

TARTU ÜLIKOOL
MATEMAATIKA-INFORMAATIKATEADUSKOND
Arvutiteaduse instituut
Tarkvarasüsteemide õppetool

Asko Seeba

**Unifitseeritud tarkvaraarendamise protsess ja selle
rakendamise juhtumianalüüs**

Magistritöö

Juhendaja: Anne Villems

Autor:“...” mai 2001
Juhendaja:”...” mai 2001
Õppetooli juhataja:”...” 2001

Tartu 2001

Sisukord

| | |
|---|----|
| Sisukord | 2 |
| 1 Sissejuhatus | 4 |
| 1.1 Proloog | 4 |
| 1.2 Käesolev väitekiri..... | 4 |
| 2 Unifitseeritud tarkvaraarenduse protsess | 6 |
| 2.1 Tarkvaratootmise kriis..... | 6 |
| 2.1.1 Tarkvaratootmise kriisi mõiste..... | 6 |
| 2.1.2 Koskmudel | 7 |
| 2.2 Tarkvaraarenduse parim tööpraktika..... | 11 |
| 2.2.1 Arenda tarkvara iteratiivselt..... | 13 |
| 2.2.2 Halda nõudmisi | 16 |
| 2.2.3 Kasuta komponendipõhiseid arhitektuure | 20 |
| 2.2.4 Modelleeri tarkvara visuaalselt | 21 |
| 2.2.5 Kontrolli tarkvara kvaliteeti | 24 |
| 2.2.6 Juhi tarkvara muudatusi..... | 25 |
| 2.3 Unifitseeritud protsess..... | 26 |
| 2.3.1 Unifitseeritud protsess lühidalt | 26 |
| 2.3.2 Unifitseeritud protsess on kasutusmalljuhitav | 27 |
| 2.3.3 Unifitseeritud protsess on arhitektuurikeskne..... | 28 |
| 2.3.4 Unifitseeritud protsess on iteratiivne ja inkrementaalne | 30 |
| 2.3.5 Elu unifitseeritud protsessi järgi..... | 31 |
| 2.3.6 Toode | 32 |
| 2.3.7 Tsüklite faasid..... | 34 |
| 3 Juhtumianalüüs esimestest katsetustest Cybernetica AS-is | 38 |
| 3.1 Protsess enne unifitseeritud protsessi..... | 38 |
| 3.2 Unifitseeritud protsessini jõudmine | 40 |
| 3.3 Projektid: TrueSign 1.0 ja TrueSign 1.1..... | 41 |
| 3.3.1 Saateks – lühidalt avaliku võtme infrastruktuurist..... | 41 |
| 3.3.2 Projekti valimine | 43 |
| 3.3.3 Protsessi rakendamine TrueSign 1.0 arendusprojektil – õppetunnid .. | 44 |
| 3.3.4 TrueSign 1.1 arendusprojekt..... | 53 |

| | | |
|-------|---|----|
| 4 | Tarkvaraarenduse protsessi rakendamine kogu organisatsioonis..... | 57 |
| 4.1 | Protsessi muutuse mõju | 57 |
| 4.2 | Protsessi arendamine ja kujutamine CMM-is..... | 58 |
| 4.2.1 | Ülevaade CMM-ist | 58 |
| 4.2.2 | CMM-i puudused | 60 |
| 4.2.3 | Pragmaatiliselt protsessi parandamisest | 61 |
| 4.2.4 | Protsessi küsimustik | 62 |
| 4.3 | Cybernetica ASi püüdlused | 72 |
| | Sõnaseletustik..... | 74 |
| | Kasutatud kirjandus | 80 |
| | Unified Software Development Process and a Case Study of It's Application | 82 |

1 Sissejuhatus

1.1 Proloog

Tarkvaraspetsialistide jaoks on hea uudis see, et igasugused majandusharud järjest rohkem sõltuvad tarkvarast. Halb uudis on see, et nende süsteemide muutumine suuruse, kompleksuse, hajutatuse ja tähtsuse suunas sunnivad järjest rohkem laiendama meie, tarkvaratööstuse spetsialistide, teadmisi selle kohta, kuidas neid süsteeme välja töötada. Püüd parendada aegunud süsteeme moodsa tehnoloogia suunas toob endaga kaasa omad probleemid. Teisiti öeldes, oleme olukorras, kus ärimaailm eeldab tarkvaratooteid kasutades üha suuremat produktiivsust ja paremat kvaliteeti, samal ajal oodates uute tarkvaratoodete kiiremat väljatöötamist ja juurutamist. Probleemi teravdab asjaolu, et kvalifitseeritud arenduspersonali juurdekasv ei pea nõudlusega sammu.

Tulemus on see, et tarkvara ehitamine ja hooldamine on raske ja muutub järjest raskemaks. Kvaliteetse tarkvara ehitamine korratavalt ja ennustatavalt on veelgi raskem... [Kruchten, 1999]

Kirjeldatud probleemist ei ole puutumata jäänud ka Cybernetica AS, kus käesoleva väitekirja autor töötab tarkvaraarendusjuhina infoturbe tooteid välja töötavas infoturbeosakonnas.

1.2 Käesolev väitekirj

Käesolev väitekirj tutvustab unifikseeritud tarkvaraarenduse arendusprotsessi ja selle rakendamist. Autor kirjeldab ka juhtumianalüüsi (*case study*) enda isiklikust kogemusest töötades Cybernetica AS-is. Väiteki on jagatud kolmeks loogiliseks osaks:

2. peatükk kirjeldab unifikseeritud protsessi ja seda, miks ta hea on.
3. peatükk kirjeldab juhtumianalüüsi esimestest katsetest rakendada seda protsessi Cybernetica AS-is paari autori juhitud projekti peal.
4. peatükk räägib sellest, kuidas organisatsiooni tasemel arendusprotsessi juhtida, kirjeldades üht skaalat, mida saab kasutada organisatsiooni taseme määramiseks, ning

seda, kuidas nüüd üritatakse Cybernetica AS-is tarkvaraarendusprotsessi organisatsiooni tasemel kontrolli alla saada.

2 Unifitseeritud tarkvaraarenduse protsess

Unifitseeritud tarkvaraarenduse protsessi kirjeldav teooria on üheksakümnendatel aastatel iteratiivse ja objektorienteeritud tarkvaraarenduse ideoloogiast välja kujunenud. Selle protsessi põhieesmärk on võime adapteeruda võimalikult paljudesse tarkvaratootmise olukordadesse ja tulla toime tänapäevani tarkvaratootjatele muret valmistava nn. “tarkvaratootmise kriisiga”¹, mille suhtes muuhulgas ka klassikaline koskmudel aidata pole suutnud.

2.1 Tarkvaratootmise kriis

Paragrahv tarkvaratootmise kriisist on lühendatud kokkuvõtte raamatu [Royce, 1998] esimesest peatükist, kuna see on seni ainus autori loetud allikas, mis tarkvaratootmise ajalugu unifitseeritud protsessi valguses pikemalt analüüsib.

2.1.1 Tarkvaratootmise kriisi mõiste

Tarkvara parim omadus on selle paindlikkus: seda saab programmeerida tegema ükskõik mida. Tarkvara halvim omadus on selle paindlikkus – omadus ükskõik mida on teinud tarkvaraarenduse raskesti planeeritavaks, jälgitavaks ja juhitavaks. 90-ndate keskel tehtud põhjalikumad analüüsid tarkvaraarenduse hetkeseisust jõudsid Walker Royce’i andmetel kõik samadele üldistele järeldustele (ta toob oma raamatu ühes lisis ära ka põhjaliku ülevaate kolmest olulisemast analüüsist):

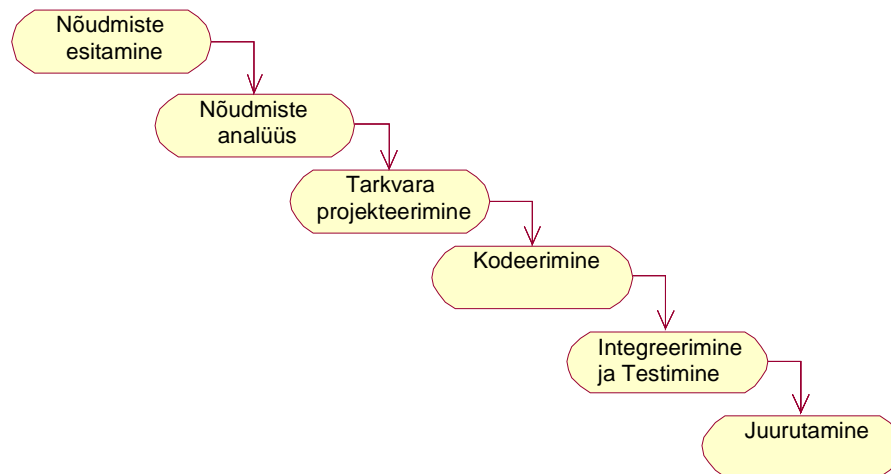
- Projektide edenemine on väga ennustamatu – ainult 10% projektidest valmib tähtajaks ja ei ületa eelarve piire.
- See, et mingi projekti juhtimise teooria aitab graafikus ja eelarves püsida, on pigem juhus, kui reegel.
- Tarkvara ümbertegemiste maht ja viis on teiste valdkondadega võrreldes omane ainult ebaküpsele protsessile.

Selline viimased 30 aastat kestnud ennustamatus ongi see, millele vihjatakse, kui räägitakse “tarkvaratootmise kriisist”.

¹ Tarkvaratootmise kriis – seda väljendit kasutatakse tarkvara tootmise raskesti planeeritavuse, jälgitavuse ja juhitavuse omadusele viitamiseks

2.1.2 Koskmudel

Käesoleva väitekirja lugejalt eeldatakse tarkvaraarenduse koskmudeli (vt. Joonis 1) üldiste põhimõtete teadmist. Hoolimata paljude tarkvaraekspertide soovitudest ja koskmudeli taga olevast teooriast, praktiseerivad paljud projektid sellist klassikalist tarkvaraarenduse juhtimist.

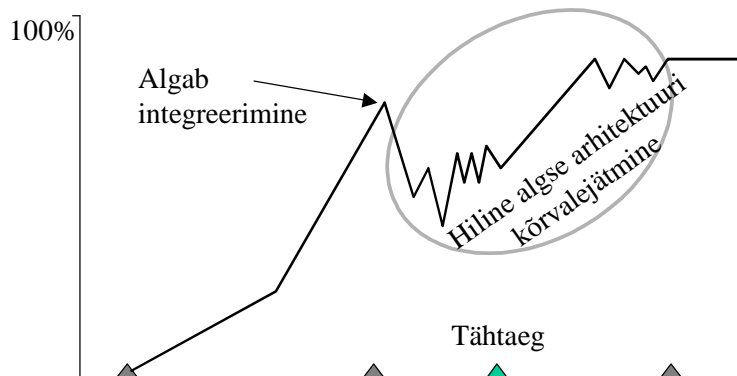


Joonis 1. Koskmudel

On kasulik pöörata tähelepanu klassikalise arendusprotsessi omadustele sellisena, nagu seda tüüpiliselt on rakendatud (mis kusjuures ei pruugi olla see, mida kavatseti). Projektid, mis hätta on sattunud, on tihti näidanud järgmisi sümptomeid:

- **Pikaleveniv eelnevalt kodeeritud süsteemiosade integreerimine ning selle käigus ja selle tõttu hiline algse arhitektuuri kõrvalejätmine.** Joonis 2 illustreerib koskprotsessiga tüüpilist arendusprojekti. Arengut kujutatakse vertikaalteljel protsentides valminud programmi koodist. Tüüpiline on järgmine sündmuste käik:
 1. Ajal, mil projekteeritakse ja koostatakse põhjalikke (sageli liigagi põhjalikke) juhendeid, kulgeb projekt edukalt.
 2. Koodi kirjutatakse elutsükli hilisemas etapis.
 3. Ettenägematute teostusküsimuste ja liideste mitmetimõistetavuste tõttu rängad probleemid integreerimisel.

4. Tuntakse rasket eelarve ja ajagraafiku survet pikaleveniva integreerimise ja testimise ajal.
5. Toimuvad hilised aja puuduse tõttu ilma projekteerimiseta ebaoptimaalsed parandused.
6. Hapra ja raskesti hallatava toote hilinenud tarnimine.



Joonis 2. Klassikalise arendusprotsessi tavaline areng

- **Liiga hiline riskide lahendamine.** Tõsine koskmudeliga seotud probleem on see, et puudub varajane riskide lahendamine. See tuleb sellest, et koskmudeli korral on tarkvara elutsükli alguses fookus pabertehistel (*paper artifacts*), kus tegelikud projekteerimise, teostuse ja integreerimise riskid on veel suhteliselt raskesti aiatavad.
- **Nõudmistest juhitud süsteemi funktsionaalne dekomponeerimine.** Tarkvaraarendusprotsessi on traditsiooniliselt juhtinud nõudmised: üritatakse saada täpseid nõudmiste definitsioone ja siis rahuldada täpselt need nõudmised. Selline tähendab enne teiste tarkvaraarendustegevuste alustamist nõuete täielikku ja üheselmõistetavat spetsifitseerimist. Kergeusklikult käsitletakse kõiki nõudmisi kui võrdseid ja loodetakse, et need nõudmised jäävad kogu tarkvaraarenduse elutsükli ajaks muutumatuteks. Reaalses maailmas esinevad sellised tingimused harva. Nõudmiste spetsifitseerimine on

tarkvaraarendusprotsessis raske, aga tähtis osa. Kõikide nõuete võrdne käsitlemine juhib suure osa tarkvara väljatöötamisele kuluvatest panustest eemale tegeliku elu jaoks tähtsatest nõudmistest, raisates seda jälgitavusega, testitavusega, logistilise toega jne. seotud paberimajanduse peale, mis hiljem, kui arusaamine olulistest nõudmistest ja arhitektuurist areneb, paratamatult kõrvale jäetakse.

Teine konventsionaalse lähenemise omadus on see, et nõudmisi spetsifitseeritakse funktsionaalsetena. Klassikalise koskmudeli protsessi sisse on ehitatud fundamentaalne eeldus, et tarkvara dekomponeerub funktsioonideks, ja nii omistatigi nõudmised vastavatele komponentidele. Selline dekomponeerimine erineb väga tihti objektorienteeritud ja komponendipõhise arhitektuuri dekomponeerimisest. Funktsionaalne dekompositsioon kinnistus ka lepingutesse, alltöölepingutesse ja **töö liigendstruktuuridesse** (*work breakdown structure*), tehes tihti juba ette võimatuks juhtida projekti arhitektuurikeskselt..

- **Osapooltevahelised vääritimõistmised.** Konventsionaalses protsessis on nõuete spetsifitseerimise raskuste ja tehnilise informatsiooni vastavas vormis kirjelduse ainult paberdokumentidena vahetamise tõttu **osanikel** (*stakeholders*; isik, keda süsteemi väljund materiaalselt huvitab) kalduvus teineteisest pidevalt valesti aru saada. Täpse sümboolika puudumise tulemusel on läbivaatused subjektiivsed ja informatsiooni vahetamine valikuline. Tüüpiline lepingulise tarkvaraarenduse sündmuste ahel on klassikalises protsessis järgmine:

1. Lepingu täitja valmistab ette tarnelepingu dokumendi ja annab selle kliendile kinnitamiseks.
2. Kliendilt saab lepingu täitja tagasi kommentaarid (tüüpiliselt 15 kuni 30 päeva jooksul).
3. Lepingu täitja liidab need kommentaarid lepingusse ja esitab (tüüpiliselt 15 kuni 30 päeva jooksul) lepingu viimase versiooni kinnitamiseks.

Selline ühekordse läbivaatusega paberite vahetamise protsess on tarkvaraarenduse tegelike eesmärkide saavutamiseks äärmiselt väikese

kasuteguriga, et mitte öelda kahjulik. Selline lähenemine on pidevalt põhjustanud kliendi ja tellimuse täitja vahelise suhte langemist vastastikkuse mitteusaldamiseni, mis samuti raskendab kalenderplaani ja hinna parema tasakaalu saavutamist.

- **Liigne kontsentreeritus dokumentatsioonil ja koosolekutel.** Konventsionaalne protsess keskendub erinevate tarkvaratoodet kirjeldada üritavate dokumentide produtseerimisele ilma piisava fookuseta toote enda käegakatsutaval juurdekasvul. Peamised **tähtpunktid** (tähtpunkt – mingi teatava projekti etapi formaalse läbimise punkt) seisnesid tüüpiliselt tseremoniaalsetes koosolekutes, kus käsitleti üksnes teatavaid dokumente. Lepingutäitjad produtseerisid sõna otseses mõttes tonnide viisi paberit, et läbida korrektselt tähtpunkte ja demonstreerida osanikele progressi selle asemel, et kulutada oma energiat tegevustele, mis vähendaksid riske ja produtseeriks kvaliteetset tarkvara. Tüüpiliselt vaatasid esitajad ja auditoorium läbi pigem lihtsaid asju, millest nad aru said, kui kompleksseid, aga tähtsaid asju. Seetõttu enamik tarkvara arhitektuuri läbivaatusi olid tehniliselt vähe kasulikud. Samas oli nende ettevalmistamine ja juhtimine kulukas. Sellised läbivaatused esitavad pelgalt arengu fassaadi. Tabel 1 annab ülevaate tüüpilise arhitektuuri läbivaatuse tulemustest.

| Näiline tulemus | | | Tegelik tulemus |
|---|---------------|----------------|---|
| Põhjalik ülevaade | mitmekesisele | auditooriumile | Ainult väike protsent auditooriumist saab tarkvarast aru. Ülevaated ja dokumendid toovad esile ainult mõned tähtsatest tarkvarasüsteemi omadustest ja riskidest. |
| Tarkvara arhitektuur, mis paistab nõuetele vastav | | | Nõuetele vastavuse kohta ei eksisteeri ühtki käegakatsutavat tõestust. Vastavus üheseltmõistetavatele nõuetele omab vähe väärtust. |
| Nõuete kaetus (tüüpiliselt sajad) | | | Vähesed (kümned) juhivad arhitektuuri. Kõigi nõudmistega tegelemine hajutab |

| | |
|---|---|
| | tähelepanu kriitilistelt nõudmistelt eemale. |
| Arhitektuur on "süütu, kuni pole tõestatud vastupidist" | Arhitektuur on alati süüdi. Arhitektuuri puudused ilmnevad elutsükli hilises staadiumis. |

Tabel 1. Konventsionaalse tarkvaraprojekti läbivaatuse tulemused

2.2 Tarkvaraarenduse parim tööpraktika

Erinevad tarkvaraprojektid ebaõnnestuvad erinevatel viisidel ja kahjuks on neid ebaõnnestuvad projekte liigagi palju. Õnneks on aga võimalik tuvastada sellistele projektidele iseloomulikke peamisi ebaõnnestumise sümptome [Kruchten, 1999]:

- Ebatäpne arusaamine lõppkasutajate vajadustest.
- Võimetus ohjata muutuvaid nõudmisi.
- Moodulid, mis ei sobi kokku.
- Tarkvara, mida on raske hooldada või laiendada.
- Tõsiste projekti vigade hiline avastamine.
- Tarkvara kehv kvaliteet.
- Vastuvõetamatult ebaefektiivne tarkvara töö.
- Meeskonnaliikmed, igapäev omal moel, teevad võimatuks tagantjärgi kindlaks teha, kes muutis mida, millal, kus ja miks.
- Ebakindel redaktsioonide kaupa väljatöötamise protsess (*build-and-release process*).

Kahjuks nende sümptomite kõrvaldamine ei kõrvalda veel haigust. Näiteks hiline tõsiste projekti vigade avastamine on ainult suuremate probleemide, nimelt subjektiivsete projekti seisu hindamiste ja projekti nõuetes, arhitektuuris ja teostuses avastamata kooskõlaprobleemide sümptom.

Ehkki erinevad projektid ebaõnnestuvad erineval moel, ilmneb, et enamus neist ebaõnnestuvad mingi järgmiste algpõhjuste kombinatsiooni tõttu [Kruchten, 1999]:

- Nõuete haldus on juhuslik.
- Suhtlemine on ebaselge ja ebatäpne.

- Arhitektuur on ebakindel.
- Komplekssus käib üle jõu.
- Nõuetes, arhitektuuris ja teostuses on avastamata kooskõlaprobleemid.
- Testimine on ebapiisav.
- Projekti seisu hindamine on subjektiivne.
- Riskide lahendamine on pealiskaudne.
- Muudatuste levitamine on juhtimata.
- Automatiseeritus on ebapiisav.

Kui nende algpõhjustega tegelda, ei kõrvalda me ainult sümptomeid, vaid satume ka kvaliteetse tarkvara väljatöötamises ja hooldamises korrataval ja ennustataval moel palju paremale positsioonile.

Tarkvaraarenduse parim tööpraktika koosneb tarkvaraarenduse võtetest, mis kommertsmaailmas koos kasutatuna on tõestatud tarkvaraarenduse probleemide algpõhjused kõrvaldanud. Neid võtteid kutsutakse “parimaks praktikaks” (*best practices*) mitte niivõrd sellepärast, et nende väärtust kvantitatiivselt täpselt mõõta saaks, kui pigem sellepärast, et need on uuringute põhjal tarkvaratööstuse edukates organisatsioonides enim kasutatud. See parim praktika koosneb järgmistest võtetest [Kruchten, 1999]:

1. Arenda tarkvara iteratiivselt.
2. Halda nõudmisi.
3. Kasuta komponendipõhiseid arhitektuure.
4. Modelleeri tarkvara visuaalselt.
5. Kontrolli tarkvara kvaliteeti.
6. Juhi tarkvara muudatusi.

Järgnevas toodud parima tööpraktika kirjeldused on tõlked vastavatest [RUP, 2000] kirjeldustest.

2.2.1 Arenda tarkvara iteratiivselt

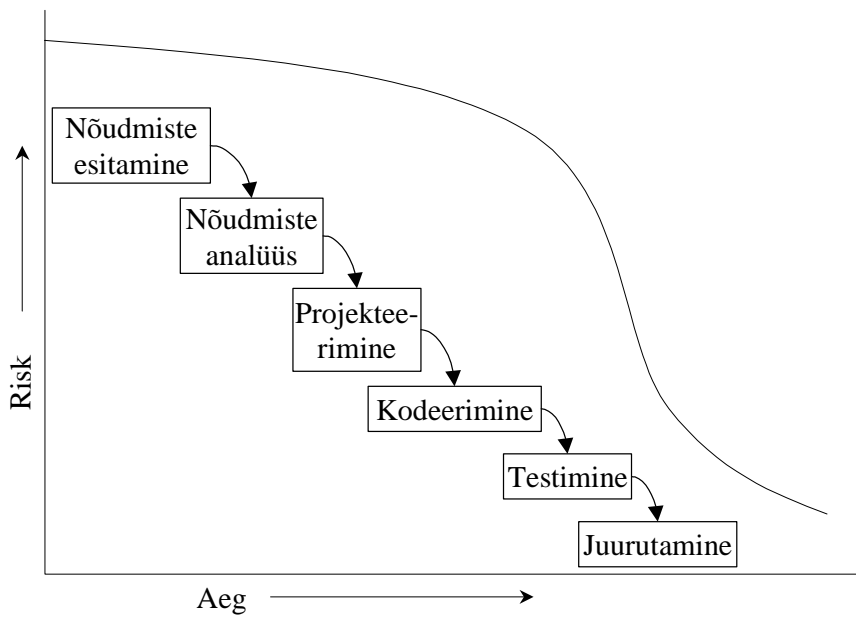
2.2.1.1 Mis on iteratiivne arendus?

Iteratiivset arendust kasutavas projektis koosneb tarkvara elutsükkel mitmest iteratsioonist. Iteratsioonid koosnevad vabas järjekorras erinevates proportsioonides talitluse modelleerimise, nõuete, analüüsi ja projekteerimise, teostamise, testimise ja evituse tegevuste hulgast, kus proportsioonid sõltuvad sellest, kus vastav iteratsioon arendustsükklis asub. Algatus- ja detailimisfaasi iteratsioonid keskenduvad juhtimise, nõuete ja projekteerimise tegevustele; konstrueerimisfaasi iteratsioonid keskenduvad projekteerimisel, teostamisel ja testimisel ning siirdefaasi iteratsioonid keskenduvad testimisel ja evitusel.

