Microsoft Visual SourceSafe tarkvara, kus hoiti kõiki dokumente ja koodiplokke. Tarkvara kasutamine õigustas ennast – vajalik info oli kõigile projekti liikmetel kättesaadav ja dokumentidest säilitati ka vaheversioonid.

Unifitseeritud protsess käsitleb üsna pikalt arhitektuuri ja komponentarhitektuuri olulisust tehniliste aspektide käsitlemisel. Autori arvates on tegemist ainult arendusprojektidele iseloomuliku teemaga. ERP-tarkvara pakettidel on olemas oma arhitektuur ja sisemine ülesehitus ning kui selle baasil arendatakse midagi juurde, on tehniline personal tavaliselt sunnitud juba olemasolevates raamides tegutsema. Vaadeldavas projektis kasutati Navisioni sisseehitatud arendusvahendeid kõikide kohanduste ja arenduste tegemiseks ning arhitektuur oli juba eelnevalt tarkvara loojate poolt väga selgelt defineeritud. Autori arvates on ERP-tarkvara puhul komponentarhitektuuri parim praktika pea alati rakendatud, kuna sellega tuleb arhitektuuri kirjeldus kohe kaasa. Ainuke väljakutse siinjuures on arendusmeeskonna harimine olemasoleva arhitektuuri osas.

Vigade haldus on iteratiivse arendusprojekti puhul väga oluline valdkond. Testimise ja varajase kasutusele võtmise käigus tekib vigu ja neid on tarvis hallata. Vaadeldavas projektis kasutati vigade haldamiseks firmasisest veahaldustarkvara BugBase. Tegu oli veebipõhiselt kasutatava süsteemiga, kuhu pääsesid vigu raporteerima ka kliendi meeskonna liikmed. Projekti lõpus vahetati BugBase välja vabavaralise tarkvara Mantis (http://www.mantisbt.org/) vastu.

Korrektsest veahaldustarkvarast siiski ei piisanud. Veahaldusega kaasnes praktikas kaks probleemi – a) raporteerijate koolitus ja b) ajaplaneerimine vigade parandamiseks. Vearaport on dokument, mille koostamine ei ole tavakasutajale alguses kohe jõukohane.

Projekti käigus selgus, et informatiivsete vearaportite saamiseks on vaja kasutajaid koolitada, et nad annaksid vea kohta piisavalt infot selle taastekitamiseks. Koolitamine vähendas oluliselt taastekitamatute probleemide osakaalu teatatud vigades. Omaette väljakutseks kujunes vigade parandamisega seotud temaatika. Nimelt ei olnud projektiplaanis piisavalt arvestatud ajakulule, mis kaasnes vigade parandamise ja paranduste testimisega.

Veaparandusteks aja planeerimine on psühholoogiline teema. Kui testimiseks ja veaparandusteks planeerida minimaalselt aega, siis on tavapäraselt tulemuseks asjaolu, et testimisega saadakse valmis, kuid parandustsükliks aega ei jää. Kui planeerida testimiseks ja veaparanduseks sama palju aega kui programmeerimisele, on tulemuseks ebakvaliteetne arendus – arendajad arvestavad, et „aega on" ning annavad pooliku koodi testimiseks. Sel viisil loodetakse arendus kiiremini valmis saada ja loodetakse, et „testimise käigus silume". Kokkuvõttes võtab teine lähenemisviis praktikas autori hinnangul rohkem aega (antud psühholoogilise efekti vastu võib aidata ehk Goldratti projektijuhtimise ideoloogia, kus arendajate eest varjatakse planeeritud ajakava). Lahenduseks on siin vaid firmasissese kvaliteedikultuuri arendamine.

Vigade halduse temaatika kokkuvõtteks saab öelda, et suureks abiks on grupitööd toetava tarkvara olemasolu, mis võimaldab leitud vigade organiseerimist, infovahetust ja protsessi läbipaistvust. Tarkvara aitab juhtimise seisukohalt vaid aimu saada, kui palju aega on vaja vigade parandamiseks eraldada. Vigade halduse lahutamatuks osaks peab olema projektiplaani kohendamine, et leitud vead saaksid ka parandatud.

Pidev kvaliteedikontroll oli projekti käigus tagatud iga iteratsiooni lõpus toimuva aktsepteerimistestidega. Kvaliteedikontrolli osaks on ka kõikide vahedokumentide kliendipoolne aktsepteerimine. Kliendipoolne esindaja pidi alla kirjutama kõikidele üleantavatele tulemitele ja see tagas ka kliendipoolse kvaliteedikontrolli.

