View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by DSpace at Tartu University Library

T A R T U   Ü L I K O O L

MATEMAATIKA-INFORMAATIKATEADUSKOND

Arvutiteaduse instituut

Informaatika eriala

Katrin Toe

Modulaarsuse mõõtmine tarkvara analüüsifaasis

Magistritöö (30 EAP)

Juhendaja: Peep Küngas

Autor: …………………………………… "….." jaanuar 2012

Juhendaja: ……………………………… "….." jaanuar 2012

Lubada kaitsmisele

Professor: ……………………………….. "….." jaanuar 2012

TARTU 2012

Sisukord

1 Sissejuhatus ... 3
2 Taust ... 6
2.1. Tarkvara modulaarsus ... 6
2.2. Modulaarsuse põhiprintsiibid ... 8
2.3. Modulaarsuse hindamine ... 12
2.4. Modulaarsusest tulenev kasu ... 13
2.5. Modulaarsus süsteemi analüüsimisel ... 14
3 Modulaarsuse mõõtmise raamistik ... 17
3.1. Tarkvara meetrika nõuded ... 17
3.2. Eesmärk-Küsimus-Meetrika ... 19
3.3. Meetrikate kirjeldus ... 23
3.4. Modulaarsuse mõõtmise tehnikad ... 29
4 Juhtumiuuring ... 32
4.1. Süsteem EMPIS ... 32
4.2. Andmete kogumine ... 33
4.3. Disaini struktuuri maatriksi automaatne koostamine ... 35
4.4. Meetrikate rakendamine ... 39
4.5. Regressioonanalüüs ... 43
4.6. Juhtumiuuringu kokkuvõte ... 51
5 Kokkuvõte ... 53
Abstract ... 54
Sõnastik ... 56
Viited ... 58
Lisad ... 63
Lisa 1. EMPIS tehnilise ülesande näidis ... 63

1 Sissejuhatus

Tarkvaratööstus püüdleb uute tehnikate ja lähenemiste poole, mis aitaks suurendada arenduse produktiivsust, vähendada tootmise aega ja parandada süsteemi kvaliteeti. Enamasti igal sellisel lähenemisviisil on olnud märkimisväärne edu, aga kuna nõuded muutuvad järjest kompleksemaks, jääb endiselt väljakutseks luua nendele ootustele vastavat tarkvara [1].

Märkimisväärselt palju aega kulutatakse tarkvara täiendus- ja hooldustöödele ehk töödele, mis tehakse pärast toote väljalaset. Suur muudatuste hulk on seletatav kahe tarkvara evolutsiooni seadusega [2]:

- Pidev muutumine - kasutatav tarkvara peab muutuma ja täienema või vastasel juhul muutub see vähem kasulikuks järjest muutuvas maailmas, kus kasutajatel on järjest suuremad nõudmised.
- Suurenev komplekssus - kuna tarkvara areneb, muutub see järjest keerulisemaks ning on vaja eraldi ressursse, et tarkvara käigus hoida ja selle struktuure lihtsustada.

Pidevate muudatuste tõttu tarkvara jõudlus ja usaldusväärsus kahaneb, samal ajal kui ülalpidamise kulud suurenevad [3]. Tekib olukord, kus ühe väikese vea parandamine toob kaasa mitu uut viga ning lihtsa täienduse realiseerimine nõuab laiaulatuslikke koodi muudatusi. Seetõttu on selge, et iga lähenemisviis, mis lihtsustab muudatuste sisseviimist, viib majanduslikust aspektist vaadatuna märkimisväärse kokkuhoiuni.

Ühe võimaliku lahendusena, kuidas suuri ja keerulisi süsteeme odavamalt hallata, viidatakse kirjanduses tihti modulaarsusele [4] [5]. Tarkvara peetakse modulaarseks, kui see on üles ehitatud omavahel vähe seotud iseseisvatest alamosadest nii, et ühe alamosa muutusel on minimaalne mõju teistele alamosadele ning seetõttu on võimalik alamosasid eraldi arendada, täiendada ja taaskasutada [6]. Modulaarsus minimeerib muudatuste mõju ja võimaldab tarkvara odavamalt hallata.

Praktikas pole modulaarsust aga kerge saavutada, ehkki võib tunduda intuitiivne, et tihedalt põimunud ja keerukad süsteemi alamosade vahelised seosed tähendavad edaspidi suuremaid halduskulusid. Probleem on tingitud eelkõige ajapuudusest. Ajasurve tõttu

pingutatakse järgmise versiooni nimel, mõeldakse välja ajutisi lihtsaid lahendusi ja ignoreeritakse häid disaini tavasid. Unustatakse tarkvara tootmise baastõde: varastes arendusfaasides tehtud vead kanduvad edasi järgnevatesse faasidesse, kus neid parandada on oluliselt kulukam.

Modulaarsuse saavutamisel peetakse kõige olulisemaks teguriks tarkvara arhitektuuri, mis määrab ära, kuidas süsteemi on võimalik üles ehitada ja millisel viisil erinevad alamosad omavahel suhtlevad [7]. On selge, et süsteemi heaolu ja töökindlus sõltub eelkõige arhitektuuriotsustest, mille vead maksavad kõige rohkem. Kirjanduses on aga erilise tähelepanuta jäetud süsteemianalüütiku roll modulaarsuse saavutamisel. Samas tuleb arvestada, et muudatuste tegemine võib olla raskendatud ka seetõttu, et tarkvara analüüsimisel pole arvestatud modulaarsuse aspektidega. Näiteks kui analüütik on süsteemi komponentidesse jaotanud nii, et komponentide vahel on palju keerulisi ja arusaamatuid seoseid, on ka edaspidi selliseid komponente raskem muuta ning hilisemat restruktureerimist tavaliselt välditakse, kuna ei suudeta hinnata muudatuse mõjuulatust. Pigem arendatakse teadaolevalt töötavat lahendust edasi, leppides, et tarkvara hoolduskulud järjest tõusevad [8]. Kui võtta aga eesmärgiks süsteemi eluea maksimeerimine, tuleks modulaarsusega arvestada juba tarkvara analüüsifaasis. Kvaliteetne analüüs on eelduseks sellele, et modulaarsus on saavutatav.

Antud töös uuritakse, kuidas süsteemianalüütik modulaarsust mõjutada saab ning kuidas tuvastada modulaarsusega seotud analüüsivigu, mis hiljem mõjutavad tarkvara kvaliteeti negatiivselt. Eesmärk on leida kvantitatiivselt mõõdetav meetrika, mis on sobiv tarkvara projekti analüüsifaasis modulaarsuse mõõtmiseks. Töö käigus konkretiseeritakse arvutuslik mudel sidestuse, kohesiooni ja komplekssuse arvutamiseks, kasutades disaini struktuuri maatriksil põhinevat lähenemist. Arvutuslik mudel võimaldab analüüsi artefakte modulaarsuse seisukohalt võrrelda, hinnata ning identifitseerida kõrge veariskiga komponendid, mida tuleks modulaarsuse ja parema kvaliteedi saavutamiseks ümber struktureerida. Mudel valideeritakse juhtumiuuringu raames reaalse projekti peal, mille käigus koostatakse ka automaatne disaini struktuuri maatriksi koostamise programm sõltuvuste esitamiseks.

Kuna modulaarsus on pigem fundamentaalne disainiprintsiip (esmakordselt mainitud juba aastal 1970 [9]), siis selle järgmine ei sõltu tehnoloogiast ega programmeerimise

paradigmast. Nimelt nii objekt-orienteeritud, komponendi-põhine kui teenustele orienteeritud arhitektuur kasutab modulaarsusega seotud printsiipe (sidestuse minimeerimine, kohesiooni suurendamine, informatsiooni varjamine jne). Ka selles töös käsitletakse modulaarse disaini põhimõtteid üldiselt, sõltumata tehnoloogiast või paradigmast, mida on kasutatud süsteemi realiseerimiseks. Juhtumiuuring baseerub aga objekt-orienteeritud süsteemi näitel.

2 Taust

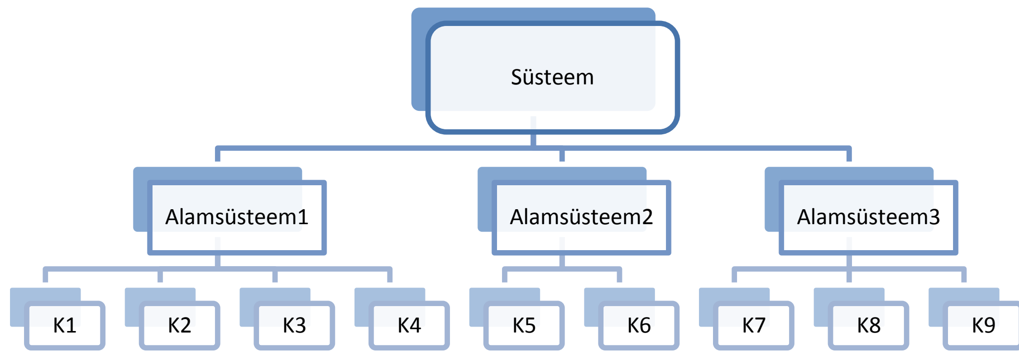
Antud peatükk sisaldab teoreetilist osa käesolevast magistritööst. Defineeritakse termin „modulaarsus" ja selgitatakse teisi olulisi modulaarsusega seotud aspekte, et lugeja saaks aru probleemi olemusest, mõistmaks antud töö praktilist osa. Tarkvara kvaliteedi tagamise seisukohalt peaks modulaarsust vaatlema mitmetest vaatepunktidest. Antud peatükk tutvustab lähenemisviise, mis katavad ära kõige olulisemad nõuded modulaarse disaini jaoks. Eesmärgiks pole täpselt kirjeldada, milline perfektne süsteem välja näeb, vaid uurida modulaarsusega seotud aspekte ja analüüsida omadusi, mida modulaarne struktuur peab rahuldama. Põhifookus on süsteemi analüüsifaasis rakendatavatel disaini meetoditel, kuid need peavad kehtima ka süsteemi realiseerimise, testimise ja hoolduse faasis.

2.1. Tarkvara modulaarsus

Kirjanduses viidatakse modulaarsusele kui ühele kvaliteedinäitajale, mille saavutamine on ihaldusväärne. Seda peetakse üheks olulisemaks tarkvara omaduseks, mis mõjutab väga tugevalt süsteemi muudetavust ja arenemisvõimelisust [3] [5].

Termin „modulaarsus" on kasutusel ka muudes eluvaldkondades, mis teeb selle raskemini mõistetavaks. Tarkvaratööstuses võib aga modulaarseks pidada toodet või süsteemi, mis on üles ehitatud omavahel vähe seotud osadest nii, et ühe alamosa muutusel on minimaalne mõju teistele alamosadele ning seetõttu on võimalik alamosasid eraldi arendada, täiendada või taaskasutada [6]. Piltlikult öeldes ehitatakse süsteem üles klotsidest ning igat klotsi saab iseseisvalt arendada ja süsteemi lisada ilma, et see teiste klotside tööd oluliselt mõjutaks. Seega modulaarsus võimaldab kompleksse süsteemi jaotada väiksemateks isoleeritud tükkideks, mida on eraldiseisvalt lihtsam lahendada.

Modulaarsust võib vaadelda erinevatel tasemetel (joonis 2.1). Toote või süsteemi tasandil nõutakse, et kompleksed süsteemid oleksid disainitud alamsüsteemidena nii, et alamsüsteemid oleks omavahel võimalikult vähe seotud, alamsüsteemide sees esineksid aga tugevad seosed [10]. Alamsüsteemi on omakorda võimalik jaotada komponentideks (joonisel 2.1 K1, K2…K9), mis koosnevad juba väiksematest mikrostruktuuridest.



Joonis 2.1: Süsteemi hierarhiline dekompositsioon

Komponent antud töös tähendab süsteemi dekompositsiooni ühikut. Sõltuvalt programmeerimise paradigmast võib komponendiks olla nii moodul, pakett, klass, teenus või mistahes eraldi seisev tükk süsteemist, mis sisaldab sarnaseid funktsioone või andmeid.

Modulaarse süsteemi disainimisel on üheks kõige suuremaks väljakutseks, kuidas süsteemi jaotada komponentidesse. Eesmärk on luua komponendid, mis rahuldavad järgnevaid tingimusi [1]:

- On taaskasutatav.
- Sisaldab omavahel seotud funktsioone.
- Iseseisev, saab eraldiseisvalt arendada ilma, et see teisi komponente mõjutaks.
- On piisavalt arusaadav ja võib kasutada kui „musta kasti“ (üksnes välise kirjelduse põhjal ilma komponendi sisemisi struktuure uurimata).

Järgnevaks väljakutseks on garanteerida, et iseseisvad alamsüsteemid ja komponendid tõesti töötavad koos. Praktikas lahendatakse see versioneerimisega: iga alamsüsteem on eraldi versioneeritav. Samuti peab olema tagatud, et kui lastakse välja uus alamsüsteemi versioon, jääb andmevahetus ülejäänud alamsüsteemidega toimima [8]. Selleks on oluline, et süsteemi igal tasandil toimub alamosade vaheline suhtlus standardiseeritud reeglite järgi [8] ja kõik seosed on hästi dokumenteeritud.

2.2. Modulaarsuse põhiprintsiibid

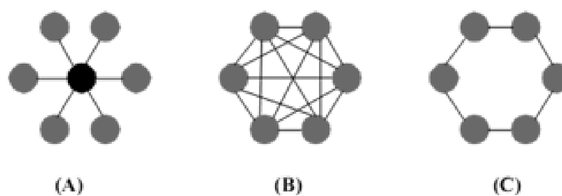
Järgnevalt antakse ülevaade kõige enam tuntumatest ja laialdasemalt kasutatud modulaarsust mõjutavatest põhiprintsiipidest, mis kehtivad nii analüüsi kui arenduse faasis. Neist kõige olulisemad on sidestus ja kohesioon.

2.2.1. Sidestus

Sidestus (ingl. k *coupling*) tähendab erinevate süsteemi alamosade vahelist vastastikust seost või sõltuvust [11] [12]. Mida rohkem seoseid, seda enam sõltub komponent teistest komponentidest ning ühte komponenti muutes võib olla vaja muuta ka sellega seotud komponente. Arenduse eesmärk on sidestuse minimeerimine, mis viib vähese või puuduva vastastikuse sõltuvuseni, kus komponendid pole kas seotud või nende vahel on lihtsad seosed [13]. Selliseid komponente on kergem ka hallata, kuna neid arendatakse, testitakse ja integreeritakse eraldi.

Komponentide vahelised sõltuvused võivad muutuda süsteemi igas versioonis. Kui komponente lisatakse, muudetakse või kustutatakse, muutuvad ka komponentide vahelised seosed. Sageli võivad need seosed olla varjatud ja ilmneda vigadena alles süsteemi kasutamisel. Keerukate sõltuvuste tõttu on ka süsteemi ülevalpidamine kulukam ning äärmuslikel juhtumitel võib süsteem minna nii keerukaks, et seda on peaaegu võimatu muuta.

Joonis 2.2 illustreerib, kuidas jõuda miinimum seosteni: kui süsteem koosneb n komponendist, siis minimaalne seoste arv nende vahel on n-1. Juhtumi A korral koosneb süsteem kesksest komponendist, mille kaudu toimub suhtlus kõikide ülejäänud komponentidega. Ka juhtumi C korral on seoste arv väike, mille korral struktuur erineb küll traditsioonilisest ülevalt-alla lähenemisest, kuid ka sellel võivad olla oma eelised. Kindlasti tuleks aga vältida lahendust B, kus seoste arv on maksimaalne [14].



Joonis 2.2: Komponentide vahelised seosed [14]

Komponendid võivad olla seotud erinevalt ning sellest sõltub ka modulaarsuse tase. Järgnevalt on toodud sidestuse erinevad tasemed järjestatult halvimast parimani [15]**:**

Sidestuse tüüp	**Tase**	**Kirjeldus**
Sisu sidestus	5	Komponent kasutab või muudab teise komponendi andmeid või kontrollib teist komponenti. Üks komponent otseselt viitab või kasutab teise komponendi sisu või muudab andmete töötlust. Muudetakse sõltumatut komponenti.
Üldine sidestus	4	Eelnevaga sarnane, kuid kasutatakse globaalseid muutujaid või andmetabeleid. Kahel või enamal komponendil on lugemise ja kirjutamise õigus samadele andmetele. Tekivad järgnevad probleemid: • Koodi raskem mõista, sest on vaja teada kõiki komponente, mis sama andmeelementi võivad muuta. • Piiratud komponendi taaskasutamisvõimalus varjatud seoste tõttu. • Võimalikud turvaprobleemid: mitteautoriseeritud juurdepääs delikaatsete andmetele. Probleemi saaks lahendada, kui samadele andmetele oleks kirjutamisõigus ainult ühel komponendil ja teistel komponentidel vaid lugemisõigus.
Juhtimissidestus	3	Kui üks komponent saab otseselt mõjutada teise komponendi käivitamist. Näiteks üks komponent edastab teisele komponendile parameetri, mille peale teine komponent käivitab teatud tegevuse. Suurim probleem on seotud koodi taaskasutusega: kaks komponenti pole enam sõltumatud.
Välissidestus	2	Komponent edastab parameetrina suure andmeühiku (nt kirje) teisele komponendile, mis kasutab ainult osa neist edastatud andmetest. Näiteks komponent A edastab komponendile B kogu isiku tabeli kirje, olgugi, et komponendil B on vaja teada ainult ühte välja (nt isiku nime). See paneb komponendi B sõltuma isiku tabeli andmestruktuurist, mis tekitab samu probleeme nagu üldise sidestuse korral.
Andmesidestus	1	Komponendid suhtlevad parameetrite kaudu, kus parameetriks on kas üksikud lihtsa andmetüübiga elemendid või juhul kui

		parameetrina edastatakse terve kirje, siis kõiki vastava kirje elemente on vaja kasutada teises komponendis. Lihtsad parameetrid, lihtsad suhtlusviisid.
Sidestus puudub	0	Komponendid pole omavahel seotud.