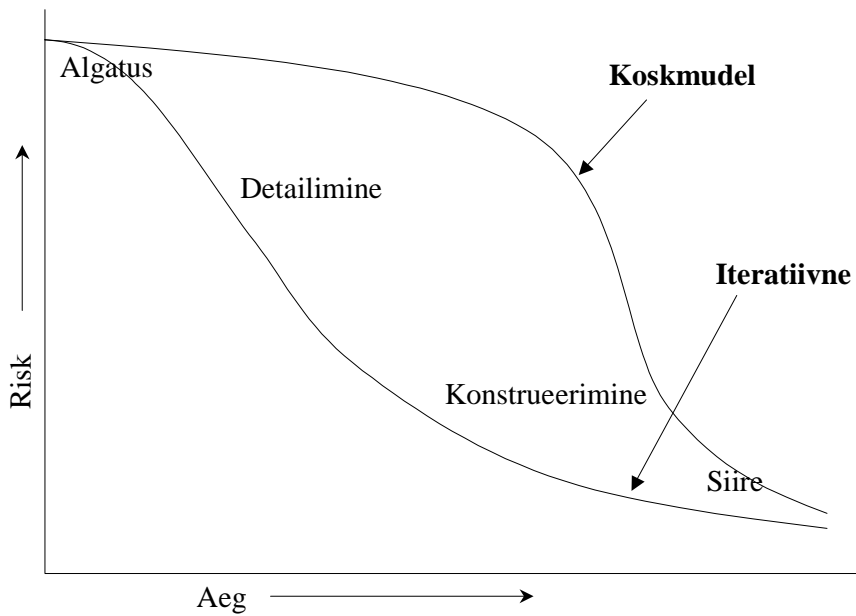
2.2.1.2 Miks arendada iteratiivselt?

Algne arhitektuur ei vasta suure tõenäosusega talle esitatud nõuetele. Hiline arhitektuuri defektide avastamine põhjustab kulusid, mis ületavad eelarvet, mõnel juhul põhjustavad isegi projekti katkestamist.

Kõikide projektidega on seotud riskid. Mida varem me suudame tarkvara elutsükli ajal veenduda, et oleme kõrvaldanud riski, seda täpsemalt me saame teha plaane. Paljud riskid ei tule isegi päevavalgele enne, kui me ei ole üritanud süsteemi integreerida. Me ei ole kunagi võimelised ennustama kõiki riske hoolimata sellest, kui kogenud arendusmeeskond meil on.



Joonis 3. Koskmudeli riskitase on kuni viimaste sammudeni väga kõrge.



Joonis 4. Iteratiivses elutsükklis saadakse valida, millist inkrementi iteratsioonis arendada, toetudes põhiriskide loetelule. Kuna iteratsioon produtseerib testitud demonstreeritava valiku funktsioonidest ja omadustest, saadakse kontrollida, kas ollakse kõrvaldanud riskid, mida selles iteratsioonis kõrvaldada kavatseti, või mitte.

2.2.1.3 Kasu iteratiivsest lähenemisest

Iteratiivne protsess on koskprotsessist üldiselt parem mitmel erineval põhjusel.

- **Iteratiivses protsessis on nõudmiste muutumine vastuvõetav.** Reaalsus on, et normaalselt nõudmised muutuvad. Nõudmiste muutumine on alati olnud projekti hädade üks peamisi allikaid, põhjustades ajagraafikus mittekinnipidamist, hilinenud tarnimist, rahulolematuid kliente ja arendusmeeskonna pettumust.
- **Elemente integreeritakse kehtvalt** – integreerimine ei ole üks “suur pauk” projekti lõpus. Iteratiivses protsessis integreeritakse peaaegu pidevalt. See, mida ollakse harjunud nägema pika, ebakindla ja kannatusterohke projekti etapina, võttes kuni 40% kogu tööpanustest projekti lõpuks, on nüüd jagatud kuueks kuni üheksaks väiksemaks integreerimiseks, mis algavad vähema arvu elementidega.
- **Riske kõrvaldatakse varem**, kuna integreerimine on üldiselt ainus aeg, kui riske avastatakse ja nendele tähelepanu juhitakse. Käies läbi varajasi iteratsioone, käiakse läbi kõik **põhivood** (*core workflows*) vaadates läbi mitmeid projekti aspekte: vahendid, inimeste kogemused jne. Mingid algselt tajutud riskid võivad osutuda üldsegi mitte riskideks ja uued ootamatud riskid võivad nähtavale tulla.
- **See võimaldab organisatsioonil õppida ja paremaks muutuda.** Meeskonna liikmed saavad kogu aeg õppida ning kogu elutsükli jooksul on erinev kompetents ja spetsialiteet rohkem rakendatud. Testijad hakkavad varem testima, tehnilised kirjutajad hakkavad varem kirjutama jne. Mitteiteratiivses arenduses peaksid samad inimesed ootama enne tööle asumist. Koolitusvajadused või vajadus (koguni välise) abi järgi avastatakse varakult hinnangute läbivaatustel.
- **See hõlbustab korduvkasutust**, kuna üldiste osade tuvastamine on kergem, kui nad on osaliselt projekteeritud või teostatud. Korduvkasutatavate osade tuvastamine ja väljatöötamine on raske. Varajaste iteratsioonide arhitektuuriläbivaatused võimaldavad tarkvaraarhitektidel tuvastada ootamatuid korduvkasutuse võimalusi ja töötada välja küps üldine kood järgnevatel iteratsioonides.

- **Selle tulemusena valmiv toode on tugevam**, kuna vigu parandatakse mitmes iteratsioonis. Vigu avastatakse isegi juba esimestes iteratsioonides peale algatusfaasi. Suutlikkuse pudelikaelad avastatakse tarnimise eelõhtu asemel ajal, mil nendega on veel võimalik tegeleda.
- **See muudab vastuvõetavaks tootega seotud taktikalised muutused.** Näiteks olemasoleva tootega konkureerimisel saab võtta vastu otsuse lasta toode välja vähendatud funktsionaalsusega varem, et arvestada konkurendi astutava sammuga, või on võimalik ühildada toode veel ühe antud tehnoloogia tootjaga.
- **Protsessi ennast on võimalik pidevalt parendada ja edasi arendada.** Iga iteratsiooni lõpus toimuv hindamine ei vaata mitte ainult projekti staatust toote ja kalenderplaani seisukohalt, vaid ka analüüsib, mida peaks muutma organisatsioonis ja protsessis endas, et tõsta selle suutlikust järgmises iteratsioonis.

2.2.2 Halda nõudmisi

2.2.2.1 Mis on nõudehaldus?

Nõudehaldus on süsteemile esitatavate muutuvate nõuete leidmise, dokumenteerimise, organiseerimise ja jälgimise süstemaatiline meetod.

Me defineerime nõudmise, kui tingimuse või suutlikuse, millele süsteem peab vastama.

Nõudehalduse me defineerime formaalselt kui süstemaatilise meetodi

- süsteemile esitatavate nõuete tuvastamiseks, organiseerimiseks ja dokumenteerimiseks ning
- kliendi ja projekti meeskonna vahelise süsteemile esitatavate nõudmiste muutumise korras kokku leppimiseks ja selle leppe ülal pidamiseks.

Nõuete kogumine võib tunduda selge ja lihtne ülesanne. Tegelikes projektides aga põrgatakse kokku raskustega:

- Nõuded ei ole alati ilmsed ja võivad olla pärit erinevatest allikatest.
- Nõudeid ei ole alati kerge selgetes sõnades väljendada.

- Nõuetel on palju erineva detailsuse astmega tüüpe.
- Kui nõuete hulka ei juhita, siis see võib muutuda juhitamatuks.
- Nõuded on seotud omavahel ja ka teiste tarkvara väljatöötamise protsessi saadustega.
- Nõuetel on unikaalsed omadused või omaduse väärtused. Näiteks ei ole nad võrdselt tähtsad või võrdselt kergesti käsitletavad.
- Nõuded muutuvad.

Nende raskustega toime tulemiseks on vaja järgmisi oskusi:

- Probleemi analüüsimine
- **Osaniku tarvetest** aru saamine
- Süsteemi defineerimine
- Projekti ulatuse juhtimine
- Süsteemi definitsiooni täpsustamine
- Muutuvate nõuete haldamine

2.2.2.2 Probleemi analüüsimine

Meeldetuletuseks mainin, et **osanik** (*stakeholder*) on isik, keda süsteemi väljund materiaalselt huvitab. **Osaniku tarve** (*stakeholder need*) on talitluslik või operatsiooniline probleem (väljavaade), mida tuleb rahuldada, et õigustada süsteemi ostmist või kasutamist.

Probleemi analüüsi on vaja probleemist ja **osanike** esialgsetest **tarvetest** aru saamiseks ning abstraktsel tasemel lahenduse välja pakkumiseks. See on arutlemine ja analüüsimine “probleemi taga probleemi” leidmiseks. Probleemi analüüsi jooksul jõutakse kokkuleppele tegelikes probleemides ja selles, kes on **osanikud**. Vaja on tuvastada, mida süsteemi kasutatav talitus (*business*) saavutada üritab ja milliseid kitsendusi ta süsteemile seab. Et oleks ka selge arusaamine, millist sissetulekut on süsteemi ehitamisele tehtavast investeringust oodata, tuleks analüüsida ka projekti äri malli (*business case*).

2.2.2.3 Osaniku tarvetest aru saamine

Nõuded tulevad erinevatest allikatest, näiteks klientidelt, partneritelt, lõppkasutajatelt ja valdkonna spetsialistidelt. Peab teadma, kuidas kõige paremini määrata, millised allikad kasutusele tulevad, kuidas nendega kontakti saada ja kuidas nendelt kõige paremini informatsiooni kätte saada. Indiviidid, kes on sellise info esmased allikad, on **osanikud**. Kui töötatakse välja enda firma sees kasutamiseks infosüsteemi, võivad arendusmeeskonna hulka kuuluda lõppkasutaja kogemustega inimesed ja talitusvaldkonna spetsialistid. Tihti alustatakse uurimist pigem talitluse mudeli (*business model*) tasemelt, kui süsteemi tasemelt. Kui arendatakse turul müüdatavat toodet, võidakse turul asuvate klientide vajadustest aru saamiseks rakendada turundusnimesi.

Tuvastamistegevuste käigus võidakse kasutada võtteid nagu intervjuud, ajurünnakud, kontseptuaalne prototüüpimine, küsimustik ja konkurentsianalüüs. Tuvastamise tulemuseks on loetelu nõuetest või vajadustest koos tekstilise ja graafilise kirjeldusega ning üksteise suhtes näidatud prioriteetidega.

2.2.2.4 Süsteemi defineerimine

Süsteemi defineerimine tähendab osanike tarvetest arusaamise teisendamist ja organiseerimist ehitatavat süsteemi kirjeldavale kujule. Süsteemi defineerimise alguses otsustatakse, mis määrab nõudmise. Lisaks otsustatakse veel dokumentatsiooni formaat, nõudmiste detailsus (kui palju ja millistes detailides), soovide prioriteetid ja hinnangulised eeldatavad tööpanused (kaks väga erinevat väärtust, mis tavaliselt antakse erinevate inimeste poolt erinevate asjade uurimise tulemusena), tehnilised ja juhtimise riskid ning esialgne ulatus. Võidakse tegeleda esialgse prototüübiga ja arhitektuuri mudelitega, mis on otseselt seotud osanike kõige tähtsamate nõuetega. Süsteemi defineerimise väljundiks on nii loomuliku keele kui graafilisel kujul olev süsteemi kirjeldus.

2.2.2.5 Projekti ulatuse juhtimine

Projekti efektiivseks läbiviimiseks on vaja kõikidelt osanikelt saadud sisendi põhjal hoolikalt prioriteerida nõuded ja määrata nende ulatus. Paljud projektid kannatavad selle all, et arendajad töötavad pigem nõ. “põnevate” asjade (erisused (*feature*), mis tunduvad arendajatele huvitavad ja väljakutsuvad) kallal, kui keskenduvad varakult

tegevustele, mis kõrvaldaksid projekti riske ja stabiliseeriks rakenduse arhitektuuri. Projekti riskide kõrvaldamiseks nii vara, kui võimalik, tuleb süsteemi välja töötada inkrementaalselt, valides iga inkrementi jaoks hoolikalt nõudmisi, mis vastavad projekti teadaolevatele riskidele. Selleks tuleb iga iteratsiooni ulatus projekti osanikega läbi rääkida. See nõuab tavaliselt häid oskusi projekti erinevatele faasidele pandavate ootuste juhtimisel. Lisaks sellele peavad kontrolli all olema ka nõuete allikad, mis ütlevad, kuidas peavad projekti saadused ja nede arendamise protsess välja nägema.

2.2.2.6 Süsteemi definitsiooni täpsustamine

Süsteemi detailne definitsioon peab olema kujul, millest osanikud aru saavad ja millega nad rahule jäävad. See peab lisaks funktsionaalsuse katmisele olema ka kooskõlas kõigi seaduslike või regulatiivsete aktide, kasutatavuse, töökindluse, suutlikkuse, toetatavuse ja hooldatavuse nõuetega. Viga, mis tihti tehakse, on uskumine, et süsteemil, mille ehitus on keeruline, peab olema ka keeruline definitsioon. See viib raskusteni projekti ja süsteemi otstarbe selgitamisel. Inimestele võib see muljet avaldada, aga neilt ei saa head sisendit, kuna nad ei saa sellest aru. Produktseeritavate **tehiste** sihtgrupist aru saamisele tuleb pöörata erilist tähelepanu – erinev auditoorium vajab tihti erinevat liiki kirjeldusi.

Näiteks **kasutusmalli** meetodika koos lihtsate visuaalsete prototüüpidega on väga efektiivne viis süsteemi otstarbe ja detailide kirjeldamiseks. Kasutusmallid aitavad asetada nõudmisi konteksti sisse – nad räägivad, kes ja kuidas süsteemi kasutab.

Teine viis süsteemi detailseks defineerimiseks on deklareerida, kuidas süsteemi testida tuleb. Testi plaanid ja definitsioonid sellest, mis teste sooritada, räägivad meile, millist süsteemi suutlikust kontrollitakse.

2.2.2.7 Muutuvate nõuete haldamine

Hoolimata sellest, kui hoolikalt nõudeid defineerida, on alati asju, mis muutuvad. Muutuvate nõudmiste haldamine ei ole keeruline mitte ainult sellepärast, et muutunud nõudmine tähendab rohkem või vähem aega, mida pühendatakse teatava uue erisuse teostamisele, vaid ka sellepärast, et muudatus ühes nõudmises võib mõjutada ka teisi nõudeid. Tuleb tagada, et nõuetel on struktuur, mida on kerge muuta, ja nõuete ning teiste arendustsükli tehiste vaheliste sõltuvuste kujutamiseks tuleb kasutada

jälitatuslinke (*traceability link*). Muudatuste haldus sisaldab endas veel tegevusi, nagu arendusaluse määramine, tuvastamine, milliste sõltuvuste jälitamine on oluline, ja seotud osade vahel jälitatusvuse määramine.

2.2.3 Kasuta komponendipõhiseid arhitektuure

2.2.3.1 Mis on komponentarhitektuurid?

Komponent on kas lähte- või täidetaval kujul hästi defineeritud liidesega ja käitumisega kapseldatud kokkukuuluv programmi kood, mis on asendatavad teiste sama liidese ja käitumisega komponentidega. Arhitektuurid, mis baseeruvad komponentidel, kalduvad olema lihtsamad tugevamad ja paindlikumad.

2.2.3.2 Arhitektuuri rõhk

Unifitseeritud arendusprotsessis juhivad süsteemi elutsükli algusest lõpuni kasutusmallid ja projekteerimise tegevused on keskendunud neid kasutusmalle teostavale tarkvara arhitektuurile. Protsessi esimestes iteratsioonides, eriti detailimisfaasi iteratsioonides, on põhirõhk tarkvara arhitektuuri produtseerimisel ja kehtestamisel. Alguses on see demonstreeritava arhitektuurilise prototüübi kujul, mis hilisemates iteratsioonides areneb järk-järgult lõplikuks süsteemiks. Demonstreeritava arhitektuuri all mõtleme süsteemi osalist teostust, mis on ehitatud valitud süsteemi funktsioonide ja omaduste demonstreerimiseks, eriti nende omaduste, mis rahuldavad mittefunktsionaalseid nõudmisi. See ehitatakse selleks, et kõrvaldada varakult suutlikkusega, läbilaskevõimega, võimsusega, töökindlusega ja teiste mittefunktsionaalsete omadustega seotud riskid nii, et süsteemi täieliku funktsionaalse suutlikkuse saab lisada konstrueerimisfaasis juba tugevale vundamendile ilma arhitektuuri kõrvale heitmise kartusteta.

2.2.3.3 Komponendipõhine arendus

Tarkvarakomponenti saab defineerida kui mittetriviaalset tarkvara tükki, moodulit, paketti või alamsüsteemi, mis kõik täidab mingit selget funktsiooni, omab selgeid piirjooni ja on integreeritav hästi defineeritud arhitektuuri.

Komponendid tulevad mitmest kohast:

- Komponente tuvastatakse, isoleeritakse, projekteeritakse, töötatakse välja ja testitakse väga modulaarset arhitektuuri defineerides. Neid komponente saab eraldi testida ja järk-järgult integreerida kogu süsteemi moodustamiseks.
- Osasid sellistest komponentidest saab välja töötada korduvkasutatavana, eriti neid, mis pakuvad hulgale üldistele probleemidele üldisi lahendusi. Need korduvkasutatavad komponendid, mis võivad olla suuremad, kui lihtsalt hulk vahendeid või klassiteeke, võivad olla korduvkasutuse aluseks kogu organisatsioonis, tõstes kogu organisatsiooni tarkvaratootmise produktiivsust ja kvaliteeti.
- Hiljutised kommertsmaailmas edukate komponendi infrastruktuuride, nagu CORBA, Internet, ActiveX ja JavaBean'id, arengud on päästnud valla terve eri valdkondade komponentide tööstuse tekkimise, mistõttu on komponente juba võimalik ise väljatöötamise asemel lihtsalt osta ja integreerida.

Ülaltoodud loetelu esimene punkt rakendab juba vana kontseptsiooni modulaarsusest ja kapseldamisest, arendades seda kontseptsiooni objektorienteeritud tehnoloogiast aluseks võttes sammu võrra edasi. Viimased kaks punkti liigutava tarkvaratootmise tarkvara reahaaval programmeerimiselt tarkvara koostamisele komponente ühendades.

2.2.4 Modelleeri tarkvara visuaalselt

2.2.4.1 Mis on visuaalne modelleerimine?

Visuaalne modelleerimine on tarkvara arhitektuuri saamiseks semantiliselt rikka graafilise ja tekstilise arhitektuurisümbolika kasutamine. Sümbolika, nagu näiteks UML, võimaldab täpse süntaksi ja semantika abil tõsta esile erinevaid abstraktsiooni tasemeid. Sel viisil paraneb arhitektuuri meeskonna omavaheline suhtlemine (arhitektuuri koostamisel ja läbivaatamisel), lugejad saavad arhitektuuri teemal arutleda ja süsteemi teostamiseks tekib üheselt mõistetav alus.

2.2.4.2 Miks modelleerida?

Mudel on süsteemi lihtsustatud vaade. See näitab süsteemi olulisi asju mingis teatavas vaates ja peidab ära mitteolulised asjad. Mudelid võivad aidata

- komplekssest süsteemist aru saamisel

- väikese ajakuluga uurida ja võrrelda arhitektuuri alternatiive
- rajada teostuse jaoks vundamenti
- tuvastada täpseid nõudmisi
- anda edasi üheseltmõistetavaid otsuseid

2.2.4.3 Komplekssest süsteemist arusaamisel aitamine

Süsteemi komplekssemaks muutumisel kasvab mudelite tähtsus. Näiteks võib koerakuuti ehitada ilma ehitusplaanita. Kui aga tegu oleks majaga või suisa pilvelõhkujaga, muutub vajadus ehitusplaani järgi tungivaks.

Sarnaselt võib olla väike ühe inimese poolt paari päevaga programmeeritud rakendus olla kergesti täielikult aru saadav. Samal ajal kümnete tuhandete koodiridadega e-äri süsteem või sadade tuhandete koodiridadega õhuliikluse juhtimise süsteem ei ole enam ühe inimese poolt kergesti arusaadav. Mudelid konstrueerides on väljatöötajal kergem keskenduda üldpildile, kujutledes, kuidas komponendid suhtlevad ja tuvastades saatuslikke vigu.

Mõned mudelite näited:

- Kasutusmallid üheseltmõistetavaks käitumise spetsifitseerimiseks.
- Klassi diagrammid ja andmemudeli diagrammid arhitektuuri saamiseks.
- Olekuskeemid dünaamilise käitumise modelleerimiseks.

Modelleerimine on tähtis, kuna see aitab meeskonnal visualiseerida, konstrueerida ja dokumenteerida süsteemi struktuuri ja käitumist ilma komplekssest segadusest sattumata.

2.2.4.4 Arhitektuuri alternatiivide uurimine madalate kuludega

Arhitektuuri alternatiivide uurimiseks saab väikeste kuludega luua ja muuta lihtsaid mudeleid. Teised väljatöötajad saavad innovatiivseid ideid läbi vaadata enne, kui investeeritakse kulukasse koodi väljatöötamisse. Koos iteratiivse arendusega aitab visuaalne modelleerimine väljatöötajatel hinnata muutusi arhitektuuris ja kirjeldada neid muutusi kogu arendusmeeskonnale.

2.2.4.5 Teostuse jaoks vundamendi rajamine

Tänapäeval kasutavad juba paljud projektid objektorienteeritud programmeerimiskeeli korduvkasutatavate, kergesti muudetavate ja stabiilsete süsteemide saamiseks. Nende hüvedeni jõudmiseks on veelgi tähtsam kasutada ka arhitektuuris objekti tehnikat.

Vastavate vahendite toel on mudelit võimalik kasutada ka teostuse esialgse koodi genereerimiseks.

2.2.4.6 Täpsete nõudmiste tuvastamine

Nõuete tuvastamine enne süsteemi ehitamist on kriitilise tähtsusega. Täpse ja üheseltmõistetava mudeli kasutamine nõuete spetsifitseerimisel aitab tagada, et kõik osanikud saavad nõudmistest aru ja on nendega nõus.

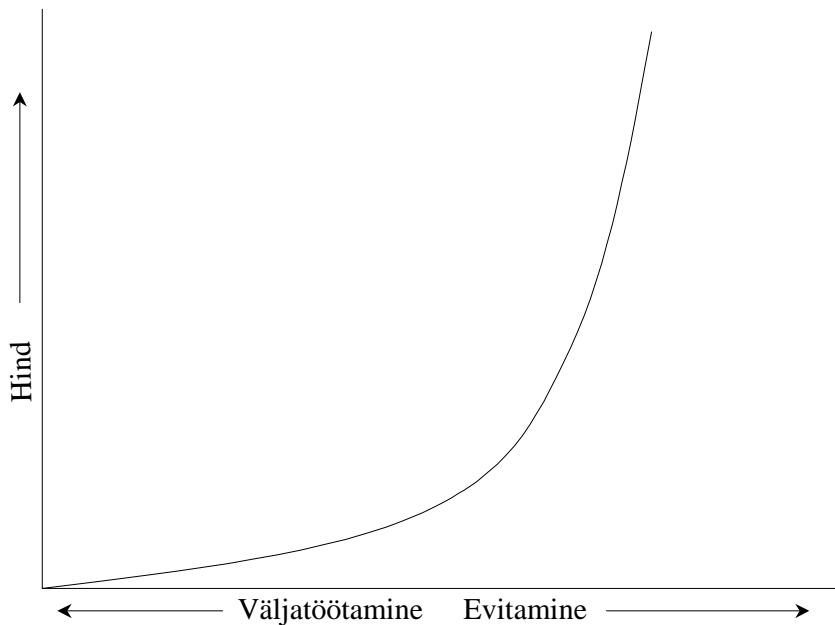
Mudel, mis eraldab süsteemi välise käitumise sisemisest teostusest, aitab keskenduda süsteemi kavatsetavale kasutamisele ilma laskumata süsteemi teostuse detailidesse.

