TARTU ÜLIKOOL
MATEMAATIKA-INFORMAATIKATEADUSKOND
Arvutiteaduse instituut
Hajussüsteemide õppetool
Infotehnoloogia eriala

Anti Kivi

Tarkvara testimissüsteemid GRID-il

Bakalaureusetöö (4AP)

Juhendaja: prof. Eero Vainikko

Autor: …….. "……" mai 2006
Juhendaja: …… "……" mai 2006

TARTU 2006

Sisukord

Sissejuhatus

Andmete ja arvutusressursi vajadus on igavesti kasvav. Seda toetab edukalt Gordon Moore'i seadus: lülitite arv protsessoritel kahekordistub iga 18 kuu järel. Näiteks osakeste kiirendi, mis valmib aastal 2007, hakkab tootma 10 PB andmeid aastas ja vajab suures koguses arvutusjõudlust. Selliseid teaduse ja ka kommertsalasid on veel mitmeid, mille realisatsioon jääb ressursi nappuse tõttu olemata. Ka ilma ennustamisel on tarvis suurt arvutusressurssi ja selle illustreerimiseks on levinud ütlus: “Ei ole ju mõtet ennustada eilset ilma”.

GRIDi infrastruktuur on peamiselt rajatud järjest suurenevate andme- ja arvutusressursside vajaduste rahuldamiseks. Seda peetakse järgmiseks suureks projektiks peale Internetti. Idee ära kasutada arvutite vabu tsükleid tekkis juba 1930.ndatel aastatel. Grid on hajusstruktuur põhimõttega, et massis peitub jõud ehk arvutuslik töö tuleks jagada pisemateks osadeks ja seejärel need laiali saata. Mida rohkem arvutuselemente, seda ühtlasemalt on jõudlus jagatud. See seab suured nõudmised Gridi vahevara dünaamilisusele.

Paljud mahukad ülesanded on taandatavad suuremahuliste lineaarvõrrandite lahendamisele. Seda omadust ära kasutades on loodud selliste võrrandite lahendamiseks programmipakett DOUG, mis lühidalt rakendab võrrandite lahendamiseks piirkondadeks jagamise meetodit. Suurema jõudluse saavutamiseks kasutab pakett MPI teateedastusteeke. DOUG on võimaline töötama ka GRID-i sõlmedel, mis suurendab programmi jõudlust veelgi.

Testimine kui tarkvara kvaliteedi tagamine on järjest tähtsamaks muutuv aspekt tarkvaraarenduses. On tekkinud suur hulk testimisvahendeid, nii kommerts- kui vabavaralisi, mis üha rohkem integreeruvad tarkvara arenduse protsessi ja püüavad automatiseerida testimist.

Regressiontestimine oma loomult on üks parimaid kandidaate testimise automatiseerimise implementatsiooniks, sest see sisaldab juba läbitud testide uuesti rakendamist iga kord, kui tarkvara muutub või lisatakse sellele uus omadus. Järelikult on tegemist korduvate protsessidega, mida on hõlbus automatiseerida.

Kuna DOUGi rakendamine toimub mahukatel ülesannetel ja on aeganõudev, siis selle testimine samadel töötingimustel ja koormustel on veelgi aeganõudvam ja võib kõvasti pärssida arendustööd.

Käesoleva töö eesmärk ongi programmipaketile DOUG automaatse tarkvaratestimise süsteemi loomine, mis täidaks teste assisteerimiseta ja pakuks võimalused tulemuste analüüsiks ning arendajate teavituseks. Lisavõimalusena implementeeritakse Gridi ressursi kasutus, mis lubab testide arvu suurendada võrreldes mõne masina lokaalse testimisega.

Töö on üles ehitatud kolmest peatükist. Esimene pakub taustinformatsoonina ülevaadet tarkvara testimisest ja selle tõlgenduste erinevustest, puudutab testimise automatiseerimist ja esitab detailsema ülevaate tarkvara testimise vahenditest.

Teises peatükis – Programmipaketi DOUG testimine – on ära toodud nimetatud paketiga seotud märksõnad ja vahendid ning mitmest neist ülevaated tehtud. Kolmas peatükk kirjeldab loodud testimissüsteemi realisatsiooni.

1 Olemasolevad testimissüsteemid

1.1 Ülevaade tarkvara testimisest

Tarkvara testimine on (tarkvara)toote kasutamine eesmärgiga leida selles puuke[5]. Testimine on tarkvaraarendusprotsessi oluline osa ning tagab arendatavate lahenduste kvaliteedi ning hiljem parema juurutamise. Testida tuleb paralleelselt tarkvara arendamisega, mitte alustada alles siis, kui rakendus on juba valmis. Paralleelne testimine aitab leitud vigu kiiremini ja odavamalt parandada ning võtta neid arvesse edasises töös.

Testimise kuldreegel on, et testimine tuleb läbi viia võimalikult kiiresti ja efektiivselt [6].

1.1.1 Terminoloogia erinevused

Kuna testimine on kiiresti arenev tarkvaratehnika haru, siis on ka selle terminoloogia kiiresti arenev. Kaks põhimõistet – valideerimine ja verifitseerimine – näivad olevat ajakohasemates määratlustes küpsuse saavutanud. Tihedamini läheb lahku testimise mõiste sidumine nendega.

Kaljula [14] defineering: testimine=verifitseerimine+valideerimine, baseerudes viitele [13]. Viide [12] käsitleb põhiliselt verifitseerimist kui kõiki tegevusi, mis on ette võetud, et kindlaks teha, kas tarkvara rahuldab oma eesmärke ja testimist selle ühe osana – dünaamilise verifitseerimisena – vältides valideerimise terminit. Viites [11] räägitakse peamiselt testimisest kui valideerimise ühest osast jättes verifitseerimise mainimata. Laquso grupp [1] tõlgendab verifitseerimist ja valideerimist vastavalt kui formaalsete ning empiiriliste analüüsi meetodite, tehnikate kogumit.