Tabel 2.1: Sidestuse tasemed

Sidestuse tase näitab, kui suure tõenäosusega tuleb ühe komponendi muutmisel teha muutus ka teise komponenti ning kui lihtne on sellist muudatust teha. Komponente sidudes peaks olema eesmärgiks saavutada andmesidestus, mis lihtsustab edaspidiste muudatuste sisseviimist ning vähendab süsteemi keerukust [15]. Kui kaks komponenti suhtlevad, peaksid nad omavahel vahetama nii vähe informatsiooni kui võimalik [14]. Mida vähem andmeid nad vahetavad, seda sõltumatumad nad on üksteisest [14].

2.2.2. Kohesioon

Kohesioon (ingl. k. *cohesion*) iseloomustab komponendi sisemisi seoseid [12]. Kohesioon näitab, mil määral on komponendi elemendid omavahel seotud. Tugev kohesioon viitab komponendi elementide omavahelisele tugevale seotusele. Arenduse eesmärk on suurendada kohesiooni nii palju kui sidestuse minimeerimiseks vajalik. Ühte ülesannet peab täitma üks komponent ning selle ülesande täitmiseks tuleb minimaalselt suhelda ülejäänud süsteemi komponentidega. Ühes alamsüsteemis võiksid asuda omavahel tugevalt seotud komponendid ning komponent võiks koosneda omavahel tugevalt seotud elementidest [12].

Komponentide osad võivad olla seotud erinevalt ning sellest sõltub ka kohesiooni tase. Järgnevalt on toodud kohesiooni erinevad tasemed järjestatult halvimast parimani [15]:

Kohesiooni tüüp	Tase	Kirjeldus
Juhuslik kohesioon	7	Komponendi osad pole omavahel seotud. Nad on seotud ainult seetõttu, et asuvad samas komponendis.
Loogiline kohesioon	6	Komponendi osad on loogiliselt seotud, kuid muus osas ei kuulu kokku. Näiteks erinevatelt sisendkandjatelt pärit andmete töötlus ühes komponendis.
Ajaline kohesioon	5	Komponendi osade vahel on loogiline kohesioon ja neid kõiki on vaja kasutada samal ajahetkel.
Protsessi kohesioon	4	Komponendi osad on seotud seetõttu, et neid tuleb täita

		teatud järjekorras ja nad kõik annavad panuse teatud protsessi. Näiteks üks element kontrollib faili juurdepääsu õigusi ja teine avab selle faili.
Suhtluse kohesioon	3	Komponendi osad on seotud, kuna nad töötlevad samu andmeid.
Järjestuslik kohesioon	2	Ühe komponendi osa väljund on teise sisendiks.
Funktsionaalne kohesioon	1	Komponendi kõik osad teostavad ühte konkreetset ülesannet.

Tabel 2.2: Kohesiooni tasemed

Eesmärgiks on funktsionaalse kohesiooni saavutamine, mis parandab süsteemi kvaliteeti tänu paremale komponentide jaotusele ja arusaadavamale struktuurile. See lihtsustab edaspidiste muudatuste ja täienduste sisseviimist, suurendab süsteemi arusaadavust ning vähendab süsteemi keerukust, mis omakorda võimaldab komponenti taaskasutada [15]. Kõrge kohesioon näitab head komponentide alamjaotust. Madal kohesioon suurendab kompleksust ja tõstab vigade arvu. Madala kohesiooniga komponendid tuleks jagada alamkomponentideks, mis on rohkem sidusad.

2.2.3. Komponendi suurus ja komplekssus

Üldine printsiip on hoida komponente väikeste ja lihtsatena, et need oleks arusaadavad ning lihtsasti arendatavad ja testitavad [16]. Suurte komponentide sidestus suureneb, mis piirab komponendi taaskasutust. Uuringud on näidanud, et suured ja kompleksed komponendid tekitavad rohkem programmeerimise vigu ning neid on raskem testida [16].

2.2.4. Autonoomsus

Autonoomsus näitab, et komponent on piisavalt iseseisev ehk teistest komponentidest ja välistest mõjudest sõltumatu ning komponendil on kontroll oma osade üle. See võimaldab komponenti paremini taaskasutada ning viitab suuremale modulaarsusele. Disaini eesmärgiks on autonoomsuse suurendamine.

2.2.5. Informatsiooni peitmine

Informatsiooni peitmine (ingl. k *information hiding*) on printsiip, mille kohaselt komponendil on ametlik kirjeldus, mis sisaldab kõige olulisemat funktsionaalsuse kirjeldust ja kõik ülejäänud süsteemi sisemisi struktuure puudutav info on peidetud.

Seega komponenti saab kasutada kui „musta kasti“. Informatsiooni peitmine võimaldab eristada komponendi sisemisi detaile selle välisest kirjeldusest. Kui komponendi sisemised struktuurid muutuvad, kuid väline mitte, siis see muutus ei tohi põhjustada seotud komponentide muudatusi. Sellega vähendatakse koordinatsiooni kulusid ja hõlbustatakse muudatuste tegemist ilma teisi komponente mõjutamata. Informatsiooni peitmisega lähedased mõisted on abstraktsioon ja kapseldamine.

2.3. Modulaarsuse hindamine

Nii analüüsi kui arenduse faasis saab modulaarsust hinnata Meyeri poolt väljapakutud viie kriteeriumi abil [14]:

1) **Modulaarne dekompositsioon** – antud kriteerium on täidetud, kui tarkvara probleemi on võimalik jagada lihtsamateks alamprobleemideks, mis on üksteisest piisavalt sõltumatud nii, et igat alamprobleemi saab iseseisvalt edasi arendada ja vajadusel veelkord dekomponeerida. Süsteem on jaotatud alamsüsteemideks ja igat alamsüsteemi võib arendada erinev meeskond, kes ei pea ootama teise alamsüsteemi arenduse järel, kuna alamsüsteemide vahelised seosed on viidud miinimumini. Samas on alamsüsteemide vahelised seosed selgelt välja toodud ja see võimaldab neid omavahel kokku panna, et moodustada terviksüsteem. Kõige tuntuim meetod selle kriteeeriumi rahuldamiseks on ülevalt-alla lähenemine. Alustatakse kõige abstraktsemast süsteemi funktsioonide kirjeldusest ja seejärel jaotatakse need erinevatesse alamsüsteemidesse ja sealt edasi minnakse järjest detailsemaks kuni elemendid on kirjeldatud piisavalt madala abstraktsioonitasemega, et neid oleks võimalik lihtsasti realiseerida.
2) **Modulaarne kompositsioon** - kui tarkvara elemente saab omavahel kombineerida ja moodustada seeläbi uusi elemente või alamsüsteeme. Selle saavutamiseks peavad komponendid olema piisavalt autonoomsed ehk üksteisest sõltumatud. Antud kriteerium on otseselt seotud eesmärgiga komponenti taaskasutada. Kompositsioon on dekompositsioonist sõltumatu.
3) **Modulaarne arusaadavus** – kui komponent on inimesele arusaadav ilma teisi komponente uurimata. Antud kriteerium muutub väga oluliseks tarkvara hooldus-

ja täiendustööde faasis, kus olemasolevatesse komponentidesse on vaja teha uusi muudatusi ja täiendusi. Sellisel juhul on oluline, et väikse muudatuse saab teha ilma teistesse komponentidesse süvenemata. Vastasel juhul võib iga väikse täienduse tegemine minna liiga kulukaks. Näiteks kui teatud komponendid on disainitud nii, et neid tuleb välja kutsuda kindlas järjekorras. Oletame, et komponent B töötab õigesti vaid juhul kui see käivitatakse pärast komponenti A ja enne komponenti C. Sellisel juhul komponendi B arusaamiseks ja muutmiseks on suure tõenäosusega vaja aru saada ka komponentidest A ja C. Järelikult komponendid A, B ja C ei täida arusaadavuse kriteeriumi.

4) **Modulaarne katkematus** - kui lihtsa muutuse korral tuleb teha muudatus vaid ühte komponenti või halvimal juhul paari. Väikesed muudatused peavad muutma üksikuid komponente süsteemi struktuuris, mitte struktuuri ennast. Näiteks selle eesmärgi saavutamiseks tuleks kasutada sümboolseid muutujaid. Süsteem ei tohiks otseselt ühtegi numbrilist ega tekstilist väärtust kasutada, vaid selle asemel sümboolseid nimesid ja tegelikke väärtusi hoitakse eraldi. Sellisel juhul on tagatud, et väärtuse muutumisel tuleb muudatus teha vaid ühes kohas.
5) **Modulaarne kaitse** - kui komponendi töös tuleb ette tõrge, mõjutab see vaid antud komponenti või halvimal juhul propageerub mõnda üksikusse naaberkomponenti. Näiteks tuleks selle eesmärgi saavutamiseks alati valideerida komponendi sisendit.

2.4. Modulaarsusest tulenev kasu

Praktikas pole modulaarsust kerge saavutada, kuid see on mitmeski mõttes kasulik. Modulaarsus loob uusi võimalusi, millest kõige olulisemad on järgnevad:

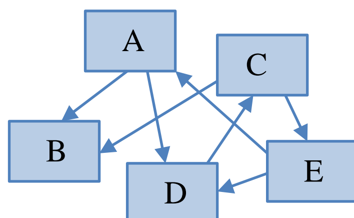
- Modulaarsus võimaldab paralleeltööd [17] [5]. Komponente on võimalik analüüsida, arendada ja testida iseseisvalt enne kui need integreeritakse kogu süsteemi. Arendaja ei pea teadma kogu süsteemi, vaid saab keskenduda väiksemale osale. Igat süsteemi alamosa võib arendada erinev meeskond erineva ajagraafiku ja elutsükliga [18]. Alamosa integreerimisel kogu süsteemi ei pea ootama teiste arendusmeeskondade järel [19].

- Modulaarsus võimaldab tarkvara tootmise kulusid vähendada paranenud produktiivsuse ja lühenenud arendusaja tõttu. Tarkvara täiendamise ja ülalpidamise kulud vähenevad, kuna tarkvara on lihtsam muuta, testida, ja taaskasutada [20].
- Modulaarsus teeb komplekssuse hallatavaks, kuna süsteem jaotatakse osadeks, mida on võimalik arendada iseseisvalt [17]. Muudatus ühte komponenti ei mõjuta teisi komponente [5]. Eriti oluline suurte ja keeruliste süsteemide korral, kus süsteemi komponentide vahelisi seoseid on nii palju, et ilma modulariseerimata muutub arendus peaaegu võimatuks: ühel hetkel võib isegi väikeste muutuste tegemine minna nii kulukaks, et odavam on süsteem ära visata ja ehitada uus [5] [21].
- Modulaarsus suurendab tarkvara kvaliteeti ja mõjutab positiivselt teisi kvaliteedinäitajaid, näiteks taaskasutatavust, hooldatavust, paindlikust, kasutuskõlblikkust, arusaadavust jne [22].
- Modulaarsus soodustab paindlikkust ja innovatsiooni [17] [19] [5].
- Modulaarsus mitmekordistab disaini valikuid, kuna komponente on võimalik segada ja sobitada [4].
- Modulaarsus kohandub ebakindlusega, sest ühe komponendi muutumisel on väike mõju teistele komponentidele [17] [10].
- Modulaarsed süsteemid on arusaadavamad [19] ja neid on lihtsam nii arendada, testida kui hooldada [62].
- Modulaarsed süsteemid kasvavad kiiremini ning on stabiilsemad [5].

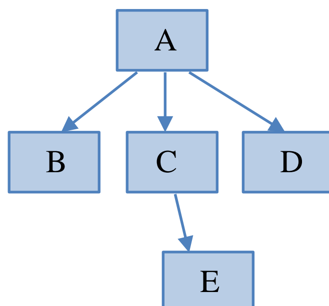
2.5. Modulaarsus süsteemi analüüsimisel

Modulaarsuse saavutamisel peetakse kõige olulisemaks teguriks tarkvara arhitektuuri, mis määrab ära, kuidas süsteemi saab üles ehitada ja millisel viisil erinevad alamosad omavahel suhtlevad [7]. On selge, et süsteemi heaolu ja töökindlus sõltub eelkõige arhitektuuriotsustest, mille vead maksavad kõige rohkem. Kirjanduses on aga erilise tähelepanuta jäetud süsteemianalüütiku roll modulaarsuse saavutamisel. Samas tuleb arvestada, et just süsteemianalüüsi faasis toimub ärinõuete põhjal süsteemi modelleerimine, mis on sisendiks programmeerijatele. Probleeme on võimalik lahendada

väga erinevalt, mis aga annab ka süsteemile erinevad omadused ja võimalused ning võib programmeerija jaoks tähendada kordades erinevat töömahtu. Ehkki ka analüütiku valikuid mõjutavad arhitektuuriotsused ja kasutatav tehnoloogia, saab analüütik süsteemi kirjeldada ja komponentidesse jaotada väga erinevalt, millest aga sõltub ka see, kas modulaarsus on saavutatav. Analüütiku valikuid iseloomustav alljärgnev näide:



Joonis 2.3: Tüüpiline tarkvara süsteem, mis iseloomustab komponentide vahelisi keerukaid ja ebaselgeid seoseid.



Joonis 2.4: Ideaalne tarkvara süsteem, kus komponentide vahel on lineaarsed ja selged seosed.

Joonis 2.3 on näide halvast disainist, kus iga süsteemi alamosa võib suhelda suvalise teise osaga, mille tõttu süsteemi sisemine struktuur muutub arusaamatuks ja lihtsasti haavatavaks. Sõltuvused lähevad süsteemi arenedes järjest keerulisemaks ja isegi väikse vea parandamine võib kaasa tuua mitu uut viga. Kujunevad välja süsteemi osad, mida ükski meeskonnaliige ei julge muuta. Halvad disainivalikud tulenevad eelkõige ajapuudusest: probleeme üritatakse lahendada nii kiiresti ja lihtsalt kui võimalik, mõtlemata, kas ka pikemas perspektiivis selline lahendus ennast õigustab.

Parema tarkvara kvaliteedi saavutamiseks (joonis 2.4) tuleb seostele mõelda juba analüüsifaasis, et sõltuvuste struktuur oleks selge ja loogiline. Analüütik peaks järgima modulaarsuse printsiipe ja kirjeldama süsteemi nii, et komponentide vaheline suhtlus toimuks standardiseeritud reeglite järgi ning kõik seosed oleks ilmsed ja hästi

defineeritud. Isegi kui ajalimiit on piiratud, saab sõltuvusi luua kontrollitud viisil, et vältida joonisel 2.3 kujutatud ebasoovitud sõltuvusi. See ei garanteeri modulaarset lähtekoodi, kuid lihtsustab süsteemi arendust ja hooldust ning võimaldab muudatusi kergemini sisse viia. See on püüdlus kvaliteetsema analüüsi poole.

Üldiselt – mida vähem sõltuvusi, seda parem. Praktikas pole see aga võimalik. Süsteemi alamosade vahelisi sõltuvusi ei saa vältida ega ette ennustada, kuid sõltuvuste struktuuri on võimalik kontrollida ning sõltuvusi isoleerida nii, et muudatuse tegemisel on muudatuse mõjuulatus väiksem.

Analüütik peaks modulaarsuse saavutamiseks juhinduma modulaarsuse printsiipidest ning järgnevatest Parnase poolt kirjeldatud [3] põhimõtetest, mida ajapuuduse tõttu kiputakse ignoreerima:

- Tarkvara disain peab olema üles ehitatud nii, et see arvestab pidevate muudatustega (ingl. k *Design for change*). Ehkki muudatusi pole võimalik ette ennustada ning kõiki komponente ei saa teha võrdväärselt lihtsasi muudetavaks, on oluline modelleerida tarkvara nii, et see arvestaks pidevate täiendustega. Selleks tuleb tähelepanu fokusseerida süsteemi pikemajajalise heaolu peale, mitte töötada ainult järgmise versiooni väljalaske nimel.
- Tarkvara peab olema hästi dokumenteeritud. Kvaliteetse dokumentatsiooni olemasolul on sõltuvusi kergem kontrollida ning muudatusi sisse viia odavam, kuna kogu vajalikku infot ei pea lähtekoodi pealt uuesti genereerima. Samuti sisaldab dokumentatsioon rohkem läbimõeldud infot. Nii uus funktsionaalsus kui ka muudatused tuleb dokumenteerida ja teha hoolikas ülevaatus. Ilma kvaliteetse dokumentatsioonita tarkvara pole arenemisvõimeline.
- Analüüsi artefaktidele tuleks teha ülevaatus teiste spetsialistide poolt. Ajasurve võib tekitada küll tunde, et nõuetekohasteks ülevaatusteks pole aega, aga analüütiku vead on ühed kulukamad.