2.2.4.7 Otsuste üheseltmõistetav edastamine

Unifitseeritud modelleerimiskeel (UML – *Unified Modelling Language*) on kooskõlaline sümboolika, mida saab rakendada nii süsteemi projekteerimisel, kui talitluse projekteerimisel. See on standardiseeritud sümboolika, mis täidab järgmisi rolle:

- Selle keele abil saab edastada projekteerimisotsuseid, mis on üheselt mõistetavad ja ei vaja selleks samas programmi koodi jälgimist.
- Selle semantika on piisavalt rikas, et kajastada kõiki olulisi strateegilisi ja taktikalisi otsuseid.
- See pakub piisavalt konkreetset vormi inimestega arutlemiseks ja vahenditega manipuleerimiseks. UML koondab endasse objektitehnika tööstuses tarkvara modelleerimise parima tööpraktika.

2.2.5 Kontrolli tarkvara kvaliteeti



Joonis 5. Tarkvara probleemide leidmine ja lahendamine on peale evitust 100-1000 korda kulukam kui projekti alguses. Kvaliteedi kontrollimine ja tagamine projekti elutsükli jooksul on õigete eesmärkide õigeks ajaks saavutamiseks eluliselt vajalik.

2.2.5.1 Mida me mõtleme projekti elutsükli jooksul kvaliteedi kontrollimise all?

On tähtis, et kõikide tehiste kvaliteeti hinnataks nende küpsemisel elutsükli erinevates punktides. Tehiseid tuleks hinnata kohe, kui tegevused, mis neid produtseerivad, lõppevad, ning iga iteratsiooni lõpus. Eriti siis, kui produtseeritakse töötavat tarkvara, tuleks seda tarkvara demonstreerida ja testida selle peal iga iteratsiooni lõpus olulisi stsenaariume, pakkudes sellega rohkem käegakatsutavat ettekujutust järeleandmiste suhtes arhitektuuris ja võimaldades arhitektuuri defekte varem kõrvaldada. See on vastupidine traditsionaalse lähenemise suhtes, mis jätab integreeritud tarkvara testimise elutsükli lõppu.

2.2.5.2 Mis asi on kvaliteet?

Kvaliteet on miski, mille poole me oma toodetes, protsessides ja teenustes püüdleme. Aga alati, kui küsida, “Mis on kvaliteet?”, on igapähe erinev arvamus. Tavalised vastused sisaldavad näiteks:

“Kvaliteet... Ma ei ole päris kindel, kuidas seda kirjeldada, aga ma tean, kui ma seda näen.”

või

“... nõuetele vastamine.”

Muide kõige sagedasem viitamine kvaliteedile, eriti tarkvara korral, toimub kõrvalmärkusena selle puudumise korral:

“Kuidas nad saavad midagi nii ebakvaliteetset välja lasta!?”

Need tavalised vastused küll räägivad kvaliteedist, aga nad pakuvad liiga vähe täpseks kvaliteedi uurimiseks ja selle tulemusel parandamiseks. Need kommentaarid kõik illustreerivad kvaliteedi defineerimise vajadust viisil, mis võimaldab seda mõõta ja saavutada.

Kvaliteet ei ole üksik iseloomustaja või atribuut. See on mitmemõõtmeline ja seda võib omada nii toode, kui protsess. Toote kvaliteet keskendub õige toote ehitamisele, protsessi kvaliteet keskendub toote ehitamise korrektsusele.

Kvaliteedi saavutamine ei ole ainult “nõuetele vastamine”, või kasutaja vajadusi ja ootusi rahuldava toote valmistamine. Kvaliteet pigem sisaldab mõõdustikku ja kriteeriume kvaliteedi saavutamise demonstreerimiseks ning protsessi teostust tagamaks, et protsessi poolt loodud toode saavutab soovitud kvaliteedi taseme ja et selle saavutamine on korratav ja juhitav.

2.2.6 Juhi tarkvara muudatusi

Peamine väljakutse tarkvarasüsteemide väljatöötamisel on see, et tuleb toime tulla erinevatesse meeskondadesse organiseerunud, võimalik, et erinevates kohtades asuvate, paljude arendajatega, kes töötavad koos mitme iteratsiooni, redaktsiooni, toote ja platvormi kallal. Distsiplineeritud juhtimise puudumisel degenerereerub arendusprotsess kiiresti kaoseks.

Arendajate ja meeskondade tegevuste ja tehiste koordineerimine tähendab tarkvara ja teiste arenduse tehiste muudatuste juhtimiseks korratavate protseduuride kehtestamist. Selline koordineerimine võimaldab projekti prioriteetidel ja riskidel baseerudes paremini ressursse hõivata. Koos iteratiivse arendusega võimaldab see tööpraktika

pidevalt jälgida muudatusi nii, et on võimalik tegeleda aktiivselt probleemide avastamisega ja neile reageerimisega.

Iteratsioonide ja redaktsioonide koordineerimine tähendab iga iteratsiooni lõpus testitud arendusaluse kinnitamist ja. Iga redaktsiooni elementide ja mitme paralleelse redaktsiooni elementide vahel jälitatavuse ülal pidamine on hindamiseks ja aktiivseks muudatuste mõjude juhtimiseks eluliselt vajalik.

Tarkvara muudatuste haldamine pakub hulga lahendusi tarkvaraarenduse probleemide algpõhjustele:

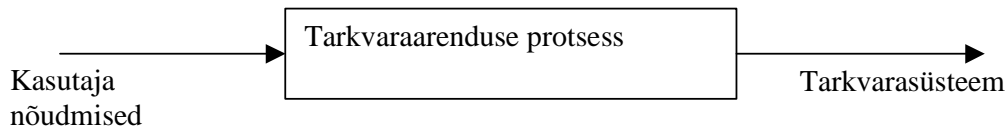
- Nõuete muutuse **töövoog** on defineeritud ja korratav.
- Muutmistaotlused hõlbustavad selget suhtlemist.
- Isoleeritud töökoopiad vähendavad paralleelselt töötavate meeskonnaliikmete omavahelist mõjutamist.
- Muudatuste hulga statistika on meetrikaks projekti seisu objektiivsel hindamisel.
- Töökoopiad sisaldavad kõiki tehiseid, hõlbustades kooskõla tagamist.
- Muutuste levitamine on hinnatav ja juhitud.
- Muudatusi on võimalik hooldada tugeva ja paindliku süsteemi abil.

2.3 Unifitseeritud protsess

Unifitseeritud tarkvaraarendusprotsess on protsess, mis ühendab ülal kirjeldatud tarkvaraarenduse parima tööpraktika ja lisab mehhanismid selle ühe tervikliku protsessina esitamiseks. Käesolev paragrahv on peamiselt ülevaade raamatu [Jacobson *et al.*, 1999] vastavatest kirjeldustest.

2.3.1 Unifitseeritud protsess lühidalt

Tarkvaraarendusprotsess koosneb hulgast tegevustest, mida on vaja läbida kasutaja vajaduste põhjal tarkvarasüsteemi väljatöötamiseks (vt. Joonis 6). Unifitseeritud protsess on rohkem, kui lihtsalt protsess – see on üldine protsessi raamistik, mida saab spetsiaalselt kohandada suure hulga erinevate tarkvarasüsteemide, rakendusvaldkondade, organisatsiooni tüüpide, kompetentsi tasemete ja projekti suuruste jaoks.



Joonis 6. Tarkvaraarenduse protsess.

Unifitseeritud protsess on komponendipõhine, mis tähendab, et tarkvarasüsteemi ehitatakse tarkvarakomponentidest, mis on omavahel hästi defineeritud liideste abil ühendatud. Unifitseeritud protsess kasutab tarkvara arhitektuuri modelleerimiseks unifitseeritud modelleerimise keelt (*Unified Modeling Language – UML*). UML, on unifitseeritud protsessi üks olulisemaid osasid – unifitseeritud protsessi ja UMLi töötati välja käsikäes.

Unifitseeritud protsessi tegelikuks eristavaks tunnuseks on kolm võtmesõna: ta on **kasutusmalljuhitav** (*use-case driven*), ta on arhitektuurikeskne (*architecture centric*) ning ta on iteratiivne ja inkrementaalne.

2.3.2 Unifitseeritud protsess on kasutusmalljuhitav

Iga **kasutusmalliga** on seotud **tegijad**. **Tegija** on keegi või miski väljaspool süsteemi või talitlust, mis suhtleb süsteemi või talitlusega. Mõiste **tegija** ei viita ainult inimkasutajatele vaid ka teistele süsteemidele. Suhtlemise näide on inimene, kes suhtleb pangaautomaadiga. Ta sisestab plastikkaardi, vastab automaadi ekraanil esitatud küsimustele ja saab mingi summa raha. Vastavalt selle inimese kaardile ja vastustele, sooritab süsteem rea tegevusi, mis tervikuna omavad kasutaja jaoks mingit kasu, antud juhul annab raha.

Sellist (tervikuna kasulikku) tüüpi teataval otstarbel suhtlust kutsutakse **kasutusmalliks**. **Kasutusmall** on tükk funktsionaalsust, mille tulemus on kasutaja jaoks mõtestatud – süsteemi funktsionaalsuse kirjeldus süsteemi kasutaja terminites. Kasutusmallid määravadki ära süsteemile esitatavad funktsionaalsed nõudmised. Kõik süsteemi kasutusmallid koos moodustavad **kasutusmalli mudeli**, mis kirjeldab süsteemi täielikku funktsionaalsust. See mudel asendab traditsioonilist süsteemi funktsionaalsuse spetsifikatsiooni. Funktsionaalsuse spetsifikatsioon vastab küsimusele, mida süsteem teeb. Kasutusmalli strateegia lisab sellele küsimusele kolm sõna lõppu: “... iga **tegija** jaoks?” Nendel kolmel sõnal on väga tähtis mõju – need

sunnivad meid mõtlema kasutaja väärtuste terminites, mitte ainult oletatavalt kasulike funktsioonide terminites.

Kasutusmallid ei ole mitte ainult vahend süsteemile esitatavate nõuete spetsifitseerimiseks, vaid need ka juhivad süsteemi arhitektuuri, teostust ja testimist. See tähendab seda, *nad juhivad väljatöötamise protsessi*. Kasutusmalli mudeli põhjal loovad väljatöötajad hulga arhitektuuri ja teostuse mudeleid, mis kasutusmalli realiseerivad. Väljatöötajad vaatavad kõiki väljapakutud mudeleid kasutusmallidele vastavuse suhtes läbi. Testijad testivad teostust kontrollimaks, et teostuse mudeli komponendid teostavad kasutusmalli korrektselt. Sel viisil kasutusmallid mitte ainult ei algata väljatöote protsessi vaid seovad selle ka kokku. **Kasutusmalljuhitav** tähendab seda, et väljatöote protsess läbib töövoogusid, mis on tuletatud kasutusmallidest. Kasutusmallid spetsifitseeritakse, kasutusmallide järgi projekteeritakse ja lõpus on kasutusmallid testijatele testjuhtumite konstrueerimise aluseks.

Kasutusmallid ei arene muudest asjadest isoleeritult. Neid töötatakse välja koos süsteemi arhitektuuriga. St. kasutusmallid juhivad süsteemi arhitektuuri ja süsteemi arhitektuur mõjutab kasutusmallide täpsustamist. Seega mõlemate, süsteemi arhitektuuri ja kasutusmallide küpsus elutsükli edenedes paraneb.

Nõuete haldusest kasutusmallide abil saab lugeda veel allikast [OPE, 2000].

2.3.3 Unifitseeritud protsess on arhitektuurikeskne

Arhitektuuri roll tarkvaratootmises on sarnane arhitektuuri rollile ehituses. Ehitust vaadeldakse erinevatest vaatenurkadest: struktuur, talitus, kliima juhtimine, veevõrk, elekter jne. See võimaldab ehitajal enne ehituse alustamist näha täielikku pilti.

Tarkvara arhitektuur sisaldab endas süsteemi kõige olulisemaid staatilisi ja dünaamilisi aspekte. Tarkvara arhitektuur kasvab välja seda tarkvara kasutama hakkava ettevõtte vajadustest sellisena, nagu kasutajad ja teised osanikud seda tunnetavad, ja nagu see kasutusmallides peegeldub. Tarkvara arhitektuuri mõjutavad ka paljud teised faktorid, nagu platvorm, mille peal tarkvara käima peab (arvuti arhitektuur, operatsioonisüsteem, andmebaasijuhtimissüsteem, võrguprotokollid), saadaolevad korduvkasutatavad komponendid (nt. graafilise kasutajaliidese vahendid), evituskaalutlused, vanemad süsteemid ja mittefunktsionaalsed nõudmised (nt. suutlikkus, töökindlus). Tarkvara arhitektuur on vaade kogu süsteemile, tuues

olulisi iseloomustajaid rohkem esile ja jättes mitteolulised detailid kõrvale. See, mis on oluline, tuleneb, arusaamisest, mis omakorda tuleb kogemusest. Arhitektuuri väärtus sõltub selle väljatöötamisega tegelevatest inimestest. Protsess aitab arhitektidel keskenduda õigetele eesmärkidele, nagu arusaadavus, paindlikkus tulevaste muutuste suhtes ja korduvkasutatavus.

Kuidas on kasutusmallid ja arhitektuur seotud? Igal tootel on funktsioon ja vorm. Kummastki eraldi ei piisa. Need kaks jõudu peavad omavahel olema tasakaalus, et toode oleks hea. Antud juhul vastab funktsioon kasutusmallidele ja vorm arhitektuurile. Kasutusmallid ja arhitektuur mõjutavad teineteist moodustades “muna ja kana” probleemi. Ühest küljest peavad kasutusmallid realiseerituna sobima arhitektuuriga. Teisest küljest peab arhitektuur jätma ruumi kõikide praeguste ja tulevaste kasutusmallide realiseerimiseks. Reaalsuses peavad nii arhitektuur kui kasutusmallid arenema paralleelselt.

Arhitektuur peab olema projekteeritud nii, et see laseks süsteemil areneda mitte ainult selle esialgsel väljatöötamisel vaid ka tulevastes põlvedes. Et sellist vormi arhitektuurile leida, peab arhitektuuri projekteerimist alustama üldistest ettekujutustest võtmefunktsionaalsusest, st. süsteemi kõige olulisematest kasutusmallidest. Need kõige olulisemad kasutusmallid võivad moodustuda ainult 5% kuni 10% kõikidest kasutusmallidest. Need moodustavad süsteemi tuumikfunktsionaalsuse. Lihtsustatult:

- Arhitekt loob arhitektuurist toore visandi, alustades arhitektuuri selle osaga, mis ei ole spetsifitseeritud kasutusmallidega (st. on spetsifitseeritud platvormiga). Ehkki see osa on kasutusmallidest sõltumatu, peab arhitektil olema enne arhitektuuri visandi loomist üldine arusaam kasutusmallidest.
- Järgmisena töötab arhitekt tuvastatud alamhulgaga kasutusmallidest, mis kujutavad arendatava süsteemi võtmefunktsionaalsust. Iga valitud kasutusmall spetsifitseeritakse detailsemalt ja realiseeritakse alamsüsteemide, klasside ja komponentide terminites.
- Kasutusmallide spetsifitseerudes ja valmides arendatakse järjest rohkem arhitektuuri, mis omakorda viib rohkemate kasutusmallide valmimiseni.

Selline protsess jätkub seni, kuni otsustatakse, et arhitektuur on stabiilne.

2.3.4 Unifitseeritud protsess on iteratiivne ja inkrementaalne

Kommertstarkvarasüsteemi väljatöötamine on suur ettevõtmine, mis võib kesta mitu kuud või isegi rohkem, kui aasta. Töö jagamine väiksemateks miniprojektideks on praktiline. Iga miniprojekt on iteratsioon, mille tulemuseks on süsteemi inkrement. Iteratsioonid viitavad töövoo sammudele ja inkrementid toote kasvule. Iteratsioonid peavad efektiivsuse huvides olema juhitud. St. neid tuleb plaanitult läbi viia. Sellepärast neid kutsutaksegi *miniprojektideks*.

Väljatöötajad baseeruvad iteratsioonis teostatavate tööde valikul kahele faktorile. Esiteks, iteratsioon tegeleb grupi kasutusmallidega, mis koos laiendavad juba senimaani väljatöötatud toote kasutatavust. Teiseks tegeleb iteratsioon kõige olulisemate riskidega. Iga järgmine iteratsioon baseerub väljatöötete tehistel, mis on olekus, kuhu nad jäid viimati lõppenud iteratsioonis. Inkrement ei pea tingimata midagi juurde lisama. Eriti elutsükli esimestes faasides võivad väljatöötajad tegeleda pealiskaudse arhitektuuri rohkem detailse või keerukamaga väljavahetamisega. Viimastes faasides on aga inkrementid tüüpiliselt funktsionaalsust juurde lisavad.

Igas iteratsioonis tuvastavad ja spetsifitseerivad väljatöötajad vastavad kasutusmallid, projekteerivad vastava arhitektuuri, teostavad selle arhitektuuri komponentidena ja kontrollivad, et need komponendid rahuldavad valitud kasutusmalle. Kui iteratsioon rahuldab talle seatud eesmärgid (mida ta tavaliselt teeb), läheb väljatöötamine edasi järgmise iteratsiooniga. Kui iteratsioon ei rahulda talle seatud eesmärgi, peavad väljatöötajad oma eelmised otsused uuesti läbi vaatama ja võibolla mingit uut lahendust katsetama.

Majanduslikel kaalutlustel plaanib meeskond ainult selliseid iteratsioone, mis aitavad projekti eesmärgi saavutada. Iteratsioone üritatakse järjestada loogiliselt. Edukas projekt ei kaldu algselt plaanitud kursist palju kõrvale. Ootamatute probleemide tõttu lisandunud iteratsioonid ja/või muutunud iteratsioonide järjekord nõuavad rohkem tööpanuseid ja aega. Ootamatute probleemide vähendamine on üks riskide vähendamise eesmärgi.

Juhitud iteratiivne protsess pakub palju hüvesid:

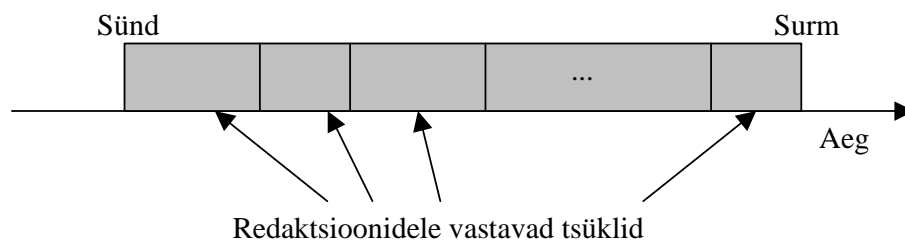
- Juhitud iteratsioon vähendab üle inkrementi kulu riski. Kui väljatöötajad peavad iteratsiooni kordama, kaotab organisatsioon kogu projekti ulatuse asemel ainult selle iteratsiooni jooksul tehtud valesti plaanitud tööpanused.

- Juhitud iteratsioon vähendab riski, et toodet ei õnnestu plaanitud ajaks turule lasta. Tuvastades riske väljatöötamise alguses kulub nende lahendamiseks aeg väljatöötamise alguses, kui inimestel pole veel nii kiire, kui väljatöötamise lõpus.
- Juhitud iteratsioon lisab kogu väljatöötamisele tempot, sest väljatöötajad jõuavad tulemusteni efektiivsemalt pideva libiseva graafiku asemel lühiajaliste selgete eesmärkide korral.
- Juhitud iteratsioon arvestab reaalsusega, mida tihti ignoreeritakse – kasutajate vajadusi ja vastavaid nõudmisi ei saa enne projekti algust täielikult defineerida. Selle asemel nad tüüpiliselt paranevad järjestikustes iteratsioonides. Seetõttu on ka sel viisil opereerides võimalik muutuvate nõudmistega kohaneda.

Need kontseptsioonid, kasutusmalljuhitavus, arhitektuurikesksus ning iteratiivne ja inkrementaalne arendamine, on võrdselt olulised. Arhitektuur annab struktuuri, mille abil iteratsioonides tööd korraldada, samal ajal, kui kasutusmallid defineerivad eesmärgid ja juhivad iga iteratsiooni töid. Eemaldades ükskõik millise neist kolmest, vähendame oluliselt unifitseeritud protsessi väärtust. See on nagu kolme jalaga tool – ilma ühe jalata, kukub tool ümber.

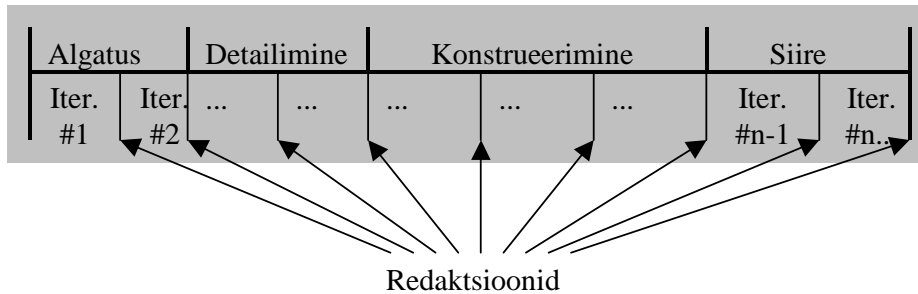
2.3.5 Elu unifitseeritud protsessi järgi

Unifitseeritud protsessis käib tarkvara sünnist surmani läbi rea tsükleid, kus iga tsükkel vastab toote kliendile üleantavale redaktsioonile (*release*) (vt. Joonis 7).



Joonis 7. Süsteemi elu, mis koosneb sünnist surmani tsüklitest.

Iga tsükkel koosneb neljast faasist: algatus (*inception*), detailimine (*elaboration*), konstrueerimine (*construction*) ja siire (*transition*). Iga **faas** on omakorda jagatud iteratsioonideks (vt Joonis 8).



Joonis 8. Elutsükkel faaside ja iteratsioonidega.

2.3.6 Toode

Iga tsükli tulemuseks on süsteemi uus tarnitav redaktsioon. See koosneb komponentide lähtetekstidest, mida saab kompileerida ja käivitada, ning õppematerjalidest ja muudest seotud saadustest. Valmis toode peab rahuldama mitte ainult kasutajate, vaid ka kõikide teiste **osanike**, st. tootega töötavate isikute, vajadusi. *Tarkvaratoode* on selles mõttes rohkem, kui ainult käivitatav masinkood.