1.1.2 Mõningaid tsitaate testimisest

- Effektiivne viis koodi testimiseks on kasutada seda selle loomulikel piiridel. *"An effective way to test code is to exercise it at its natural boundaries*" Brian Kernighan
- Testimine on nähtamatu ebaselgega võrdlemise protsess nii, et vältida mõeldamatu juhtumist anonüümsetega. "*Testing is the process of comparing the invisible to the ambiguous, so as to avoid the unthinkable happening to the anonymous.*" James Bach
- Testimine on organiseeritud skeptitsism."*Testing is organised skepticism.*" James Bach
- Programmi testimist saab kasutada puukide kohaloleku näitamiseks, mitte kunagi nende puudumise näitamiseks. "*Program testing can be used to show the presence of bugs, but never to show their absence!*" Dijkstra

- Hoiduge puukide eest ülemises koodis; ma olen ainult tõestanud selle korrektsust, mitte proovinud seda."*Beware of bugs in the above code; I have only proved it correct, not tried it.*" Knuth
- Tarkvara testijad: kõlbeliselt kõlvatud(rikutud) mõistused, kasulikult tööle palgatud. "*Software Testers: Depraved minds, usefully employed."* Rex Black

1.2 Tarkvara testimise automatiseerimisest

1.2.1 Mõttekus

Igat testijuhtumit ei ole otstarbekas automatiseerida. Automatiseeritud testi loomine on tavaliselt aeganõudvam kui selle jooksutamine. Ajaline võit saavutatakse testide kordamisest ja enamasti pikemas perspektiivis. Seega teste, mida teostatakse ainult mõned korrad, ei ole mõttekas automatiseerida.

On ka teste, mida ei ole võimalik automatiseerida selle keerukuse tõttu. See garanteerib manuaalse testimise koha testimisprotsessis ka tulevikus.

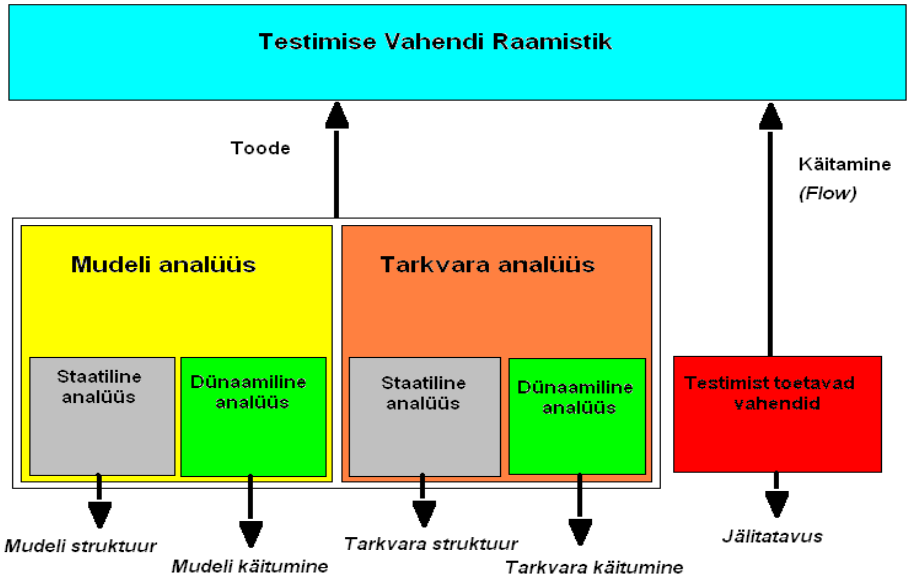
1.2.2 Kasulikkus

Automatiseeritud tarkvara testimine aitab parandada testijate produktiivsust ja vähendada vajaminevaid ressursse. Oma loomult suurendab automatiseeritud tarkvara testimine koodikaetuse taset, vähendab testimisaega ja maksumust[7].

1.3 Testimisvahendite klassifikatsioon

Valideerimis- ja verifitseerimisvahendid on laialt saadaval. Enamik vahenditest toetavad valideerimist või testimist. Harvemini saavad vahendid täheldada midagi toote kvaliteedi kohta. Verifitseerimisvahendeid leidub vähemal määral. Klassifikatsioon baseerub viitel [1] ja sõltub seal kasutatavast terminoloogiast, mida on mainitud 1.1.1 alapunktis.

Testimisvahendi raamistik joonisel 1, milles vahendid on klassifitseeritud, näitab kahte klassi: vahendid, mis toetavad toote kvaliteeti ja vahendid, mis toetavad toote käitamist (ingl. k. *flow*) [1]. Reeglina on nende kahe klassi vahendite kasutamise vahetalaks juurutamisprotsess või otsus. Järgnevalt jagunevad vahendid, mis toetavad toote kvaliteeti, mudeli analüüsi ja tarkvara analüüsi vahenditeks ja mõlemad omakorda staatilise ja dünaamilise analüüsi vahenditeks. Nagu nimigi ütleb, teostab mudeli analüüs analüüsi tarkvara mudelitel, mis võimaldab testimisega alustada tarkvara arenduse varasemates etappides ja vahendid on seega paremini integreerutavad arenduse tehnikatesse, mis püüavad testimisega alustada võimalikult vara. Mudeli analüüs on olemuselt formaalsem kui tarkvara analüüs. Viimase põhjuseks, miks seda varasemates etappides rakendada ei saa, ja eelduseks on, et mingi osa tarkvarast peab olema implementeeritud.



Joonis 1. Testimisvahendite klassifikatsioon.

Staatilise analüüsi vahendid teostavad analüüsi vastavalt mudeli struktuuril ja tarkvara struktuuril, st. lähtekoodil või spetsifikatsioonil. Dünaamilise analüüsi vahendid omakorda mudeli või tarkvara käitumisel.

Antud klassifitseerimisele võrdlusena on viites [11] testimisvahendid klassifitseeritud kahte rühma vastavalt sellele, millist analüüsi nad teostavad: staatilise ja dünaamilise analüüsi vahendid, kusjuures testimist toetavaid vahendeid ei loeta testimisvahendite hulka. Neid määratletakse kui grupp testimisega seotud vahendid, mis ei teosta otseseid teste ega kasuta ühtegi otsest testimise tehnikat. Antud viites mudeleid ei mainita.

1.4 Testimisvahendite kategoriseerimine

Kategooriad on esitatud toetudes viitele [1] ja sama viite eespool mainitud klassifikatsioonile. Samuti on viites ära toodud mõned olemasolevad vahendid, mis on peamiselt arendatud akadeemilistes ringkondades.
Mudeli analüüsi vahendid hõlmavad mõlema alajaotuse, nii staatilise kui dünaamilise mudeli analüüsi vahendeid.

Mudeli analüüsi vahendid:

- Petri võrgu analüüsi vahendid – raamistik paralleelsete ja hajussüsteemide analüüsimiseks ja konstrueerimiseks. Vahendid võimaldavad Petri võrgu redigeerimist, simuleerimist ja analüüsimist.
- Töövoo analüüsi vahendid – äriprotsesside salvestamiseks, analüüsimiseks ja haldamiseks
- Arhitektuuri analüüsi vahendid – tarkvarasüsteemi analüüsimiseks ja kirjeldamiseks, sisendiks enamasti UML mudel.
- Mudeli meetrika genereerimise vahendid
- Reegli kontrolli vahendid

- Andmeanalüüsi vahendid – analüüsides andmemudelit võimaldavad leida vastuolusid ja peidetud kitsendusi.
- Teoreemi tõestajad – abistavad matemaatiliste teoreemide tõestamisel või teooriate disainimisel.
- Mudeli kontrollijad – verifitseerib kõik võimalikud sooritused disainil baseeruval lihtsustatud mudelil.
- Testijuhtumi generaatorid – pakuvad automaatse protsessi, mille sisenditeks on formaalne tarkvara mudel ja testijuhtumite genereerimise direktiivid ning väljundiks testijuhtumite kogum, mis sisaldab stiimulite järjekorda testitavale süsteemile ja mudeli ennustatud oodatavaid vastuseid nendele stiimulitele[10].