4.3. Järeldused – iteratiivne ja inkrementaalne ERP-tarkvara juurutusprojekti metoodika

Rakendusanalüüsi baasil tehtud järeldustest tulenevalt saab anda soovitusi ERP-tarkvara juurutusprojektide metoodika osas. Eelkõige on soovitused mõeldud väikese ja

keskmise mahuga projektidele, kus soovitakse kasutada iteratiivset ja inkrementaalset lähenemist.

4.3.1. Projektorganisatsioon

Projektorganisatsiooni koostamisel on soovitatav lähtuda unifitseeritud protsessi põhimõttest, et projektis oleks sees nii finantseerijad, kasusaajad kui ka lõppkasutajad. Projektorganisatsiooni struktuurina on soovitatav kasutada OnTarget-is kirjeldatud sümmeetrilist organisatsiooni.

Projekti üldküsimustega tegeleb juhtkomitee, kelle pädevuses on eelarve-, aja- ja ressursiküsimused. Juhtkomiteele alluvad teostaja ja tellija poolsed projektijuhid. Teostaja poolne projektijuht vastutab projekti üldise käekäigu, planeerimise ja juhtimise eest. Teostaja poolsele projektijuhile allub arendusmeeskond. Tellijapoolne projektijuht vastutab tellija võtmeisikute kaasamise eest projekti, korraldab infrastruktuuri (kui see on tellijapoolne) ning organiseerib kokkusaamisi. Tellijapoolne projektijuht võib olla kliendi organisatsiooni jaoks väljastpoolt, et projektiga tegelemiseks oleks piisavalt ressurssi.

4.3.2. Projekti üldine planeerimine ja ülesehitus

ERP-tarkvara juurutusprojekti on soovitatav teha iteratiivse ja inkrementaalsena, et maandada arenduse ja juurutuse käigus esile kerkivaid riske. Iteratiivne lähenemine aitab kaasa ka juhtimise läbipaistvusele ning annab kõigile projekti osalistele pidevat tagasisidet projekti edusammude kohta. Projekti ajakava iteratsioonideks tükeldades tuleks arvestada iteratsiooni pikkuseks 2-4 nädalat.

Projekti alguses või eel on vaja teha üks iteratsioon, kus tegeletakse ainult nõuete kirjeldamisega. Esimese iteratsiooni lõpus tuleks teha üldine iteratsioonide plaan ning allkirjastada projekti nõuete spetsifikatsioon ja projekti skoop. Sellega maandatakse nõuete „paisumise” risk kohe projekti alguses. Projektiplaani koostades tuleb silmas pidada ka töökoormust, puhkuseid, riigipühasid ja muid asjaolusid, mis võivad inimeste saadavust piirata.

Järgnevad iteratsioonid peaks olema funktsionaalsuste kaupa (see sõltub juurutatavast ERP-tarkvarast), vajadusel hõlmates väikesemahulisi kohandustöid. Projektiplaani koostamise lähtepunktiks võib kasutada projektiplaani joonisel 4.1.

Eraldi tuleks käsitleda nn arendusiteratsioone, kus ei seadistata olemasolevat paketti, vaid luuakse uut moodulit või tarkvara. Arendusiteratsioonide planeerimisel tuleb olla väga konservatiivne, kuna arenduse puhul on riskid suuremad, kui standardpaketi konfigureerimisel.

ERP-tarkvara tuleb tavaliselt lansseerida „suure paugu" meetodil. Autor soovitab lansseerimise jaoks planeerida eraldi iteratsioon pikkusega vähemalt nädalat, kus tegeletakse ainult lansseerimise ettevalmistuse ja kasutajatoega. See tagab tarkvara kasutuselevõtul tekkivate vigade kiire lahendamise ja ei pane ülejäänud projektiplaani surve alla, kui tarkvara lansseerimisel ootamatusi tekib. Praktika näitab, et lansseerimisel tekib pea alati ootamatusi.

Lansseerimisele võib järgneda iteratsioone, mis tegelevad aruandluse ja muu temaatikaga, mis ei puuduta igapäevatööd. Samuti saab lansseerimise järel arendada juurde mõne mooduli, mida soovitakse hiljem kasutusele võtta. Soovitatav on kõigepealt lansseerida standardpakett ja seejärel järk-järgult lisamoodulid.