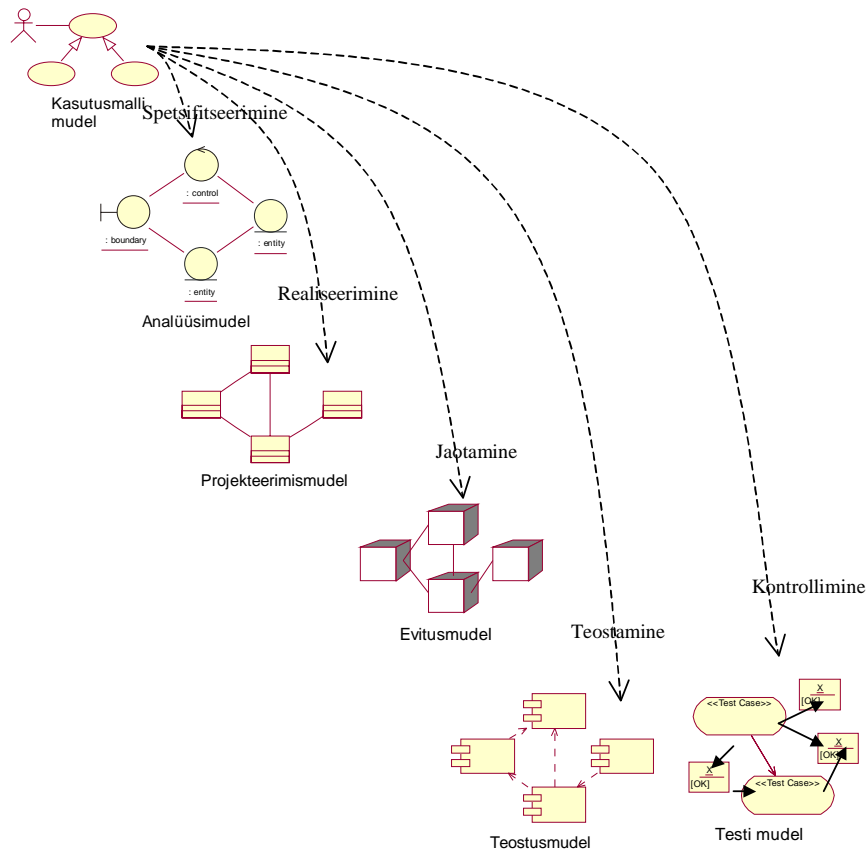
Valmistoode sisaldab nõudeid, kasutusmalle, mittefunktsionaalseid nõudeid ja testjuhtumeid. See sisaldab arhitektuuri ja visuaalseid mudeleid – UML-iga modelleeritud tehiseid. See sisaldab kõiki tehiseid, millest me selles paragrahvis (“Unifitseeritud protsess”) rääkinud oleme, sest need on need asjad, mis võimaldavad osanikel (kliendid, kasutajad, analüütikud, projekteerijad, teostajad, testijad ja juhatus) spetsifitseerida, projekteerida, teostada, testida ja süsteemi kasutada. Veelgi enam, need asjad võimaldavad osanikel süsteemi muuta ja kasutada **põlvest põlve**.

Ehkki käivitatavad komponendid on kõige tähtsamad kasutaja vaatenurgast, neist üksi ei piisa, sest keskkond muutub. Operatsioonisüsteemid, andmebaasisüsteemid ja kasutatavad arvutid arenevad. Kui eesmärgist paremini aru saadakse, võivad nõuded muutuda. See, et nõudmised muutuvad on tarkvaratootmises konstantne nähtus. Selle tulemusena peavad väljatöötajad võtma ette uusi tsükleid ja juhud peavad seda finantseerima. Et järgmist tsüklit efektiivselt läbi viia, vajavad väljatöötajad kõiki tarkvaratoote esitusi (vt. Joonis 9):

- Kasutusmalli mudel koos kõigi kasutusmallidega ja nende kasutajate vaheliste seostega.
- Analüüsimudel, millel on kaks eesmärki: kasutusmallide detailsem täpsustamine ja esialgse süsteemi käitumise kirjelduse seda käitumist võimaldavate objektide abil esitamine.
- Projekteerimismudel, mis defineerib süsteemi staatilise struktuuri alamsüsteemidena, klassidena ja liidestena ning alamsüsteemide, klasside ja liideste vahelise **koostööna** realiseeritud kasutusmallid.
- Teostusmudel, mis sisaldab (lähtetekste esitavaid) komponente, ja klasside komponentideks kujutamist.
- Eviitumudel, mis defineerib arvutite füüsilised sõlmed ja komponentide kujutamise nendeks sõlmedeks.
- Testi mudel, mis kirjeldab testjuhtumeid, mis kontrollivad kasutusmalle.
- Arhitektuuri esitus.

Süsteemil võib olla ka valdkonnamudel või talitluse mudel, mis kirjeldab süsteemi talitluskonteksti.

Kõik need mudelid on seotud. Nad koos esitavad süsteemi kui tervikut. Et aidata mudelit ühendada teiste mudelitega, on selles mudelis asuval elemendil jälitusseosed teiste mudelite elementidega nii päri-, kui pöördsuunas. Näiteks võib kasutusmall (kasutusmalli mudelist) olla jälitav kasutusmalli realisatsioonini (projekteerimismudelis) ja testjuhtumini testi mudelis. Jälitavus hõlbustab arusaamist ja muutmist.



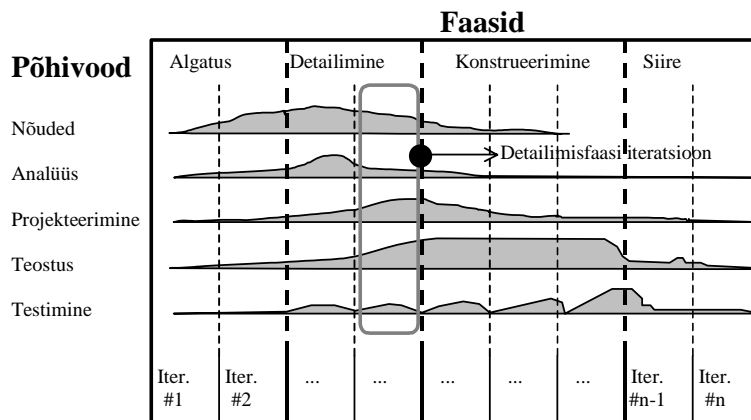
Joonis 9. Unifitseeritud protsessi mudelid. Paljude mudelite vahel on seosed. Näitena on esitatud kasutusmallide seosed teiste mudelitega.

2.3.7 Tsükli faasid

Iga tsükkel kestab mingi aje. See aeg on jagatud neljaks faasiks (vt Joonis 10). Mudelite abil osanikud visualiseerivad seda, mis neis faasides toimub. Igas faasis võivad juhid või väljatöötajad töö liigendada edasi veel iteratsioonideks ja nendest tulenevateks inkrementideks. Iga faas lõppeb **tähtpunktis** (*milestone*). Iga tähtpunkt on defineeritud mingi tehiste hulga olemasoluga. St. teatavad mudelid või dokumendid peavad olema omandanud eelnevalt kirjeldatud seisundi.

Tähtpunktid teenivad mitut eesmärki. Kriitilisim on see, et juhid peavad enne töö jätkamist järgmise faasiga tegema teatavaid eluliselt tähtsaid otsuseid. Tähtpunktid võimaldavad nii juhtidel kui ka väljatöötajatel endil jälgida neid nelja tähtsat punkti läbides töö arengut. Lisaks, jälgides igale faasile kulutatud aega ja panuseid, omandame me informatsiooni, mida saame kasutada teistele projektidele kuluva aja hindamisel, meeskonna plaanimisel, meeskonna vajaduste ennustamisel kogu projekti jooksul ja projekti arengu juhtimisel nende ennustuste järgi.

Joonis 10 loetleb vasakul veerus üles töövood: nõuete tuvastamine, analüüs, projekteerimine, teostamine ja testimine. Paremal pool toodud kõverad iseloomustavad (seda pilti ei ole soovitatav võtta liiga täpselt), mil määral vastavad töövood igas faasis esindatud on. Meenutagem, et iga faas on jagatud iteratsioonideks või miniprojektideks. Tüüpiline iteratsioon käib läbi kõik viis töövoogu, nagu näiteks näitab Joonis 10 detailimisfaasi iteratsiooni kohta.



Joonis 10. Viit töövoogu, nõuete tuvastamist, analüüsi, projekteerimist, teostust ja testimist läbitakse neljas faasis, algatusfaasis, detailimisfaasis, konstrueerimisfaasis ja siirdefaasis.

2.3.7.1 Algatusfaas

Algatusfaasis töötatakse heast ideest välja lõpptoote **nägemus** ja esitatakse toote **ärimall**. See faas vastab vähemalt järgmistele küsimustele:

- Mida süsteem peamiselt tegema hakkab iga tema peamise kasutaja jaoks?
- Milline selle süsteemi arhitektuur välja näha võiks?
- Mis on plaan ja mida selle toote väljatöötamine maksma läheb.

Lihtsustatud kasutusmalli mudel, mis sisaldab kõige kriitilisemaid kasutusmalle, vastab esimesele küsimusele. Sellest staadiumis on arhitektuur katseline. Tüüpiliselt on see lihtsalt visand, mis sisaldab kõige olulisemaid alamsüsteeme. Selles faasis tuvastatakse ja prioritseeritakse kõige olulisemad riskid, plaanitakse detailselt detailimisfaas, ja kogu projekti hinnatakse umbkaudselt.

2.3.7.2 Detailimisfaas

Detailimisfaasis spetsifitseeritakse detailselt enamus toote kasutusmallidest ja projekteeritakse süsteemi arhitektuur. Süsteemi arhitektuuri ja süsteemi enda vaheline seos on ülimalt tähtsusega. Analoogia võiks olla skelett, mis on kaetud nahaga, aga millel on väga vähe lihaseid (tarkvara) luude ja naha vahel, ainult piisavalt liha, et skelett saaks teha algelisi liigutusi. Süsteem ise on kogu keha koos skeleti, naha ja lihastega.

Arhitektuur esitatakse vaadetena kõikidest süsteemi mudelitest, mis koos moodustavad terve süsteemi – eksisteerivad arhitektuurivaated kasutusmalli mudelist, analüüsimudelist, projekteerimismudelist, teostusmudelist ja evitusmudelist. Teostusmudeli vaade sisaldab komponente, mis tõestavad, et arhitektuur on käivitav. Selles väljatöötamise faasis tuvastatakse kõige kriitilisemad kasutusmallid ja realiseeritakse need. Selle faasi tulemuseks on arhitektuuri **arendusalus** – läbivaadatud ja vastuvõetud tehise redaktsioon, mis vastab kokkuleppele edasise arenduse suhtes ja mida saab muuta ainult formaalse protseduuri kaudu, nagu **muutusehaldus** või **konfiguratsioonihaldus**.

Detailimisfaasi lõpus on projektijuhi ülesanne projekti lõpuleviimiseks vajalike tegevuste plaanimine ja vajalikke ressursside hindamine. Võtmeküsimuseks on siin, kas kasutusmallid, arhitektuur ja plaanid on piisavalt stabiilsed ja kas riskid on piisavalt kontrolli all, et oleks võimalik kogu arendustöö lepinguna kinnitada.

2.3.7.3 Konstrueerimisfaas

Konstrueerimisfaasis ehitatakse toode – skeletile (arhitektuurile) lisatakse lihased (tarkvara). Selles faasis kasvab arhitektuuri arendusalus küpseks süsteemiks. Nägemus areneb tooteks, mis on valmis siirduma kasutajate kätte. Selles väljatöötamise faasis kulutatakse suurem osa kogu projekti jaoks vaja minevatest ressurssidest. Kuigi süsteemi arhitektuur on stabiilne, võivad väljatöötajad süsteemi struktureerimiseks paremaid viise avastades soovitada arhitektidele väiksemaid arhitektuurimuudatusi. Selle faasi lõpuks sisaldab toode kõiki kasutusmalle, mis on juhtkonna ja kliendi poolt selles redaktsioonis väljatöötamiseks kokku lepitud. Sellele vaatamata ei pruugi see defektidest vaba olla. Siirdefaasi ajal leitakse ja parandatakse veel defekte. Tähtpunkti küsimus on, kas toode vastab mõne kliendi jaoks kasutajate vajadustele piisavalt, et seda juba kasutada.

2.3.7.4 Siirdefaas

Siirdefaas katab perioodi, mille jooksul tootest saab beta-redaktsioon. Beta-redaktsioonis üritab väike kogunud kasutajate hulk toodet kasutada ning teatab defektidest ja puudujääkidest. Siis väljatöötajad parandavad probleemid ja lisavad mõned soovitud täiendused laiemale kasutajate ringile mõeldud üldisesse redaktsiooni. Siirdefaas sisaldab tegevusi nagu näiteks pakendamine, kliendi personali koolitamine, telefoniabi pakkumine ja tarnejärgselt avastatud defektide parandamine. Hooldusmeeskond jagab need defektid sageli kahte kategooriasse: need, mis mõjutavad toote töötamist piisavalt, et sobitada see kohesesse delta-redaktsiooni, ja need, mida saab parandada järgmises regulaarses redaktsioonis.

2.3.7.5 Integreeritud protsess

Unifitseeritud protsess on ka ise komponendipõhine “toode”. See kasutab uut visuaalse modelleerimise standardit, UMLi, ja baseerub kolmel põhiideel: kasutusmallid, arhitektuur ja iteratiivne inkrementaalne arendus. Et need ideed tööle panna, on vaja protsessi, mis võtaks korraga arvesse tsüklid, faasid, töövood, riskide ära hoidmise, kvaliteedijuhtimise, projekti juhtimise ja konfiguratsioonihalduse. Unifitseeritud protsess on paika pannud raamistiku, mis integreerib kõik need erinevad aspektid. Selle raamistiku järgi saavad töövahendite tootjad ja väljatöötajad ehitada vahendeid, mis aitavad seda protsessi automatiseerida, toetada üksikuid töövoogusid, ehitada erinevaid mudeleid ja integreerida töö üle elutsüklite ja üle kõigi mudelite.

3 Juhtumianalüüs esimestest katsetustest Cybernetica AS-is

Siin peatükk kirjeldatakse juhtumianalüüsi esimestest katsetest rakendada seda protsessi Cybernetica AS-is paari autori juhitud projekti peal.

Alustuseks olgu veel mainitud, et Cybernetica AS tegelikult koosneb kolmest sõltumatust osakonnast, mis midagi toodavad:

- Navigatsioonisüsteemide osakond (NSO) – tegutseb elektroonikaseadmete tootmise vallas, peamiselt mereseire süsteemidega ja valgusdioodlampidega.
- Infosüsteemide osakond – nagu nimigi ütleb, tegutseb infosüsteemide vallas peamiselt tolliamet infosüsteemi ja hotellide infosüsteemi arendusega.
- Infoturbe osakond – tegeleb infoturbe toodete väljatöötamisega, peamised tooted on sideturvet ja avaliku võtme infrastruktuuri toetavad tooted.

Kõik need osakonnad (ka elektroonikaseadmetega tegelev NSO) on seotud tarkvaratootmisega, aga kuna need osakonnad tegutsevad teineteisest peaaegu sõltumatult (eraldi eesmärgid, isikkoosseis, ruumid, eelarved; tegevuste kokkupuutepunkte väga vähe) ning autor töötab ja omandab kogemusi infoturbe osakonnas, siis ka see peatükk kirjeldab ainult elu infoturbe osakonnas. Seega, kui siin peatükis räägitakse Cybernetica AS-ist, siis mõeldakse Cybernetica AS-i infoturbe osakonda, kui ei ole muud täpsustatud. Seda, kuidas infoturbe osakonnas järgiproovitud kogu Cybernetica ASis rakendada kavatsetakse, räägitakse järgmises peatükis: “Tarkvaraarenduse protsessi rakendamine kogu organisatsioonis”.

Olulise asjana olgu mainitud ka see, et praktikas on alati väga olulisel kohal olnud mitte ainult olukorra formaalne kirjeldamine ja range teoreetikute välja töötatud meetodiline lähenemine, vaid ka olustik, inimeste omavahelised suhted, kogemused jne. Seetõttu, et käesoleva magistritöö lugeja paremini siinset juhtumianalüüsi mõistaks, peab autor vajalikuks lisaks meetodikale ja statistikale ka ära tuua vabas vormis kirjeldatuna sündmuste käik ajalises järjestuses.

3.1 Protsess enne unifitseeritud protsessi

Enne, kui Cybernetica AS-is hakati unifitseeritud protsessi uurima, sarnanes sealgi arendusprotsess koskmudelile, mis ideaalis pidi välja nägema järgmine:

1. Cybernetica ASis on tegu teadusarendusasutusega, st. ise teadusuuringuid läbiviiva ja nende tulemuste põhjal tooteid arendava asutusega ühest küljest, ja mitte konkreetse kliendi tellimuse peale tootmise, vaid nõ. “oletatavale kliendile” tootmisega tegeleva asutusega teisest küljest. Seetõttu on seal reeglina toote väljatöötamise alguses esimese asjana olemas nägemused mingite subjektide (potentsiaalsete klientide) lahendamata, aga lahendamist vajavatest teatavatest probleemidest ja teadusartiklid sellest, kuidas neid probleeme lahendada saaks.
2. Nendes nägemustes ja teadusartiklites kirjeldatud ideede teostamiseks pannakse paika projekti plaan.
3. Et ülesande püstitus oleks konkreetsem, kui hängusad nägemused sellest, mida klient vajada võiks, kirjeldatakse projekti käivitades turundusmeeskonna abil esimese asjana tarkvaranõuete spetsifikatsioon. Turundusmeeskonna abil sellepärast, et nende vastutusel on arendusmeeskonna jaoks selle oletatava kliendi soovi “ära arvamine” ja meile kirjeldamine.
4. Tarkvaranõuete spetsifikatsiooni järgi koostab arendusmeeskond tarkvaralahenduse kirjelduse.
5. Tarkvaralahenduse kirjelduse järgi programmeerivad programmeerijad süsteemi komponendid valmis.
6. Valmis programmeeritud komponendid integreeritakse ja testitakse.
7. Valmis toote jaoks kirjutatu tugimaterjal, pakendati, müüdi, tehti koolitusi ja müüdi kasutajatuge.

Tundub ilus ja loogiline ja nagu maailma tasemel autoriteetid kirjeldavad, läks sealgi alguse ots alati suhteliselt ladusalt: nägemused ja teadusartiklid olid dokumenteeritud, nõuded kenasti spetsifitseeritud ja tarkvara lahendus kirjeldatud. Nende jaoks kujunesid välja ka kõik vajalikud mallid ja eeskirjad, kuidas neid tehiseid produtseerida – võta ja kirjuta ainult.

Paraku aga läks tüüpstsenaarium ka edasi, nagu raamatust loetud. Juba programmeerimise ajal hakati tehnilistest probleemidest tingituna tegema järeleandmisi kirja pandud nõudmistes. Õigemini üritati kokku leppida uutes, aga kuna kellelgi polnud aega iga sellise muutuse peale spetsifikatsioone ümber kirjutada, siis

jäidki need suusõnalisteks kokkulepeteks, mis halvimal juhul ununesid, heal juhul mäletas neid kuu aja pärast iga osanik isemoodi. Sellised muutused said tavalisteks peale esimesi integreerimiskatseid, kus tegelikud probleemid alles selguma hakkasid.

Kuigi Cybernetica AS ei tegutse konkreetsete klientide tellimuse alusel, on siiski ka seal vaja planeerida tähtaegadega, mis on tingitud turul toimuvast võidujooksust, paika pandud ürituste plaanidest vms. Paraku aga on nende tähtaegadega olnud ka Cybernetica ASis nii, nagu koskmudeli maailmapraktika näitab.

3.2 Unifitseeritud protsessini jõudmine

Nagu iga teinegi tarkvaraarendusfirma, varustab ka Cybernetica AS oma töötajaid tööks vajalikke erialaseid teadmisi sisaldavate raamatutega. Erinevus teistest tarkvarafirmadest seisneb ehk selles, et Cybernetica ASis propageeritakse töötajaid peale raamatu läbi lugemist loetud raamatust ettekannet tegema. Seda ei ole küll õnnestunud saavutada, et 100% kõigi loetud raamatute kohta ettekanne oleks tehtud, aga mingis ulatuses siiski.

Sarnaselt tegi 1999 aasta talvel raamatust [Royce, 1998] ettekande Arne Ansper, sideturbetoodete tooteliini juht, autoriteetseim tarkvaraarenduse spetsialist Cybernetica ASis. Kuna see ajalugu on dokumenteerimata ja mina sellele ajal veel Cybernetica ASis ei töötanud (mina liitusin selle meeskonnaga 1. augustil 1999), siis ma ei tea rohkem, kas tol korral sellele ka mingit arengut töökorralduse muutmiseks toimus. Tõenäoliselt ainus, Arne Ansper, kes selleks piisavalt pädev oleks olnud, oli liialt hõivatud jooksvate probleemide lahendamise ja arendusprojektide juhtimisel, et ümberkorraldusteks ei jäänud aega.

Kuna Cybernetica ASi infoturbe osakonna meeskond oli viimase paari aastaga suurenenud neljakordselt (5-6 inimeselt umbes 25ni) ja kuna suurema meeskonna juhtimisel hakkas vanast ajast jäänud stiiliga tekkima segadus, siis 1999 sügise esimesel poolel tegeldi Cybernetica ASi infoturbe osakonna struktuuri ümberkorraldamisega, mille käigus käesoleva töö autor määrati tarkvaraarenduse meeskonna juhiks. Autori rolliks pidi olema sellise ressursi, nagu arendusmeeskond, projektide jaoks ette valmistamine ja projektide vahel jagamine. Selle ettevalmistuse hulka kuulus muuhulgas ka arendusvahendite ja metoodikaga tegelemine.

Tunnetades, et arendusprotsessile tuleb rohkem tähelepanu pöörata, tegi Arne Ansper sellest samast raamatust uuesti ettekande 1999 aasta novembris. Ettekande lõpus

arutati, mida edasi peaks tegema, et teema selle seminari järel õhku rippuma ei jääks. Raamat, mida Arne tutvustas, rõhutas, et iteratiivne arendus ei ole ilma heade protsessi toetavate integreeritud ja automatiseeritud töövahenditeta võimalik. Autori jaoks oli sellel ajal veel unifitseeritud protsess võõras teema. Arne oli kursis pisut ka sellega, et unifitseeritud protsessi autoriteedid töötavad Rational Software Corporation'is ning seetõttu kõige metoodilisemalt on iteratiivse protsessi kirjeldamisele ja arendusvahendite väljatöötamisele lähenetud seal. Olles sel teemal pisut arutanud, otsustati, et autor kontakteerub info saamiseks Rational Software'i Soome kontoriga.

Enne aastavahetust jõudsid Arne Ansper ja autor veel käia ühel Rational'i ühepäevasel tutvustusseminaril Stockholmis, kus nad said aru ainult slaididest, sest suuline jutt oli rootsi keeles. 2000 aasta jaanuaris käisid Monika Oit (infoturbe osakonna juhataja), Arne Ansper ja autor privaatsetl ainult neile pühendatud kokkusaamisel Rational'i Soome kontoris, kus neile tehti põhjalik ülevaade sellest, kuidas unifitseeritud protsess (täpsemalt *Rational Unified Process* – Rational Software Corporation'i poolt välja töötatud unifitseeritud protsessi edasiarendus, mis katab protsessi ka töövahenditega) nende toodetud vahenditega kaetud on. Aastavahetuse ajal luges autor läbi raamatud [Fowler, 1999] ja Arne tutvustatud raamatu [Royce, 1998] ning veebruariks kujunes kindel otsus katsetada *Rational Unified Process*'i (RUP). Tellisime Rational Software'ilt vajalikud arendusvahendid.

3.3 Projektid: TrueSign 1.0 ja TrueSign 1.1

3.3.1 Saateks – lühidalt avaliku võtme infrastruktuurist

Digitaalsignatuuride loomiseks ja verifitseerimiseks (kontrollimiseks) kasutatakse võtmepaari, mis koosneb teineteisega matemaatiliste arvutuste abil seotud salajasest ja avalikust võtmest. Elektrondokumendi digitaalsignatuur saadakse elektrondokumendi spetsiaalse algoritmi abil kodeerimisel, kusjuures see algoritm mõjutab kodeeringu tulemust ainult signatuuri looja valduses oleva (teistele mitte teadaoleva) salajase võtme abil. Algoritm ja salajane võti on sellised, et salajase võtme tuvastamine selleks volitamata isikul ainult dokumenti ja selle digitaalsignatuuri teades on praktiliselt võimatu (kiireimad teadaolevad algoritmid võtavad mõttetult kaua aega).