Allpool kirjeldatud staatilise ja dünaamilise testimise vahendid kuuluvad tarkvara analüüsi klassi.

Staatilise testimise vahendid:

- Struktuuri analüüsi vahendid – pakuvad (visuaalset) lähtekoodi tõlgendust või struktuuri, hindavad meetrika baasil tarkvara kvaliteeti ja keerukust.
- Süntaksi analüüsi vahendid – kontrollivad üldiste programmeerimistavade rikkumisi.
- Käitumise analüüsi vahendid – püüavad avastada käitamisaegseid (ingl. k. *run-time*) vigu kompileerimise ajal.
- Väite kontrolli vahendid – väidete kontroll vastavalt väite keelele. Väide on avaldis, mis täpsustab mingi programmi muutuja tingimuse või seose. Väite avaldised lisatakse lähtekoodi kommentaaridena [11].
- Meetrika genereerimise vahendid
- *Refactoring tools* – rakendatakse eelprotsessina programmi funktsionaalsuse laiendamisele, kuna parandab tarkvara laiendatavust, või järelprotsessina programmi puhastamiseks peale uue funktsionaalsuse implementeerimist.

Dünaamilise testimise vahendid:

- Koormustesti vahendid – süsteemi käitumine koormuse all. Aitab leida pudelikaelu; oluline mitmekasutaja süsteemide testimisel. Koormustestimisega on väga lähedalt seotud stresstestimine, kus koormus tõstetakse üle normide, ebatavaliselt kõrgele. Selliste testide oodatavad tulemused on erindolukorrad [9].
- Monitooringu vahendid – võimaldavad kindlaks teha süsteemi tervise olukorda ja identifitseerida potensiaalseid probleeme enne vea esinemist; vea esinemisel aitavad ka teavitada administraatorit.
- Silujad – võimaldavad defekte lokaliseerida ja eemaldada; samuti inspekteerida käitamisseisundit sümboolsel viisil, valida käitamislaadi (samm-sammult, katkestuspunktide abil) [12].
- Testi kaetuse vahendid – võimaldavad määratleda testi efektiivsust: kui palju lähtekoodist mingi test katab või läbib? Kaetust mõõdetakse mitmel viisil: lause- e. reakaetus, haru-, tingimus- ja teekaetus [5].
- Draiver/tupik (*stub*) vahendid – kasutatakse moodultestimisel asendades väljaarendamata mooduleid, mis on vajalikud mingi mooduli testiks [11].
- Väljundi võrdlejad(*comparators*) – võrdlevad testi väljundit oodatava väljundiga.
- Makro salvestamise/taasesituse vahendid – salvestavad testijuhtumeid jäljendades inimtestija tegevusi; taasesitus viiakse läbi saates rakendusele

salvestatud nupuvajutuste ja hiire liikumise jms sündmusi ja nii kontrollides rakenduse käitust.
- Andmebaasi manipulatsiooni vahendid – võimaldavad andmeid kopeerida, võrrelda, muundada, redigeerida; samuti kujutada ja visualiseerida andmete struktuuri andmebaasis.
- Veebi testimise vahendid – sisuliselt kõik tarkvara analüüsi vahendid, mille testitavaks süsteemiks on veebisüsteem.

Testimistoe vahendeid üldiselt eraldiseisvatena pole; kuuluvad testimissüsteemide koosseisu.

Testimistoe vahendid:
- Defekti haldamise vahendid – aitavad defekte koguda, salvestada, organiseerida, sisaldavad infot defekti taasesituseks, haldavad nimekirja leitud, parandatud defektidest.
- Testi disaini vahendid
- Planeerimise vahendid
- Nõuete haldamise vahendid – võimaldavad salvestada ja hallata nõudeid; analüüsida, kategoriseerida, muutusi jälitada ja nende mõju analüüsida. Parandavad süsteemi spetsifikatsiooni kvaliteeti.
- Konfiguratsiooni haldamise vahendid – võimaldavad salvestada tarkvara erinevaid konfiguratsioone; jälgida, esitada, võrrelda muutusi erinevate konfiguratsioonide vahel.

Kui antud kategoriseerimine on tuntud pigem akadeemilistes keskkondades, siis kommertsmaailmas on levinud mõnevõrra erinev jaotus:

- Lähtekoodil opereerivad vahendid
- Funktsionaalsuse testimise vahendid
- Koormustestimise vahendid
- Andmebaasi testimise vahendid
- Veebi testimise vahendid
- Vea jälituse (ingl. k. *bug tracking*) vahendid

Mudeli analüüsi vahendeid mainitakse vähem, sest need esindavad formaalsemaid testimise tehnikaid, mille rakendamine organisatsioonides nõuab suuremaid kulutusi.

1.5 Testimisvahendite võrdlus

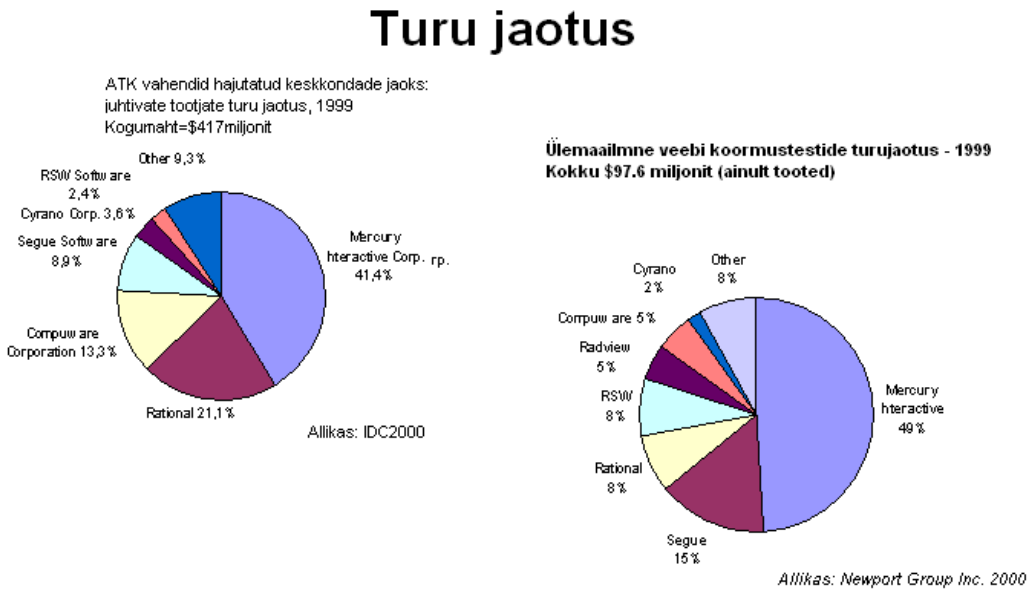
Igaüks, kes on kaalunud testimisvahendi kasutamist, on kiirelt mõistnud rohket mitmekülgset valikut turul, nii vahendite sortimendis kui ka tarnijate arvus. Parim vahend ükskõik millisele üksikasjalikule situatsioonile sõltub rakendatud süsteemitehnika keskkonnast ja kasutatavast testimismetoodikast, mis omakorda dikteerib, kuidas automatiseerimist rakendatakse protsessi toetuseks.[3]