Iga iteratsioon peaks sisaldama vähemalt ühte kontrollpunkti analüüsidokumendi kinnitamisel. Vajadusel võib kontrollpunkti panna ka testijuhtumite kirjelduste valmimise järele. Iga iteratsioon peab lõppema kliendipoolsete aktsepteerimistestidega. Iteratsioonis peaks teostatama tulemid, mis on kirjeldatud järgnevas punktis.

4.3.3. Tulemid

ERP-tarkvara juurutusprojektis on tulemite valikul kõige olulisem, et loodavad dokumendid (välja arvatud tehnilised spetsifikatsioonid) oleksid mõlemale poolele arusaadavad ning projekti seisukohalt vajalikud. Soovitatav on teha järgmised tulemid:

- Projektiplaan (sh. ka lansseerimisplaan);
- Nõuete spetsifikatsioonidokument kogu projekti kohta;

- Analüüsi-disainidokument iga funktsionaalse mooduli (iteratsiooni) kohta, mis sisaldab äriprotsesside kirjeldusi, *gap-fit* analüüsi ja lahenduse kirjeldust;
- Kasutusjuhend (vajadusel);
- Testjuhtumite kirjeldused;
- Aktsepteerimistestide protokoll.

Ülalloetud dokumentide loetelu ja sisu võib vastavalt projektile kohandada, kuid põhiolemuselt on kõik dokumendid rohkemal või vähemal määral projekti edukaks lõpetamiseks vajalikud.

4.3.4. Parimad praktikad

Parimate praktikate all mõistetakse siinkohal unifitseeritud protsessis ülesloetud praktikaid, mida on võimalik kohaldada ka ERP-tarkvara juurutusprojektidele. Parimad praktikad on:

- Visuaalne modelleerimine – Visuaalset modelleerimist on mõttekas rakendada vaid juhul, kui projekti osapooled peavad seda kasulikuks. Modelleerimine on soovitatav, kui on tarvilik arendada uut funktsionaalsust või lisamooduleid.
- Muudatuste haldus – Projekti alguses fikseeritud nõuetest kinnipidamist tuleb jälgida kogu projekti vältel. Kui selgub mõne nõude muudatuse vajadus, tuleb see kooskõlastada juhtkomitees ja vajadusel korrigeerida projektiplaani.
- Komponentarhitektuur – ERP-tarkvara arendustööde puhul peab silmas pidama, et kõik arendusmeeskonna liikmed oleksid tuttavad kasutatava tarkvara arhitektuuri ja arenduspõhimõtetega. Arenduse käigus tuleb kontrollida, et arendusmeeskond ka tegelikult arhitektuuri põhimõtteid rakendab.
- Vigade haldus – Vigade haldamiseks on soovitatav kasutada lihtsalt ligipääsetavat veahaldustarkvara. Kliendi meeskonda tuleb koolitada vearaportite kirjutamise osas. Projektiplaanis tuleb arvestada vigade

parandamisele kuluva ressursi- ja ajakuluga, et tagada vigade raporteerimise kõrval ka nende lahendamine.

- Pidev kvaliteedikontroll – Pidev kvaliteedikontroll on tagatud iga iteratsiooni lõpus toimuvate aktsepteerimistestidega, mis annavad avalikku infot tarkvara kvaliteedist. Kõik tulemid on soovitatav lasta kliendi poolt kirjalikult aktsepteerida – see tagab kliendipoolse motivatsiooni dokumendid ka sisuliselt läbi vaadata.

KOKKUVÕTE

Tänapäevane ERP-tarkvara on keeruline valmistoode, mille kasutusele võtmiseks on vaja süstemaatilist lähenemist. Standardsena on saadaval juba valmissüsteemid, mis hõlmavad kogu ettevõtte jaoks tarvilikku ärifunktsionaalsust. Standardsed tarkvarapaketid pakuvad oluliselt madalamaid juurutuskulusid võrreldes sarnase funktsionaalsuse arendamisega uude süsteemi. ERP-tarkvara pakettides on tavaliselt olemas võimalus funktsionaalsust kohendada, et ettevõtted saaksid oma spetsiifikale ka tarkvaralist tuge.

ERP-tarkvara juurutamise edutegurid langevad üldiselt kokku tarkvaraarendusprojektidega. ERP-tarkvara juurutades on tähtsad muuhulgas juhtkonna toetus, lõppkasutajate haaratus, juurutusmeeskonna koosseis ja selged ärieesmärgid. Edutegurite mõjutamiseks on loodud mitmesuguseid tarkvara arendus- ja juurutusmetoodikaid.