Digitaalsignatuuri verifitseerimiseks kasutatakse salajase võtmega seotud avalikku võtit, mille järgi on samuti salajase võtme tuvastamine praktiliselt võimatu, aga mille abil saab kontrollida, kas digitaalsignatuur on loodud antud elektrondokumendi põhjal või mitte. Selline avalik võti on teada vähemalt neile, kes on huvitatud digitaalsignatuuri õigsusest (digitaalsignatuuri kirjeldatud seosest elektrondokumendiga).

Meil on praegu kirjeldatud kaks subjekti: signeerija ja huvitatud osapool. Suures kommunis (nt. Eesti Vabariik) võib selliseid subjekte palju olla. Selleks, et avalike võtmete levitamine suures kommunis mugavam oleks, kasutatakse spetsiaalset kolmandat rolli, sertifitseerimiskeskust (*Certification Authority – CA*). Lihtsamal juhul on see ainus subjekt, kelle avalik võti peab olema avalikkusele teada ja kelle avalikku võtit avalikkus usaldama peab. Ülejäänud avalike võtmete jaoks annab sertifitseerimiskeskus välja sertifitseerimiskeskuse poolt allkirjastatud sertifikaate, mis kinnitavad mingi avaliku võtme seotust vastava signeerijaga. Keerulisemal juhul võivad erinevad CA moodustada hierarhia, kus osade CAde avalikud võtmed on samuti sertifitseeritud, aga mingi hierarhias kõrgemal asuva CA juures. Siis peab sertifitseerimisteenuse kasutaja usaldama vähemalt tipmise CA avalikku võtit. Tänapäeva CA pakub ka interneti vahendusel *on-line* teenuseid (nt. sertifikaadi tühistusnimekirjad – *Certificate Revocation List – CRL*) kontrollimaks, ega sertifikaati tühistatud ei ole. Kui vastava signeerija digitaalsignatuurist huvitatud osapool selle CA sertifikaate usaldab, siis piisab tal signeerija avaliku võtmega seotud sertifikaadist, et veenduda, et elektrondokumendil on tegu selle signeerija digitaalsignatuuriga.

Sellist infrastruktuuri kutsutaksegi avaliku võtme infrastruktuuriks. Kuigi selles kontekstis räägitakse digitaalsignatuuridest (digitaalallkirjadest), on siiski tänapäeva PKI tavamaailmas kasutatava allkirja väljavahetamisest veel kaugel. Nimelt tavamaailmas kasutatakse allkirju pikaajalise dokumentaalse tõestusväärtuse tagamiseks (nt. Eesti Vabariigis peavad kõik raamatupidamislikku väärtust omavad allkirjastatud dokumendid säilima vähemalt seitse aastat, mõned kauemgi). Praegune PKI aga ei suuda tagada, et digitaalsignatuur oleks usaldusväärne kauem, kui mõneks hetkeks – peale avaliku võtmega seotud sertifikaadi kehtivuse lõppemist (sertifikaadiga on seotud kehtivusaeg) või selle tühistamist (tühistada peab saama võtme omanik suvalisel hetkel, kuna tal peab olema kaitse salajase võtme

kompromiteerumise, st. volitamata isikute kätte sattumise, vastu) ei ole võimalik enam tuvastada, kas digitaalalkiri anti võtme kehtivuse ajal või peale seda. Seega praegune PKI võimaldab digitaalsignatuure kasutada ainult lühiajalise autentimise otstarbel.

Infoturbe osakonna teadlased on juba mõned aastad tegelenud avaliku võtme infrastruktuuri valdkonna uurimisega (*Public Key Infrastructure – PKI*) just sellest aspektist – nad on välja töötanud teoreetilised alused aktsepteeritava ressursinõudlikkusega tehnoloogiale, mis võimaldab tagada digitaalsignatuuride pikaajalise tõestusväärtuse. Rohkem infot sellel teemal pakub näiteks allikas [TrueSign, 2000]

2000 aasta veebruariks oli teadlaste töö ja organisatsiooni tegevus jõudnud olekusse, kus pidi algama selle tehnoloogia demonstreerimist võimaldava tarkvara teostamine.

3.3.2 Projekti valimine

Protsessi katsetamiseks esimese projekti valimisel on propageeritud kahte lähenemist:

1. Rational'i müügiesindajad, et mitte anda välja riskantseid lubadusi, soovitasid seda protsessi alustuseks proovida mõne pilootprojekti peal, millest ei tekiks suurt kahju, kui tähtaegadest kinnipidamine ei õnnestu.
2. Unifitseeritud protsessi teoreetikud/ideoloogid aga soovivad kasutada just kõige riskantsemat projekti, sest iteratiivset protsessi juhivad riskid ja kuna pilootprojekt on juba oma olemuselt ilma reaalse riskideta, siis ei pruugi ettevõtte tippjuhtkond iteratiivse protsessi väärtusest aru saada. Lisaks sellele, kuna selline "mänguprojekt" ei ole prioriteetne, võib sellele ressursside eraldamine raskusi tekitada, kui samal ajal käivad mingid "tegelikud" projektid, millel tähtaegadega raskusi on.

Meil ei langetatud projekti valikul otsust kummagi põhimõtte järgi – valiti ajaliselt esimene projekt, mis polnud veel alanud. Juhuslikult osutus selleks kõige riskantsem – firma edasise tegevuse suhtes võtmetähtsust omava uue toote, Cybernetica ASi teadurite poolt uuritud ja arendatud digitaalsetele signatuuridele pikaajalist tõestusväärtust tagava teooria demonstreeritava tarkvarana väljatöötamine. Riskantne oli see ettevõtmine mitmest küljest:

1. Projekt, mille käigus ei arendata edasi mingit juba olemasolevat tarkvara, vaid tehakse uue tarkvara esimest versiooni, on juba loomu poolest riskantne tegevus – tegu on uue valdkonnaga, mille riskid on veel enamuses avastamata.
2. Sellel valdkonnal on firma edasise tegevuse suhtes strateegiline tähtsus – strateegid loodavad, et selle tehnoloogiaga on lahendatud praeguste digitaalsignatuuride juures veel alles jäänud olulisimad probleemid, mis on takistanud neid kasutada muuks, kui lühiajaliseks autentimiseks, nt. pikaajalise tähtsusega dokumentide allkirjastamiseks (vt. ka [TrueSign, 2000]).
3. Tehnoloogiliselt on tegu olemasolevatele avaliku võtme infrastruktuuridele (PKI – *Public Key Infrastructure*) lisaväärtuse lisamisega, mitte väljavahetamisega. St. väga kesksel kohal on olemasolevate PKI tarkvaratoodetega liidestumise probleemid (standardid, tegelikkus, konkreetsed näited, kasutajate nõuded jne.), mis kõik peidavad endas ootamatuid üllatusi ja vajavad seetõttu nende uurimisel mingit väga kindlat meetodit.
4. See ei too esimesel paaril aastal rahaliselt midagi märkimisväärset sisse.

Projekti algatus viibis alguses mõnda aega, kuna üritati leida projektile unifitseeritud protsessi kogemustega juhti. Kuna teda ei leitud (tõenäoline põhjus on see, et Eestis eelmise aasta alguses neid veel ei olnudki), siis anti see töö autorile – valiti firma sees parajasti selline, kes polnud projekti juhtimisega hõivatud, ja kes oli jõudnud selleks ajaks unifitseeritud protsessi kohta kõige rohkem materjali läbi lugeda.

3.3.3 Protsessi rakendamine TrueSign 1.0 arendusprojektil – õppetunnid

3.3.3.1 Sümptomid

Iteratiivsus. Iteratiivset arendust tegelikult ei toimunud. Kuigi me üritasime rääkida projektist unifitseeritud protsessi terminites, nagu faasid ja iteratsioonid, ei toimunud *sisuliselt* itereerimist – algatusfaasis tegeldi *ainult* nõuete spetsifitseerimisega, detailimisfaasis tegeldi *ainult* arhitektuuri projekteerimisega jne.

Detailimisfaasi iteratsioonid. Plaani järgi pidi detailimisfaasis olema kaks (vajadusel kolm) iteratsiooni, kus iga iteratsiooni sisuks pidid olema vastavate kasutusmallidega

tegelemine. See ei läinud nii – kuna oli tegu uue metoodikaga nii protsessi, visuaalse modelleerimise keele, kui töövahendite mõttes, ei saanud arhitekt, ega ka autor täpselt aru protsessi kirjeldustest, mida konkreetselt mingil sammul teha tuleb, ning tegelikkuses läks see faas kontrolli alt välja – tegevused ei toimunud plaanitud järjekorras, esimese iteratsiooni kasutusmallide valiku asemel projekteeriti sisuliselt ikkagi tervet süsteemi, mistõttu iteratsioon lõppes plaanitud faasi lõpust nädal aega hiljem ja rohkem iteratsioone selles faasis ei toimunud.

Konstrueerimisfaas. Tänu tarkvara arhitektuuri visuaalsele objektorienteeritud mudelile oli programmeerimistegevusi kergem eristada (tükeldada) ja nende loogilist järjekorrad graafi kergem plaanida, kui varem. Plaanitud sai kolm iteratsiooni tegevusi ning programmeerimisülesannete kestvuste ennustused baseerusid programmeerijate isiklikul tunnetusel. Faas väljus kontrolli alt, kulutades ära plaanitud 15 inimkuu asemel 34 inimkuud tööjõu ressursi – erinevus (*hic!*) 227%. Kuna kõik programmeerimistegevused juba faasi alguses võtsid umbes kaks korda rohkem aega, kui plaanitud, siis oli juba alguses näha, et iteratsioon plaanitud ajaks ei lõppe ja kartuses kulutada uue arendusprotsessi tundmaõppimisele liialt palju programmeerijate aega, kujunes ka selles faasis välja plaanitud kolme iteratsiooni asemel üks ühtlane etapp.

Projekti tähtaeg ja kulutatud ressurss. Projekt hilines plaanitud ühe kalendrikuu võrra (kestis septembri lõpu asemel oktoobri lõpuni). Kogu projekti tegelik töömaht oli vaatamata kokkuhoiule viimases faasis plaanitud 34 inimkuu asemel 47,5 – erinevus 13,5 inimkuud – 28%.

Muutused plaanis. Olid enamasti halvad – seisnesid lõpu edasilükkamises TrueSigni tehnoloogia demonstreeritavuse huvides (tarkvarale esitatavad nõuded täideti peaaegu kõik). Igasugustele muudatusettepanekutele reageeriti konstrueerimisfaasis liiga kiiresti – ei analüüsitud piisavalt muudatusettepaneku sisu ja olulisust, põhjustades sellega liigseid ümbertegemisi. Testimise käigus esile tulevaid probleeme lahendati käigu pealt, mis ei andnud aega enne probleemi lahendamist hinnata tõsidust ja lahendamisele minevat ajakulu.

3.3.3.2 Põhjused

Konstrueerimisfaasis olnud ülesannete lahendamiseks intuitsioonile tuginedes pakutud ajahinnangud osutusid ebarealistlikeks. Kui programmeerija käest küsida, kui

kauda tal võtab aega mingi antud omadustega klassi programmeerimine, siis ta ei pruugigi valessti vastata eeldusel, et tal selle programmeerimise käigus ootamatuid probleeme ei teki. Paraku osutub, et vähemalt teine samapalju aega, mis kulub puhtalt programmi koodi kirjutamisele, kulub ka mingite kompileerimise ja linkimise probleemidega tegelemisele, kolmanda partei komponentidega seotud integreerimisprobleemidele, mingitele enda tehtud programmivigade otsimisele jne.

Süsteemiseeritud testimise ja muudatusettepanekutele reageerimise korraldamine eeldanuks kontrolli all olevat konstrueerimisfaasi.

Arendus ei toimunud üldises mõttes iteratiivselt (iga faasi igas iteratsioonis tuleb tegeleda kõikide töövoogudega ja iga iteratsiooni lõpus peab demonstreerima mingit käivitavat arhitektuuri osa, või selle kandidaati algatusfaasis) selle tõttu, et selle projekti juhtimisega seotud uusi aspekte ja ootamatuid probleeme oli palju ja need juhtisid autori tähelepanu arenduse iteratiivsest planeerimisest ja selle mõistmisest eemale – uut, mida korraga jälgida, oli liiga palju. Tulemus oli see, et töö toimus praktiliselt koskmudeli põhimõttel.

Ennustamisoskuse puudumise tõttu konstrueerimisfaasi kontrolli alt väljumise näol oli tegu liiga hilja (materialiseerumise hetkel) avastatud materialiseerunud riskiga, mida oleks saanud avastada ja ennetada, kui oleks arendusele lähenenud iteratiivselt – oleks lasknud programmeerijatel juba varajases staadiumis proovida mõnd mudeli elementi programmeerida, mis omakorda oleks andnud informatsiooni selle kohta, kui kiiresti programmeerijal võtab mingi teatud omadustega spetsifitseeritud ülesande täitmine aega.

3.3.3.3 Lahendused

3.3.3.3.1 Protsessi juurutamine

Peamine viga, mis tehti ja millest kõik teised hädad alguse said, oli see, et ei uuritud enne projekti algust seda, kuidas RUPi organisatsioonis rakendamist organisatsiooni iseärasusi arvesse võttes läbi viia. RUP nimelt räägib ka sellest. Põhjus, miks ei uuritud, oli selles, et seda temakohast materjali, mida läbi töötada tuli, oli väga palju ja enne projekti algust polnud piisavalt aega, et ka seda aspekti endale selgeks teha. Ilmselt poleks seda viga tehtud, kui oleks olnud tol ajal kasutada mõnd kogenud konsultanti, kes teab, millest alustada tuleb, sest kogenud inimese abil on esimeste

sammude tuvastamine suurusjärgu võrra kiirem, kui kogenematul suures inforägastikus üksikut vihjet otsides.

Protsessi juurutamisel on mitu aspekti:

- **Organisatsiooni hetkeolukorra hindamine.** Kõigis RUPi poolt pakutavates juurutusstsenaariumites tuleb organisatsiooni tarkvaraarenduse hetkeolukorda hinnata ja kaardistada selle suurimad probleemid. Seda Cybernetica ASis süstemaatiliselt ei toimunud. Deklareeriti lihtsalt, et olemasolev protsess ei rahulda, kuna projektid venivad ja ei püsi eelarves. Tulemus oli see, et keskendumus suurimatele sisulistele probleemidele oli hajutatud. Näiteks selgus autori jaoks alles projekti konstrueerimisfaasis, et programmeerija programmeerimiskiiruse ennustamiseks ei saa kasutada programmeerija enda ütlust vaid selles saab usaldusväärset tugineda ainult kogutud statistikale.
- **Protsessi ja töövahendite inkrementaalne juurutamine.** Et projektis osalevad inimesed ei oleks liialt ülekoormatud liiga paljude uute faktoritega korraga, tuleb protsessi ja vahendeid juurutada vähehaaval, mitte korraga. See haakub ka eelmise punktiga – sedasi on võimalik plaanida protsessi juurutamist nii, et kõige problemaatilisemad kohad saaksid esimesena lahendatud. Cybernetica ASis sellist plaani ei peetud, et protsessi võiks kuidagi sammhaaval rakendada, vaid lihtsalt asuti projekti kallale ja püüti protsessist ilma valimata aru saada niipalju, kui võimalik. Tulemus on sama, mis eelmises punktis. Lisaks eeldavad ühed protsessi osad teisi, ja seega suure tõenäosusega tehti osade asjade kallal ka liigset tööd.
- **Väljatöötusjuhtumi (*Development Case*) kasutamine.** Väljatöötusjuhtum, tehis, mis käib iga projektiga kaasas, on mõeldud just projektiga seotud protsessi kajastamiseks ja konfigureerimiseks ning on seega sisendiks projekti plaanimisel ja juhtimisel. Cybernetica ASis seda ei rakendatud. Tulemus on see, et projekti plaanimises on protsessi vaade selgelt esile tõstmata, põhjustades projekti märkamatu (märkamatu selles mõttes, et tähelepanu on muude esiplaanil olevate probleemide tõttu hajutatud) kõrvalekaldumist protsessist ja seega suurendades projekti käitumise ennustamatuse määra.

Kõik see kokku tähendab, et kulutati väikese efektiivsusega kallist aega protsessi parandamisele samal ajal, kui seda oleks saanud teha suurema efektiivsusega.

3.3.3.3.2 Tulevaste projektide tarvis statistiline analüüsimine

Et programmeerijate töökiiruse probleem ei oleks järgmiste projektide jaoks enam nii suur risk, kui ta seni oli olnud, kogus autor TrueSign 1.0 arendusprojekti pealt vastavaid statistilisi andmeid, mida edaspidi kasutada. Statistilise analüüsime tulemus võib lisaks realistlike ajagraafikute planeerimisele olla sisendiks ka uute ja kiiremate programmeerijate töövõtete väljatöötamisel. Samas järgmise projekti plaanimisel arvestada seda, et järgmises projektis kiiremini tegutsetakse, ei saa, kuna pole teada, kui palju see efektiivsus tõuseb – eelmise projekti tulemusest latti kõrgemale asetades toetume ainult intuitsioonile, mis on riskantne.

Konstrueerimisfaasi on vaja detailselt planeerida enne konstrueerimisfaasi algust, st. konstrueerimisfaasi ressursinõudlikkust peab suutma hinnata nende tehiste põhjal, mis on olemas detailimisfaasi lõpuks. Selleks võttiski autor programmeerimise ajakulu ja arhitektuuri mudeli vahelise seose uurimise aluseks arhitektuuri mudeli selle versiooni, mis oli olemas detailimisfaasi lõpus enne konstrueerimisfaasi algust. Sellest mudelist vaatlesin 15 komponenti:

- Kokku 76 klassi, millel oli kokku 295 meetodit. Nende klasside programmeerimiseks läks kokku aega 151 inimpäeva.
- Keskmiselt oli igas klassis selleks ajaks projekteeritud 3,88 meetodit.
- Vaadeldud komponentide programmeerimisele kulunud aeg konstrueerimisfaasi alguseks projekteeritud klasside vahel jaotatuna oli 1,99 päeva 1 klassi kohta.
- Sama aeg konstrueerimisfaasi alguseks projekteeritud meetodite vahel jaotatuna oli 0,51 päeva 1 meetodi kohta.

Illustreerimaks seda, kuidas tegelikkus ennustustest erines, näitab Tabel 2 konstrueerimisfaasi alguses tehtud ennustusi tegelikkusega võrreldes.

| Komponendi ehk programmeerimisülesande nimi | Esialgne töömahu ennustus päevades | Tegelik ajakulu päevades |
|--|---|---------------------------------|
| Misc. classes | 3 | 5 |
| Protocol request library | 3 | 5 |

| | | |
|---------------------------|----|------|
| Certificate library | 1 | 4 |
| Cryptographic library | 2 | 6 |
| Document library | 10 | 2×11 |
| Timestamp library | 10 | 28 |
| Merkle tree | 3 | 2 |
| Authenticated search tree | 5 | 10 |
| Round info | 1 | 7 |
| jne. | | |

Tabel 2. Konstrueerimisfaasi töödele kuluva ajakulu ennustus vs. tegelikkus.

Autor uuris ka, millele kulub programmeerijate aeg. Selleks lasi ta igal programmeerijal vabas vormis kirjutada logi, kus iga kirje sisaldab mingit tegevust ja sellele kulunud aega. “Näide 1” on näide sellest, mida kirjutati. Kuna selline logi kirjutamine oli pisut tülikas, siis toimis see lühikest aega (paar nädalat), aga mingi arvudes väljendatava aimduse, kuhu programmeerijate aeg kulub, sellest sai.

Joonis 11 näitab seda tulemust tulpdiagrammil. Vertikaalteljel on eristatud järgmised tegevuste tüübid:

- Koodi kirjutamine – siia alla kuulub nii uue koodi kirjutamine, kui vana parandamine. Ülesanne on teada, sellest aru saamisele kuluv aeg on eraldatud teistesse tegevuse tüüpidesse.
- Teistega suhtlemine – siia alla kuulub kogu vabas vormis toimuv kommunikatsioon töö asjus. Nt. arhitektiga suhtlemine, et arhitektuurist aru saada, teiste programmeerijatega suhtlemine, teiste programmeeritud komponentide asjus, mida endal kasutada tuleb, ja enda programmeeritud komponentide asjus, mida teistel kasutada tuleb, ning omavaheline üldhariv õpetamine. Vabas vormis suhtlus on suusõnaline vestlus (ka telefonitsi) ja vestlus meilitsi.
- Muu 3. partei – siia alla kuuluvad väljaspool Cybernetica ASi toodetud teekide jms. arendusvahendite tundma õppimine, va. OpenSSL. Tema jaoks on eraldi tegevusetüüp välja toodud.

- Kompileerimine – erinevate tootjate teekide kompileerimine ja kokku linkimine enda rakendustega on alati toonud esile hulga probleeme, mille lahendamine võtab aega.
- Standardid – kuna projekti käigus valmib toode, mis peab olemasoleva PKI tarkvara tavadega võimalikult palju arvestama, siis tuleb neid tavasid ka tundma õppida – sellele kulub samuti aega. Konstrueerimisfaasis küll vähe, sest selleks ajaks on suurem osa asju juba selged, aga siiski.
- OpenSSL – OpenSSL on C teek, mida me kasutame krüptoprimitiivide kogumina, ja kuna see on meie jaoks väga oluline, samas keerulise ülesehitusega, pidas autor vajalikuks tuvastada, kui palju ikkagi selle käsitlemisega seotud probleemide lahendamisele aega kulub.
- Raport – et autor oleks asjade käiguga pidevalt kursis, kirjutasid igal hommikul programmeerijad talle aruande sellest, mida nad “eile” tegid, mida nad “täna” kavatsevad teha ja mida nad “homme” kavatsevad teha.
- Ülesanne – siia alla kuulub aeg, mis kulus ülesandega tutvumisele.
- Projekteerimine – siia alla kuuluvad arhitektuuri detailimisega seotud tegevused. Need leidsid aset, sest paljud arhitektuuriga seotud kajastamist vajavad nüansid selguvad alles teostamise käigus.
- Projektiväline – aegajalt tuli mõnel projekteerijal ka projektiväliste asjadega tegeleda. Nt. Cybernetica ASi infoturbeosakonna Tartu labori arvutivõrgu toimimise eest vastutavad programmeerijad (neid on kaks) pidid mõnikord võrguprobleeme lahendama, arhitekt juhtis enne TrueSign 1.0-is osalemist ise üht teist projekti ja seetõttu pidi aegajalt sellega veel tegelema jne.
- Dokumenteerimine – kuigi arhitektuur on kõik mudelina kirjeldatud, tuli peale komponendi programmeerimist mõnda asja kommentaaridega varustada, et tagada teistele kindlam arusaamine asjast.

Joonis 11 näitab, et tegevused, mis ei ole koodi kirjutamine, võtavad üksikuna väga vähe aega. Samas näitab Joonis 12, et kõik kokku võtavad nad peaaegu teise samapalju aega ära, kui koodi kirjutamine.

Autor spekulereib, et sümptomites kirjeldatud 227% erinevus ennustatud ajakulu ja tegeliku ajakulu vahel peitubki selles, et programmeerija intuitsiooni järgi hinnates

arvestab ainult koodi kirjutamise tegevusega. Toodud statistika järgi kulub küll koodi kirjutamisele rohkem, kui pool ajast, aga see statistika on koostatud ainult konstrueerimisfaasi lõpus väikse osa (paarinädalase perioodi) põhjal – konstrueerimisfaasi lõpupoole oli programmeerijate jaoks tundmatuid faktoreid vähem, kui alguses. Kui me eraldaksime toodud statistikas koodi kirjutamisest muude tegevuste hulka veel silumise tegevusi, millega samuti intuitsioon ei kipu arvestama, läheneks muude tegevuste osakaal arvatavasti veelgi koefitsiendile 2,27.