3 Modulaarsuse mõõtmise raamistik

Kirjanduses leidub hulganisti materjale, mõistmaks, miks modulaarsus on hea ja millist kasu see pakub. Aga vähe on süstemaatilisi uurimusi, kuidas otsuse tegijad jaotavad süsteemi komponentideks ja millised riskid kaasnevad, kui jaotada valesti. Ühelt poolt on modulaarne disain vastuseks komplekssuse haldamisele, teiselt poolt on vaja aru saada, kuidas „head või sobivat“ modulaarset disaini saavutada [5].

Kõige lihtsam on see, kui analüütik järgib modulaarsuse printsiipe ja jaotab nõuded alamsüsteemidesse ja komponentidesse nii, et komponentide vahel on võimalikult vähe seoseid. Paraku pole süsteemi disain täpne teadus, mis alati põhjustab optimaalse lahenduse. Nõuete jaotus komponentidesse pole iseenesest ilmne, sest kompleksse süsteemi disain sisaldab rohkelt valikuid ning tulevikku pole võimalik ette ennustada [21].

Õigete otsuste tegemiseks on vaja hinnata ja mõõta analüüsi tulemeid juba enne realiseerimist. Vajalik on luua raamistik modulaarsuse mõõtmiseks analüüsifaasis. Eesmärgiks pole leiutada uusi meetrikaid, vaid valida olemasolevate meetrikate kollektsioonist antud töö jaoks kõige sobivamad, võrrelda meetrikaid ja nende potentsiaalset kasutamist ning pakkuda välja erinevaid meetrikate hübriide. Tulemused võimaldavad avastada analüüsi vigu palju efektiivsemalt.

Kirjanduses on toodud hulganisti modulaarsuse mõõtmise meetrikaid, kuid enamasti põhinevad need lähtekoodi mõõtmises, kus mõõtmise objektiks on klassid, meetodid ja atribuudid. Neid meetrikaid saab rakendada aga alles pärast seda, kui kood on valmis. Käesolevas töös toodud meetrikad keskenduvad aga analüüsi ja disainifaasile - kuidas modelleerida süsteemi nii, et programmeerijatele antav sisend võimaldaks luua modulaarsemat süsteemi.

3.1. Tarkvara meetrika nõuded

Tarkvara kvaliteedi järjest suurenev tähtsus nõuab reeglistikku, et hinnata tarkvaratoote kvaliteeti objektiivselt ja kvantitatiivselt. Selleks otstarbeks on alates 1960-ndate keskelt välja töötatud tuhandeid meetrikaid [23]. Mõistega „tarkvara meetrika“ viidatakse mitmesugustele tarkvara osade mõõtmistele, mis aitavad hinnata ja ennustada tarkvara

kulusid ja kvaliteeti [23]. IEEE defineerib meetrika kui kvantitatiivse mõõdiku, mis näitab, mil määral süsteem, komponent või protsess vastab teatud nõuetele [24]. Meetrika rakendamisel teostatakse teatud reeglite järgi mõõtmine ning tulemuste põhjal hinnatakse, mil määral mõõdetav alamsüsteem või komponent järgib hea disaini printsiipe. Tulemused aitavad kontrollida arendusprotsessi, parandada tarkvara kvaliteeti ja produktiivsust ning teha otsuseid tarkvara ümberstruktureerimiseks. Eesmärk on luua juhtimisotsuste abistamise vahend, mis kombineerib erinevaid tarkvaraarenduse aspekte ning võimaldab teha mitmeid ennustusi ja anda hinnanguid [23].

Sõltuvalt mõõtmise eesmärgist jaotatakse tarkvara meetrikad 3 kategooriasse [23]:

1. *Protsesside mõõtmine* – arendusprotsesside efektiivsuse hindamiseks. Mõõdetakse arenduse aega, raporteeritud vigade arvu, muudatuste arvu jne. Tulemused aitavad vastata küsimustele, kui kaua aega läheb protsessi lõpuni jõudmiseks, kui palju see maksab, kas protsess on piisavalt efektiivne jne.
2. *Toote mõõtmine* – tarkvara tulemite hindamiseks. Mõõdetakse nii tarkvara koodi kui dokumentatsiooni, et hinnata tarkvara kvaliteeti.
3. *Ressursside mõõtmine* – protsessi tegevuste jaoks vajalike ressursside hindamiseks.

Antud töö eesmärgiks on hinnata tarkvara toote kvaliteeti. Toote kvaliteedi mõõtmiseks pakutakse kirjanduses välja kõige enam meetrikaid objekt-orienteeritud süsteemidele. Enamasti mõõdetakse programmi lähtekoodi – klasse, meetodeid ja nende vahelisi seoseid.

Tarkvaratööstus on võtnud vastu lihtsamad meetrikad, sest need on arusaadavamad ja lihtsamini kogutavad, kuid keeldunud keerulistest [1]. Paljud akadeemilised meetrikad on praktikas ka sobimatud, kuna nende skoop ja sisu ei lähe kokku reaalsusega. Akadeemilised meetrikad on fokuseeritud väikestele programmidele, samas kui tarkvaraarenduses on meetrikaid mõistlik rakendada suurte ja keeruliste süsteemide puhul. Paljud akadeemilised mudelid põhinevad parameetritel, mida praktikas mitte kunagi ei mõõdeta.

Selleks, et meetrikaid oleks võimalik praktikas rakendada, peavad need vastama järgnevatele nõuetele [25]:

- *Objektiivsus* - mõõtmine peab andma sama tulemuse sõltumata sellest, kes mõõtmise teostab. Subjektiivseid mõjutusi tuleb vältida.
- *Täpsus* - samades tingimustes mõõtmise teostamine peab andma sama tulemuse.
- *Valiidsus* - meetrika väärtused peavad vastama teatud kriteeriumidele.
- *Kasu* – meetrikast peab olema praktiline kasu.
- *Ökonoomsus* – mõõtmiste pealt saadav kasu peab olema suurem kui mõõtmiste läbiviimiseks tehtavad kulud. Sõltub otseselt sellest, kui lihtne on mõõta andmeid või kas meetrikate rakendamine on automatiseeritud.
- *Võrreldavus* – erinevate elementide või protsesside tulemusi on võimalik võrrelda.
- *Standardsus* – meetrika väärtused peaksid olema kaardistatud mõõtkavana.

Objektiivsus, täpsus ja valiidsus on kolm peamist nõuet, mida meetrika peab täitma.

3.2. Eesmärk-Küsimus-Meetrika

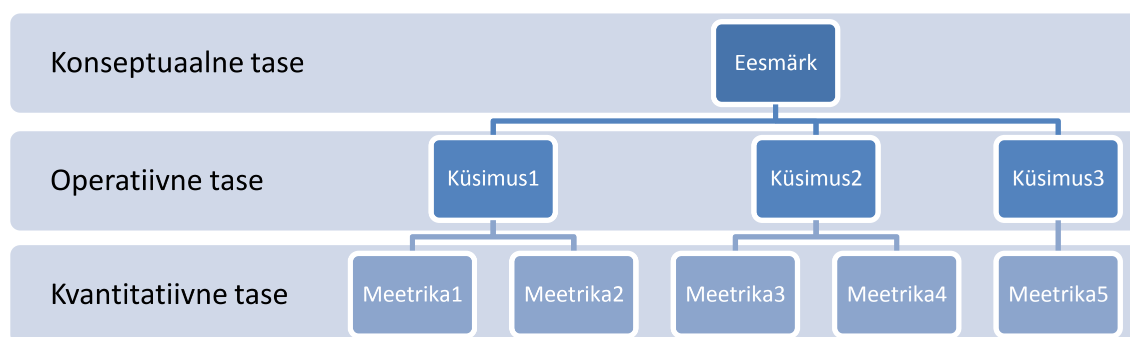
Kirjanduses leidub palju erinevaid meetrikaid, kuid need ei anna juhiseid, kuidas meetrikaid rakendada ja millistes tingimustes. Meetrikate arendamisel ja kasutamisel peaksid olema kindlad eesmärgid. Vajalik on välja arendada mõõtmise raamistik, mis aitab luua tegevuskava, kuidas mõõtmisi läbi viia.

Mõõtmise raamistikud nõuavad suuri hulka andmete kogumist ja analüüsimist. Kahjuks paljud raamistikud on tavaliselt liiga laialivalguvad, mille tõttu paljud andmed jäävad kasutamata ja analüüsimata. Teiseks probleemiks on kirjanduses toodud meetrikate suur hulk, kust konkreetse situatsiooni jaoks konkreetset meetrikat üles leida on väga raske, kuna selle kohta puuduvad juhendid.

Antud töö fookuses on modulaarsuse suurendamine tarkvara analüüsifaasis. Modulaarsuse mõõtmise raamistiku väljatöötamiseks tuleb esmalt sõnastada eesmärgid ja nendest eesmärkidest lähtuvalt arendada välja meetrikad. Selleks kasutan Basili, Caliera ja Rombach-i poolt aastal 1994 väljatöötatud süstemaatilist lähenemisviisi: Eesmärk-Küsimus-Meetrika (ingl. k. *Goal Question Metric (GQM)*) [26]. Antud lähenemisviis aitab tuvastada sobivaid meetrikaid ja kirjeldab kogutud andmete põhjused, kasutades

ülevalt-alla lähenemist: eesmärgi põhjal genereeritakse küsimused ja küsimuste põhjal genereeritakse meetrikad (vt joonis 3.1). Selline lähenemisviis aitab arendada eesmärkidele vastavaid meetrikaid.

Analoogseid lähenemisviise on teisigi: näiteks Eesmärk-Atribuut-Mõõdik (ingl. k. *GAM - Goal-Attribute-Measure*) või BSC (ingl. k *Balanced Scorecard*). Mõõtmiste raamistike väljatöötamiseks on GQM neist aga kõige sobivaim ja enam kasutatavam [27]. GQM peaks vähendama mõõtmise arvu, kuna igal meetrikal peab olema põhjus. Samas jääb mitmete autorite ([28] [29]) arvates püsima risk, et GQM käigus genereeritakse rohkem meetrikaid kui on võimalik andmeid koguda ja analüüsida. Selle riski maandamiseks on GQM lähenemisviisi pidevalt täiustatud, et välistada liiga suure hulga meetrikate rakendamist. Näiteks [29] pakub välja prioritiseerimise sammu, et viia meetrikate arv miinimumini.



Joonis 3.1: Eesmärk-Küsimus-Meetrika lähenemine

3.2.1. Eesmärgi defineerimine

Kontseptuaalsel tasemel defineeritakse mõõtmise eesmärgid. Koosneb viiest erinevast dimensioonist: objekt, põhjus, fookus, vaatepunkt, keskkond [26].

Antud töös on nendeks dimensioonideks:

- Objekt: komponent
- Põhjus: hindamine ja parandamine
- Fookus/probleem: modulaarsus
- Vaatepunkt: tarkvara analüütik
- Keskkond/kontekst: tarkvara analüüsi- ja disainifaas

Seetõttu võib mõõtmise eesmärgi sõnastada järgnevalt: *hinnata ja parandada komponentide modulaarsust tarkvara analüüsi- ja disainifaasis analüütiku vaatepunktist.*

3.2.2. Küsimuste sõnastamine

Operatiivsel tasemel sõnastatakse küsimused, et iseloomustada defineeritud eesmärgi saavutamist [26]. Vastates küsimustele, saab öelda, kas eesmärk on täidetud. Enne küsimuste sõnastamist tuleb aru saada, millised näitajad ja disaini printsiibid mõjutavad modulaarsust kõige rohkem. Selleks võib viia läbi intervjuusid ekspertidega, nende vastused sõnastada kui hüpoteesid ning hüpoteeside põhjal genereerida küsimused [27]. Oluline on leida ka õige abstraktsioonitase.

Vastavalt antud töö teoreetilisele osale ja enda kogemusele sõnastan küsimused järgnevalt:

Küsimus	**Alamküsimused**
Kui seotud on komponent teiste komponentidega?	• Kui suur on seoste hulk? Mitmest komponendist antud komponent sõltub ja mitu komponenti sõltub antud komponendist? • Kas ühe muudatuse tegemiseks on vaja muuta teisi komponente? • Kui lihtne on uut komponenti süsteemi lisada? • Kui lihtne on komponenti ära kustutada? • Kui komponent ei tööta, mitut komponenti see mõjutab? • Kui komponent suhtleb teiste komponentidega, kui palju andmeid nad vahetavad? • Kas samu andmeid kasutab mõni teine komponent?
Kui sarnased on komponendid omavahel?	• Kas komponent sarnaneb mõnele teisele komponendile? Kui sarnased need on? • Kas ühe komponendi muutmisel on vaja muuta ka komponenti, mis pole antud komponendiga otseselt seotud? • Kas komponenti on võimalik jaotada veel edasi väiksemateks tükkideks, mis on omavahel sõltumatud või vähe seotud?
Kui suur on komponent?	• Kui suur on komponenti kirjeldav dokumentatsioon?

Kui keeruline on komponent?	• Kas komponenti on lihtne muuta või täiendada? • Kui keerulised on seosed? • Kui automatiseeritud on komponent?
Kui arusaadav on komponent?	• Kas komponendi muutmiseks on vaja aru saada teistest komponentidest? • Kas on ilmne, kuidas on komponendid omavahel seotud või esineb varjatud seoseid?
Kas komponenti on võimalik taaskasutada?	• Kas komponendi kasutamiseks on vaja eelnevalt käivitada teisi komponente? Kas komponent sõltub mõne teise komponendi sisendist? • Kas mõni teine komponent sõltub antud komponendi sisendist?

Tabel 3.1: Eesmärk-Küsimus-Meetrika: küsimuste sõnastamine

3.2.3. Meetrikate valimine

Kvantitatiivsel tasemel vastatakse küsimustele, kasutades erinevaid meetrikaid [26]. Ühele küsimusele võib vastata mitu meetrikat ja ühte meetrikat võib kasutada mitmele küsimusele vastamiseks. Ei ole soovitatav defineerida vaid ühte meetrikat, kuna see ei suuda katta kõike võimalikke aspekte. On vaja erinevaid meetrikaid, igaüks keskendub erinevatele küsimustele [27]. Samas peaks meetrikate arv jääma nii väikseks kui võimalik. Seda on võimalik saavutada prioritiseerimise tulemusena [29].

Meetrikate valik sõltub ka andmetest, mille peal meetrikaid rakendatakse. Seetõttu tuleb antud tasemel määratleda meetrikad abstraktsemalt, kuid eeldusel, et neid on võimalik rakendada analüüsitulemite peal.

Modulaarsuse mõõtmiseks tuleks mõõta komponendi sidestust, kohesiooni, arusaadavust, suurust ja kompleksust. Tabelis 3.2 on toodud meetrikad, mida saab analüüsifaasis rakendada küsimustele vastamiseks. Peatükis 3.3 antakse valitud meetrikate täpsem kirjeldus.

Küsimus	Meetrika kategooria	Meetrikad
Kui seotud on komponent teiste komponentidega?	Sidestus	fan-in, fan-out, seoste indeks, komponendi ebastabiilsus
Kui sarnased on komponendid omavahel?	Kohesioon	Jaccardi indeks
Kui suur on komponent?	Suurus	dokumentatsiooni suuruse meetrikad
Kui keeruline on komponent?	Komplekssus	IFC, fan-in, fan-out, seoste indeks
Kui arusaadav on komponent?	Arusaadavus, sidestus	dokumentatsiooni suuruse meetrikad, seoste indeks
Kas komponenti on võimalik taaskasutada?	Sidestus, suurus	fan-in, fan-out, seoste indeks, dokumentatsiooni suuruse meetrikad

Tabel 3.2: Eesmärk-Küsimus-Meetrika: meetrikate valimine

3.3. Meetrikate kirjeldus

Antud peatükis defineeritakse meetrikad, mis aitavad vastata alampeatükis 3.2.2 püstitatud küsimustele. Kuna kirjanduses toodud meetrikaid on liiga palju, kirjeldatakse siin vaid need meetrikad, mida on võimalik kohandada analüüsi artefaktide mõõtmiseks. Lähtutud on sellest, et meetrikaid peab olema nii palju, et need võimaldaks vastata peatüki 3.2.2 küsimustele, kuid nii vähe kui võimalik, et maksimeerida nendest saadav kasu. Lisaks peab andmete kogumine olema ökonoome ja meetrikad peavad vastama meetrikate nõuetele (vt alampeatükk 3.1). Sellise sõelumise tulemusena valiti välja järgnevad meetrikad:

3.3.1. Fan-in ja Fan-out

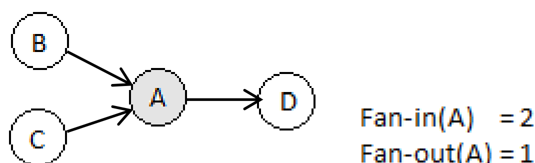
Hanry ja Kafura poolt [30] pakutakse komponendi sidestuse ja komplekssuse mõõtmiseks välja järgnevad meetrikad:

- Fan-in – komponentide arv, mis juhivad antud komponenti või andmestruktuuride arv, kust antud komponent saab andmeid. Kõrge fan-in väärtusega komponendid

on tavaliselt alamkomponendid, mille funktsionaalsust vajavad mitmed suuremad komponendid [31]. Ühtlasi on kõrge fan-in indikaatoriks, et komponent on taaskasutatav, kuid komponendi muudatus võib mõjutada paljusid teisi komponente. Seega kõrge fan-in väärtus võib olla probleemiks juhul kui komponentide vahelise sidestuse tase on kõrge (vt 2.2.1).