Järgnev lõik annab juhiseid, et aidata testimise professionaalidel valida, mõista ja implementeerida testimisvahendeid organisatsioonides ning baseerub viitel [2]. Kuigi võrdlus on koostatud aastal 2001, on see teistega võrreldes kirjeldavam, võrdlevam ja kokkuvõtlikum. Tehnilisem võrdlusinformatsioon on saadaval viites [3] ja [15] (koos hindadega).

Tänu suurele arvule vahenditele on järgitud Pareto 80/20 printsiipi ja võrdluse baasiks valitud populaarseimad kommertsvarad. Baas sisaldab järgmisi rakendusi:

- Rational Robot (sisaldub Rational TestStudios)
- Rational Visual Test 6.5
- Mercury WinRunner 7
- Segue SilkTest 5
- Compuware QARun 4.7.1 (sisaldub QACenter'is)

1.5.1 Turg ja automaatne testimine



Joonis 2. Turuosa jaotused; vasakul automatiseeritud tarkvara kvaliteedi vahendite joatus turul tarnijate järgi ja paremal veebikeskkondade koormustesti vahendite jaotus tarnijate kaupa.

Jooniselt selgub, et populaarseimad on Mercury vahendid hajusate keskkondade ja veebikeskkondade testimiseks omades turuosasid ligikaudu vastavalt 40 ja 50 protsenti.

Enamik hinnatud vahenditest pakuvad sama funktsionaalsust erineval viisil. Paljud tarnijatest pakuvad väljalaskeid vähemalt 2 korda aastas koos veaparandustega. Turg pole veel lõppfaasis, vaid kasvav, ning prognoositakse rohkem ühinemisi.

1.5.2 Testimisvahendid

1.5.2.1 Funktsionaalsete testimisvahendite maatriks

Tabelis 1 esitatud vahendite maatriks pakub kiiret ja lihtsat viidet testimisvahendite võimalustest. Igale kategooriale maatriksis on antud hinnang ühest viieni:
1 = suurepärane tugi antud funktsionaalsusele,

2 = hea tugi, aga esinevad puudused või muu vahend pakub efektiivsemat tuge,
3 = põhiline tugi ainult,
4 = kõne all olev funktsionaalsus on ainult toetatud API vahendusel või kolmanda osapoole lisana, kuid ei sisaldu üldises vahendis/allpool keskmist,
5 = tugi puudub.

Valikukriteeria kogumiku võib üles ehitada kasutades seda maatriksit ja arvtulemust abistamaks vahendi hindamisprotsessi. Üldiselt mida madalam on arvtulemus, seda parem, kuid see on subjektiivne ja põhineb võrdluse autori[2] ja teiste professionaalide arvamustel, mida on kasutatud selle võrdluse loomisel.

	Salvestus&Taasesitus	Veebi testimine	Andmebaasi testid	Abdmefunktsioonid	Objekti kaardistus	Kujutise testimine	Test/Vea taastatavus	Objekti nime kaart	Objekti identiteet	Laiendatav keel	Keskkonna tugi	Terviklikkus	Maksumus	Kasutamise kergus	Tugi	Objekti testid
WinRunner	2	1	1	2	1	1	2	1	2	2	1	1	3	2	1	1
QA Run	1	2	1	2	1	1	2	2	1	2	2	1	2	2	2	1
Silk Test	1	2	1	2	1	1	1	1	2	1	2	3	3	3	2	1
Visual Test	3	3	4	3	2	2	2	4	1	2	3	2	1	3	2	2
Robot	1	2	1	1	1	1	2	4	1	1	2	1	2	1	2	1

Tabel 1. Testimisvahendite võrdluse maatriks.

Maatriksi arvtulemused:

- WinRunner = 24
- QARun = 25
- SilkTest = 24
- Visual Test = 39
- Robot = 24

1.5.3 Kokkuvõte

Tugevused/Nõrkused:

Rational
Tugevused: ökonoomne, palju toetavaid vahendeid, hea laiendatav keel(VisualBasic baasil), soodsad andmeloome tingimused, hea online kogukond.
Nõrkused: tugi, keel on vaikeväärtustena natuke algeline ja segane.

Mercury
Tugevused: arvatavasti kõige populaarsem testimisvahend, küllaldaselt töid, hea tugi, hea online kogukond, hea brauserite vaheline tugi.
Nõrkused: "karbist võetuna" pole Java toetust, kaldub kallima lahenduse poole.

Compuware
Tugevused: tugevad professionaalsed teenused, hea integratsioon arenduse toodetega, ökonoomne.
Nõrkused: vaene online kogukond, toote kinnijooksmine.

Segue
Tugevused: hea arenduskeel, hea online kogukond, taastesüsteem.
Nõrkused: helpdesk, kaldub kallima lahenduse poole.

1.6 Testimisvahendi valik ja hindamine

Järgnevalt on esitatud viieastmeline protsess organisatsioonile sobiva testimisvahendi võrdlemiseks, hindamiseks ja lõpuks valimiseks. Baseerub viitel [4].

Samm 1: Defineeri algnõuded

Enne testimisvahendi otsimist on tähtis luua nimekiri algnõuetest. Mis probleemid vahend aitab lahendada? Mis võimalusi vahend vajab, et olla efektiivne organisatsiooni keskkonnas? Milliseid kitsendusi tööriist peab rahuldama?