Levinumaid tarkvara arendusmetoodikaid maailmas on unifitseeritud protsess. Tegemist on iteratiivse ja inkrementaalse protsessiga, mis kirjeldab parimaid praktikaid tarkvaraprojektide teostamisel. Parimad praktikad on nõuete haldus, komponentarhitektuur, visuaalne modelleerimine, pidev kvaliteedikontroll ja muudatuste haldus. Unifitseeritud protsessi kohaselt jaguneb tarkvaraprojekt faasideks ja iteratsioonideks, mis aitavad projekti kulgu paremini jälgida ja juhtida. Unifitseeritud protsess on pälvinud ka kriitikat, kuna tegemist on raskepärase ja mahuka materjaliga, mida on keeruline rakendada.

Navision OnTarget on koskmudelil baseeruv juurutusprotsess, mis on välja töötatud silmas pidades ERP-tarkvara Microsoft Navision. OnTarget jagab projekti kuueks faasiks: diagnoos, analüüs, disain, arendus ja testimine, juurutusfaas ning kasutajatugi. Autori arvates on OnTarget hästi rakendatav metoodika, kuna selles ei nähta ette erilist kohandamist. Samuti on metoodikaga kaasa on pandud rohkelt erinevaid dokumendimalle, mis teevad metoodika rakendamise lihtsamaks.

Käesolevas töös vaadeldi projekti mobiiltelefonide distributsioonifirma kassasüsteemi juurutus- ja arendustööde näitel. Projekti ülesehituses kasutati iteratiivset lähenemist, mis on omane unifitseeritud protsessile. Autori arvates aitas see hästi riske maandada, kuid projekti alguses oleks pidanud eraldi iteratsiooniga nõuded kirjalikult fikseerima.

Iteratsioonide sisemine ülesehitus oli sarnane – igas iteratsioonis teostati analüüsi- ja disainitööd, arendati ning testiti. Iteratsioon lõpetati aktsepteerimistestidega. Iteratsioonide siseselt oli alati paigas üks vaheetapp – analüüsidokumendi aktsepteerimine, mis fikseeris antud iteratsiooni nõuded. Selline lähenemine ei õigustanud ennast, kuna iteratsiooni käigus võis muutuda selle töömaht ja kestvus.

Projektis kasutati OnTarget-ist pärinevat projektorganisatsiooni ülesehitust, mis koosnes juhtkomiteest, kliendi ja teostaja poolsetest projektijuhtidest, arendusmeeskonnast ja kliendi võtmeisikutest. Selline projektorganisatsioon toimis hästi ning soodustas kommunikatsiooni.

Projekti tulemeid koondati loetavuse huvides ja iteratiivsuse tõttu. Igas iteratsioonis tehti analüüsidokument, testjuhtumid, aktsepteerimistestide protokoll ja kasutusjuhend. Analüüsidokumendi sisu oli inspireeritud OnTarget-ist ning sisaldas endas nõudeid, äriprotsessi, *gap-fit* analüüsi ja pakutava lahenduse disaini. Ülejäänud dokumendid olid vastavad OnTarget metoodikale. Selline tulemite ülesehitus ja iteratsioonide vaheline tükeldamine oli projektile kasulik ning andis väljundina süstematiseeritud dokumentatsiooni.

Vaadeldavas projektis kasutati unifitseeritud protsessis nimetatud parimatest praktikatest põhjalikumalt vaid kahte – vigade haldus ja pidev kvaliteedikontroll. Vigade halduse puhul oli kasuks tarkvara kasutamine vigade raporteerimisel ja haldamisel, kuid raskusi tekitas vigade lahendamisega seotud töö planeerimine ja teostamine. Pidev kvaliteedikontroll tagati projektiplaani ülesehituse ja kõikide dokumentide kirjaliku kliendipoolse aktsepteerimise nõudega.

Keskmise ja väikese suurusega tarkvaraprojektides soovitab autor kasutada inkrementaalset ja iteratiivset lähenemist, kasutades 2-4 nädalasi iteratsioone. Kõige esimeses iteratsioonis peaks tegelema vaid nõuete spetsifikatsiooni koostamise ja

kinnitamisega. Järgnevad iteratsioonid peaksid sisaldama konkreetsete funktsionaalsete moodulite juurutamist ja kohandamist. Iteratsioonide järjestamisel tuleks standardne funktsionaalsus lansseerida võimalikult varakult ning järk-järgult lisanduvad moodulid järgmiste iteratsioonidega lisada.