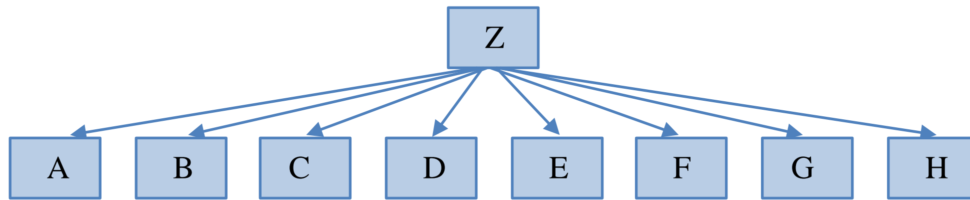
- Fan-out – komponentide arv, mida vaadeldav komponent juhib või andmestruktuuride arv, mida uuendatakse antud komponendi poolt. Kõrge fan-out väärtusega komponendid võivad olla tuumikkomponendid, kuid üldiselt näitab kõrge fan-out, et komponent on halvasti disainitud.

Fan-in ja fan-out väärtuste arvutamist saab visualiseerida graafi näitel. Kui komponente kujutada graafi tippudena ning komponentide vahelisi seoseid graafi servadena, siis komponendi fan-in on komponenti sisenevate servade arv ning fan-out komponendist väljuvate servade arv. Näiteks joonisel 3.2 kujutatud graafil on komponendi A fan-in=2 ja fan-out=1.

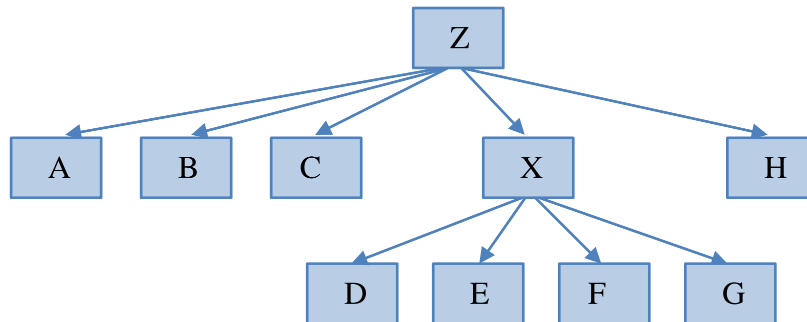


Joonis 3.2: Fan-in ja Fan-out arvutamine

Antud meetrikad võimaldavad eristada, millistest objektidest komponent sõltub ning milliseid objekte komponent juhib. Kõrge fan-in ja fan-out väärtustega komponentide restruktureerimine võib aidata suurendada modulaarsust [31]. Oluline on aga siiski mõista seose tüüpe ning komponentide hierarhiat. Analüütik peaks püüdma disainida loogilise komponentide struktuuri, kus komponentide hierarhia ülemisel tasemel asuvad kõrge fan-out väärtusega komponendid ning hierarhia alumisel tasemel asuvad kõrge fan-in väärtusega komponendid [32]. Joonisel 3.3 kujutatud komponendi Z fan-out väärtus on liiga kõrge. Probleem lahendatakse komponendi X lisamise ja komponentide restruktureerimisega (joonis 3.4). Seega fan-in ja fan-out kõrged väärtused võivad olla indikaatoriks, et komponenti on vaja ümber struktureerida. On selge, et analüüsifaasis selliste muudatuste tegemine on oluliselt odavam kui pärast koodi realiseerimist.



Joonis 3.3: Komponendi Z fan-out väärtus on liiga kõrge



Joonis 3.4: Komponendi Z fan-out väärtuse vähendamine komponendi X lisamise ja komponentide restruktureerimise kaudu

Fan-in ja fan-out meetrikad on piisavalt universaalsed - neid on võimalik rakendada erinevate tarkvara tulemite peal. Samuti võimaldavad need võrrelda erinevate komponentide sidestust, kompleksust ning taaskasutatavust.

3.3.2. Seoste indeks

Süsteemi alamosade vaheline seoste paljusus mõjutab modulaarsust negatiivselt, ehkki seoste mõju sõltub ka sidestuse tasemest. Samas tuleks uurida erinevaid seose tüüpe, sest süsteemi alamosade vahelise suhtluse põhjused on erinevad. Ainult komponentide vaheliste seoste hindamine ei pruugi anda adekvaatset tulemust, sest komponent võib olla seotud ka süsteemi teiste elementidega, mis mõjutavad komponendi modulaarsust erinevalt. Lisaks komponentide omavahelisele suhtlusele võib komponent olla seotud erinevate andmestruktuuridega, väliste liidestega jne. Seega tuleks arvestada kõikide seoste tüüpidega, mida analüüsifaasis on võimalik identifitseerida. Komponentide vahel ei pruugi olla otseseid seoseid, kuid nad võivad kasutada samu andmeid või muutujaid. Näiteks kui komponent A ja B pole omavahel otseselt seotud, kuid A muudab ja B kasutab sama andmeühikut X, siis tegelikult on nad tihedalt seotud [14]. Seetõttu on vaja

meetrikat, mis erineva tüübi ja tasemega seoseid arvestab ning võimaldab erinevaid komponente modulaarsuse seisukohalt paremini võrrelda. Selleks defineerin seoste indeksi kui erinevat tüüpi seoste summa:

$$\text{seoste indeks} = \sum_{i=0}^{n} k_i \text{fanIn}_i + l_i \text{fanOut}_i$$

kus n viitab erinevat tüüpi seoste arvule ning k_i ja l_i on koefitsiendid. Konkreetne koefitsiendi väärtus sõltub sellest, millist objekti mõõdetakse ning kuidas seose tüübi mõju hinnatakse. Näiteks kui mõõdetakse komponendi ja andmestruktuuride vahelisi seoseid ja komponent saab andmeid uuendada, võib koefitsient olla suurem kui ainult andmete lugemisel, kuna andmete uuendamine mõjutab komponenti rohkem. Kui erinevat tüüpi seoseid ei suudeta prioritiseerida, võib koefitsiendid väärtustada ühega.

Seoste indeks võimaldab erinevaid seoseid vaadelda kui tervikut. Seoste mõõtmise tulemus näitab, kui palju on analüütik ette näinud seoseid erinevate komponentide ja teiste süsteemi alamosade (nt andmestruktuuride) vahel.

3.3.3. Komponendi ebastabiilsus

Komponendi ebastabiilsus (ingl.k. *instability of a component*) on üks sidestuse mõõtmise meetrikatest, mis defineeritakse järgneva valemiga [33]:

$$ebastabiilsus = \frac{FanOut}{FanIn + FanOut}$$

Ka ebastabiilsust võib mõõta erinevat tüüpi seoste pealt. Stabiilsus ennustab komponendi muudatuse tõenäosust. Mida raskem on komponenti muudatust teha, seda suurem on tõenäosus, et muudatust ei tehta, mistõttu komponent on stabiilsem. Meetrika väärtused on vahemikus [0,1]: 0 tähendab maksimaalset stabiilsust, 1 maksimaalset ebastabiilsust [33]. Madal ebastabiilsus näitab, et komponent sõltub vähestest komponentidest, samas kui paljud teised komponendid sõltuvad antud komponendist. Sellise komponendi muutmine on keeruline ja kulukas, kuna komponendi muudatused mõjutavad mitmeid teisi komponente.

Kahe komponendi sõltuvused on kahjulikud kui stabiilsem komponent sõltub vähem stabiilsest komponendist [33]. Nii propageeruvad muudatused vähem stabiilsest

komponendist rohkem stabiilsesse komponenti, mõjutades ka neid komponente, mis on stabiilsemast komponendist sõltuvuses. Selliste laiaulatuslike kõrvalmõjude vältimiseks peaks komponent sõltuma ainult temast stabiilsematest komponentidest [33].

3.3.4. Jaccardi indeks

Komponentide sarnasuse ja kohesiooni mõõtmiseks on kõige tuntum ja laialdasemalt kasutatud meetrika Chidamber and Kemerer-i poolt pakutud LCOM (ingl. k. *Lack of Cohesion of Methods*), mis mõõdab ühe klassi meetodite sarnasusi atribuutide ja muutujate kaudu [34]. Enamik teisi kohesiooni meetrikaid baseeruvad LCOM-il [35]. Paraku analüüsifaasis ei saa LCOM-i rakendada, kuna see on liialt koodi spetsiifiline. Analüüsifaasis on komponentide abstraktsioonitase oluliselt kõrgem.

Kui koodi tasemel on oluline, et samas klassis asuksid sarnased meetodid, mis kasutavad samu atribuute ja muutujaid, siis analüüsifaasis peaks analüütik komponendi koostama nii, et arendajal oleks lihtsam luua kõrge kohesiooniga klasse. Ühes komponendis peaksid asuma omavahel tihedalt seotud elemendid. Kui samas komponendis saab elemente jaotada vähemalt kahte alamhulka nii, et kumbki alamhulk ei kasuta samu andmeid, tuleks nendest alamhulkadest luua eraldi komponendid.

Analüüsifaasis on oluline ka aru saada, millised komponendid on semantiliselt seotud. Semantilisest seosest aitab paremini aru saada järgnev näide: komponendid A ja B on nii funktsionaalsuse kui kasutajaliidese poolest peaaegu identsed, kuid neid on analüüsitud erinevate analüütikute poolt erinevate komponentidena ja seetõttu ka realiseeritud eraldi. Seega nii komponendile A kui B vastab eraldi kood, andmestruktuurid ning nad pole omavahel otseselt seotud. Samas on nad sisu poolest äärmiselt sarnased, mistõttu tekib suur tõenäosus, et ühe komponendi muutmisel tuleb ka teise komponenti teha sama muudatus. Analüütikutele on suureks väljakutseks, kuidas tuvastada sarnaseid objekte ja neid komponentidesse jaotada nii, et sidestus ja keerukus ei suureneks.

Sarnaste objektide komponentidesse jaotamisel mängib kõige suuremat rolli analüütiku kogemus, kirjanduses pakutud meetrikatest ei pruugi olla kasu. Objektiivsetest mõõdikutest annab aga kõige rohkem lootust Jaccardi indeks [36], mis mõõdab kahe komponendi A ja B sarnasust nende ühisosa ja ühendi kaudu:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Jaccardi indeks on piisavalt universaalne: mõõta võib nii komponentide kui nende mikrostruktuuride (nt atribuudid) sarnasust. Võib täheldada, et Jaccardi indeks võrdleb kahte komponenti. Kõikide komponentide omavahelise sarnasuse mõõtmiseks võib koostada komponentide sarnasuste ruutmaatriksi.

3.3.5. Informatsioonivoo kompleksus (IFC)

Komplekssuse meetrikaid leidub kirjanduses kõige rohkem. Osalt seetõttu, et ka sidestuse, kohesiooni või suuruse meetrikad sobivad komplekssuse mõõtmiseks. Näiteks mida suurem komponent, seda komplekssem. Samas võib komponent olla kompleksne ka keerulise äriloogika tõttu, kus komponendis läbitakse mitmeid mahukaid kontrolle ja arvutusi. Enamus kirjanduses toodud komplekssuse meetrikad on aga antud töö jaoks sobimatud, kuna mõõdetakse lähtekoodi mikrostruktuure (koodi struktuure, pärinevust) [37]. Analüüsifaasis on andmete mõõtmine mikrostruktuuride tasemel liiga kulukas.

Kafura ja Henry [30] pakuvad välja informatsioonivoo komplekssuse meetrika (ingl.k. IFC - *information flow complexity*), mis defineeritakse järgnevalt:

$$IFC = (FanIn * FanOut)^2$$

Fan-in ja fan-out korrutis annab kõikide võimalike sisendite ja väljundite kombinatsioonide arvu [25]. Meetrikat saab automaatselt rakendada fan-in ja fan-out tulemuste põhjal.

3.3.6. Suuruse meetrikad

Kõige fundamentaalsem ja lihtsam modulaarsuse mõõtmise meetrika põhineb erinevate tarkvara tulemite suuruse mõõtmises. Koodi tasemel mõõdetakse klasside või meetodite koodiridade arvu, mille põhjal saab öelda, kas klass/meetod on liiga suur või väike [38]. Tarkvara analüüsis saab komponendi suurust mõõta analüüsi artefaktide pealt, näiteks dokumentatsiooni sõnade arv või komponendiga seotud andmestruktuuride arv. Konkreetsemad meetrikad sõltuvad projektis koostatavatest analüüsi tulemitest.

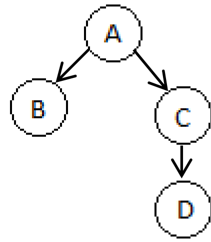
3.4. Modulaarsuse mõõtmise tehnikad

Selgub, et enamike välja pakutud meetrikate väärtuste leidmiseks on vajalik mõõta komponentide vahelisi seoseid. Seega tuleb otsustada, kuidas analüüsitulemite pealt neid seoseid mõõta. Võib tekkida probleem, et neid meetrikaid ei saa praktikas rakendada, kuna mõõtmise aluseks olevate andmete leidmine võib osutuda liiga kulukas. Koodi tasemel on arendatud mitmesuguseid vahendeid, et automatiseerida meetrikate rakendamist. Analüüsitasemel on automatiseerida keerulisem, kuigi mitte võimatu.

Analüüsitasemel komponentide vaheliste seoste mõõtmiseks ei piisa tavapärasest dokumentatsioonist, protsessiskeemidest ega andmemudelist. Dokumentatsioonist seoste leidmine võtab liiga palju aega ja muutub seetõttu kulukaks. Protsessiskeemid on liiga üldised ning ei pruugi kajastada kõiki seoseid erinevatel tasanditel. Andmemudel näitab seoseid andmestruktuuride vahel, kuid mitte seda, millised süsteemi komponendid on vastavate andmestruktuuridega seotud.

Kõige lihtsam on komponentide vahelisi seoseid mõista kui andmed on esitatud graafiliselt. Sõltuvuste esitamiseks võib joonistada komponentide vahelise sõltuvuste graafi. Suurte süsteemide korral selline lähenemisviis pole aga põhjendatud, sest graafide arusaadavus väheneb ja seoste lugemine muutuks liialt ajakulukaks.

On olemas paremaid tehnikaid. Kolm kõige enam levinud meetodit modulaarsuse mõõtmiseks on aksiomaatiline disaini meetod [39], funktsionaalsete struktuuride meetod [40] ja disaini struktuuri maatriks [41]. Neist kõige rohkem annab lootust disaini struktuuri maatriks (edaspidi DSM), mis visualiseerib maatriksis asuvate elementide omavahelised sõltuvused [21]. Sisuliselt on tegemist maatriksiga, mille read ja veerud kajastavad süsteemi erinevate elementide vahelisi seoseid. DSM on piisavalt universaalne - võimalik on disaini seoseid võrrelda mitmesugustel tasanditel, alustades alamsüsteemidest, lõpetades mikrostruktuuridega. Modulaarsus ei sõltu mitte ainult nende elementide vahelisest sõltuvusest, vaid ka nende jaotuse mustrist [21]. DSM illustreerimiseks sobib joonis 3.5.



	A	B	C	D
A	0	1	1	0
B	0	0	0	0
C	0	0	0	1
D	0	0	0	0

Joonis 3.5: Komponentide vaheliste seoste graaf ja DSM kujutis

Joonisel 3.5 on kujutatud nelja komponendi vahelisi seoseid: vasakul pool on toodud komponentide seoste graaf ning paremal DSM. Võib eristada otseseid seoseid (nt komponent A sõltub otseselt komponentidest B ja C) ning kaudseid seoseid (nt komponent A sõltub kaudselt komponendist D, st muudatus komponendis D võib mõjutada komponenti A).

Joonisel 3.5 toodud DSM mõõdab otsest seotust. Kaudse seotuse mõõtmiseks tuleb maatriksit mitmekordistada (ingl. k *matrix multiplication*).

	A	B	C	D
A	0	1	1	0
B	0	0	0	0
C	0	0	0	1
D	0	0	0	0

M^1

	A	B	C	D
A	0	0	0	1
B	0	0	0	0
C	0	0	0	0
D	0	0	0	0

M^2

	A	B	C	D
A	0	1	1	1
B	0	0	0	0
C	0	0	0	1
D	0	0	0	0

$V=\sum M^n$; $n = [1, 2]$

Joonis 3.6: Otsesed ja kaudsed seosed

Joonisel 3.6 toodud maatriks M^1 näitab otseseid seoseid, maatriks M^2 seoseid kaugusega 2 ning V on maatriksite summa, mis võimaldab näha kõiki otseseid ja kaudseid seoseid kokku.

DSM võib olla eraldi analüüsitulem. DSM on piisavalt universaalne ja seda saab kohandada vastavalt vajadusele: maatriksisse võib kanda nii komponendid, komponentide alamtegevused- või elemendid, andmestruktuurid jne. DSM-i võib võtta kasutusele ka juba käimasolevates projektides. Erinevad alamsüsteemide DSM-id võib kanda erinevatele maatriksitele ning ainult alamsüsteeme siduvaid komponente näidata mõlemal DSM-il.