Et saada vihjeid, kuidas kiirendada programmeerimist, otsis autor ka seoseid komponendi programmeerimise kiiruse ja muude tegevuste osakaalu vahel. Midagi uut ei leidnud, küll aga sai kinnitust kolmele intuiitselt aimatavale seosele:

- Aeglasemalt valminud komponentide omapäraks oli keeruline siseehitus (peaasjalikult need teadlaste poolt välja mõeldud uued matemaatilised konstruktsioonid): timestamp library – 2,55 päeva/meetod, timestamp klass doclib-is – 1,10 päeva/meetod, authenticated search tree – 1 päev/meetod.
- Aeglasemad programmeerijad suhtlesid teistega vähem – iga vähem suheldud 1 minuti kohta programmeeriti 25 minutit kauem. Autorile pole paraku selge, milline on siin põhjuse ja tagajärje suund. Autor kahtlustab, et kindlat vastust siin ei olegi – olenevalt olukorrast võib see suund nii või naa olla.
- Suurem segadus oli komponentidega, mille täpsem modelleerimine oli edasi lükatud. Nt. sertifikaadi *on-line* tühistusteenuse serveripoolne komponent: 5,25 päeva/meetod.

*11.07 10.08 Asun kahte TODO'd tegema - üks exception ilusaks ja üks korduv kood eraldi funktsiooni

*11.07 10.23 Muudatused tehtud, testitud, kommititud

*11.07 10.24 Asun raportit kirjutama

*11.07 10.47 Raport kirjutatud ja teele saadetud

*11.07 10.48 Asun mudelit uurima CertUpdateMgr'i asjus, et mis klasse mul täna vaja veel teha on.

*11.07 10.49 Helistab Margus. PKIHeader'i destructor astub pange. Na"eb ma"lusodimise moodi va'lja. Nyid la"heb pool tundi sellele, et uurin asja oma testprogrammiga. (rahulikult saab to"o"d teha vaid o-htuti ja na"dalavahetustel)

*11.07 10.59 ko~ne Margusega lo~ppes

*11.07 11.00 asun puuke trackima

*11.07 12.15 suht edutu olen olnud. minu testkood funktsioneerib. asun Marguse koodiga proovima.

*11.07 13.16 jah. Marguse kood kra"shib yhes kindlas kohas ja sigsegv tuleb minu destruktorist. Uurin.

*11.07 13.32 tuli Terminus ja la"ks la'buks. Kasutan pausi ja la"hen so"o"ma

*11.07 13.59 Naasesin so"o"mast, Majas on vaikus. Hakkan to'o"le

*11.07 14.36 Margus kirjutab, et leidis vea yles. Hingame kergendatult. Ajan Meelisega Juttu

*11.07 14.47 jutuajamise ka"igus leiame doclibist bugi yles

*11.07 14.48 asun oma asjade kallale

*11.07 14.49 Otsustan et olen va"sinud ja puhkan 10 minti.

*11.07 15.16 Puhkamise ka"igus sai doclibist veel bugisid leitud ja neid tuleb nyid parandama hakata. SignerInfos on paar kysitavust.

*11.07 17.34 Uskumatu, aga siiamaani olen vahetpidamata koodi kirjutada ragistanud. OK. vahepeal oli kymneminutiline ekraanijo~llitamise paus...

*11.07 17.34 Ja"tkan tegutsemist

*11.07 18.40 Asi valmis. kohe saab kommititud. Siis uimerdan natuke siin, vaatan, kas see lahendas Meelise mured. Kui ei siis rabelen veel. Kui jah, siis la"hen puhkan ja tulen hiljem tagasi. Vahepealset to"o"tegemist segas yks kymneminutiline telefoniko~ne vanalt so~bralt.

*11.07 20.32 Olen tagasi. Vaatan nyid autentimisteed yle. Nende ASN struktuur on natuke naljakas ja tahab kohendamist.

*11.07 21.50 Vaidleme Meelisega yhe SInfo jupi yle. Osutub et olen teinud vea

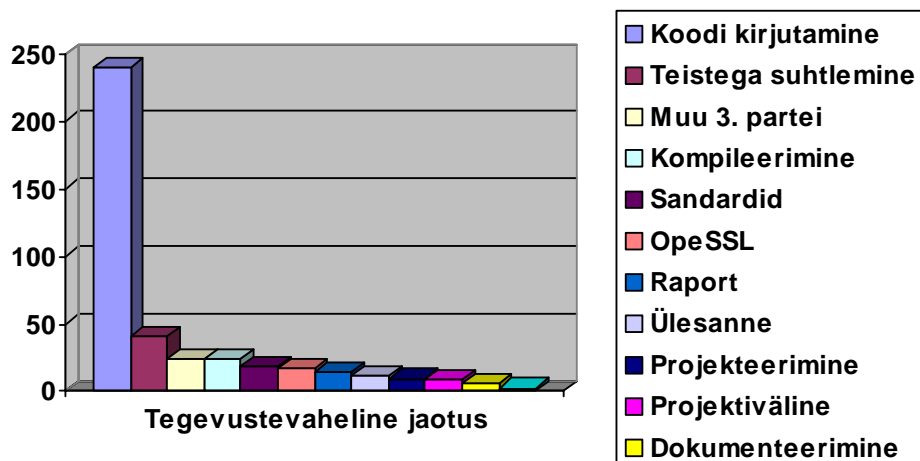
*11.07 22.12 Lo~petame vaidluse ja ma asun viga parandama. Enne lo~petan AuthPath asjad.

*11.07 22.23 SInfo bug osutus lihtsaks (Ma olin OpenSSL'i koodist valesti aru saanud). Parandus on valmis ja kohe commitin.

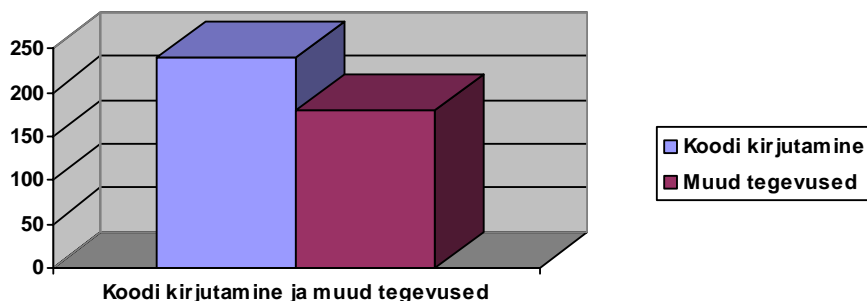
*11.07 22.30 Kommititud. Vahepeal kompileeris ja mina uimerdasin niisama. Nyid loen natuke meili.

*11.07 22.40 Ja nyid la"hen koju. Natuke nukker tunne on, aga ega ma ta'na enam midagi kasulikku teha ei jaksaks.

Näide 1. Ühe programmeerija vabas vormis kirjutatud päevane tegevuste logi.



Joonis 11. Programmeerijate ajakulu (päevas keskmiselt kulutatud minutite arv) jaotus tegevuse tüüpide vahel konstrueerimisfaasis.



Joonis 12. Muude tegevuste ajakulu (päevas keskmiselt kulutatud minutite arv) osakaal koodi kirjutamise suhtes.

3.3.4 TrueSign 1.1 arendusprojekt

3.3.4.1 TrueSigni 1.1 sisuline eesmärk

Taustainfoks kirjeldab lühidalt, millised on TrueSigni arendusega seotud seniste projektide sisulised eesmärgid.

TrueSign 1.0 strateegiline eesmärk oli tehnoloogia demonstratsioon. Selleks loodi üks lihtne demonstreerimise otstarbel kasutatav sertifitseerimiskeskuse tarkvara, TrueSigni server, mis koosneb notari ja ajatempli serverist, *on-line* sertifikaadi tühistusteenuse server, ning demonstreerimise otstarbel kasutatav TrueSigni kliendirakendus. CA tarkvara väljastab veebiliidese abil avaliku võtme sertifikaate ja publitseerib need muuhulgas TrueSigni notari serverile. Kliendirakendusega saab allkirjastada dokumendi ning saata digitaalsignatuuri koos sertifikaadiga notari

serverile, kes saadab vastuseks allkirjastatud ja ajatembeldatud kinnituse selle kohta, kas esitatud sertifikaat kehtis hetkel, mil notarile allkiri esitati, või mitte.

TrueSign 1.1 eesmärgiks on tootestada TrueSigni poolt pakutav tehnoloogia, kui teenus. St. teha TrueSign selliseks, et selle poolt PKI-le lisatavad teenused oleksid kasutatavad ka teistele, mitte ainult Cybernetica ASile. Selleks esiteks tuli stabiliseerida TrueSigni komponentide omavaheliseks suhtluseks kasutatav protokollistik, et TrueSigni edasine arendus ei hakkaks segama vanematest versioonidest pärit TrueSigni komponentide tööd (nt. siis kui vana versiooni kliendiga loodud signeeritud dokumendil olevat notari kinnitust soovitakse pikendada uue versiooni notari serveri juures). Teiseks, et võimaldada TrueSigni teenuse kasutamist kõikvõimalikes infosüsteemides jms. rakendustes, tuleb luua mingi korralikult spetsifitseeritud API (*Application Programming Interface* – rakenduse programmeerimise liides). Esimeses lähenduses otsustasime TrueSign 1.1 raames kirjutada C++-i teegi.

Seega, TrueSign 1.1 projekti sisuks on

- klienditeegi projekteerimine ja teostamine,
- protokollistiku spetsifitseerimine ja serveritarkvara uue protokollistiku järgi töötama panemine ning
- kliendirakenduse muutmine selliseks, et ta kasutaks oma ülesannete sooritamiseks väljatöötatud APIt.

3.3.4.2 TrueSign 1.0 kogemused rakendatuna TrueSign 1.1-s ja uued õppetunnid

Kuna TrueSign 1.1 on alles lõppemata projekt (lõpp on plaanitud juuni alguseks), siis pole paljud tulemused veel teada, aga niipalju, kui praeguseks näha on, saab siin kirjeldada.

3.3.4.2.1 Iteratiivne arendus

Selle projekti algatamisel tehtud suurim viga oli see, et algatusfaas jäeti ära. Kuna seda ei olnud, siis ei olnud enam ka aega plaanida täpselt, kuidas detailmisfaasis itereerima hakata, ja seetõttu kukkus detailimisfaas jällegi välja selliselt, et ilma demonstreeritavate teostatavuse katseteta projekteeriti kogu uus klienditeek valmis.

Külla aga toimus detailimine ise seekord kahe iteratsioonina, kus esimeses iteratsioonis lisaks projekteerimisele selgus ka hulk projekteerimistöid, mida alguses ei osanud ette näha, ja mis siis läbiti enne konstrueerimisfaasi eraldi iteratsioonina.

Konstrueerimisfaasis toimus kolm iteratsiooni (praegu on käimas kolmas), kus esimeses iteratsioonis teostati API see osa, mis on seotud dokumendi nende tegevustega, mis ei vaja TrueSigni serveriga suhtlemist (dokumendi loomine, signeerimine, verifitseerimine jms.). Teises iteratsioonis teostati API see osa, mis eeldab serveriga suhtlemist (notari kinnitused, ajatemplid), ning alustati kliendirakenduse integreerimist uue APIga. Kolmandas iteratsioonis toimub kliendirakenduse klienditeegi ja serveritarkvara integreerimine.

Nagu näha, on toimunud mingid itereerimise alged faasisiseselt, aga kogu projekti ulatuses toimib ikka koskmudel – kõigepealt arhitektuuri projekteerimine, siis alles teostamine. Nähes praeguseid integreerimisega seotud probleeme (teatavate kiipkaartide liidestamise eriomadused, kasutatava kommerts-CA tarkvara eriomadused jms.) arvab autor, et nende probleemidega oleks osatud arvestada juba detailimisfaasis, kui oleks toimunud mingeid tehnilisi nõudeid tuvastavad integreerimiskatsed, aga neid ei toimunud, kuna neid ei plaanitud.

Siit moraal – kuigi olemasoleva tarkvara uue versiooni arenduse ülesande püstitus võib tunduda selge ja riskivaba, ei tohi projekti algusest ära jätta algatusfaasi, mis on just ette nähtud kui sissejuhatus projekti riskidest juhitud plaanimisele. Ka autoriteedid kirjandusest väidavad, et see algatusfaas võib tarkvara edasiarenduste puhul küll lihtsam ja kergekaalulisem olla, aga ta peab ikkagi olema.

3.3.4.2.2 Kasu TrueSign 1.0 konstrueerimisfaasi statistikast

TrueSign 1.1 konstrueerimisfaasis sai tegevuste ajakulu planeeritud TrueSign 1.0 konstrueerimisfaasi statistilise analüüsi põhjal. Tulemuseks on see, et töömahtude hinnangud vastavad seni väga täpselt tegelikkusele. Küll aga on selle projekti puhul olnud probleeme konstrueerimisfaasieelsest paika pandud ajagraafikust kinnipidamisega kahel põhjusel:

- Programmeerijate tööjõud ei olnud saadaval kogu aeg sellises ulatuses, nagu alguses plaaniti – osad programmeerijad suunati juhtkonna poolt tegelema teiste projektidega. Sellistest ootamatustest üle saamine eeldaks juba arendusprotsessi kontrolli alla saamist juhtkonna tasemel – peab olema

defineeritud, kuidas käib projektidele ressursi jagamine ja millistel alustel seda muuta saab jne. Hetkel Cybernetica ASis käib see lihtsalt turundusosakonna poolt tuleva nõudluse, juhtkonnapoolse intuiitvise prioriteeditunnetuse ja läbirääkimiste alusel. Juhtkonna tasemel arendusprotsessi defineerimisest, mille poole praegu Cybernetica ASis liigutakse, räägib järgmine peatükis.

- Autor jagas idealistlikult inimkuudes antud töömahu hinnangu inimeste arvuna antud tööjõu hulgaga ja sai nö. kalendri aja, millele täpselt vastavalt sai defineeritud iteratsioonide tähtajad. Tõde on paraku see, et reaalsuses ei õnnestu töid nii palju paralleelselt teha, et kogu programmeerijate meeskond oleks koormatav antud iteratsiooni ülesannetega, eriti iteratsiooni lõpus, kus on jäänud teha veel vähe, aga teineteisest sõltuvaid asju. See põhjustab selle, et kuigi programmeerijate koormatuse huvides saab neid koormata ülesannetega järgmisest iteratsioonist, siis jooksva iteratsiooni lõpp võtab rohkem aega, kui plaanitud. Selle asjaolu tõttu on soovitatav plaanida ajagraafikut mingi ajavaruga, mille täpselt tuvastamiseks autor praegu veel ei oska kindlat meetodit välja pakkuda. Tuvastanud on ta ainult selle, et püüd iteratsiooni alguses iteratsiooni kulgu ära aimata tegevuste sõltuvuste ahela ja ajahinnangute abil, ei toimi, kuna
 - tegelikkuses on paljud ülesannetevahelised sõltuvused ignoreeritavad ja
 - kasutatavad statistilised ajahinnangud on keskmised – üksikute ülesannete suhtes on tegelikkuses kõikumine väga suur lüües faasi alguses ajateljele paigutatud tegevuste sõltuvusgraafi segi.

4 Tarkvaraarenduse protsessi rakendamine kogu organisatsioonis

See peatükk räägib kõigepealt sellest, milline võiks üldse olla ühe tarkvarafirma eesmärk oma arendusprotsessi juhtides, ja milline on maailmatasemel tunnustatud skaala tarkvarafirmade arendusprotsessi tasemete hindamiseks. Seejärel kirjeldab, mida parajasti Cybernetica ASis selles osas tehakse ja mida peaks tegema.

4.1 *Protsessi muutuse mõju*

Protsessi muutmine võib olla raske ja selle tegelike tulemuste nägemine võib võtta aega. Mingi uue töövahendi kasutuselevõtt on suhteliselt lihtne – see tuleb paigaldada, lugeda kasutajajuhendit, teha läbi mõni näide ja võibolla osaleda mõnel kursusel. Üleminek võib kesta mõnest tunnist mõne nädalani. Samas tarkvara väljatöötamisprotsessi muutmine tähendab tihti inimeste fundamentaalsete uskumuste ja väärtushinnangute muutmist. See on kultuuri muutmine, mis on oma loomuselt küllaltki poliitiline või filosoofiline.

Protsessi muutus mõjutab inimesi ja organisatsiooni rohkem sügavuti, kui tehnoloogia või vahendite muutmine. Seda tuleb plaanida ja juhtida hoolikalt. Tuleb tuvastada muudatuse väljavaated ja oodatav kasu, teha see huvitatud osalistele selgeks, tõsta nende teadlikkuse taset ja siis järkjärgult liikuda seniselt tööpraktikalt uuele.

Protsessi juurutades tuleb tegeleda järgmiste asjadega:

- Inimesed ja nende kompetents, oskused, motivatsioon ja suhtumine: igaüks peab olema adekvaatselt koolitatud ja motiveeritud.
- Toetavad töövahendid: uute vahenditega vanu välja vahetades tuleb neid olukorra jaoks kohandada.
- Tarkvara väljatöötamise elutsükli mudel, temaga seotud organisatsiooniline struktuur, tegevused ja tööpraktika koos tekkivate tehistega.
- Tarkvara väljatöötamisprotsessi tegelik kirjeldus.

Lisaks neile, on veel asju, mis mõjutavad seda, kuidas inimesed töötavad. Näiteks füüsiline töökeskkond, organisatsiooniline kultuur ja poliitika ning premeerimise süsteem.

Lisaks protsessis osalevatele inimestele, tuleb arvestada ka inimestega väljastpoolt organisatsiooni, keda need muutused mõjutavad:

- Juhid, kes on tarkvaraarendusorganisatsiooni töövõime eest vastutavad, peavad aru saama, miks protsessi muudetakse ja miks rakendatakse uusi vahendeid. On oluline, et nad saaksid aru, kas ja kuidas toimub progress. Igal protsessi parandamise projektil peab olema tippjuhtkonna heakskiit. Juhtkond peab aru saama, et protsessi muudatus on tasuv ja et eesmärkide saavutamist tuleb hoolikalt juhtida.
- Võibolla on vaja informeerida kliente, et organisatsiooni protsess on muutunud, kuna see võib mõjutada, kuidas ja millal nende sisendit käsitletakse.
- Muutusest võivad olla mõjutatud ka muud tarkvaraarendusorganisatsiooni osad. Mõnikord võib muudatus ühes osas põhjustada vastureaktsiooni ja skeptitsismi teistelt organisatsiooni osadelt. Põhjus on tihti selles, et nad ei saa aru muudatuse põhjustest. Isegi, kui nad ei saa sellesse otseselt sekkuda, võib see põhjustada poliitilisi probleeme.

Sellest, mida projektijuhi jaoks tähendab üleminek koskprotsessilt iteratiivsele ja mis ohud selle ülemineku juures varitsevad, saab lugeda allikast [Waterfall to Iterative, 2000].

4.2 Protsessi arendamine ja kujutamine CMM-is

Järgnev jutt on peamiselt suhteliselt vaba ja lühendatud tõlge raamatust [Royce, 1998]. CMM-i küsimustiku vastustes on kohati asendatud Walker Royce'i nägemuse sellega, mida *Rational Unified Process* (RUP) ja *MS Project* pakuvad.

Sellest, kuidas RUPi abil CMMi skaalale ennast kujutada, võib lugeda ka allikast [CMM].

4.2.1 Ülevaade CMM-ist

Suutvusküpsuse mudel (*Software Engineering Institute's Capability Maturity Model – SEI CMM*) on levinud mall tarkvaraarendusprotsessi küpsuse hindamiseks. CMM defineerib arendusprotsessi küpsusele viis taset, mis näitavad, milliseid protsessi võtmekohti (*key process areas – KPAs*) vaadeldav organisatsioon toetab:

1. Algne tase – KPA-sid pole.
2. Protsess on korratav. 2. taseme KPA-d: nõuete haldus, projekti planeerimine, tarkvaraprojekti jälgimine ja ülevaadete tegemine, alltöövõtude haldus, tarkvara kvaliteedikontroll, tarkvara konfiguratsioonihaldus.
3. Protsess on defineeritud. 3. taseme KPA-d: organisatsiooni fookus protsessil kui tervikul, organisatsioonipoolne protsessi defineerimine, koolitusprogrammid, terviklik tarkvaraarenduse juhtimine, tarkvaratoote arendus, gruppidevaheline koordineerimine, vastastikused läbivaatused.
4. Protsessi juhitakse. 4. taseme KPA-d: protsessi hindamine ja analüüs, kvaliteedijuhtimine, toote defektide ärahoidmine.
5. Protsessi optimeeritakse. 5. taseme KPA-d: tehnoloogia innovatsioon, protsessi muudatuste haldus.

Enamik edukusele pretendeerivaid tarkvarafirmasid peaksid endale eesmärgiks seadma protsessi 3. CMM taseme. Arendusprotsessi suutlikkuse hindamise (*software capability evaluation – SCE*) käigus tavaliselt tuvastatakse, kas “organisatsioon ütleb, mida ta teeb ja teeb, mida ta ütleb”, uurides selleks organisatsiooni sisepoliitikat ja projektide praktikat. Neid hinnatakse KPA raamistiku järgi. Hindamisprotsess ei ole perfektne, aga hea suhteline indikaator.

Tüüpiline SCE kasutab täieliku auditi osana SEI protsessi küsimustikku (*SEI Maturity Questionnaire*). Hindamine sisaldab, detailset analüüsi, intervjuusid ja muid võtteid. Küsimustikku kasutatakse tavaliselt hindamise alguses konteksti tekitamiseks.

Tarkvaraorganisatsioonide jaotuse kohta CMM'i tasemete järgi on tehtud palju erinevaid uuringuid. Tabel 3 annab umbkaudse pildi tarkvaraorganisatsioonidest 1995 aastal.

| CMM tase | Sagedus | Käitumise tase |
|-----------------|----------------|-------------------------------|
| 1 Algne | 70% | Ennustamatu, kõrge risk |
| 2 Korratav | 15% | Raskustes, aga jäävad ellu |
| 3 Defineeritud | <10% | Stabiilne, ennustatav, arenev |
| 4 Juhitav | <5% | Väga kindel, usaldusväärne |

Tabel 3. Tarkvaraorganisatsioonide jaotus CMM-i skaalal aastal 1995.

4.2.2 CMM-i puudused

1. Üks peamisi SEI CMMi puudusi on see, et KPA-d fookuseeruvad peamiselt klassikalises protsessis eksisteerivatele dokumendi kujul tehisele (*artifacts*): arhitektuur, nõudmised, alltöö lepingud, lepingud, plaanid ja raportid. Väga vähe KPA-sid puudutavad arenevaid väljatöötate tehiseid (nõuete mudel, projekteerimismudel, lähtekood, programmid), protsessi automatiseerimise keskkonda jms.
2. Teine puudus on CMMile omane viis kirjeldada konfiguratsioonihaldust ja kvaliteedikontrolli mitte teiste protsessi tegevuste juurde kuuluvatena vaid kui kõikidest teistest protsessi tegevustest eraldiseisvaid distsipliine.

Praktikas on protsessi küpsuse tegelikuks indikaatoriks projekti käitumise ennustatavuse tase. Projekti käitumise kolme näitaja suhtes (projekti hind, ajagraafik ja tarkvarakvaliteet) ja CMM-i tasemete vaheline korrelatsioon peaks näitama järgmisi trende:

1. Protsess on juhuslik ja käitub ennustamatult.
2. Protsess on erinevates projektides korratav.
3. Protsess on defineeritud, projektid käituvad aja möödudes hinna, ajagraafiku või tarkvarakvaliteedi suhtes vähehaaval järjest paremini.
4. Protsessi juhitakse. Aja möödudes projekti käitumine paraneb kas ühe näitaja suhtes oluliselt või mitme näitaja suhtes korraga (näiteks hinna ja tarkvara kvaliteedi suhtes).
5. Protsessi optimeeritakse. See tase vastab projekti käitumise aja möödudes väga kiirele paranemisele – iga järgnev projekt käitub oluliselt paremini igas mõttes. Viienda taseme organisatsioonid saavad tõenäoliselt tegutseda väga kitsas nišis.