Projektorganisatsiooni ülesehitusel soovitab autor kasutada OnTarget metoodikas kirjeldatud lähenemist juhtkomitee ja sümmeetrilise projektorganisatsiooniga. See tagab piisava infovahetuse ja kliendi suurema osaluse projektis. Kliendipoolne projektijuht võib olla ka kliendiorganisatsiooni jaoks väljast, et tagada piisava ajaressursi olemasolu projekti vältel.

Autor soovitab projekti käigus luua kogu projekti kohta käiva nõuete spetsifikatsiooni. Seejärel tuleb igas iteratsioonis teha nõuete analüüsidokument, mis sisaldab äriprotsesside analüüsi, *gap-fit* analüüsi ning pakutavaid lahendusi. Lisaks tuleb igas iteratsioonis teha ka testjuhtumite kirjeldused ja aktsepteerimistestide protokoll. Kasutusjuhendi koostamine sõltub projekti vajadustest ning ei pruugi olla alati kohustuslik.

Unifitseeritud protsessis toodud parimate praktikate kohandamine ERP-tarkvara juurutusprojektides on autori arvates kasulik. Visuaalset modelleerimist tuleks kasutada eelkõige keeruliste äriprotsesside kirjeldamisel ja lisamoodulite arendamisel. Projekti käigus tekkinud muudatusettepanekud tuleks lahendada juhtkomitees ning muudatuste aktsepteerimisel tuleb kohendada ka projektiplaani. Arendusmeeskond peab olema tuttav ERP-tarkvara arhitektuuriga ning arendused sellele vastavalt teostama. Vigade halduseks on soovitatav kasutada tarkvaralist lahendust ja vigade lahendamiseks tuleb projektiplaanis aeg planeerida. Pideva kvaliteedikontroll on võimalik tagada, kui kõik dokumendid tuleb kliendil kirjalikult aktsepteerida.

KASUTATUD KIRJANDUS

Boehm, Barry W. – A spiral model of software development and enhancement, IEEE Computer, 21 (5), 61-72, 1988.

Chaos report 2000, The Standish Group International, 2000.

Fowler, M., Kendall, S. UML Distilled. USA: Addison-Wesley, 1997, 183 p.

Hidding, Gezinus J. – Reinventing Methodology: Who Reads It and Why?, Communications of the ACM 40, No. 11, 102-109, 1997.

ISO 9001:2000, International Standards Organization 2000

Jacobson, I., Booch, G., Rumbaugh, J. The Unified Software Development Process. USA: Addison-Wesley, 1999, 463 p.

Kruchten, P. From Waterfall to Iterative Lifecycle – A tough transition for project managers. Rational Software Corporation 2000, 10 p.

Kudrjavets, Gunnar – Microsofti metoodikast mõjutatud tarkvara testimisprotsess. Magistritöö, Tartu Ülikool, 2002.

Littover, M. UML sõnastik 2000. [http://www.cc.ioc.ee/uml/]. 10/2/2002.

Manifesto for Agile Software Development, http://agilemanifesto.org/ 2001 (17.05.2003)

Navision OnTarget 2.0. Microsoft Corporation 2003.

Nah, Fiona F.-H., Lau, Janet L.-S. - Critical factors for successful implementation of enterprise systems. MBC University Press, 2001, Business Process Management, Journal vol. 7 No 3, pp. 285-296

Otsason, Veljo – Ülevaade väledatest tarkvaraarenduse metoodikatest. Bakalaureusetöö, Tartu Ülikool, 2003

Rational Unified Process, Version 2001A.04.00.

Royce, W. Software Project Management: A Unified Framework USA: Addisson Wesley Longman, 1998, 406 p.

SOFTWARE DEVELOPMENT PROCESS IN ERP-SOFTWARE IMPLEMENTATION PROJECTS

Kaspar Loog

Master thesis

SUMMARY

Enterprise resource planning (ERP) software implementation project plays a big role in overall software-related projects in the whole world. However, ERP implementation projects have the same flaws as software development projects that are started from scratch. Slipping schedules, growing budgets and reduced functionality is a common denominator for both software development and ERP implementation projects.

Improving software development using various quality benchmarks and processes has gained huge interest during the last years. As a result, several methodologies or processes have been brought out. The dominant view on software development process implies using incremental and iterative development processes. ERP-software cannot boast with the available options in implementation methodologies – at best the package includes an in-house implementation process which is available only to certified partners.