DSM maatriksit on võimalik organiseerida nii, et saab eristada aktsepteeritavaid ja mitteaktsepteeritavaid sõltuvusi. On olemas mitmeid algoritme, mis aitavad maatriksit organiseerida ning märgistada mustrid ja probleemsed seosed või sarnased komponendid [42]. Sobiva DSM tabeli välja töötamine sõltub projektist ja kasutatavast tehnoloogiast. Väiksemate projektide korral ei pruugi DSM rakendamine olla otstarbekas.

4 Juhtumiuuring

Väljapakutud meetrikate ettepanekutest pole kasu, kui neid pole praktikas kasutatud. Järgnevalt rakendatakse peatükis 3 välja pakutud arvutuslikku mudelit reaalse süsteemi peal. Mõõdetakse analüüsi artefaktide sidestust, suurust ja komplekssust. Mõõtmise tulemused võimaldavad analüüsi tulemeid modulaarsuse seisukohalt võrrelda ja hinnata ning identifitseerida kõrge veariskiga komponendid, mida tuleks modulaarsuse ja parema kvaliteedi saavutamiseks ümber struktureerida. Mõõtmise tulemuste peal viiakse läbi lihtne regressioonanalüüs, leidmaks, kas suur muudatuste arv võib tuleneda sellest, et analüüsi artefaktides on ignoreeritud modulaarsuse printsiipe.

4.1. Süsteem EMPIS

Juhtumiuuring viidi läbi Eesti Töötukassa infosüsteemi „EMPIS“ peal, mis on arendatud AS Webmedia poolt. Käesoleva töö autor on osalenud projektis analüütikuna süsteemi loomisest saadik. EMPIS on veebipõhine infosüsteem, mille kasutajad on töötukassa ametnikud. Süsteem võimaldab registreerida töötuid, määrata neile toetusi, pakkuda erinevaid teenuseid, vahendada tööpakkumisi, sõlmida asutustega lepinguid, menetleda hankeid jne. Samuti vahendatakse andmeid riigi teiste infosüsteemide ja töötukassa siserakendustega. EMPIS lihtsustab asjaajamist nii töötutel kui töötukassa ametnikel ning muudab ametnike tööd lihtsamaks ja efektiivsemaks.

Tegemist on objekt-orienteeritud süsteemiga, mis on realiseeritud Java platvormil ja Oracle andmebaasil. Seisuga 01.01.2012 on projekt kokku kestnud ligi 3 aastat, arendustöödele on kulunud umbes 40 000 töötundi ja süsteem sisaldab 270 882 rida puhast Java koodi. Arendus jätkub samas tempos. Eesti mõistes on tegemist suure ja keeruka projektiga.

Arendus on toimunud agiilse arendusmetoodika järgi ja rakendatud ka SCRUM raamistikku. Süsteem on kasutusele võetud etappidena, et vana süsteem järk-järgult asendada uuega. Esmalt realiseeriti kõige kriitilisemad ja enamkasutatavad äriprotsessid vaid 8 kuuga, misjärel süsteem võeti esmakordselt kasutusele. Järgnevad etapid realiseeriti lähtuvalt äriprotsesside prioriteetsusest. Selline etapiviisiline arendus on paindlikum, samas suureneb ka muudatuste ja täienduste hulk. Kui esimesel aastal kulus

enamik ressurssidest uue funktsionaalsuse arendamiseks, siis 3. aastal kulub märkimisväärselt palju aega täiendustele, muudatustele ja vigade parandamisele. Selline jaotus on tingitud ühelt poolt sellest, et on toimunud mitmeid mahukaid seadusemuudatusi, mis on põhjustanud nõuete muudatuse. Teiselt poolt on süsteem suur, kompleksne ning automaatika tõttu on komponentide vahel palju sõltuvusi. See tingibki olukorra, kus ühe muudatuse tegemisel tuleb muudatusi teha mitmesse komponenti ning keeruliste sõltuvuste tõttu testimise ja stabiliseerimise faas pikeneb.

4.2. Andmete kogumine

Peatükis 3.3 valitud meetrikate rakendamiseks tuleb koguda andmeid, et teostada mõõtmisi. Oluline on planeerida, milliseid andmeid koguda ja kuidas seda kõige efektiivsemalt teha. Andmete kogumine on väärtusetu, kui nende pealt mõõtmisi ei teostata. Mõõtmisi saab teostada aga ainult andmete pealt, mis on täielikud ja uuendatud.

Süsteemis EMPIS on analüüsi tulemuseks järgnevad artefaktid:

1) Tehnilised ülesanded – detailne komponendi kirjeldus, mis on sisendiks programmeerijale. Ühest tehnilisest ülesandest võib mõelda kui ühest komponendist. Tehnilised ülesanded on koostatud ühtse malli järgi ja sisaldavad järgnevaid alampeatükke: muudatuste ajalugu, eesmärk, prototüübi link, privileegid, pealkiri, atribuudid, kontrollid, tegevused, salvestusreeglid. Ühtset malli ei järgi mõningad algoritmi kirjeldused ja üldised dokumendid. Tehnilisi ülesandeid uuendatakse iga muudatusega, millega kaasneb ka uus versioon. Tehnilise ülesande näide on toodud lisas 1.
2) Liideste kirjeldused - liideste kirjeldused nii töötukassa siserakendustega kui väliste süsteemidega. Sisaldab ka veebiteenuste kirjeldusi. Dokumendi malli poolest sarnanevad tehnilisele ülesandele.
3) Prototüüp – koosneb staatilistest HTML lehtedest, mis näitavad, kuidas rakendus ja selle komponendid hakkavad välja nägema ja kuidas protsesse läbitakse. Ühest prototüübi ekraanikuvast võib mõelda kui ühest komponendist. Ühele kuvale võib vastata mitu tehnilist ülesannet. Kui ühele tehnilisele ülesandele vastab mitu ekraanikuva, on tegemist komponendi taaskasutamisega, st need ekraanikuvad

peavad olema suurel määral sarnased. Lisaks on prototüübi kuvade erinevad variandid, mille pealt pole kunagi tehnilist ülesannet kirjutatud.

4) Andmemudel – enamasti üks andmetabel koos paari alamtabeliga vastab komponendi tasemele.
5) UML skeemid – UML skeemidest on kõige enam koostatud protsessiskeeme. Paraku ei kata need kogu süsteemi funktsionaalsust, mistõttu pole andmed täielikud. Lisaks võivad skeemid olla aegunud.
6) Koosoleku protokollid – sisuliselt on tegemist nõuete loeteluga.
7) Kasutusjuhendid – sisaldab protsessi läbimiseks vajalike tegevuste detailset kirjeldust ja ekraanipilte.
8) Testjuhtumid – testjuhtumite kirjelduste põhjal realiseeritakse ühiktestid. EMPIS-es koostavad testjuhtumeid nii analüütikud kui testijad. Kuigi testjuhtumeid saab realiseerida alles pärast seda, kui sellega seotud komponendid on realiseeritud ja stabiliseerimise faas on möödas, võib neid kirjeldada varem. Analüütik saab testjuhtumeid kirjeldades protsessid veel täpsemalt läbi mõelda ja vajadusel teha muudatusi tehnilistesse ülesannetesse.

Lisaks on olemas keskkond tööülesannete halduseks, kuhu on raporteeritud kõik nõuded, arendustööd, täiendused, muudatused ja vead. Selle kaudu on võimalik teha mitmesuguseid väljavõtteid erinevate versioonide mahukuse, tööülesannete tüüpide kohta jne.

Andmete koosseisu analüüsides selgub, et meetrikate rakendamiseks sobivad kõige paremini tehnilised ülesanded. Esiteks on tehnilised ülesanded sisendiks programmeerijatele ja need vastavad kõige paremini komponendi tasemele. Teiseks on need analüüsi tulemitest kõige detailsemad, uuendatud ja täielikud. Kolmandaks, nõuete jaotus tehnilistesse ülesannetesse annab modulaarsuse seisukohalt suure efekti. Seega sobivad tehnilised ülesanded kõige paremini modulaarsuse mõõtmiseks analüüsifaasis ja nende pealt on võimalik koostada disaini struktuuri maatriks.

4.3. Disaini struktuuri maatriksi automaatne koostamine

4.3.1. Eesmärk

Disaini struktuuri maatriksi automaatseks koostamiseks leidub erinevaid programme (nt Lattix LDM, IntelliJ IDEA, Eclipse plugin Structure101), kuid kõikidel juhtudel võetakse sisendiks lähtekood. Analüüsitulemite põhjal DSM koostamine on uudne lähenemine, mille kohta Internetis materjale ei leitud.

Antud eksperimendi käigus uuriti EMPIS analüüsitulemeid ja tehti järeldus, et kõige paremini on komponentide vahelised seosed kirjeldatud tehnilistes ülesannetes. Seetõttu otsustati sealt sõltuvused välja võtta staatilise analüüsi meetodil ja kanda DSM-i. Selle tegevuse käigus aga selgus, et tegemist on liiga ajamahuka tööga ning analüütikud ei saa endale võtta täiendavaid dokumenteerimiskohustusi. Lisaks tekib oht, et DSM-i ei uuendata piisavalt tihti. Seega otsustati DSM koostada automaatselt tehniliste ülesannete pealt. Eesmärk on luua kaks disaini struktuuri maatriksi:

- Komponentide vaheliste seoste ruutmaatriks
- Komponentide ja andmetabelite seoste maatriks

4.3.2. DSM automaatne koostamine

Programmi lähtekood on koostatud Java programmina ja koosneb järgnevatest klassidest:

- Klass, mis võtab dokumentide viimased versioonid dokumendihaldussüsteemist ja teeb nendest eraldi kataloogi lokaalsesse arvutisse.
- Klass, mis parsib iga dokumendi lahti (jaotab peatükkideks, võtab välja võtmesõnad) ja kannab tulemused vahetabelisse.
- Klass, mis paneb vahetabeli tulemuste põhjal kokku CSV failid, kus kajastuvad erinevad seosed.

Dokumentide lahti parsimise seadistuste kohta on eraldi fail, mida kasutajal on võimalik täiendada uute reeglitega.

EMPIS dokumentide parimisel kasutatakse järgnevaid reegleid:

- Komponentide vaheliste seoste ruutmaatriksi koostamiseks parsitakse tehniliste ülesannete koodid kujul [A-Z]{2,}_?[0-9]+\.[0-9]+(\.[0-9]+)* ja viited kirjutatakse väljund CSV-sse täpsustamata viitena "v". Seega esialgu tuuakse välja, et komponentide vahel on seos, täpsustamata, milline see seos täpselt on (kas sisalduvus, viide). Edaspidi on plaanis dokumentide keelelist konteksti rohkem analüüsida, et tuvastada täpsem seose tüüp.
- Komponentide ja andmetabelite seoste parsimisel eristatakse erinevat tüüpi seoseid:
    - u – update ehk andmetabeli uuendamine, st komponendis toimub kirje lisamine/muutmine või kustutamine.
    - r - read ehk lugemine, st komponent ainult loeb andmetabelist, aga ei uuenda andmeid.
    - v - täpsustamata viide, st regulaaravaldisega ei suudeta tuvastada seose tüüpi, kuid suure tõenäosusega on tegemist seosega r.

Täpsemad parsimise reeglid on järgnevad (tehnilise ülesande näide toodud lisas 1):

1) Dokumendist otsitakse võtmesõnad kujul [A-Z_]+
2) Keeleline tabelinime tuvastus: "tabel<järelliide> VÕTMESÕNA" või "VÕTMESÕNA kirje<järelliide>".
    a. järelliide IN ('s', 'st') - lugemine - väljund-csv's "r"
    b. järelliide = 'sse" - kirjutamine - väljund-csv's "u“
3) Kui seadistus ja situatsioon on (writeRuleRequired=false OR (<salvestusreeglite peatükk on tühi> AND writeRuleLinguisticFallback=true)), siis rakendatakse keelelist tabelinime tuvastust kõigile dokumendis leiduvatele võtmesõnadele.
4) Kui seadistus ja situatsioon on (writeRuleRequired=true AND <salvestusreeglite peatükk ei ole tühi>), siis keeleline tabelinime tuvastus tehakse ainult salvestusreeglite peatükis (vt. järgmine punkt).
5) Salvestusreeglite peatükist parsitakse tabelinimesid kahel viisil:
    a. Lõikudest, mille stiil on "Heading 3..." (tabeli nimeline alampeatükk), leitakse võtmesõnad ja märgitakse, et tegemist on tabelisse kirjutamisega.

   b. Lõikudest, mille stiil ei ole "Heading 3..." parsitakse võtmesõnad ja märgitakse tabelina koos võimaliku lugemis-kirjutamislipuga keelelise tabelinimetuvastusega. Kui lugemis-kirjutamisjärelliidet pole, tuvastatakse tabel täpsustamata viitena - väljund-csv's "v".
6) Lisaks märgitakse kõik dokumendist leitud võtmesõnad, mis leiduvad seadistatud ühenduse abil baasist saadud tabelinimede loetelus, täpsustamata tabelinime viiteks - väljund-csv's "v".

Kuna samas tarkvaraarendusfirmas järgitakse sarnast dokumentatsiooni malli, on võimalik automaatset DSM koostamise vahendit rakendada ka teiste projektide peal.

4.3.3. DSM väljund

DSM automaatse koostamise programm väljastab kaks CSV faili, mille sisu kopeeritakse või loetakse tabeltöötlusprogrammi, kus on võimalik seoseid paremini visualiseerida:

1) Komponentide vaheliste seoste ruutmaatriks annab ülevaate tehniliste ülesannete vahelistest seostest ehk sisuliselt komponentide vahelistest seostest. Näiteks joonise 4.1 põhjal saab välja lugeda, et komponent AV_4.1 sõltub komponentidest AV_4.1.2 ja AV_4.2 (rida AV_4.1) ning komponendid AV_4.1.1, AV_4.2, AV_4.5 ja AV_6.1 sõltuvad komponendist AV_4.1 (veerg AV_4.1).

2) Komponentide ja andmetabelite seoste maatriks annab ülevaate tehniliste ülesannete ja andmetabelite seostest. Näiteks joonise 4.2 põhjal saab välja lugeda, et komponendis AV_4.1 loetakse või uuendatakse tabelite *emp_avaldus, emp_avaldus_puudus, emp_avaldus_toend, emp_avaldus_tt_hoive* kirjeid (rida AV_4.1). Tabelit *emp_avaldus_toend* kasutatakse aga komponentides AV_4.1, AV_4.1.1, AV_4.1.2, AV_6.1, AV_6.1.1 ja AV_7.1 (veerg *emp_avaldus_toend*). Kui tabeli struktuur muutub, tuleb kindlasti üle vaadata komponendid AV_4.1.2 ja AV_7.1, kuna seal toimub kirjete uuendamine.

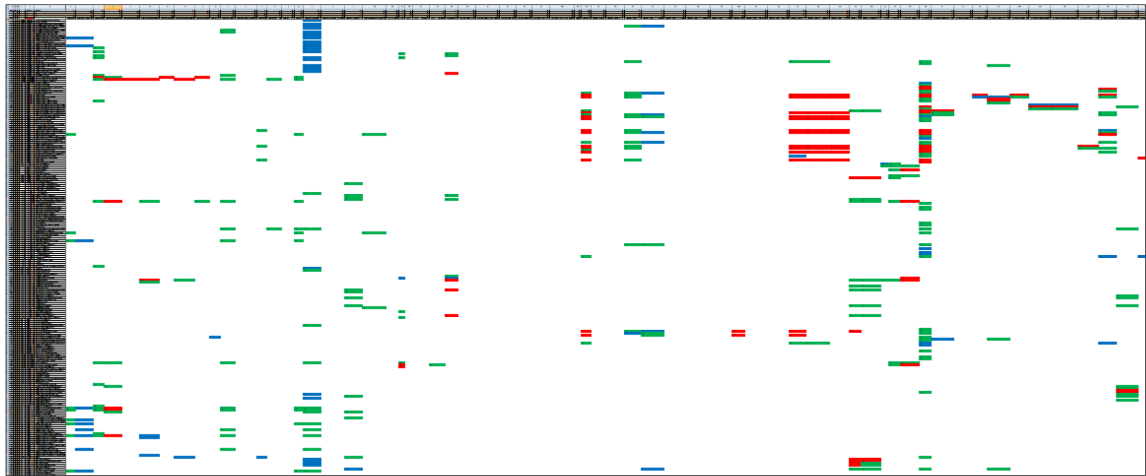
	AV_4.1	AV_4.1.1	AV_4.1.2	AV_4.2	AV_4.3	AV_4.4	AV_4.5	A
AV_4.1			v	v				
AV_4.1.1	v				v		v	
AV_4.1.2						v		
AV_4.2	v							
AV_4.3								
AV_4.4								
AV_4.5	v			v	v			
AV_5.1								
AV_6.1	v	v	v					
AV_6.1.1		v						

Joonis 4.1: Komponentide vaheliste seoste ruutmaatriksi fragment

	EMP_AVALDUS	EMP_AVALDUS_PUUDUS	EMP_AVALDUS_TOEND	EMP_AVALDUS_TT_HOIVE
AV_4.1	u	r	r	u
AV_4.1.1	u	v	r	r
AV_4.1.2	u	u	u	u
AV_4.2	v			r
AV_4.3	r			r
AV_4.4				v
AV_4.5	v	v		
AV_5.1	v			
AV_6.1	u		r	r
AV_6.1.1	u	v	r	r
AV_7.1	u		u	

Joonis 4.2: Komponentide ja andmetabelite seoste maatriksi fragment

DSM koostamise vahendit on võimalik edasi arendada vastavalt vajadusele ja projekti spetsiifikale. Samuti võib maatriksi graafilist väljundit täiendada nii, et visuaalselt oleks paremini eristatavad probleemsed sõltuvused. Näiteks joonisel 4.1 viitab komponent AV_4.1 komponendile AV_4.2 ja komponendis AV_4.2 viidatakse komponendile AV_4.1 – tegemist on tsüklilise seosega, mis tuleks maatriksis esile tõsta, sest see rikub analüüsi eeldusi oodatud ja mõistlikest lahendustest. On olemas mitmeid algoritme, mis aitavad maatriksit organiseerida ning märgistada mustrid ja probleemsed seosed või sarnased komponendid [69]. Nii saab analüütik kiire ülevaate, milliseid komponente võib olla vaja restruktureerida (joonis 4.3). Lisaks peab analüütikul olema võimalus kontrollida ja täiendada disainireegleid.