Ühilduvusprobleemid

Iga vahend, mis valitakse, peab ühilduma loodavate rakenduste toetatavate operatsioonisüsteemidega, rakenduste loomisel kasutatavate arenduskeskkondadega ja kolmanda osapoole tarkvaraga, millega loodav rakendus integreerub.

Vahendi audients

Algnõuete nimekirja defineerimisel tuleb silmas pidada vahendi igapäevaseid kasutajaid, võimalikku aja ja energia investeerimist personali.

Eelarve piirangud

Enne ostmist peab määrama eelarve vahendi jaoks. Maksumus on tõenäoliselt tähtis tegur ja sellel on suur mõju vahendi valikul. Kui pannakse paika algne automaatse testimisvahendi eelarve, tuleks arvestada pakkumise võimalikku ülemmäära ja tegelikku väärtust. Need sõltuvad organisatsiooni suurusest ja juba olemasolevatest vahenditest. Tuleb arvestada organisatsiooni loodetav kasu vahendist ja määratleda, kas see on mõõdetav.

Arvestades vahendi kasu ja maksumust, on võimalik arutada esialgset investeeringu tasuvuse(ROI – *return on investment*) projektsiooni.
Samuti peab määrama, kuidas organisatsioon mõõdab edu.

Äri nõuded

Suure organisatsiooni või reguleeritud tööstuse puhul võib vaja minna täiendavaid nõudeid, et organisatsioon kvalifitseeruks kui "heakskiidetud varustaja".

Samm 2: Valikute uuring

Uurides valikuid tuleks võrgu otsingud levitada nii kaugele ja laialt kui võimalik. Allikad: uudisgrupid, veebilehed, konverentsid, ajakirjad, kaastöötajad, toodete tarnijad. Sellel etapil tuleb suhelda vahendi müügiinimestega ja varustada neid valikukriteeriumi infoga. Suheldes müügipersonaliga:

- Kuidas on võimalik võrrelda seda vahendit turul?
- Millistel juhtudel on see vahend parim valik?
- Millistel juhtudel ei ole see vahend parim valik?
- Millised omadused eristavad seda vahendit võistlevatest?
- Mis teile ei meeldi selle vahendi puhul?

Sel tasemel võivad müügiesindajad survet avaldada. Praeguse sammu ajal kogutakse informatsiooni, millise toote demo on mõtet sisse tuua.

Samm 3: Nõuete täpsustamine

Selle sammu jooksul lisatakse nõuded, mis avastati turu uuringus või ei teatud nendest varem või unustati. Tuleb suhelda kõigi inimestega, keda vahendi implementatsioon mõjutab. See annab neile võimaluse arutada muresid ja alusetuid arvamusi.

Sel etapil võib välja tulla eelarve ebareaalsus – vahendid, mis rahuldavad funktsionaalseid nõudeid ei pruugi rahuldada eelarve nõudeid. Sel juhul tuleb suurendada eelarvet või vähendada funktsionaalseid nõudeid.

Samm 4: Nimekirja ahendamine

On võimalik, et turul on ainult üks toode, mis rahuldab püstitatud nõudeid. Sel juhul on praegune samm kerge. Tõenäoliselt on nimekirja jäänud 4 või 5 reaalset võimalust ja mõned osalised lahendused.

Kasutades nõuete nimekirja tuleb reastada uuritud vahendid. Ülemised 2 või 3 vahendit saavad olema lõplikud kandidaadid, mida hakatakse uurima käsitsi. Kuna käsitsi hindamine on aeganõudev, siis sellest tuleneb ka piiratud valik.

Samm 5: Finalistide hindamine

Esimene samm finalistide hindamisel on tegevuste kogumiku identifitseerimine. Valida paar testi, mis esitavad huvipakkuvaid väljakutseid vahenditele. Järgmisena kontakteeruda iga vahendi müügiesindajaga ja hankida hindamiskoopiad. Iga vahendit tuleks kasutada vähemalt nädal, soovitavalt kuu. Mõned pakkujad nõuavad, et üks inimene nende personalist võtaks osa hindamisest. Hindamise ajal on mõttekas modelleerida kogu protsess, mida hindamisgrupp kavatseb järgida. See tähendab, et GUI automatiseerimise tööriista jaoks:

- Loo test
- Lisa test lähtekoodi kontrolli
- Jooksuta testi
- Hinda tulemusi
- Oleta, et kasutajaliides muutus ja test vajab vastavat modifitseerimist

Kujutle eelmisi tegevusi mitte ühe testi, vaid sadade testide korral.

Peale otsust

Viimaste sammudena tuleb reserveerida treeningu aeg ja suhelda selgelt ja sageli inimestega organisatsioonis, keda mõjutab uus vahend. Soovitav on implementeerida sammhaaval ning võtta kasutusele abinõud vahendi kasutuse edukuse mõõtmiseks.

1.7 Vabavara vahendid

Vabavara vahendite kvaliteet ja funktsionaalsus kõigub laialt. Tihti on limiteeritud toetatava platvormi või keele seisukohast või hoopis aegunud. Enamasti on platvormi tugi vabavaralistele Linux platvormidele. Vabavaraline vahend toetab üldjuhul ühte populaarsematest keeltest , milleks on Java või C++. Enamus vabavara vahendid on arendatud uurimisprojektide raames ülikooli kogukondades või üksikute rohkem või vähem professionaalide poolt. Akadeemiliste ringkondade arendatud vahendid kalduvad olema spetsiifilise funktsionaalsusega. Piltlikult väljendades – kui kommertsvahendid püüavad funktsionaalsust arendada laiuti, siis teaduslik arendus sügavuti.

Kategooria, millel on rohkeim vabavara valikuid, on vea jälituse vahendid (D. Faught 2001).

2 Programmipaketi DOUG testimine

2.1 Programmipakett DOUG

Paljudes teadusarvutuslikes ning insenertehnilistes ülesannetes tuleb lahendada lineaarvõrrandite süsteeme, mis on oma mahult tihti suuremad kui ühte arvutisse või ajapiiridesse on võimalik paigutada. Selliste ülesannete lahendamiseks on loodud programmipakett DOUG.

DOUG (*Domain decomposition On Unstructured Grids*) pakett realiseerib piirkondadeks jagamise meetodid (*Domain Decomposition Methods*) ebakorrapäraste piirkondade korral suurte lineaarsete võrrandisüsteemide lahendamiseks hajussüsteemides. Tegemist on automaatselt paralleliseeruva programmiga, kus algülesanne jaotatakse eri protsessorite vahel ära ning rakendatakse iteratiivseid meetodeid, kus kiirema koondumise huvides kasutatakse igal alampiirkonnal oma lahendajat. Paralleliseerimine on implementeeritud läbi sõnumiedastusteegi (*Message Passing Interface*, MPI). Programmipaketis DOUG on uudne ka võimalus lahendada ülesandeid, mille puhul maatriks omab blokk-struktuuri – sellised süsteemid on raskesti lahenduvad[28][33].