The objective of the thesis is to propose a new approach for ERP-software implementation and customization projects using Microsoft Navision as an example. The theoretical standpoint for the process will be Navision OnTarget implementation process and Rational Unified Process. The practical evaluation will be based on real-world Navision implementation project, which is a mixture of both processes that have been used. The approach proposed can be applied as a starting point for an implementation process for an ERP-software project or implementation organization.

Modern ERP software is a complex package which requires a systematic approach to implementation. Systems incorporating functions that most companies require are

widely available. Standard packages offer also lower implementation costs compared to development that are started from scratch. ERP packages usually include the possibility to customize the functionality, so that the companies can support their specific functions in software.

The success criteria for an average software development project and ERP implementation project are widely the same. Having a successful ERP package implementation requires good top-level management support, involvement of end-users, selection of implementation consultants and clear business objectives. Several implementation and software development processes have been created to improve these success criteria in a systematic way.

Unified process is one of the best known methodologies among the software development community. The methodology praises an iterative and incremental approach to projects and describes best practices for software development. The best practices are namely requirements management, component architecture, visual modeling, continuous quality control, and change management. Unified process divides the software project into phases and iterations, which help monitor and manage software projects better. While having wide recognition throughout the world, unified process has gained criticism for being too heavyweight and difficult to implement.

Navision OnTarget is a waterfall-based software implementation process, specially created having Microsoft Navision in mind. OnTarget divides the software project into six phases: diagnosis, analysis, design, implementation and testing, deployment and on-going support. The author perceives OnTarget simple to implement, because it is intended to be used straight from the book– no customization is needed. The package also includes plenty of templates, which further simplifies the use of the methodology.

The thesis has an analysis of an ERP software implementation project for a mobile phone retail company in Estonia. Iterative and incremental approach was used to manage the project. The author discovered that applying this reduces risks as the methodology suggests. However, a separate iteration for requirements specification would have been useful in the beginning of the project.

The internal structure of the iterations was consistent – every iteration included analysis and design, implementation and testing activities were concluded by acceptance tests done by the client. Acceptance of the analysis document worked as a major milestone in every iteration. However, this approach did not work because the document included requirements for the iteration at hand. Prior to the beginning of the iteration, it was not possible to estimate the timeframe and amount of work.

The project in focus used project organization structure from OnTarget. The organization consisted of steering committee, client and implementation project manager, development team and the client's key people. The approach justified itself because of good communication between team members.

The artifacts created during the projects were grouped by iterations to ensure readability. Every iteration resulted in the creation of an analysis document, test cases, protocol of acceptance tests and a user manual. The analysis document was inspired by Navision OnTarget and contained requirements, business process descriptions, gap-fit analysis and the design of a proposed solution. The rest of the documents corresponded to OnTarget methodology. Iteration-based approach to creating artifacts worked well for the project and resulted in well structured documentation.

Two best practices from the unified process were used in the project - issue management and continuous quality control. Special supporting software was used to track issues and this helped to manage the project much better. However, there were problems with planning and doing the work to fix unresolved issues. Continuous quality control was ensured by the project plan, acceptance testing and signing of all documents by the client.

The author recommends using iterative and incremental approach with 2-4 week iterations in mid-sized ERP software implementation projects. The first iteration should only deal with specifying and signing off the requirements. Next iterations should include implementation and customization of the software module by module. Standard functionality should be first configured and deployed, later adding customized modules.

Symmetric project organization described in Navision OnTarget suits well with mid-sized ERP projects. This ensures good flow of information and greater involvement of the client. A special client project manager position should be established and filled with a person outside of the organization. This approach guarantees the time that the project requires by the client project manager.

The author recommends creating a single specification document for the whole project. In all iterations a separate analysis document should be created, which includes business process descriptions, gap-fit analysis and proposed solutions. All iterations should include test-case descriptions and acceptance test protocols. A user manual might not be needed in all projects.

According to the author the use of best practices described in the unified process pays off. Visual modeling should be used in understanding business processes and developing add-on modules. Change requests should be discussed and resolved within the project steering committee. When a change is approved by the steering committee, the project plan should be adjusted accordingly. The development team must be acquainted with the architecture of ERP software and do all customization work according to the specified architecture. Special issue tracking software is recommended to track bugs. Time should be allocated in the project plan to fix bugs and deal with issues. Continuous quality control is ensured when all documents are signed by the client.