Kuigi pole välistatud, et paljud organisatsioonid suudaksid tekitada endale näo, mis võimaldab seda organisatsiooni hinnata kolmanda taseme vääriliseks, sealjuures protsess tegelikkuses ei pruugi hea olla, peaks tõesti hea protsess kolmanda taseme

hinde kergesti saama. Walker Royce väidab, et omadest tosinate protsesside hindamise kogemustest on ta selgeks saanud veel mõned indikaatorid, mis tõeliselt vastavad küpsele protsessile:

- Objektiivne arusaamine protsessi praegusest küpsusest.
- Objektiivne arusaamine projekti kvantitatiivsest käitumisest hinna ja kvaliteedi suhtes.
- Tegelik projekti käitumise paranemine.
- Aeg, mida vajatakse hindamise ettevalmistamiseks, on minimaalne.

Tasemel organisatsioon ja tasemel projektid teavad protsessi ja järgivad seda. Nad ei vaja aega auditi ettevalmistamiseks. Kui te arvate, et teie organisatsioon on kolmandal tasemel, siis vastake küsimusele: Kas Te taluksite üllatusauditi?

4.2.3 Pragmatiliselt protsessi parandamisest

4.2.3.1 Protsessi küpsus

Vastavus SEI CMM-ile ei too tingimata kaasa kvaliteetsete toodete arendamist. Küll aga protsess, mille käigus valmivad kvaliteetsed tooted, hinnatakse küpseks. Enamike protsesside paikapandud raamistike peamiseks puuduseks on see, et nad määravad kvaliteedikontrolli programmi staatiliselt kellegi eraldi tööülesandeks selle asemel, et integreerida kvaliteedikontroll dünaamiliselt igapäevaste tööülesannete hulka.

4.2.3.2 Küpse protsessi hind

Küps protsess ei ole kallim. Pikemas perspektiivis see hoopis säästab raha. Parandades ebaküpset protsessi, organisatsioon sageli tajub muutustega seotud kulutusi – protsessi juurutamine juba jooksva projektil, mida valitsevad lühiajalise perspektiiviga kulukaalutlused, on väga raske. Samas on protsessi juurutamine pikaajaliste äriliste püüdlustega organisatsioonis ja pikaajalistes projektides, mis on alles planeerimise staadiumis, kergem.

4.2.3.3 Tarkvaraarenduse meetrika

Tarkvaratoote kvaliteedi ja progressi (kaks erinevat perspektiivi tarkvaratootmise püüdlustes) hindamiseks on vaja objektiivseid mõõdupuid. Arhitektid hoolivad rohkem kvaliteedinäitajatest samal ajal, kui juhid hoolivad tavaliselt rohkem töö

edenemise näitajatest. Iga sellise arendusprotsessi korral, kus hindamiseks vajalikku infot kogutakse käsitsi, on edu piiratud. Kõige tähtsam hindamiseks vajalik info on lihtsad objektiivsed näitajad selle kohta, kuidas erinevad toote/projekti vaated muutuvad. Absoluutsed näitajad on tavaliselt palju vähem tähtsad, kui suhtelised muutused ajas. Tarkvaraprojekti dünaamilise loomuse tõttu peavad need näitajad olema saadaval igal ajal, saadavad suvalise alamhulga arendatavate toodete, alamsüsteemide, väljalasete, versioonide, komponentide ja meeskondade kohta ning hallatavad nii, et oleks võimalik hinnata trende (esimesi ja teisi tuletsi). Selline pidev info saadavus on praktikas saavutatud ainult siis, kui neid näitajaid hallatakse arenduskeskkonna abil *on-line*'is automaatselt genereeritava kõrvalproduktina.

4.2.3.4 Protsessi juurutamine

Erinevad eesmärgid tarkvaratootmises vajavad erinevaid protsesse. Koos universaalsete teemade ja tehnikatega eksisteerivad ka olukorrast sõltuvad erinevused tehnikates, prioriteetides, rituaalides ja rõhuasetustes. Tootearendusega tegeleva firma protsess ei ole täpselt sama, mida kasutab firma, kes arendab suurt spetsiifilist süsteemi välise kliendiga sõlmitud lepingu põhjal.

4.2.3.5 Protsess vs. meetod

Projekti juhtimise protsess tegeleb paljude erinevate probleemidega, millest igäihe lahendamiseks kasutatakse mingit tehnilist meetod. Protsessi iseloomustavad näiteks iteratiivne arendus, demonstratsioonipõhine hindamine ja riskide haldamine. Erinevates kohtades rakendatavateks meetoditeks võivad olla objektorienteeritud tehnika, arhitektuuriline lähenemine ja UML-keelne kujutamine. Kuigi halba protsessi ei päästa tõenäoliselt kunagi hea meetod, võib hea protsess olla edukas enamuse tehniliste meetoditega. On selge, et mõned meetodid on paremad, kui teised. Hea meetodi tulemus koos hea juhtimisprotsessiga on parim. See on eesmärk.

4.2.4 Protsessi küsimustik

Autor toob siinkohal ära ainult CMMi teise taseme küsimustiku koos selgitustega, kuidas neile küsimustele potentsiaalselt vastata tuleks.

4.2.4.1 Nõuete haldus

1. *Kas tarkvarale esitatavaid süsteeminõudeid kasutatakse tarkvara arenduse ja selle juhtimise alusena?*

Tarkvarale esitatavad nõuded pannakse kirja **nägemuses** ja **kasutusmallimudelis**. Igat **iteratsiooni** saadab **redaktsiooni** (*release*) **spetsifikatsioon**, mis sisaldab **vahepealsete tähtpunktide** (*milestones*) eesmäärke.

2. *Kui tarkvarale esitatavad süsteeminõuded muutuvad, kas siis tehakse ka vastavad muudatused tarkvaraarendusplaanidesse, tegevustesse ja toodetesse?*

Iteratiivses arenduses on iga iteratsioon varustatud uue redaktsioonispetsifikatsiooniga ja muudatustega tehnilistes **tehistes** (*artifacts*). **Muutmistaotluse** (*change request*) tüübid on **defekt** (*defect*) ja **täiendustaotlus** (*enhancement request*). Nõudmiste muutumisest põhjustatud muutusi kajastavad täiendustaotlused.

3. *Kas projekt järgib tarkvarale esitatud süsteeminõuete haldamiseks organisatsiooni poolt kehtestatud korda?*

Organisatsiooni poolt kehtestatud kord peaks sisaldama selgesõnalist meetodikat projekti tehiste, sh. nõuetega seotud tehiste, haldamiseks.

4. *Kas projektis nõudehaldusega tegelevad inimesed on vastavate protseduuride läbiviimiseks koolitatud?*

Koolitus on organisatsioonispetsiifiline asi.

5. *Kas nõudehaldusega seotud tegevuste seisundi hindamiseks kasutatakse mingeid mõõtusid (nt. muutmistaotluste koguarv, avatud, heaks kiidetud ja arendusalusesse liidetud muutmistaotluste arv)?*

Tuleks jälgida täiendustaotlusi ja nende olukorra hinnangut perioodiliselt raporteerida.

6. *Kas projekti nõudehaldusega seotud tegevused on allutatud tarkvara kvaliteedikontrolli (Software Quality Assurance – SQA) läbivaatustele?*

Kvaliteedikontroll on kõikide meeskondade kohustus. Sõltumatu testimise üksus, kes on tarkvara kvaliteedikontrollis peamine vastutaja, mitte ainult ei vaata üle nõudehaldust, vaid osaleb aktiivselt redaktsiooni spetsifikatsioonide, redaktsiooni kirjelduste loomisel ja nende nõuete suhtes **jälitavuse** (*traceability*) tagamisel. Muutmistaotlustes kajastatud nõuete muutusi vaatab üle ka **muutmisenõukogu** (*Configuration Control Board* [Royce, 1998], *Change Control Board* [RUP, 2000] – mõlemal juhul lühendatult *CCB*). Nõuetega seotud tehiseid paratamatult “vaadatakse läbi” ka kasutusklassi mudelitega, **projekteerimistehistega**, **teostuse** tehistega ja **evitustehistega** seotud väljatöötamise tegevuste käigus.

4.2.4.2 Tarkvaraprojekti plaanimine

1. *Kas ennustused (nt. projekti suurus, hind ja ajagraafik) on planeerimiseks ja projekti jälgimiseks dokumenteeritud?*

Gantti graafi abil (*Gantt chart*) *MS Project*’i abil defineeritakse hinna plaani, ajagraafiku, iteratsiooni sisu ja suuruse alused. Lisaks Gantti graafile on iteratsioonide kalenderplaan koos väljatöötete loetelu ja mahuga formaalselt kinnitatud tellimuslepingu lisana. Projekti jälgida ning progressi ja kvaliteeti aluseks olevate plaanidega ja plaanimuutustega võrrelda aitavateks vahenditeks on **seisuhinnangud** (*status assessments*) ja **projekti mõõdustik** (*project metrics*). Madalamal tasemel detailselt dokumenteeritud hinnangud, plaanid ja tegelikkus kajastuvad muutmistaotlustes.

2. *Kas projekti plaanid dokumenteerivad neid tegevusi, mida tehakse, ja korraldusi, mis on antud projektiga seoses antud?*

Ärimall ja **tarkvara väljatööteteplaan** kirjeldavad üldisi tegevusi ja on välja pandud tarkvaraprojekti juhi poolt kui kohustuslik projekti alus. Gantti graafid dokumenteerivad hinna aluseid ja kohustusi kõikidel juhtimise tasemetel.

Töökäsud (*work orders*) dokumenteerivad madalama taseme tegevusi ja kohustusi.

3. *Kas kõik grupid ja isikud on nõus neid mõjutavate tarkvaraprojektiga seotud kohustustega?*

Gantti graafi kasutades saab tarkvaraprojekti juht pidada alluvate juhtidega läbirääkimisi kohustuste asjus. Töökäsud ja muutmisnõukogu on vahendid madalama taseme kohustuste osas läbi rääkimiseks.

4. *Kas projekt järgib kirja pandud organisatsiooni tarkvaraprojekti planeerimise poliitikat?*

Organisatsiooni poliitika peaks pakkuma tarkvaraprojekti planeerimiseks organisatsioonilisi aluseid. Organisatsiooni infrastruktuur peaks võimaldama ligipääsu varasemale kogemusele ja vaikimisi kasutatavatele plaanimismärgistele (*planning benchmarks*).

5. *Kas tarkvaraprojekti plaanisel on kasutada adekvaatsed ressursid (st. finantsvahendid ja kogenud isikud)?*

Tarkvaraprojekti juht, kes on plaani eest vastutav, peaks selle ressursi tekitama ja oma valdusesse võtma. Et määrata tööpanuse (*effort*) tulusust (*return on investment – ROI*), sisaldab ärimall organisatsiooni jaoks ootusi ja kohustusi. Plaanimisressursside adekvaatsust ei defineeri poliitika. Plaanimine peaks moodustama umbes 10% projekti tööpanustest. Kuna adekvaatsete ressursside tuvastamine on projekti spetsiifiline, peaks see olema organisatsiooni **projekti läbivaatusorgani** (*project review authority – PRA*) vaatluse all ja seda tuleks läbi vaadata iga peamise tähtpunkti (*major milestone*) ajal.

6. *Kas tarkvaraprojekti tegevuste plaanamise seisu tuvastamiseks kasutatakse mõõtmisi (st. tähtpunktidesse jõudmisel võrreldakse projekti plaanamise tegevusi plaaniga)?*

Plaani kriitiliste omaduste võrdlemiseks tegelikkusega töötakse välja spetsiaalne arengut iseloomustav meetrika (väljatöötate areng, testimise areng, läbitud hindamiskriteeriumid, täidetud stsenaariumid, väljatöötatud SLOC (*source lines of code – koodiridade arv*), suletud vs. avatud SCO-d jne.)

7. *Kas projekti juht vaatab läbi tarkvaraprojekti plaanamise tegevusi nii perioodilisel alusel kui ka sündmuste alusel?*

Vajalikele juhtimisindikaatoritele tähelepanu suunamise ja riskide perioodilise hindamise eest hoitakse **seisu hindamise** (*status assessment*) abil vastutavana

projekti juht. Sündmuste alusel hindamiseks on lihtsad sundimise vahendid peamised tähtpunktid ja redaktsioonikirjeldused.

4.2.4.3 Tarkvaraprojekti järelvalve

1. *Kas projekti tegelikke tulemusi (graafik, suurus ja hind) võrreldakse hinnangutega tarkvara plaanides?*

Plaanitud tulemusi võrdlevad tegelike arengu indikaatorite tulemustega seisuhinnangud. Plaanitud kvaliteedi indikaatoreid (hindamise kriteeriume) võrdleb tegelike tulemustega redaktsiooni kirjeldus (*release description*). Detailsema muutusehalduse jaoks dokumenteerivad plaanitud tulemuste vastavust tegelikele SCO-d.

2. *Kui tegelikud tulemused erinevad projekti tarkvaraplaanidest märkimisväärselt, kas siis võetakse ette parandamistegevusi.*

Ebaõnnestunud hindamiskriteeriumitele juhitakse tähelepanu redaktsioonikirjeldustes ja järgnevatel iteratsioonides. Muudele kõrvalekalletele plaanist juhitakse tähelepanu seisuhinnangutes, kus nõutakse ja jälgitakse probleemi lõpuni lahendamist.

3. *Kas muutused tarkvarale esitatavate nõuete osas kooskõlastatakse kõikide nende muutuste poolt mõjutatavate gruppide ja isikutega?*

Nõudmiste muutusi kooskõlastatakse arenevate WBS-ide (*Work Breakdown Structure – töö liigendusstruktuur*), tarkvara väljatöötamise ja seisuhinnangute kaudu. Madalatasemeliste nõudmistele juhitakse tähelepanu CCB-s, jälitatakse SCO-des ja võetakse arvesse redaktsiooni kirjeldustes.

4. *Kas projekt järgib kirja pandud organisatsiooni poliitikaid nii tarkvara väljatöötamise tegevuste jälgimise kui juhtimise osas?*

Organisatsiooni poliitika peaks teatavate teemade hulga jaoks defineerima standardse seisuhinnangu formaadi, et projektide omavaheline võrdlemine oleks võimalik.

5. *Kas spetsiaalselt tarkvara väljatöötamise tegevuste ja toodete (st. tööpanused, graafik ja eelarve) jälgimise eest on keegi vastutav?*

Vastutav isik on tarkvaraprojekti juht. Mehhanism perioodiliste läbivaatuste ja seletatavuse tagamiseks on seisu hindamine koos aluseks võetud WBS-iga.

6. *Kas tarkvara väljatöötamise jälgimise ja järelevaatuse oleku määramiseks kasutatakse mõõtmisi (st. kogu jälgimise ja järelevaatuse tegevuste peale kulutatud tööpanused)?*

Mehhanismid tegevuste oleku jälgimiseks ning tarkvara väljatöötamise tööpanuste järelevaatuseks on WBS-i kulutuste ja arengu meetrika.

7. *Kas tarkvaraprojekti jälgimise ja järelevaatuse tegevusi vaadatakse perioodiliselt läbi tippjuhtkonnaga (st. projekti suutvus, avatud probleemid, riskid ja astutavad sammud)?*

See on täpselt see põhjus, miks on olemas äri mall (mida uuendatakse iga faasi lõpus), seisuhinnangud ja peamiste tähtpunktide läbivaatused.

4.2.4.4 Alltöövõtude haldus

Kuna protsessi raamistik spetsiaalselt alltöövõtude halduseks midagi ette ei näe, siis selleks, et protsess jääks homogeenseks, eeldame, et kõik tehnikad, vahendid ja mehhanismid on ka alltöövõtjatel kasutusel. Kui seda ei õnnestu saavutada, või kui peatöövõtja ei suuda defineerida küpse protsessiga alltöövõtja jaoks tööst hästi eraldatud tükki, tuleks alltöövõttu vältida. Et riske efektiivselt hallata, tuleb organisatoorsete liideste arvu ja kompleksust hallata. Kõik alltöövõttudega seotud otsused peavad olema dokumenteeritud äri mallis.

1. *Kas alltöövõtjate töö tegemise võimekuse järgi valimiseks kasutatav protseduur on dokumenteeritud?*

Organisatsiooni poliitika peaks nõudma kogu projekti personalilt, sh. Tarkvara alltöövõtjalt, ühe ja sama väljatöötelaani järgimist. Projektidesse palgatud alltöövõtjate protsessid peaksid olema hinnatud vähemalt sama küpseks, kui see on projekti tipmisel organisatsioonil. (Teisiti öeldes, 3. tasemel organisatsioon ei tohiks palgata 2. taseme organisatsiooni.)

2. *Kas muutused alltöövõttudes toimuvad peatöövõtja ja alltöövõtja kokkuleppel?*

Kaine mõistus ütleb, et see peaks alati nii olema.

3. *Kas alltöövõtjatega viiakse läbi perioodilisi tehnilisi kooskõlastusi?*

Alltöövõtjad, kes järgivad sama väljatöõteplaani, peaksid osalema samadel tehnilistel kooskõlastustel, peamiste tähtpunktide läbivaatustel ja seisu hindamistel.

4. *Kas tarkvara alltöövõtja tulemusi ja suutvust jälgitakse talle pandud kohustuste suhtes?*

Alltöövõtjaid, kes järgivad sama väljatöõteplaani, tuleks kohustuste suhtes jälgida samamoodi, nagu jälgitakse peatöövõtjat.

5. *Kas projekt järgib tarkvara alltöövõtude haldamisel kirja pandud organisatsiooni poliitikat?*

Poliitikat kirjeldav dokument nõuab, et alltöövõtja järgib sama protsessi, mida peatöövõtjagi.

6. *Kas inimesed, kes vastutavad tarkvara alltöövõtude haldamise eest, on tarkvara alltöövõtude haldamiseks koolitatud?*

Koolitamine on organisatsioonispetsiifiline asi.

7. *Kas tarkvara alltöövõtude haldamise tegevuste seisu määramiseks kasutatakse mõõtmisi (st. kalenderplaani seis plaanitud üleandmiskuupäevade suhtes ja alltöövõtu haldamiseks kulutatud tööpanus)?*

Alltöövõtjaid tuleks hallata samal homogeensel viisil, naid peatöövõtjatki.

8. *Kas tarkvara alltöövõtja tegevusi vaadatakse läbi projektijuhiga nii perioodilisel alusel kui ka sündmuste alusel?*

Tarkvaraprojektijuht peaks alltöövõtjaid juhtima samal viisil, kui ülejäänud projekti meeskonda. Kõik alltöövõtjate kohustustega seotud otsused on dokumenteeritud ärimallis, mida uuendatakse elutsükli ühest faasist teise minekul.

4.2.4.5 Tarkvara kvaliteedikontroll

Kõik tegevused ja inimesed on tarkvara kvaliteedikontrolliga (*software quality assurance – SQA*) seotud. Eraldiseisvat hindamissmeeskonda soovitatakse kasutada selliste kvaliteedi hindamise tegevuste jaoks, nagu näiteks testimine ja meetrika

analüüs, mida (kalenderplaani efektiivsuse huvides) viiakse läbi paralleelselt ja (tehniliste aspektide erinevuse huvides) sõltumatult. Kvaliteedi eest vastutamine lasub organisatsiooni erinevatel meeskondadel. Sellele küsimustikule vastamise eesmärgil vastavad CMM-i SQA definitsioonile kõige rohkem sõltumatu hindamismeeskonna tegevused.

1. *Kas SQA tegevusi plaanitakse?*

Testimise tegevusi, meetrikat ja kvaliteedikontrolli tegevusi kirjeldab tarkvara väljatöötaja plaan. Töö liigendstruktuur kajastab juba paljusid plaani detaile. SQA tegevuste plaanimise mehhanismiks on ka redaktsioonikirjeldused.

2. *Kas SQA tegevused võimaldavad objektiivselt kontrollida et tarkvaratooted ja tegevused vastavad rakendatavatele standarditele, protseduuridele ja nõudmistele.*

Iteratsiooni eesmärgid tuvastatakse redaktsiooni spetsifikatsioonides. Projekti standardid ja protseduurid tuvastatakse tarkvara väljatöötaja plaanides. Redaktsiooni spetsifikatsioonid kirjeldavad vahepealsete toodete kvaliteeti ka eesmärgi täidetuse/mittetäidetuse vaates. Muutmisnõukogud ja muutmistaotlused tagavad, et madalataseme standardid ja protseduurid oleksid kontrollitud ja jälgitud. Automatiseeritud vahendid keskkonnas (kompilaatorid, dokumentatsiooni produtseerimine, muudatuste haldus) peaksid tagama, et tooted vastavad rakendatavatele standarditele.

3. *Kas neid gruppe ja isikuid, keda SQA läbivaatused ja auditid puudutavad, varustatakse SQA läbivaatuste ja auditite tulemustega (st. neid kes töö teostasid ja neid, kes töö eest vastutavad)?*

SQA läbivaatusteks on kõik protsessi kontrollpunktid ja muutmisnõukogu tegevused. Vahepealsed tulemused dokumenteeritakse perioodiliselt redaktsiooni kirjeldustes. Kõiki muutmistaotlused vaadatakse muutmisnõukogu poolt muutusi puudutavate gruppide osalusel läbi.

4. *Kas projektis lahendamata jäänud mittevastavuste küsimusi (st. kõrvalekalded rakendatavatest standarditest) vaadeldakse tippjuhtkonna tasemel?*

See on üks põhjusi, miks on väljatöötaja plaanidele ja seisuhinnangutele vaja projekti läbivaatusorgani heakskiitu. Projekti läbivaatusorgani poolsetel

perioodilistel läbivaatustel käsitletakse projekti arenedes tehtud organisatsiooni poliitikast kõrvale kaldumise ettepanekuid.

5. *Kas projekti järgib SQA teostamisel organisatsiooni kirja pandud poliitikat?*

Organisatsiooni ja projekti SQA poliitikad on kirjas vastavalt organisatsiooni poliitikas ja tarkvara väljatöötetlaanis.

6. *Kas SQA tegevuste sooritamiseks on saadaval piisavalt ressursse (st. rahaline ressurss ja määratud juht, kes tegeleb tarkvara mittevastavuse küsimustega)?*

SQA ressursside adekvaatsust ei spetsifitseerita poliitikaga. Hea mallina võib kasutada seda, et umbes 25% projekti tööpanustest peaks kuluma hindamismeeskonna tegevustele (testimine, hindamine, meetrika, muutmisenõukogu). Kuna adekvaatse ressursside ja isikute määramine on projektspetsiifiline, siis peaks need hindamised selgelt käima projekti läbivaatusorgani ametliku kontrolli alt läbi.

7. *Kas SQA sooritatavate tegevuste seisu ja hinna määramiseks kasutatakse mõõtmisi (st. tehtud töö, tööpanused ja kulutatud rahaline ressurss plaaniga võrreldes)?*

Töö liigendstruktuur on tegevuste aluseks ja kõikide tegevuste tegelikkuse vastamist plaanile jälgivad perioodilised seisu hindamised.

8. *Kas SQA tegevusi vaadatakse tippjuhtkonna poolt läbi perioodilisel alusel?*

See on üks projekti läbivaatusorgani läbivaatuste ja projekti läbivaatusorgani poolse tarkvara väljatöote plaani heakskiitmise otstarvetest.

4.2.4.6 Tarkvara konfiguratsioonihaldus

Nii nagu SQA korral, on ka tarkvara **konfiguratsioonihaldusega** (*software configuration management – SCM*) seotud kõik tegevused ja inimesed. Konfiguratsioonihalduse tegevuste, sh. Muutmistaotluste andmebaasi halduse, muutmisenõukogu administreerimise ja **arendusaluse** haldamise vastutus lasub sõltumatul hindamismeeskonnal. Neid tegevusi tuleks sooritada paralleelselt (kalenderplaani efektiivsuse huvides) ja sõltumatult (tehniliste aspektide erinevuse huvides). Üldiselt tegelevad SCM-i tegevustega kõik tarkvarainsenerid ja on selleks peamiselt varustatud vastava tarkvaraarenduskeskkonnaga. Sellele küsimustikule

vastamise eesmärgil, vastavad CMM-i SCM-i definitsioonile kõige rohkem hindamismeeskonna tegevused.