DOUG on porditud kaasaegsesse objekt-orienteeritud programmeerimiskeskkonda, kasutades Fortran95 moodulite kontseptsiooni.

Iseseisva rakendusena on DOUG paketile realiseeritud tõrkekindel kommunikatsioonimudel, mis on võimeline täielikult taastama väljalangenud arvutuslikke sõlmi uutel protsessoritel [28].

DOUGi arendus toimub hajusalt asetsevate arendusmeeskondade ühisel panusel ja koostöö optimiseerimiseks kasutatakse Subversioni. Tartu Ülikoolis toimub DOUGi arendus TÜ Suuremahuliste paralleelarvutuste ja GRIDi uurimisgrupi poolt prof. E. Vainikko juhendamisel. Samas uurib grupp ka GRIDi rakendusvõimalusi, sh. DOUGi ühildamist GRID keskkonnaga.

Järgnevates alapunktides esitatakse ülevaated programmipaketiga DOUG seotud rakendustest.

2.2 Versioonikontrolli süsteem

Versioonikontroll on informatsiooni muutuste haldamise kunst. See on kriitiline vahend programmeerijatele, kes kulutavad palju aega koodi muutustega toimetamiseks. Versioonikontrolli tarkvara ületab kaugelt tarkvaraarenduse maailma. Kus iganes kasutatakse arvutit sageli muutuva informatsiooni haldamiseks, seal on ruumi versioonikontrolli vahendile. [29]

Versioonikontroll salvestab kõik muutused, mis on tehtud mingile failide kogumikule aja jooksul; olenevalt süsteemist ka kaustade struktuuri muutused. See võimaldab turvaliselt taastada andmete vanemaid versioone ja uurida andmete muutuste ajalugu.

Enam kasutavad süsteemi tarkvara arenduse meeskonnad, mis lubab neil koordineerida arendajate individuaalseid muudatusi andmetes – vahendid konfliktsete muudatuste lahendamiseks ja ühendamiseks. Enamlevinud süsteemid omavad ka võrgust juurdepääsu vahendeid. Kaks eelmist omadust teevad võimalikuks arendustöö suurtes ja hajusates gruppides. Laiemalt kasutatavad ja omaksvõetud vahendid on CVS, Subversion, GNU Arch, Bazaar, Arx ja CVSNT [9].

Failid andmetega salvestatakse spetsiaalses kogumikus, mida nimetatakse hoidlaks (*repository*). See salvestab kõik muudatused failide ja struktuuriga. Hoidla võib baseeruda kas andmebaasil või kõvakettal. Hoidla seisukohalt kasutatakse kahte fundamentaalset versioonikontrolli süsteemi disaini: tsentraliseeritud ja detsentraliseeritud e. hajus. Tsentraliseeritud (CVS, Subversion) vahendi puhul hoitakse andmed ühes keskses hoidlas (võib omada backup'e), mis sarnaneb eriomadustega failiserveriga. Hoidlasse muudetud andmete salvestamiseks antakse arendajatele autentimistunnused ja iga arendaja vastutab enda tehtud muudatuste eest ise. Detsentraliseeritud (GNU Arch, Bazaar, Arx) disaini korral saavad arendajad ise omale hoidla kloonida mingist olemasolevast ja isiklikud muudatused kloonitud hoidlasse salvestada. Samuti on võimalik muudatusi hoidlate vahel levitada. Et arendaja muudatused programmi põhiliinis avalduksid, peab peahoidla hooldaja need sinna liitma. Lühidalt saab öelda, et Arch on tsentraliseeritud koodi integreerija ümber ja Subversion (ja CVS) on tsentraliseeritud hoidla ümber [30].

Hoidlas võib eksisteerida mitmeid paralleelseid arendusliine, mida nimetatakse harudeks (*branch*). See teeb võimalikuks hoida põhi arendusharu stabiilsena, samal ajal töötades uue kallal või luua uus haru mingi katselise omaduse (*feature*) väljatöötamiseks [31].

Versioonikontrolli süsteem võimaldab kasutajatel märgistada väljavõtteid haru mingist hetkeolukorrast lihtsustades hilisemaid teatud osa eraldusi harust. Väljavõtteid nimetatakse *tag*'ideks [31].

2.2.1 CVS

CVS, Concurrent Versions System, on hetkel populaarseim avatud lähtekoodiga versioonikontrolli süsteem. Oma täbarate puuduste tõttu – muutusi arvestatakse faili seisukohalt, mitte muutuse enda seisukohalt; andmete sisestused hoidlasse pole atomaarsed; failide ja kaustade ümbernimetamised on kohmakad; harundamise (*branching*) piirangud – hakkab vananemise märke näitama. CVS süsteemi kasutavad sellised populaarsed avatud lähtekoodi projektid nagu Mozilla, Gimp, XEmacs, KDE ja GNOME. Mõned originaalse CVS'i hooldajad on deklareerinud, et CVS'i kood on muutunud liiga konarlikuks, et seda efektiivselt säilitada. Need probleemid viisid Subversioni sünnini [30][9].

2.2.2 Subversion

Subversion on vabavaraline avatud lähtekoodiga versioonikontrolli süsteem. See on mõeldud välja vahetamaks CVS'i. Subversion on põhiliselt uuesti-teostatud CVS

põhinedes samuti tsentraliseeritud disainil. Töötab peamiselt samal viisil ja elimineerib CVS'i suuremad puudused.

Subversion võimaldab ligipääsu hoidlale ka võrgu vahendusel, mis võimaldab seda kasutada erinevatel inimestel erinevate arvutite taga. Juurdepääs võrgust on realiseeritud klient/server mudelil. Võimalus muuta ja hallata sama andmekogumit mitmetel inimestel oma vaatepunktist soosib koostööd. Progress saab toimuda kiiremini ilma ühe kindla juhtkanalita, mille kõik muutused peavad läbima. Kuna tehtud töö on versioonidesse jagatud, ei pea muretsema selle üle, et kvaliteet on kompromiss sellisele juhtkanalile – kui mingi ebaõige muudatus on andmetes tehtud, saab alati selle tagasi võtta.

Mõned versioonikontrolli süsteemid on ka tarkvara konfiguratsiooni haldamise süsteemid (*software configuration management*, SCM). Need süsteemid on spetsiaalselt kohandatud, et hallata lähtekoodi kaustade puid ja omavad palju omadusi, mis on spetsiifilised tarkvara arendusele. Autorite arvates aga pole Subversion sellist tüüpi süsteem, vaid üldine vahend, millega on võimalik hallata ükskõik millist failide kogumit. Mõne jaoks võivad need failid olla lähtekood, teiste jaoks kõik alates toidukaupade ostunimekirjast lõpetades digitaalse video seguni ja edasi. Tarkvara arenduse kogukondades klassifitseeritakse Subversion kui SCM süsteem.