Joonis 4.3: Fragment EMPIS tehniliste ülesannete-andmemudeli tabelite seostest, kus punasega on toodud probleemsed seosed.

Loodud maatriksid on kasulikud nii probleemsete komponentide ja sõltuvuste tuvastamiseks kui ka muudatuse mõjuulatuse hindamiseks. Iga uue muudatuse või täienduse korral on lihtne kontrollida, milliste komponentidega on muudetav komponent või andmetabel seotud. Nii on ka uutel analüütikutel projekti kergem sisse elada, kuna sõltuvuste mõistmiseks pole tarvis tervet dokumentatsiooni läbi lugeda. Samuti saavad testijad kiirelt hinnata, millised on potentsiaalsed komponendid, mis muudatuse tõttu võivad olla vigased. Kuna seosed on tehtud ilmseks, on neid ka lihtsam kontrollida.

4.4. Meetrikate rakendamine

Enamikke peatükis 3.3 välja pakutud meetrikaid on võimalik rakendada disaini struktuuri maatriksi pealt automaatselt, st lisamõõtmisi pole vaja teha. Vaid suuruse ja kohesiooni meetrikad nõuavad eraldi andmete kogumist.

Meetrikaid rakendati 247 komponendi peal. Järgnevalt kirjeldatakse, kuidas meetrikate väärtused leiti ning esitatakse mõõtmise tulemused.

4.4.1. Fan-in ja Fan-out

Eelnevalt koostatud DSM maatriksid on fan-in ja fan-out mõõtmiseks sobivad, kuna teevad erinevate komponentide ja andmestruktuuride vahelised seosed ilmsiks. DSM

maatriksit on sidestuse mõõtmiseks vajalik täiendada veergudega, mis summeerivad seoste arvu ridade ja veergude lõikes.

1) Komponentide vaheliste seoste ruutmaatriksis on fan-in väärtuseks veeru summa ja fan-out väärtuseks rea summa. Näiteks joonisel 4.4 on komponendi AV_4.1 fan-in väärtuseks 6 ja fan-out 5.
2) Komponentide ja andmetabelite seoste maatriksis on fan-in väärtuseks andmestruktuuride arv, kust komponent saab andmeid ehk seoste tüübiga „r“ ja „v“ summa. Fan-out väärtuseks on andmestruktuuride arv, mida uuendatakse komponendi poolt ehk seoste „u“ summa. Näiteks joonisel 4.5 on komponendi AV_4.1 fan-in väärtuseks 7 ja fan-out väärtuseks 10.

			5	8	11	6	4	3	2
			AV	AV	AV	AV	AV	AV	AV
			AV_1.1	AV_1.2	AV_1.3	AV_4.1	AV_4.1.	AV_4.1.	AV_4.2
5	AV	AV_4.1	v	v	v			v	v
6	AV	AV_4.1.1			v	v			
4	AV	AV_4.1.2		v					
1	AV	AV_4.2				v			
0	AV	AV_4.3							
0	AV	AV_4.4							
5	AV	AV_4.5				v			v
0	AV	AV_5.1							
4	AV	AV_6.1		v		v	v	v	
4	AV	AV_9.2	v		v	v			
8	ISIK	ISIK_6.1				v			

Joonis 4.4: Komponentide vaheliste seoste fan-in ja fan-out arvutamine

Kokku					Kokku	61	14	14
	u				u	17	3	5
		r			r	9	3	8
			v		v	35	8	1
Kok						EMP_AVALDUS	EMP_AVALDUS_PUUDUS	EMP_AVALDUS_TOEND
17	10	2	5	AV	AV_4.1	u	r	r
15	5	4	6	AV	AV_4.1.1	u	v	r
12	6	0	6	AV	AV_4.1.2	u	u	u
4	0	2	2	AV	AV_4.2	v		
3	0	2	1	AV	AV_4.3	r		
1	0	0	1	AV	AV_4.4			
8	0	2	6	AV	AV_4.5	v	v	
6	4	0	2	AV	AV_5.1	v		
15	8	4	3	AV	AV_6.1	u		r

Joonis 4.5: Komponentide ja andmetabelite seoste fan-in ja fan-out arvutamine

4.4.2. Seoste indeks

Seoste indeksi leidmiseks liidetakse kokku erinevate maatriksite fan-in ja fan-out väärtused. Näiteks joonise 4.4 ja 4.5 põhjal on komponendi AV_4.1 seoste indeksi väärtus 28 (= 5 + 6 + 17).

4.4.3. Komponendi ebastabiilsus

Komponendi ebastabiilsuse saab automaatselt arvutada fan-in ja fan-out pealt, kasutades peatükis 3.3.3 toodud valemit.

4.4.4. Jaccardi indeks

Jaccardi indeksit ei saa rakendada automaatselt, kuna võrrelda tuleb komponentide mikrostruktuure. Kõikide komponentide omavahelise sarnasuse mõõtmiseks tuleks eraldi koostada sarnasuste maatriks. Jaccardi indeksi väärtuste leidmine nõuab eraldi andmete kogumist, mis käesoleva magistritöö skoobist jääb välja.

4.4.5. IFC

Komponendi informatsioonivoo komplekssuse saab automaatselt arvutada fan-in ja fan-out väärtuste pealt, kasutades peatükis 3.3.5 toodud valemit

4.4.6. Komponendi suurus

Arvestades, et ülejäänud meetrikate mõõtmise aluseks on olnud tehnilised ülesanded, tuleb ka komponendi suuruse mõõtmiseks sama analüüsi tulemit kasutada, et teha järeldusi meetrikate omavahelistest seostest (täpsemalt peatükis 4.5). Komponendi suurust iseloomustab dokumentatsiooni suurus. Seetõttu teostatakse uus andmete kogumine ning leitakse kõikide komponentide dokumentatsiooni (tehnilised ülesanded) sõnade arv, mis on komponendi suuruse indikaatoriks.

4.4.7. Meetrikate rakendamise kokkuvõte

Meetrikate rakendamise järel valmib eraldi mõõtmiste tulemuste tabel, mida on võimalik töödelda ning tulemusi sorteerida ja filtreerida. Tabelis 4.1 on rakendatud meetrikate väärtuste kokkuvõte. Kokku mõõdeti 247 komponenti, mis on seotud 228 andmetabeliga.

	Sõnade arv	Seoste indeks	Komponentide vahelised seosed				Komponentide-andmetabelite seosed			
			Fan-in	Fan-out	Eba-stabiilsus	IFC	Fan-in	Fan-out	Eba-stabiilsus	IFC
Aritmeetiline keskmine	929,3	9,7	1,8	1,8	0,5	154,6	4,5	1,5	0,2	640,5
Mediaan	610	7	1	1	0,5	0	4	1	0,04	0
Miinimum	106	0	0	0	0	0	0	0	0	0
Maksimum	7834	47	15	13	1	17424	29	14	1	50176

Tabel 4.1: Rakendatud meetrikate väärtuste kokkuvõte

Tabelis 4.1 toodud kokkuvõttest nähtub, et keskmiselt on komponendil seoseid 9,7, mis jaotuvad järgnevalt: komponentide vahelisi seoseid on keskmiselt 3,6 (komponent sõltub 1,8 komponendist ning juhib 1,8 komponenti) ning komponentide-andmetabelite seoseid keskmiselt 6 (komponent loeb andmeid keskmiselt 4,5 tabelist ning kirjutab 1,5 tabelisse). Keskmine dokumentatsiooni maht on ligikaudu 929 sõna. Kõigi vaadeldavate tunnuste puhul on mediaan aritmeetilisest keskmisest väiksem viidates sellele, et tunnuste jaotus on positiivse asümmeetriaga. Teisisõnu on enamik väärtusi keskmisest väiksemad, kuid üksikud väga kõrge väärtusega juhtumid kallutavad keskmist kõrgema väärtuse suunas.

Mõõtmise tulemuste põhjal on võimalik leida kõige vähem modulaarsed komponendid, mille meetrikate väärtused on süsteemi keskmisest mitu korda kõrgemad. Tabelis 4.2 on toodud 20 kõige suurema seoste indeksiga komponenti, mis viitab nende vähesele modulaarsusele. Ühelt poolt võivad need olla äriloogiliselt keerukad komponendid või tuumikkomponendid, mille puhul võibki eeldada, et modulaarsus pole saavutatav. Teiselt poolt viitavad meetrikate väärtused halvale disainile, st komponente teisiti struktureerides saab modulaarsust parandada. On selge, et sellised paljude seostega komponendid muudavad süsteemi haprамaks, sest suurendavad tõenäosust, et muutuste efekt propageerub edasi ebaproportsionaalselt suurele osale süsteemist. Ka praktikas on enamus välja toodud 20 komponendist olnud probleemsed ja nõudnud palju muudatusi ja veaparandusi. Seega välja pakutud meetrikate rakendamine enne komponentide realiseerimisse suunamist võimaldab paremini hinnata ja võrrelda komponentide modulaarsust. Keskmisest oluliselt kõrgemad näitajad on heaks indikaatoriks, et komponenti on vaja restruktureerida.

Komponent	Sõnade arv	Seoste indeks	Komponentide vahelised seosed				Komponentide-andmetabelite seosed			
			Fan-in	Fan-out	Eba-stabiilsus	IFC	Fan-in	Fan-out	Eba-stabiilsus	IFC
AV_3.1	4076	47	12	11	0,48	17424	13	11	0,46	20449
LEP_1.2	7834	47	11	6	0,35	4356	16	14	0,47	50176
TV_1.1	1836	36	3	4	0,57	144	29	0	0,00	0
YL_3.1	5166	35	7	4	0,36	784	23	1	0,04	529
ISIK_2.2	4160	33	6	8	0,57	2304	12	7	0,37	7056
ISIK_10.8	4082	32	2	8	0,80	256	13	9	0,41	13689
OTS_1.6	4137	31	2	5	0,71	100	23	1	0,04	529
YLD_1.1	3174	31	15	1	0,06	225	15	0	0,00	0
LEP_1.9	4310	29	4	3	0,43	144	14	8	0,36	12544
AV_4.1	3836	28	6	5	0,45	900	7	10	0,59	4900
ISIK_10.3	2469	28	5	4	0,44	400	13	6	0,32	6084
TP_1.2	1988	28	3	7	0,70	441	14	4	0,22	3136
AV_10.2.1	1213	27	0	13	1,00	0	9	5	0,36	2025
ISIK_5.2	1023	27	11	2	0,15	484	14	0	0,00	0
TP_1.4	1354	27	1	6	0,86	36	19	1	0,05	361
OTS_1.2	1628	26	14	2	0,13	784	4	6	0,60	576
AV_4.1.1	1635	25	4	6	0,60	576	10	5	0,33	2500
TV_3.2	2666	25	5	7	0,58	1225	8	5	0,38	1600
YL_4.1	1897	25	0	4	1,00	0	21	0	0,00	0
LEP_1.4.2	2894	24	6	0	0,00	0	12	6	0,33	5184

Tabel 4.2: 20 kõige suurema seoste indeksiga komponenti

4.5. Regressioonanalüüs

Testimaks, kas ja kuidas mõjutab komponendi dokumentatsiooni mahukus seoste paljusust ning see omakorda muudatuste ja täienduste arvu, viiakse läbi lihtne regressioonanalüüs.

Eeldatavalt mõjutavad süsteemi modulaarsust kõige enam komponentide vahelised seosed – mida rohkem seoseid, seda vähem modulaarne süsteem. Suur seoste arv suurendab komponendi komplekssust, vähendab arusaadavust ning komponenti on raskem taaskasutada. Lisaks on komponent haavatavam, mis põhjustab rohkem vigu. Ka muudatuste ja täienduste hulk suureneb, sest komponenti võib olla vaja muuta ainult seetõttu, et muutus sellega seotud komponent. Komponendi suur seoste arv on omakorda põhjustatud keerulisest äriloogikast ja komponendi suurusest – mida suurem komponent,

seda rohkem seoseid. Modulaarsust tahame saavutada, et vähendada vigade ja täienduste arvu.

Seega on eeldatavad seosed sõnastatavad kahe hüpoteesina (H1 ja H2) ja neid hüpoteese välistavate nullhüpoteesidena ($H1_0$ ja $H2_0$):

- H1: Mida suurem on komponendi dokumentatsioon, seda kõrgem on komponendi seoste indeksi väärtus.
- $H1_0$: Komponendi dokumentatsiooni suurus ei mõjuta komponendi seoste indeksi väärtust.

- H2: Mida kõrgem komponendi seoste indeksi väärtus, seda rohkem tekib tulevikus komponendi muudatusi ja täiendusi.
- $H2_0$: Komponendi seoste indeks ei mõjuta komponendi muudatuse ja täienduste arvu.

Nende hüpoteeside kehtivuse kontrollimiseks testitakse kahte regressioonmudelit:

- Esimeses mudelis on sõltuvaks muutujaks komponendi seoste indeks ning seletavaks muutujaks komponendi dokumentatsiooni suurus.
- Teised mudelis on sõltuvaks muutujaks muudatuste ja täienduste arv ning seletavaks muutujaks komponendi seoste indeks.

Regressioonanalüüsi teostamiseks kasutatakse andmeanalüüsiprogrammi STATA IC 12.

4.5.1. Kirjeldav statistika

Enne hüpoteeside testimist antakse lühiülevaade regressioonmudelitesse hõlmatud tunnuste arvnäitajatest (tabel 4.3). Kokku mõõdeti 247 komponenti.

	Dokumentatsiooni suurus (sõnade arv)	Seoste indeks	Versioonide arv
Aritmeetiline keskmine	929,3	9,7	6,3
Mediaan	610	7	4
Miinimum	106	0	1
Maksimum	7834	47	53

Tabel 4.3: Tunnuste arvnäitajad

Dokumentatsiooni suurust näitab komponenti kirjeldava tehnilise ülesande sõnade arv (vt 4.4.3). Seoste indeksi väärtuse leidmine on kirjeldatud peatükis 4.4.2.

Kuna modulaarsusega püütakse optimeerida tarkvara evolutsiooni ehk vähendada muudatuste mõjuulatust ja muudatuste arvu, siis käesoleva regressioonanalüüsi käigus sooviti leida korrelatsioon modulaarsuse ja muudatuste-täienduste vahel. Seetõttu teostati täienduste ja muudatuste arvu mõõtmiseks uus andmete kogumine, mille käigus leiti dokumentatsiooni versioonide arv. Kasutati sama valimit nagu seoste indeksi ja dokumentatsiooni suuruse leidmisel.

Tabelis 1 toodud kokkuvõttest nähtub, et dokumentatsiooni sõnade arvu varieeruvus on väga suur. Kõige mahukam dokumentatsioon sisaldab 7834 sõna, mida on 74 korda enam kui sõnu kõige väiksemas dokumentatsioonis. Keskmine dokumentatsiooni maht on ligikaudu 929 sõna.

Võrreldes dokumentatsiooni mahu erinevustega, on seoste indeksi ja versioonide arvu variatsioon veidi väiksem. Seoste indeksi väärtused varieeruvad nullist kuni 47 ning versioonide arv ühest kuni 53. Keskmiselt on ühel komponendil ligikaudu 10 seost ning 6 versiooni.

Kõigi vaadeldavate tunnuste puhul on mediaan aritmeetilisest keskmisest väiksem viidates sellele, et tunnuste jaotus on positiivse asümmeetriaga. Teisisõnu on enamik väärtusi keskmisest väiksemad, kuid üksikud väga kõrge väärtusega juhtumid kallutavad keskmist kõrgema väärtuse suunas.

4.5.2. Dokumentatsiooni suurus ja komponendi seoste indeks

Tabelis 4.4 toodud tulemustest nähtub, et dokumentatsiooni sõnade arv ning komponendi seoste indeks on omavahel statistiliselt oluliselt seotud. Iga täiendav sõna dokumentatsioonis suurendab seoste arvu 0,006 võrra ehk 167 täiendavat sõna toovad kaasa ühe lisaseose. Seejuures on statistilise usaldusväärsuse näitaja *p-väärtus* väiksem kui 0,001, mis tähendab, et tõenäosus, et eksime, kui eeldame positiivse seose olemasolu, on väiksem kui 0,1%. Seega vähem kui 0,1% tõenäosusega on valimis nähtav seos saadud juhuslikult. Järelikult võib nullhüpoteesi $H1_0$ ümber lükata ning järeldada, et dokumentatsiooni suurus ja komponendi seoste indeks on positiivselt seotud.