1. *Kas tarkvara konfiguratsiooni halduse tegevusi plaanitakse?*

Tarkvara väljatöötelaan peaks SCM-i tegevusi plaanima.

2. *Kas projekt tuvastab, haldab ja tegeleb tarkvara väljatöötetud saadavaks konfiguratsioonihaldust kasutades?*

Muutmistaotlusi ja kõiki tehiseid hallatakse konfiguratsioonihalduse abil.

3. *Kas konfiguratsiooni elementide/ühikute muutmise haldamiseks järgib projekt dokumenteeritud protseduuri?*

Nagu on dokumenteeritud organisatsiooni poliitikas ja tarkvara väljatöötelaanis, on muutmistaotlused *on-line* mehhanismiks muudatuste haldusel?

4. *Kas tarkvara arendusaluse standardaruandeid (st. tarkvara muutmisenõukogu protokollid, muutmistaotluste koond ja seisuhinnangud) jagatakse neile gruppidele ja isikutele, keda need puudutavad?*

Seisuhinnang, mis sisaldab muutmisenõukogu tulemusi standardmeetrika vormis, peaks olema saadaval kõigi osanike ja projekti meeskondade jaoks.

5. *Kas projekt järgib tarkvara konfiguratsiooni halduse tegevuste teostamisel kirja pandud organisatsiooni poliitikat?*

Seda tagavad organisatsiooni poliitika ja tarkvara väljatöötelaan.

6. *Kas projekti personal on koolitatud sooritama neid tarkvara konfiguratsiooni halduse tegevusi, mille eest nad vastutavad?*

Koolitus on organisatsioonispetsiifiline asi. Üldiselt tegelevad keskkonnast tingituna tarkvaraorganisatsioonis konfiguratsiooni haldusega kõik. Sõltumatu testi meeskonna formaalsed konfiguratsiooni halduse tegevused on peamiselt administratiivne juhtimine ja aruandlus.

7. *Kas tarkvara konfiguratsioonihalduse tegevuste oleku määramiseks kasutatakse mõõtmisi (st. tarkvara konfiguratsioonihalduse tegevustele kulutatud tööpanused ja rahalised ressursid)?*

Töö liigendstruktuur on tegevuste aluseks ja kõikide tegevuste tegelikkuse vastamist plaanile jälgivad perioodilised seisu hindamised.

8. *Kas tarkvara arendusaluse vastamist seda defineerivale dokumentatsioonile kontrollitakse perioodiliste audititega (st. SCM grupi poolt)?*

Muutmisnõukogud on kõige sagedasemad auditid, mis kontrollivad kooskõllalisust muutmistaotluse kaupa. Redaktsiooni kirjeldused sisaldavad peamiste tähtpunktide jaoks loodud vahepealsete arendusaluste jaoks integreeritud kvaliteedi, täielikkuse ja kooskõllalisuse auditit.

4.3 Cybernetica ASi püüdlused

Cybernetica ASis on praegu käsil olemasolevate väljatöötetprotsesside standarditele vastav defineerimine, aga kuna sellega pole veel lõpetatud, siis selle mõjusust Cybernetica ASi suutlikkusele autor veel hinnata ei oska ja seetõttu piirdub siin ainult kirjeldusega sellest, mida on ettevõttes otsustatud ja mida peaks veel tegema.

Kui eesmärk oleks ainult Cybernetica ASi infoturbe osakonnas kasutatav väljatöötetprotsess defineeritult kontrolli alla võtta, siis tuleks lihtsalt juhendada CMMi skaalast, RUPi soovitustest protsessi juurutamisel ja RUPi kirjeldustest selle kohta, kuidas RUP CMMi skaalale kujutub (vt. [CMM]).

Cybernetica ASis on aga otsustatud kõikide osakondade efektiivsuse tõstmise ning selleks vajaliku teadmuse ja juhtimise üleorganisatsioonilise ühtlustamise nimel, et väljatöötetprotsesside defineerimisega ja tippjuhtkonna tasemel juhtimisega tuleb tegeleda kõikides osakondades sünkroonselt. Eesmärk on hea, aga olukord eesmärgi saavutamiseks on keeruline, kuna Cybernetica ASi kõik kolm osakonda on oma sõltumatu tegutsemise tõttu väga erinevad.

Kui informatsioonisüsteemide ja infoturbe osakonna ühtlustamist saab veel vaadelda ainult RUPi ja CMMi keskselt, siis navigatsioonisüsteemide osakonnaga see ei õnnestu. Navigatsioonisüsteemide osakonnas tegeldakse elektroonikaseadmete tootmisega – selle poolest Cybernetica AS erineb tavalisest tarkvarafirmast. Tarkvaratootmine selles osakonnas seisneb peamiselt seal toodetavate elektroonikaseadmete juhtimistarkvara kirjutamises. Kuna CMM on ainult tarkvaraprotsessile orienteeritud, siis navigatsioonisüsteemide osakonda see ei

rahulda. Küll aga katab 2. taseme CMMi nõudmised ja ka navigatsioonisüsteemide osakonna protsessid adekvaatselt ISO 9001:2000 kvaliteedijuhtimissüsteemi standard.

ISO 9001:2000 kvaliteedijuhtimissüsteemi järgi on kogu organisatsiooni väljatöote protsesside juhtimise defineerimisel kõige peamiseks dokumendiks “Kvaliteedikäsiraamat”, mis loetleb üles kõik protsessi valdkonnad või tegevused, mille jaoks on defineeritud mingi kord. Iga sellise valdkonna kohta käiv kord on omakorda lahti kirjutatud vastavas kvaliteedikäsiraamatu juhendis. Olulisemad peatükid (lisaks peatükkidele, millel on pigem dokumendi struktuurne tähendus), millest ISO 9001:2000-le vastav kvaliteedikäsiraamat koosneb, on

- “Juhtkonna kohustused”,
- “Ressursside juhtimine”,
- “Toote valmistamine” ning
- “Möötmise, analüüs ja parendamine”.

Et ka teistele potentsiaalsetele partneritele, kes oma partnereid standardile vastavuse järgi valivad, oma taset tõendada, võib peale protsessi juhtimise korda tegemist tellida vastavalt organisatsioonilt auditi, kes positiivse tulemuse korral väljastab ettevõttele sertifikaadi. Negatiivse tulemuse korral ei saa järgmist auditit enne kuue kuu möödumist tellida. Kuna tõenäosus, et esimesel katsel kogemuste puudumisel võib selline audit ebaõnnestuda, on väga suur, on soovitatav alguses tellida eelaudit, mis on pisut kergekaalulisem ning mille tulemusel antakse soovitusi, kuidas olukorda parandada, ja kolm kuud selleks parandamiseks aega. Seejärel tuleb “tegelik” audit.

Nagu RUP ja CMM, nii soovitab ka ISO alustada mitte hüpoteetilise ideaalsüsteemi kirja panekust, vaid olemasoleva kirjeldamisest ISO kvaliteedikäsiraamatu struktuuri järgi – see saabki edasise protsessi esimeseks lähendiks. Cybernetica ASile tähendab see tarkvaratootmise osas paratamatult esialgu iga osakonna jaoks eraldi juhendite kirjutamist, sest igas osakonnas käivad praegu asjad isemoodi. Edasi tuleb tuvastada iga osakonna senises protsessis kõige suuremad probleemid ja hakata neid probleeme ükshaaval parandama kasutades selleks juba unifitseeritud protsessi vahendeid.

Sõnaseletustik

andmetüüp (*datatype*) väärtuste hulga kirjeldaja. Andmetüübid on näiteks primitiivsed eeldefineeritud tüübid ja kasutaja defineeritavad tüübid. Eeldefineeritud tüübid on näiteks numbrid, sõned ja aeg.

arendusalus (*baseline*) läbivaadatud ja vastuvõetud tehise redaktsioon, mis vastab kokkuleppele edasise arenduse suhtes ja mida saab muuta ainult formaalse protseduuri kaudu, nagu **muutusehaldus** või **konfiguratsioonihaldus**.

arhitektuur (*architecture*) mingis keskkonnas asuva süsteemi kõrgeima taseme kontseptsioon. Tarkvarasüsteemi arhitektuur (mingil antud hetkel) on selle oluliste **liideste** vahendusel suhtlevate **komponentide** korraldus või struktuur.

defekt (*defect*) toote või pooltoote toimimise kõrvalekalle või puudujääk. Defektid on näiteks elutsükli varastes faasides leitud tegematajätmised ja puudused ning testimiseks või kasutamiseks valmis oleva tarkvara vigade sümptomid.

erisus (*feature*) süsteemi poolt pakutav mõtestatud teenus, mis otseses mõttes rahuldab **osaniku tarbe**. Klassifikaatorisse (nagu näiteks liides, klass või andmetüüp) kapseldatud omadus (nagu näiteks operatsioon või atribuut).

evitus (*deployment*) üks **põhivoog** tarkvara väljatööteprotsessis, mille eesmärk on tagada väljatöötatud süsteemi edukas **siire** kasutajatele. Väljunditeks on **tehised**, nagu õppematerjalid ja paigaldusmeetodid.

faas (*phase*) kahe peamise **tähtpunkti** vaheline aeg, mille jooksul saavutatakse hulk hästidefineeritud eesmärke, valmistatakse **tehiseid** ja tehakse otsuseid, kas liigutakse edasi järgmisse faasi või mitte.

Gantti graaf (*Gantt chart*) visualiseerib tööülesannete kestvusi ja omavahelisi ajalisi sõltuvusi.

interaktsioon (*interaction*) spetsifikatsioon, kuidas teatud ülesande sooritamiseks **isendite** vahel sündmust saadetakse.

iteratsioon (*iteration*) aluseks võetud plaani järgi toimiv ja hindamiskriteeriumitega selge tegevuste jada, mille tulemuseks on (seesmine või väline) **redaktsioon**.

jälitatavus (*traceability*) võimalus tuvastada projekti elemendiga seotud teisi projekti elemente, eriti neid, mis on seotud **nõuetega**.

kasutusmall (*use case*) kasutusmall defineerib hulga **kasutusmalli isendeid**, kus iga esindaja on süsteemi sooritav tegevuste järjekord, mis tervikuna omab teatud **tegija** jaoks mõtestatud väärtust. Kasutusmall sisaldab kõiki peamisi ja alternatiivseid sündmuste voogusid “mõtestatud väärtuse” saamiseks. Tehniliselt, kasutusmall on klass, mille isendid on stsenaariumid. Tegevuste järjekorra spetsifikatsioon koos variantidega, mida süsteem (või muu olem) võib sooritada, süsteemi **tegijatega** suheldes.

kasutusmalli isend (*use case instance*) süsteemi poolt sooritav tegevuste jada, mis tervikuna omab **tegija** jaoks mõtestatud väärtust. Tegevuste järjekorda spetsifitseerib **kasutusmall**.

kasutusmallimudel (*use case model*) mudel, mis kirjeldab süsteemi funktsionaalseid **nõudeid kasutusmallide** terminites.

kasutusmalljuhitav protsess (*use case driven process*) tarkvara väljatööteteprotsess, milles plaanitakse projekti edasisi tegevusi **kasutusmallide** põhjal.

klass (*class*) samu atribuute, operatsioone, meetodeid, seoseid ja semantikat jagavate objektide kirjeldus. Klass võib kasutada keskkonnale pakutavate operatsioonide kogumiku spetsifitseerimiseks mingit liideste hulka.

klassifikaator (*classifier*) mehhanism, mis kirjeldab käitumislikke ja struktuurseid erisusi (*features*). Klassifikaatorid on **liidesed**, **klassid**, **andmetüübid** ja **komponendid**.

komponent (*component*) mittetriviaalne peaaegu sõltumatu asendatav süsteemi osa, mis täidab hästidefineeritud **arhitektuuri** kontekstis selget ülesannet.

konfiguratsioon (*configuration*) (1) üldine: funktsionaalsete osade olemuse, numbrite ja peamiste omadustega defineeritud süsteemi või võrgu korraldus. Käib nii riistvara kui tarkvara konfiguratsiooni kohta. (2) Nõuded, arhitektuur ja teostus, mis defineerivad süsteemi või süsteemi komponendi mingi versiooni.

konfiguratsioonihaldus (*configuration management*) tugiprotsess, mille eesmärk on tuvastada, defineerida ja võtta asju arendusaluseks; hallata nende asjade muutusi ja redaktsioone; raporteerida ja salvestada nende asjade olekut ja **muutmistaotlusi**; tagada nende asjade täielikkus, kooskõlalatus ja korrektsus ning hallata nende asjade hoidmist ja käsitlemist.

koostöö (*collaboration*) mingis kontekstis mingi tegevuse teostamiseks omavahel suhtlevate objektide kogumi kirjeldus. Kajastab tegevuse teostamises korraga kolme vaadet: andmestruktuurid, sündmuste järjekord ja andmevood. Koostööl on staatiline ja dünaamiline osa. Staatiline osa kirjeldab objektide ja nendevaheliste seoste rolli koostöös. Dünaamiline osa koosneb ühest või enamast dünaamilisest **interaktsioonist** mis näitavad teadete voo ajalist järjestust koostöös. Spetsifikatsioon, kuidas mingi **operatsioon** või **klassifikaator**, nagu näiteks **kasutusmall**, teostatakse klassifikaatorite ja **sidemete** poolt.

liides (*interface*) klassi või komponendi teenuse spetsifitseerimiseks kasutatav **operatsioonide** kogumik. Nimetatud operatsioonide hulk, mis iseloomustab elemendi käitumist.

muutmisnõukogu (*change control board – CCB*) keskne juhtimismehhanism tagamaks, et iga **muutmistaotlus** saab vastavalt käsitletud.

muutmistaotlus (*change request – CR*) üldine termin suvalise **osanikult** pärit **tehise** või **protsessi** muutmise taotlemiseks.

muutusehaldus (*change management*) **tehiste** muutmise juhtimise ja jälgimise tegevus.

nõue (*requirement*) kirjeldab tingimust või suutlikkust, millele süsteem peab vastama; pärineb otse kasutaja vajadusest või on määratletud lepingus, standardis, spetsifikatsioonis või teises formaalselt seotud dokumendis.

nägemus (*vision*) väljatöötatava toote kasutaja või kliendi vaade, mis on spetsifitseeritud süsteemi põhiliste **osaniku tarvete** ja **erisustena**.

projekteerimine (*design*) tarkvaraväljatöote protsessi osa, mille peamine eesmärk on otsustada, kuidas süsteem teostatakse. Projekteerimise ajal tehakse süsteemi taodeldava funktsionaalsuse ja kvaliteedinõuete tagamiseks strateegilisi ja taktikalisi otsuseid.

põhivoog (*core workflow*) üks üheksast *Rational Unified Process*'i põhivoogudest: talitluse modelleerimine (*business modeling*), nõuded (*requirements*), analüüs ja projekteerimine (*analysis & design*), teostamine (*implementation*), test (*test*), evitus (*deployment*), konfiguratsiooni- ja muutusehaldus (*configuration &*

change management), projektijuhtimine (*project management*). Tarkvara väljatöötamistalitluse abstraktne **talitluse kasutusmall**.

põlv (*generation*) lõplik redaktsioon elutsükli lõpus.

operatsioon (*operation*) teenus, mida saab objektilt küsida. Operatsioonil on signatuur, mis kirjeldab selle operatsiooni päist (nimi, tagastatava väärtuse tüüp, argumendid jne.).

osanik (*stakeholder*) isik, keda süsteemi väljund materiaalselt mõjutab.

osaniku tarve (*stakeholder need*) talitluslik või operatsiooniline probleem (väljavaade), mida tuleb rahuldada, et õigustada süsteemi ostmist või kasutamist.

projekti läbivaatusorgan (*project review authority – PRA*) organisatsiooniline olem, kellele projektijuht aru annab. PRA on vastutav selle eest, et tarkvaraprojekt oleks kooskõlas poliitikatega, tööpraktikatega ja standarditega.

projekti mõõdustik (*project metrics*) projekti hindamiseks kasutatavate näitajate skaalade valik.

protsess (*process*) (1) operatsioonisüsteemi protsess: tegevuse juhtimisharu, mida saab käivitada paralleelselt teiste protsessidega. (2) Hulk osaliselt järjestatus samme mingi eesmärgi saavutamiseks. Tarkvaratehnikas on eesmärgiks valmistada uus tarkvaratoode või muuta olemasolev paremaks. Protsessitehnikas on eesmärgiks töötada välja või parandada protsessi mudelit. Vastab **talitluse kasutusmallile** talitlustehnikas.

redaktsioon (*release*) alamhulk lõppproduktist, mida hinnatakse peamise **tähtpunkti** juures.

redaktsiooni spetsifikatsioon (*release specification*) nõuete kogum, millele peab antud redaktsioon vastama.

seisu hindamine (*status assessment*) tegevus, mille käigus tuvastatakse, mil määral on projekt talle antud hetkeks esitatud nõudmised täitnud.

seisuhinnang (*status assessment*) tehis, mille abil dokumenteeritakse **seisu hindamise** tulemus.

side (*association*) seos, mis modelleerib isenditevahelist kahesuunalist semantilist ühendust.

siire (*transition*) protsessi neljas **faas**, mille käigus liigub tarkvara kasutajate kätte.

stsenarium (*scenario*) **kasutusmalli isendi** kirjeldus, **kasutusmalli** alamhulk. Kindel tegevuste järjekord käitumise illustreerimiseks. Stsenariumit saab kasutada interaktsiooni või kasutusmalli isendi käivitamise illustreerimiseks.

talitlus (*business*) terviklik organisatsioon, infrastruktuur või keskkond, milles vaadeldakse väljatöötatava süsteemi toimimist. Näiteks selleks, et töötada välja videolaenutuse infosüsteemi, on kasulik vaadelda süsteemi kasutama hakkava videolaenutuse firma tegutsemist – videolaenutuse talitlust.

talitluse kasutusmall (*business use case*) **kasutusmall**, kus vaadeldavaks süsteemiks on **talitlus** ja **tegijaks** on **talitluse** kasutajad (nt. videolaenutuse **talitlusele** on **tegijateks** videolaenutuse kliendid).

tarkvara väljatöötetepaan (*software development plan*) **tehis**, millesse on koondatud kogu projekti juhtimiseks vajalik informatsioon.

tegija (*actor*) keegi või miski väljaspool süsteemi või talitlust, mis suhtleb süsteemi või talitlusega.

tehis (*artifact*) informatsioonitükk, mida produtseeritakse, muudetakse või kasutatakse protsessi poolt, defineerib vastutusala ja on versioonihalduse subjekt. Tehis võib olla nt. mudel, mudeli element, dokument, dokumendis sisalduv teine dokument, kirjeldus, tarkvara jms.

teostus (*implementation*) tarkvara väljatööteteprotsessi **põhivoog**, mille eesmärk on teostada ja testida klasse. Defineerib, kuidas midagi konstrueeritakse või arvutatakse. Näiteks klass on tüübi teostus, meetod on operatsiooni teostus.

tähtpunkt (*milestone*) mingi teatava projekti etapi formaalse läbimise punkt; vahefiniš; unifikatsioon protsessis on see punkt, kus lõpeb formaalselt iteratsioon.

täiendustaotlus (*enhancement request*) **osaniku** taotluse tüüp, mis spetsifitseerib süsteemi uut **erisust** või funktsionaalsust.

töö liigendstruktuur (*work breakdown structure – WBS*) plaanimisraamistik; projekti dekompositsioon väiksemateks töö ühikuteks, millest tuletatakse ja jälgitakse projekti hinda, **tehiseid** ja tegevusi.

töökäsk (*work order*) projektijuhi vahend vastutavale personalile teatamiseks, mida ja millal teha tuleb.

töövoog (*workflow*) talitluses sooritatavate tegevuste jada, mille tulemus omab talitluse **tegija** jaoks mõtestatud väärtust.

vahepealne tähtpunkt (*minor milestone*) kõik tähtpunktid, mis ei ole **faasi** lõpu tähtpunktid.

väljatöötusjuhtum (*development case*) tarkvaratehnika protsess, mida organisatsioonis kasutatakse. See töötatakse välja kui unifitseeritud protsessi konfigureerimise vahend, mida kohandatakse vastavalt projekti vajadustele.

ärimall (*business case*) **tehis**, mis sisaldab äri seisukohalt vajalikku infot, mille abil määrata, kas antud projekt on investeerimist väärt või mitte.

Kasutatud kirjandus

- [Booch *et al.*, 1999] Booch, Grady, Ivar Jacobson, James Rumbaugh, "The Unified Modeling Language User Guide," The Addison-Wesley object technology series. Copyright © 1998 by Addison Wesley Longman, Inc.
- [CMM] "Reaching CMM Levels 2 and 3 with the Rational Unified Process", <http://www.rational.com/media/whitepapers/tp174.pdf>, Copyright © 1987-2000 Rational Software Corporation.
- [Fowler, 1999] Fowler, Martin "UML Distilled, Second Edition: A Brief Guide to the Standard Object Modeling Language," The Addison-Wesley Object Technology Series, Copyright © 1999 by Addison Wesley Longman, Inc.
- [Jacobson *et al.*, 1999] Jacobson, Ivar, Grady Booch, James Rumbaugh, "The Unified Software Development Process," The Addison-Wesley object technology series. Copyright © 1999 by Addison Wesley Longman, Inc.
- [Kruchten, 1999] Kruchten, Philippe, "The Rational Unified Process: an Introduction," The Addison-Wesley object technology series. Copyright © 1999 by Addison Wesley Longman, Inc.
- [Royce, 1998] Royce, Walker, "Software Project Management: a Unified Framework," The Addison-Wesley object technology series. Copyright © 1998 by Addison Wesley Longman, Inc.
- [RUP, 2000] Rational Software Corporation, "Rational Unified Process Version 2001.03.00," Copyright © 1987-2000 Rational Software Corporation, Portions © Copyright IBM Corporation 1999-2000
- [OPE, 2000] Roger Oberg, Leslee Probasco, Maria Ericsson, "Applying Requirements Management with Use Cases", <http://www.rational.com/media/whitepapers/apprmuc.pdf>, © Copyright 2000 by Rational Software Corporation.
- [TrueSign, 2000] Privador AS "TrueSign: Under the Hood", http://gns.privador.com/tutorial/under_the_hood/under_the_hood.pdf, Copyright © 2000 by Privador AS.

[Waterfall to Iterative, 2000] Philippe Kruchten, “From Waterfall to Iterative Lifecycle – A Tough Transition for Project Managers”, <http://www.rational.com/media/whitepapers/TP173A.pdf> Copyright © 1987 - 2000 Rational Software Corporation.

Unified Software Development Process and a Case Study of It's Application

Asko Seeba

Abstract

On last three decades the software development projects has been very difficult to plan, track and control – only 10% of software projects has been in budget and schedule. This unpredictability has been called as a “software development crisis”. Even classic waterfall approach to software development hasn't been much helpful. The same problem has been met also in Cybernetica AS, the software developing company where the author of the thesis works as a software development manager.

Theory describing unified software development process was formed during last decade and is based mostly on iterative and object-oriented software development methodologies. The main purpose for this process is to adopt to as much software development situations as possible and also eliminate the problem of “software development crisis” that until now makes headaches to software developers.

The purpose of the thesis is to describe the unified software development process, a case study about how this process has been experimented in Cybernetica AS and how Cybernetica AS intends to advance its software development process in all over the company. The case study about the authors experience consists of management of two projects where he tried to apply the process. As it wasn't totally successful (the first project wasn't in budget and schedule), it describes the symptoms of the deviation, reasons that caused the deviation and solutions that would eliminate the problems.

The most important mistake that was made, is that there wasn't any systematic approach to implementing new process in the company. The only statement was that current process doesn't satisfy and new one is needed. Actually, before implementing new process, current organization must be assessed very carefully, most critical gaps must be identified, and start to implement a new process step-by-step from these most problematic issues. Otherwise the development team is too busy in learning new process and tools, focus will be faded away from important issues and thus, project will suffer anyway.