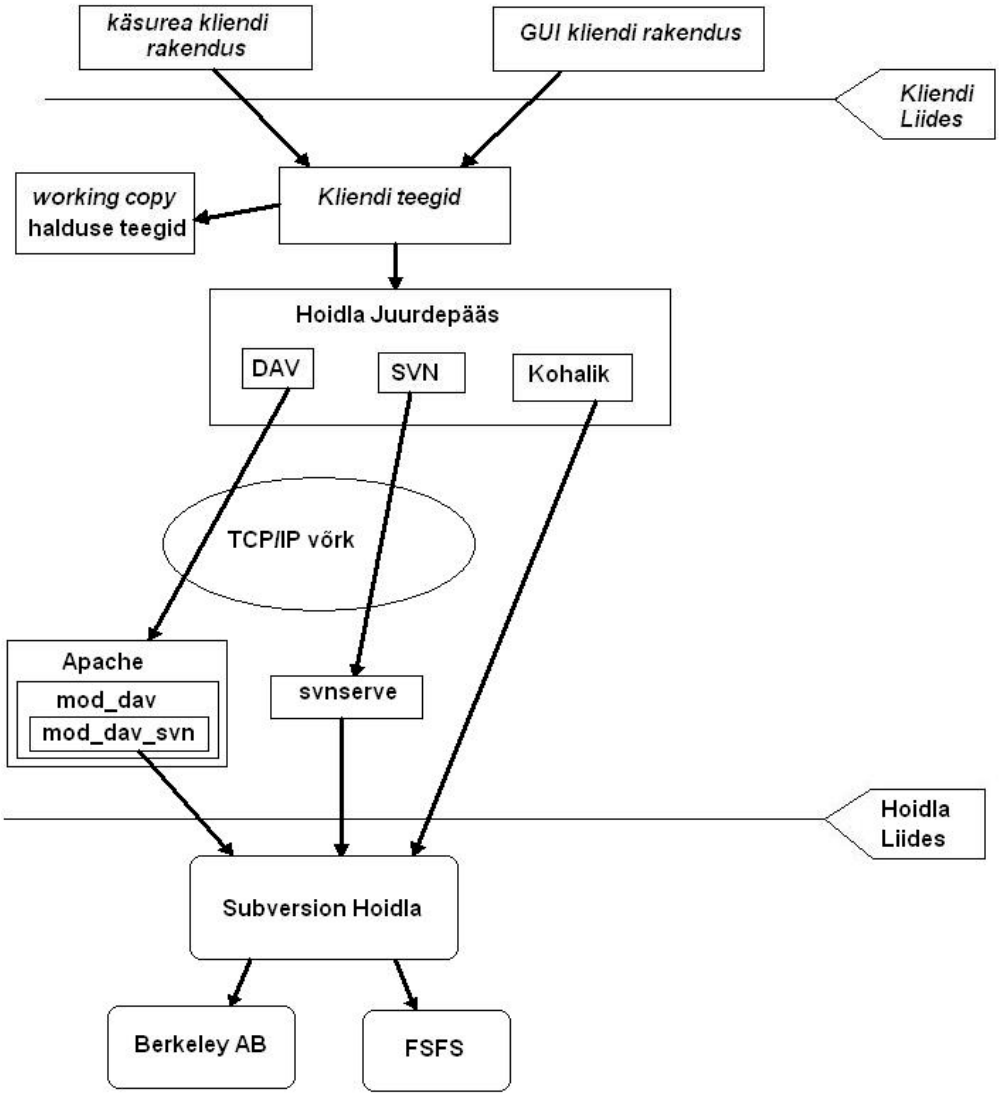
2.2.2.1 Subversioni arhitektuur

Joonis 3 illustreerib Subversioni arhitektuuri. Ühes otsas asub hoidla, mis hoiab kõiki versioonidesse seatud andmeid. Teises otsas aga klientprogramm, mis haldab kohalikke peegeldusi nendest andmetest, nn töökoopiaid (*working copy*). Nende kahe äärmuse vahel on mitu teed läbi Hoidla Juurdepääsu (*Repository Access*, RA) kihtide. Mõned nendest teedest lähevad risti läbi võrkude ja võrguserverite, mis omavad juurdepääsu hoidlale. Teised jätavad võrgu üldse kõrvale ja sisenevad hoidlasse otse [29].

2.3 GRID

Grid on termin, mis tähistab arvutuste ja andmete haldamise infrastruktuuri – geograafiliselt eri kohtades paiknevad arvutid, superarvutid ja spetsiaalseadmed (andmehoidlad, sensorid jms) moodustavad ühtselt vaadeldava ressursi nii, et süsteemi kasutaja ei pea mõistma, kus täpselt tema arvutus- ja andmeanalüüsi ülesandeid lahendatakse ning kuidas täpselt toimub andmete haldamine [16]. Lühidalt, kui veeb on teenus informatsiooni jagamiseks üle Interneti, siis Grid on teenus arvutusjõudluse ja andmemahu jagamiseks üle Interneti [17].

Gridi peamised eesmärgid on suure jõudluse pakkumine ressursinõudlikele töödele ning erinevate arvutite, arvutuskeskuste, andmehoidlate ja spetsiaalseadmete ühendamine. Tehes seda nii, et nende kasutamine oleks lihtne ja mugav ka ilma sügavate teadmisteta infotehnoloogias, super- ja hajusarvutustes [16].



Joonis 3. Subversioni arhitektuur.

Idee sai alguse 1970ndate algul ARPANET'is mõttest ära kasutada arvutite vabu tsükleid. Arenduse põhjuseks on üha suurenev nõudlus arvutusjõudluse ja andmemahu järele.

Superarvutitega võrreldes on GRID odavam ja paindlikum viis geograafiliselt hajusalt asetsevate teadusgruppide(ka kommertsorganisatsioonide) suurte arvutuste ning ressursside vajaduste rahuldamiseks. Ideoloogia kohaselt kasutatakse odavaid personaalarvutite kogumeid ehitusblokkidena, mis ühendatakse Interneti teel läbi nn. Gridi vahevara ja saadakse selliselt "virtuaalne superarvuti". Sügavam ülevaade Gridi ideoloogiast ja tehnoloogiast on antud Berman, Fox, Hey(2003)[6] ja Foster, Kesselman(2003)[5] raamatutes.