	Koefitsient (standardviga)
Sõnade arvu kordaja	0,006 (0,0003)*
Konstant	3,76 (0,4244)*
R^2	0,63
Jääkväärtus	24,06
Valimi suurus	247

*p<0,001%

Tabel 4.4: Regressioonmudel: sõltuvaks muutujaks komponendi seoste indeksi väärtus

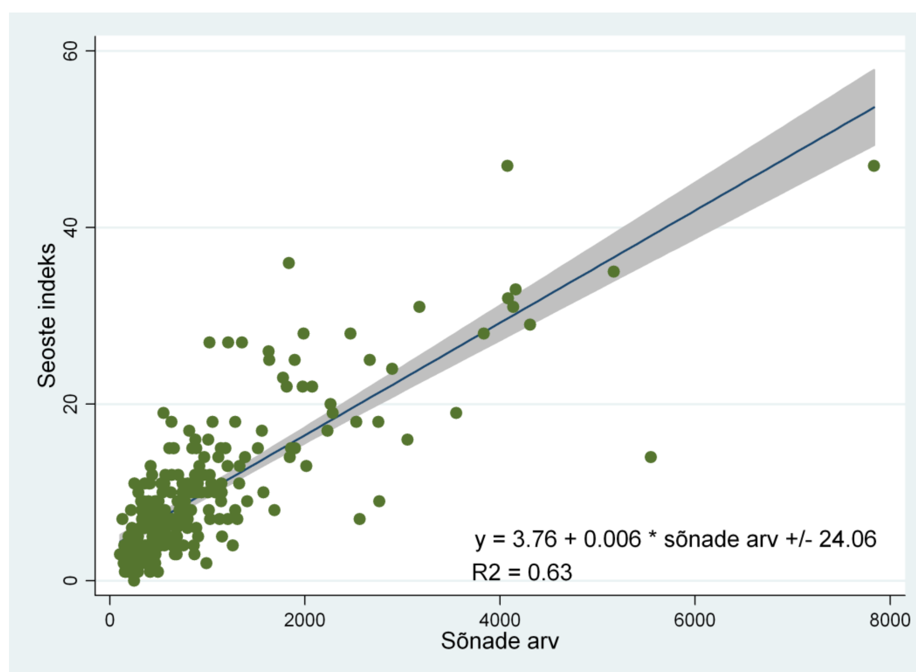
Mudeli kirjeldusmäära (R^2) väärtus viitab mudeli küllaltki tugevale kirjeldusvõimele: dokumentatsiooni sõnade arv seletab ära 63% seoste indeksi väärtuste variatiivsusest.

Regressioonanalüüsi tulemused võib kokkuvõtvalt väljendada järgmise valemiga:

*seoste indeks = 3,76 + 0,006*sõnade arv*

Valem võimaldab kõikidele lisanduvatele komponentidele ennustada seoste indeksi väärtust dokumentatsiooni suuruse põhjal. Kui komponendi dokumentatsiooni sõnade arv suureneb, tõuseb ka seoste indeks. Näiteks kui komponendi dokumentatsioon on 2000 sõna, on seoste eeldatav arv 3,76 + 0,006*2000 = 15,76. Standardviga (=24,06) on osa tunnuse väärtusest, mis sõltub ainult juhuslikest mõjudest.

Tunnuste vahelise sõltuvusest visuaalse ülevaate saamiseks koostatakse regressioonanalüüsi tulemuste alusel hajuvusdiagramm (joonis 4.6), kuhu kantakse punktidena kõigi mõõdetud komponentide väärtused ning variatsiooni kõige paremini iseloomustav regressioonsirge koos selle 95-protsendilise usaldusintervalliga.



Joonis 4.6: Komponentide seoste indeksi ja dokumentatsiooni sõnade arvu vaheline hajuvusdiagramm

Jooniselt 4.6 on näha, et valdaval osal komponentidest on dokumentatsioon väiksem kui 2000 sõna ning seoste arv alla 30. Kuna suurema sõnade ja seoste arvuga vaatlusi on märgatavalt vähem, on ka regressioonsirge 95% usaldusintervall suuremate väärtuste korral suurem. Seega suudetakse antud valemiga ennustada täpsemalt kuni 2000 sõna pikkuse dokumentatsiooniga komponentide seoste arvu kui väga mahuka dokumentatsiooniga komponentide seoste arvu.

4.5.3. Komponendi seoste indeks ja versioonide arv

Tabelis 4.5 kajastatud tulemustest on näha, et komponendi seoste indeks ning muudatuste ja täienduste arv on omavahel statistiliselt oluliselt seotud. Iga täiendav seos suurendab muudatuste ja täienduste arvu 0,68 võrra ehk 1,5 täiendavat seost toovad keskmiselt kaasa ühe dokumentatsiooni versiooni. Seejuures on seose statistilise usaldusväärsuse näitaja p väärtus väiksem kui 0,001, mis tähendab, et tõenäosus, et eksime, kui eeldame

positiivse seose olemasolu, on väiksem kui 0,1%. Järelikult võib nullhüpoteesi $H2_0$ ümber lükata ning järeldada, et komponendi seoste indeks ja versioonide arv on positiivselt seotud.

	Koefitsient (standardviga)
Seoste indeksi kordaja	0,68 (0,0368)*
Konstant	3,76 (0,4642)*
R^2	0,58
Standardviga	21,82
Valimi suurus	247

*p<0,001

Tabel 4.5: Regressioonmudel: sõltuvaks muutujaks komponendi versioonide arv

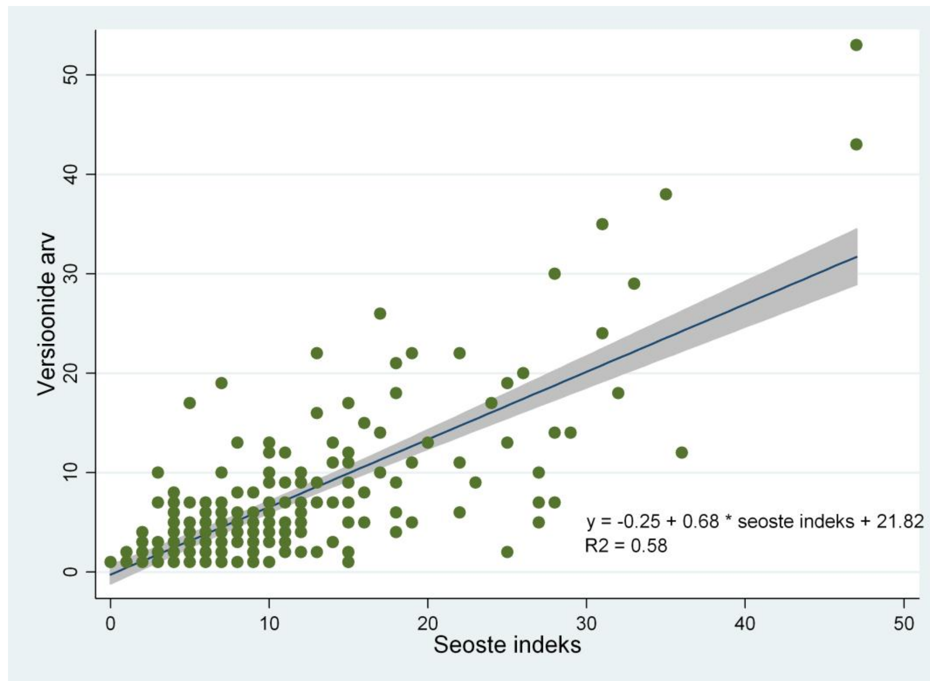
Mudeli kirjeldusmäära (R^2) väärtus viitab mudeli küllaltki tugevale kirjeldusvõimele: seoste arv seletab ära 58% versioonide väärtuste variatiivsusest.

Regressioonanalüüsi tulemused võib kokkuvõtvalt väljendada järgmise valemiga:

*komponendi versioonide arv = -0,25 + 0,68*seoste indeks*

Valem võimaldab kõikidele lisanduvatele komponentidele ennustada versioonide arvu komponendi seoste arvu põhjal. Kui komponendi seoste arv suureneb, tõuseb ka versioonide arv. Näiteks kui komponendi seoste arv on 10, on versioonide eeldatav arv -0,25 + 0,68*10 = 6,6.

Tunnuste vahelistest sõltuvustest visuaalse ülevaate saamiseks koostatakse regressioonanalüüsi tulemuste alusel hajuvusdiagramm (joonis 4.7), kuhu kantakse punktidena kõigi mõõdetud komponentide väärtused ning variatsiooni kõige paremini iseloomustav regressioonsirge koos selle 95-protsendilise usaldusintervalliga.



Joonis 4.7: Komponentide versioonide arvu ja dokumentatsiooni sõnade arvu vaheline hajuvusdiagramm

Joonis 4.7 ilmestab tugevat lineaarset seost: kui seoste indeks tõuseb, suureneb ka versioonide arv (st täienduste ja muudatuste arv). Samuti viitab joonis sellele, et regressioonvalemi alusel tehtav ennustus on kõige täpsem juhul, kui seoste indeksi arv on ligikaudu 10 (95% usaldusintervall on selles vahemikus kõige kitsam). Väga kõrge seoste indeksi väärtuse korral on 95% usaldusintervall laiem ning seega ka seoste indeksi alusel versioonide arvu ennustus ebatäpsem.

EMPIS-e puhul võib tulemust tõlgendada järgnevalt: kui analüütik on kirjeldanud komponendid, milles on seoste indeks suur, tähendab see tõenäoliselt seda, et ka tulevikus on selle komponendiga kaasnevate muudatuse ja täienduste hulk suur. Suur seoste indeks on indikaatoriks, et tuleviku muudatuste vältimiseks tuleks komponenti ümber struktureerida nii, et seoste indeks väheneks.

4.5.4. Regressioonanalüüsi tulemused

Regressioonianalüüsi tulemusena selgub, et mida suurem dokumentatsioon, seda suurem on seoste indeks, mis vähendab komponendi modulaarsust. Suurem seoste arv põhjustab ka täienduste ja muudatuste arvu kasvu. Seega kehtivad joonisel 4.8 kujutatud seosed:

* p<0,001

Joonis 4.8: Komponendi dokumentatsiooni suuruse, seoste indeksi ja versioonide vahelised seosed

Kuna regressioonanalüüsi tulemusel leitud koefitsientide statistiline usaldusväärsus on väga kõrge ning mudelite kirjeldusaste samuti suhteliselt kõrge, võib olla kindel, et testitud seosed kehtivad ka juhul, kui kontrollida veel võimalike teiste faktorite mõju. Näiteks võivad komponendi muudatuste ja täienduste arvu mõjutada veel järgnevad aspektid:

- Äriloogika muutus - organisatsiooni struktuuri või protsesside muutumine
- Komponendi kasutamise aeg – mida kauem on komponent kasutusel, seda suurema tõenäosusega on seda vaja muuta. Uute komponentide puhul on versioonide arv väike.
- Komponendi keerukus – teatud komponentide puhul võib seoste indeks olla madal, kuid keeruline äriloogika tekitab vajaduse muudatuste ja täienduste järele.
- Komponendi olulisus – tuumikkomponente ja üldisi dokumente on vaja tihedamini muuta, kuigi seoste arv ei pruugi tõusta.

Seega komponendi mahukas dokumentatsioon ning sellest tulenev suur seoste hulk on indikaatoriks, et komponenti tuleb tulevikus rohkem muuta, kuid lisaks võib muudatuste ja täienduste arvu mõjutada ka mõni muu faktor, mille mõju antud töös ei olnud võimalik testida.

Lisaks tuleb arvestada, et tulemus on saadud suhteliselt väikese valimi pealt (247 komponenti), mis kõik asuvad samas süsteemis. Üldisemate järelduste tegemiseks tuleks valimit suurendada ning teostada mõõtmisi erinevate süsteemide peal.

4.6. Juhtumiuuringu kokkuvõte

Arvutusliku mudeli rakendamine projekti „EMPIS“ analüüsitulemite peal andis objektiivse hinnangu komponenti modulaarsuse tasemest ja seega kinnitas, et antud töös kirjeldatud lähenemist on võimalik praktikas kasutada.

Järgnevalt antakse täpsem kokkuvõte juhtumiuuringu tulemustest.

Juhtumiuuringu käigus rakendati peatükis 3 välja pakutud mudelit reaalse süsteemi peal. Analüüsi artefaktidest kajastasid komponentide vahelisi seoseid kõige paremini tehnilised ülesanded, mis olid mõõtmisteks sobilikud ka uuendatud ja täielike andmete tõttu. Kuna tehnilistest ülesannetest seoste tuvastamine on liialt ajamahukas, otsustati sõltuvuste paremaks visualiseerimiseks koostada DSM. Selle tegevuse käigus aga selgus, et DSM käsitsi koostamisel pole praktilist väärtust, vaid pigem tekitab analüütikutele juurde täiendava dokumenteerimiskohustuse ja tekib oht, et maatriksit ei uuendata piisavalt tihti. Seetõttu arendati välja automaatne DSM koostamise programm, mis võtab sisendiks tehnilised ülesanded ning väljastab komponentide vaheliste seoste ruutmaatriksi ning komponentide-andmetabelite seoste maatriksi.

Loodud maatriksid on kasulikud nii probleemsete sõltuvuste tuvastamiseks kui ka muudatuse mõjuulatuse hindamiseks. Iga uue muudatuse või täienduse korral saab kiiresti kontrollida, milliste komponentidega on muudetav komponent või andmetabel seotud. Dokumentatsioonist sõltuvuste otsimine oleks tunduvalt ajakulukam. Kuna seosed on tehtud ilmseks, on neid ka lihtsam kontrollida.

DSM peal rakendati erinevaid meetrikaid analüüsitulemite sidestuse ja komplekssuse mõõtmiseks ning eraldi mõõdeti komponentide suurust. Mõõtmise tulemused võimaldavad analüüsi artefakte modulaarsuse seisukohalt võrrelda, hinnata ning identifitseerida kõrge veariskiga komponendid, mida tuleks modulaarsuse ja parema kvaliteedi saavutamiseks restruktureerida.

Testimaks, kas ja kuidas mõjutab komponendi dokumentatsiooni mahukus seoste paljusust ning see omakorda muudatuste ja täienduste arvu, viidi läbi lihtne regressioonanalüüs. Selgus, et komponendi mahukas dokumentatsioon ning sellest tulenev suur seoste hulk on indikaatoriks, et komponenti tuleb tulevikus rohkem muuta,

kuid lisaks võib muudatuste ja täienduste suurt arvu mõjutada ka mõni muu faktor, mille mõju antud töös ei olnud võimalik testida.

5 Kokkuvõte

Antud töös uuriti modulaarsusega seotud aspekte süsteemi analüüsifaasis, et avastada struktuurseid vigu võimalikult varakult. Tutvustati modulaarsusega seotud põhimõisteid ja printsiipe ning selgitati, kuidas modulaarsust hinnata. Ehkki ka analüütiku valikuid mõjutavad arhitektuuriotsused ja kasutatav tehnoloogia, saab analüütik süsteemi kirjeldada ja komponentidesse jaotada väga erinevalt, millest aga sõltub ka see, kas modulaarsus on saavutatav. Loogiline ja selge komponentide jaotus ning ilmsed seosed lihtsustavad edaspidist süsteemi arendust ja hooldust.

Töö raames arendati välja modulaarsuse mõõtmise raamistik, kasutades eesmärk-küsimus-meetrika lähenemisviisi. Selle käigus konkretiseeriti arvutuslik mudel analüüsifaasis sidestuse, kohesiooni ja komplekssuse arvutamiseks. Mudelit rakendati reaalse projekti peal. Esmalt arendati sõltuvuste visualiseerimiseks välja eraldi programm, mis tagastab analüüsidokumentide põhjal komponentide vaheliste sõltuvuste disaini struktuuri maatriksi. Maatriksi peal rakendati erinevaid meetrikaid analüüsitulemite sidestuse ja komplekssuse mõõtmiseks ning eraldi mõõdeti komponentide suurust. Mõõtmise tulemused võimaldavad analüüsi artefakte modulaarsuse seisukohalt võrrelda, hinnata ning identifitseerida kõrge veariskiga komponendid, mida tuleks modulaarsuse ja parema kvaliteedi saavutamiseks restruktureerida. Analüütik saab mudelit rakendada enne komponentide realiseerimisse suunamist ja nii avastada analüüsivead palju efektiivsemalt.

Tulevikus on kavas antud arvutuslikku mudelit rakendada EMPIS projektis ning vastavalt vajadusele edasi arendada, et praktikas oleks sellest võimalikult palju kasu. Ka disaini struktuuri maatriksi graafilist väljundit on vajalik täiendada probleemsete mustrite ja sõltuvuste tuvastamiseks. Lisaks juba koostatud maatriksitele on plaan arendada komponentide-liideste ning komponentide-testjuhtumite seoseid kajastav maatriks, et kõik analüüsifaasis identifitseeritavad seosed oleks ilmsed. Kui arvutuslikku mudelit õnnestub EMPIS projektis edukalt juurutada, võib mudelit rakendada ka teiste projektide peal, et teha üldisemad järeldused mudeli valiidsusest.

Measuring modularity in software analysis phase

Master thesis (30 EAP)

Katrin Toe

Abstract

Modularity is a general set of design principles for managing the complexity of large systems. It involves breaking down the system into separate, independent components that communicate with each other through standardized interfaces or rules and specifications. Modularity creates options and makes development and maintenance process much easier to handle.

The topic of modularity is more often the focal point when designing the software architectures. However, the aim of this work is to investigate aspects related to modularity in the system analysis phase that precedes and dominates coding. Although the decisions of system analyst are influenced by software architecture and development environment, there are still many choices how to divide the system into components and how the components are connected. If the aspects of modularity are ignored by analysts, those decisions could have cascading negative effect on the system quality while logical and clear structure of components and dependencies will facilitate further development and maintenance process.

In this thesis we investigate how modularity can be achieved on a system analysis phase. To assess design modularity quantitatively and objectively, we propose a framework to measure the modularity properties in a software analysis phase. A goal-question-metric approach is used to derive the different metrics in this framework. More specifically, we define measures to quantify coupling, cohesion, complexity and size of components based on the artifacts of system analysis.

In order to validate the framework, a case study is conducted. Firstly, dependency analysis tool was developed that represents components dependencies in a dependency structure matrix form. The input of this tool is documentation. Secondly, the matrix is processed according to the coupling and complexity metrics defined before. Size of the

component is measured separately. The measurement results can then be used to compare different artifacts of system analysis in terms of their modularity and identify high-risk components that might require redesign. System analyst can apply the method early before coding in order to indicate the possible problem areas of analysis more efficiently.

Sõnastik

Analüüsifaas - tarkvaraarenduse faas, kus määratletakse kliendi vajadused ja võimalused tarkvara süsteemi arendamiseks ning projekteeritakse ja modelleeritakse süsteem. Modelleerimisel on oluline mõista nii süsteemi protsesside omavahelisi seoseid kui ka väliseid süsteeme hõlmavaid seoseid.

Analüütik – isik, kes määratleb kliendi vajadused ja võimalused tarkvara süsteemi arendamiseks ning projekteerib ja modelleerib süsteemi. Analüütiku poolt loodud artefaktid on programmeerijatele sisendiks, mille põhjal toimub süsteemi realiseerimine.

Artefakt – tarkvaraarenduse käigus loodav ja/või kasutatav tulem.

Dekompositsioon – süsteemi ülesehitus, kus tervik jaotatakse osadeks (ülevalt-alla lähenemine).

DSM – disaini struktuuri maatriks, mis kajastab süsteemi erinevate elementide vahelisi seoseid

Hüpotees – mingi nähtuse seletamiseks esitatud tõestamata, ent ka kummutamata teaduslik oletus [43].

Integratsioon - tervikliku süsteemi järk-järguline koostamine komponentidest [11].

Kohesioon - iseloomustab komponendi sisemisi seoseid [12]. Kohesioon näitab, mil määral on komponendi elemendid omavahel seotud.

Komponent - antud töös tähendab süsteemi dekompositsiooni ühikut. Sõltuvalt programmeerimise paradigmast võib komponendiks olla nii moodul, pakett, klass, teenus või mistahes eraldi seisev tükk süsteemist, mis sisaldab sarnaseid funktsioone või andmeid.

Kompositsioon – süsteemi ülesehitus, kus osadest pannakse kokku tervik (alt-üles lähenemine).

Meetrika - kvantitatiivse mõõdik, mis näitab, mil määral süsteem, komponent või protsess vastab teatud nõuetele [24]. Meetrikad aitavad hinnata ja ennustada tarkvara kulusid ja kvaliteeti [23].

Modulaarsus – mõõt, mis näitab, millises ulatuses koosneb programm sellistest osadest, mille hulgast ühe muutmine mõjutab teisi minimaalselt [11]. Modulaarseks peetakse toodet või süsteemi, mis on üles ehitatud omavahel vähe seotud osadest nii, et ühe alamosa muutusel on minimaalne mõju teistele alamosadele ning seetõttu on võimalik alamosasid eraldi arendada, täiendada või taaskasutada [6].

Nullhüpotees – *hüpoteesi* vastandhüpotees. Konservatiivne väide, mis eeldab tavaliselt, et muutusi ei ole, erinevus puudub jne [44].

Sidestus – erinevate süsteemi alamosade vaheline vastastikune seos või sõltuvus [11].

Süsteemianalüütik - antus töös sama tähendusega nagu *analüütik.*

Tarkvara arhitektuur – süsteemi baasstruktuur, mis hõlmab programmeerimiskeelte valikut, võrgu- ja andmebaasistruktuuri, raamistike valikut jne.

Versioneerimine – tarkvara muutuste ajaloo pidamise võime [11].

Viited

[1] N.S.Gill, P.S.Grover, „Component-based measurement: few useful guidelines," *ACM SIGSOFT Software Engineering Notes,* kd. 28, nr 6, 2003.

[2] M. M. Lehman, „On Understanding Laws, Evolution, and Conservation in the Large-Program Life Cycle," *Journal of Systems and Software,* kd. 1, pp. 213-221, 1980.

[3] D. L. Parnas, „Software aging," *Proceedings of the 16th International Conference on Software Engineering*, Sorento, Italy, pp. 279–287., 1994.

[4] C. Y. Baldwin, K. B. Clark, „Design Rules: The Power of Modularity", MIT Press, 2000.

[5] S. K. Ethiraj, D. Levinthal, „Modularity and Innovation in Complex Systems," *Management Science,* kd. 50, nr 2, 2004 http://www.jstor.org/pss/30046056 [viimati vaadatud 01.01.2012].

[6] R. Langlois, „Modularity in technology and organization," *Journal of Economic Behavior & Organization,* kd. 49, 2002.

[7] L. Bass, P. Clements, R. Kazman, „Software Architecture in Practice vol. 2", Addison Wesley, 2003.

[8] Sun Microsystems, „The benefits of modular programming," 2007. http://netbeans.org/project_downloads/usersguide/rcp-book-ch2.pdf [viimati vaadatud 01.01.2012].

[9] Gauthier, Richard, Stephen, „Designing Systems Programs", Prentice-Hall, Englewood Cliffs, N.J.„ 1970.

[10] M. E. Sosa, S. D.Eppinger, C. M. Rowles, „A Network Approach to Define Modularity of Components in Complex Products," 2007.

[11] „IT terministandardi sõnastik“ http://www.keeleveeb.ee/dict/speciality/itstandard/ [Viimati vaadatud 01.01.2012]

[12] W. Stevens, G. Myers, L. Constantine, „Structured design“, Yourdon Press, 1979.

[13] D. Poshyvanyk, A. Marcus, „The Conceptual Coupling Metrics for Object-Oriented Systems,“ Philadelphia, 2006.

[14] B. Meyer, „Object-Oriented Software Construction (second edition)“, Santa Barbara, California: ISE Inc., 1997.

[15] N. Fenton ja A. Melton, „Deriving structurally based software measures,“ *Journal of Systems and Software - An Oregon workshop on software metrics,* kd. 12, nr 3, 1990.

[16] „Systems Design: Principles of Hierarchical Decomposition,“ http://it.toolbox.com/blogs/enterprise-solutions/systems-design-principles-of-hierarchical-decomposition-48204 [Viimati vaadatud 01.01.2012].

[17] K. B.Clark, C. Y.Baldwin, „The Option Value of Modularity in Design: An Example from Design Rules, Volume 1: The Power of Modularity,“ 2002.

[18] L. Marengo, G. Dosi, P. Legrenzi, C. Pasquali, „The structure of problem-solving knowledge and the structure of organizations,“ Industrial & Corporate Change.

[19] D.L.Parnas, „On the criteria to be used in decomposing systems into modules,“ *Communications of the ACM,* kd. 15, nr 12, 1972 http://www.cs.umd.edu/class/spring2003/cmsc838p/Design/criteria.pdf [Viimati vaadatud 01.01.2012].

[20] K. Ulrich, S. Eppinger, „Product Design and Development“, McGraw-Hill/Irwin, 1999.

[21] A. MacCormack, J. Rusnak, C. Baldwin, „The Impact of component modularity on design evolution: evidence from the software industry,“ 2007. http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1071720 [Viimati vaadatud 01.01.2012]

[22] S. Huynh, Y. Cai, „An Evolutionary Approach to Software Modularity Analysis,“ *ACoM '07 Proceedings of the First International Workshop on Assessment of Contemporary Modularization Techniques*, Washington, DC, USA, 2007.

[23] N. E. Fenton ja M. Neil, „Software metrics: roadmap,“ *ICSE 00 Proceedings of the Conference on The Future of Software Engineering*, New York , 2000 http://www.cs.ucl.ac.uk/staff/A.Finkelstein/fose/finalfenton.pdf [Viimati vaadatud 01.01.2012].

[24] IEEE, „IEEE Standard Glossary of Software Engineering Terminology,“ 1990.

[25] H. Balzert, „Lehrbuch der Software-Technik. 2“, Berlin, Heidelberg, 1998.

[26] V. R. Basili, G. Caldiera, H. D. Rombach, „The goal question metric approach,“ Encyclopedia of Software Engineering, 1994 http://www.cs.umd.edu/~mvz/handouts/gqm.pdf [Viimati vaadatud 01.01.2012].

[27] R. V. Solinger, E. Berghout, „Goal/Question/Metric Method: A Practical Guide for Quality Improvement of Software Development“, McGraw-Hill Companies, 1999 http://www.iteva.rug.nl/gqm/GQM%20Guide%20non%20printable.pdf [Viimati vaadatud 01.01.2012].

[28] Y. Wang, Q. He, „A Practical Methodology for Measurement Deployment in GQM,“ *Proceedings of the Canadian Conference on Electrical and Computer Engineering (IEEE CCECE 2003)*, Montreal, Canada, 2003.

[29] P. Berander, P. Jonsson, „A Goal Question Metric Based Approach for Efficient Measurement Framework Definition,“ *ISESE 06 Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*, New York, 2006.

[30] S. Henry, D. Kafura, „Software Structure Metrics Based on Information Flow,“ *IEEE Transactions on Software Engineering* , kd. SE-7, nr 5, pp. 510-518, 1981.

[31] V. Gruhn, R. Laue, „Complexity Metrics for Business Process Models,“ 2006.

http://ebus.informatik.uni-leipzig.de/~laue/papers/metriken.pdf [Viimati vaadatud 01.01.2012]

[32] C. Borysowich, „Design Principles: Fan-In vs Fan-Out,“ 2007. http://it.toolbox.com/blogs/enterprise-solutions/design-principles-fanin-vs-fanout-16088 [Viimati vaadatud 01.01.2012]

[33] R. C. Martin, „Stability,“ 1997. http://www.objectmentor.com/resources/articles/stability.pdf [Viimati vaadatud 01.01.2012]

[34] S. R. Chidamber, C. F. Kemerer, „A Metrics Suite for Object Oriented Design,“ *IEEE transactions on software engineering,* kd. 20, nr 6, 1994.

[35] L. C. Briand, J. W. Daly, J. Wüst, „A Unified Framework for Cohesion Measurement in Object-Oriented Systems,“ *Empirical Software Engineering,* 1998.

[36] „Jaccard index,“ http://en.wikipedia.org/wiki/Jaccard_index [Viimati vaadatud 01.01.2012]

[37] Y. Ma, K. He, J. Liu, X. Zhou, „A Hybrid Set of Complexity Metrics for Large-scale Object-oriented Software Systems,“ *IEEE International Conference on Computer and Information Technology (CIT 2006)* , 2006.

[38] P. K. Suri, N. Garg, „Software Reuse Metrics: Measuring Component Independence and its applicability in Software Reuse,“ *IJCSNS International Journal of Computer Science and Network Security,* kd. 9, nr 5, 2009.

[39] N. O. Suh, „Axiomatic Design: Advances and Applications“, Oxford University Press, USA, 2001 http://www.axiomaticdesign.com/technology/ADSChapter5.html [Viimati vaadatud 01.01.2012].

[40] K. Otto, K. Wood, „Product Design,“ Prentice Hal, New Jersey, 2001.

[41] D. V. Steward, „The Design Structure System: A Method for Managing the Design of Complex Systems,“ *IEEE Transactions on Engineering Management,* kd.

EM_28, nr 3, pp. 71-74, 1981.

[42] N. Sangal, E. Jordan, V. Sinha, D. Jackson, „Using dependency models to manage complex software architecture," 2005 http://sdg.csail.mit.edu/pubs/2005/oopsla05-dsm.pdf [Viimati vaadatud 01.01.2012].

[43] „Eesti keele seletav sõnaraamat," http://www.keeleveeb.ee/ [Viimati vaadatud 01.01.2012]

[44] „Hüpoteesid," http://www.e-ope.ee/_download/euni_repository/file/47/Loeng_3.PDF [Viimati vaadatud 01.01.2012].

Lisad

Lisa 1. EMPIS tehnilise ülesande näidis

Tehniline ülesanne ADM_3.3 - Otsuse põhjuse lisamine-muutmine

Versioon nr.	Kuupäev	Autor	Kommentaar
V0.01	20.11.2009	Katrin Toe	Alusdokumendi loomine
V0.02	24.11.2011	Katrin Toe	Pealkiri, task 157108

1. Eesmärk

Eesmärgiks on anda kasutajale võimalus lisada-muuta otsuse põhjuse sisuteksti. Komponent avatakse modaaldialoogis.

2. Privileegid

- *adm.otsuste_haldus*

3. Sisendid

- Lisamisel: DOKUMENT_KIRJELDUS.ID
- Muutmisel: DOKUMENT_KIRJELDUS_POHJUS.ID

4. Pealkiri

- Otsuse põhjuse lisamine-muutmine

5. Atribuudid

Kuvatakse järgnevad otsuse põhjuse lisamine-muutmise väljad:

5.1. Põhjus – valikmenüü loendi POHJUS_KOOD kehtivatest väärtustest [seotud väljaga POHJUS_KOOD]

5.2. Vanemotsuse suund – väli kuvatakse vaid juhul kui põhjusega seotud dokumendi kirjelduse suunaks on „KEHTETUKS" või „MUUTMINE". Kuvatakse valikmenüü loendi OTSUS_SUUND väärtustest [seotud väljaga VANEM_OTSUS_SUUND_KOOD. Kuvatakse juhul kui põhjusega seotud DOKUMENT_KIRJELDUS.DOKUMENT_SUUND_KOOD in ('KEHTETUKS', 'MUUTMINE')].

5.3. On süsteemne – checkbox [ON_SYSTEEMNE]

5.4. On rikkumine – checkbox [ON_RIKKUMINE]

5.5. Trükise pealkiri – tekstiväli, otsuse pealkiri sõltuvalt põhjusest [PEALKIRI]

~~5.5.~~5.6. Seaduse punktid – tekstiväli [SEADUSE_PUNKTID]

~~5.6.~~5.7. Kehtiv alates – kuupäeva väli [KEHTIV_ALATES]

~~5.7.~~5.8. Kehtiv kuni – kuupäeva väli [KEHTIV_KUNI]

~~5.8.~~5.9. Sisu – tekstiala [SISU]

7. Kontrollid

7.1. Kohustuslikud väljad:

- Põhjus
- Kehtiv alates
- Vanemotsuse suund – juhul kui väli kuvatakse

7.2. Kehtiv kuni ei tohi olla varasem kui kehtiv alates

7.3. Unikaalsuse kontroll – DOKUMENT_KIRJELDUS_ID, POHJUS_KOOD, VANEM_OTSUS_SUUND_KOOD on unikaalsed

7.4. Tuleb kontrollida, kas sisuteksti sisestatud muutuja eksisteerib [kontrollida, kas sisutekstis loogelistes sulgudes olev {muutuja} leidub ka loendis OTSUS_SISUTEKST_MUUTUJA (=KOOD väärtus)]

8. Tegevused

8.1. Nupp „Salvesta"

Teostataks kontrollid, salvestatakse andmed.

8.2. Nupp „Katkesta"

Sulgeb modaaldialoogi ilma andmeid salvestamata.

9. Salvestusreeglid

9.1. DOKUMENT_KIRJELDUS_POHJUS

Välja nimi	Tüüp	Täpsustus
ID	NUMBER(18)	Unikaalne võti
DOKUMENT_KIRJELDUS_ID	NUMBER(18)	Viide DOKUMENT_KIRJELDUS tabeli kirjele
POHJUS_KOOD	VARCHAR2(30)	Kasutaja sisestatud väärtus (loendi OTSUS_POHJUS väärtus)
VANEM_OTSUS_SUUND_KOOD	VARCHAR2(30)	Kasutaja sisestatud väärtus (loendi OTSUS_SUUND väärtus)
ON_SYSTEEMNE	NUMBER(1)	Kasutaja valitud väärtus
ON_RIKKUMINE	NUMBER(1)	Kasutaja valitud väärtus
SEADUSE_PUNKTID	VARCHAR(255)	Kasutaja sisestatud väärtus
KEHTIV_ALATES	DATE	Kasutaja sisestatud väärtus
KEHTIV_KUNI	DATE	Kasutaja sisestatud väärtus
SISU	CLOB	Kasutaja sisestatud väärtus
PEALKIRI	VARCHAR2(255)	Kasutaja sisestatud väärtus