Nüüdisaegsetes terminites, Grid on standardiseeritud tarkvara kiht (*layer*) kasutajate ja Gridi ressursside vahel. Grid vahevara on komponent, mis seob igat tüüpi ressursid (erinevad riistvara arhitektuurid, operatsioonisüsteemid jne) ühtseks süsteemiks standardiseeritud vahenditega. Selle jõud peitub paralleliseerimises ja massiivses arvutusülesannete, nn. Gridi tööde (Grid jobs), sooritamises. See võimaldab kasutajatel luua arvutustöö, jagada see nii mitmeks iseseisvaks alamtööks

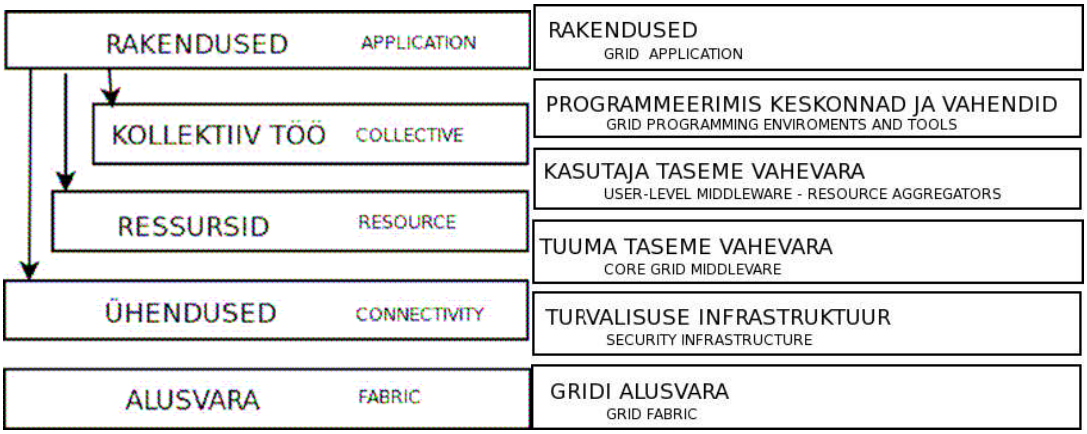
ja seejärel saata need Gridi. Lõpuks kui tööd on sooritatud, alla laadida tulemused ja analüüsida neid [18].

Tüüpiline juhtum teaduslikus uurimuses – liiga palju inimesi arendavad liiga palju võrdselt arukaid lahendusi. Üle maailma on käimas sadu erinevaid GRIDi projekte, mis viivad tehnoloogiliste lähenemiste mitmekesisusele; näiteks UNICORE, Globus, Legion, Gridbus jne. Ühest küljest on positiivne näha mitmeid loovaid lähenemisi võistluskeskkonnas, teisest küljest ootavad tööstus ja uurimuse kasutajad standardeid, et alustada programmide kirjutamist [19]. Sellega seoses loodi 1990ndate lõpus Global Grid Forum (GGF) [22], mille ülesandeks on Gridi ühtsete standardite ja protokollide väljatöötamine. Praeguseks arendussuunaks on Gridi vahevara, täpsemalt Globuse funktsionaalsuse, ühtlustamine veebiteenuste (*Web Services*) WSDL (*Web Services Description Language*) kontseptsiooniga. Selle tulemusena on arendatud Gridi teenuste arhitektuur OGSA (*Open Grid Services Architecture*) [23].

Rüdiger Berlich [19] hinnangul on enimkasutatud vahevara Globus Toolkit, arendatud Globus projekti raames, arvestades tollaseid(juuli 2004) installeerimisarve, ja CERN (*Centre Europeenne pour la Recherche Nucleaire*) Gridi arvutuste üks edasiviivaid jõude [19].

2.3.1 Gridi arhitektuur

Grid koosneb erinevatest komponentidest – alates põhiressurssidest kuni lõppkasutaja rakendustarkvarani välja, mida tihti kujutatakse kihtidena *(layers)*. Joonisel 4 on välja toodud kaks levinumat kihistruktuuri. Vasakul pool on Foster, Kesselman ja Tuecke nägemus Gridi koostisosadest. Paremal oleva 6-kihilise interpretatsiooni autorid on Rajkumar Buyya ja Madhu Chetty.



Joonis 4. Vasakul on Foster, Kesselman ja Tuecke 5-kihiline Gridi struktuur ning paremal Buyya ja Chetty 6-kihiline[23].

Ülaltoodud joonisel olevad struktuurid on loodud fundamentaalsete süsteemi koostisosade identifitseerimiseks, funktsionaalsuse ja eesmärkide spetsifitseerimiseks ning komponentide omavaheliste interaktsioonide selgitamiseks. Ülesehituse vaatenurk on seatud virtuaalsete organisatsioonide (VO) vastastikuse koostöövõime tagamisele. Koostöö tähendab eelkõige ühiseid protokolle, sest Gridi arhitektuur on peamiselt protokollide arhitektuur. See on tarkvaraline ehituskunst, kus defineeritakse

VO kasutajate ja ressursi vahelised loomis-, haldamis- ning kasutamisseoste mehhanismid.

VO moodustavad indiviidid või organisatsioonid, kes jagavad oma ressursse (tarkvara, arvuteid, andmeid). VO abil toimub juurdepääsude andmine ressurssidele. Kes, mis, millal ja kuidas saab ressursile juurdepääsu, määrab omanik[23].

Täielikult funktsionaalne Grid vajab nelja komponenti(loetletud GGF'is)[18]:

- *Cluster Element* (CE) – tegelik sõlm/kobar, milles Gridi tööd sooritatakse. Selle eesliides (*front-end*) on ühendatud Gridi, kus ta aktsepteerib Gridi töid ja sooritab need kohalike töödena. Kui töö on lõpetatud, tagastab väljundid täpsustatud viisil.
- *Storage Element* (SE) – salvestuselemendi sõlm, millele on lisatud mingi andmemahu ressurss ja on kättesaadav Gridi kasutusel.
- QDRL(*Quasy Dynamic Resource Locator*), tuntud ka kui GIIS(*Grid Index Information Service*) – selle ülesandeks on Gridi ressursside informatsiooni levitamine üle terve võrgu. CE ja SE registreerivad oma ressursi informatsiooni kohalikus GRIS'is (*Grid Resource Information System*), mis asub tüüpiliselt nendelsamadel CE ja SE sõlmedel. GRIS registreerib oma aadressi lähimas QDRL'is, mis seejärel levitab antud GRIS'i asukohta oma ahelat pidi üles.
- UI (*User Interface*) – käsurea või graafiline kasutajaliides, mis võimaldab kasutajatel sisestada uusi Gridi töid, jälgida olemasolevaid, tagastada väljundeid, lõpetada käimasolevaid töid jne.

2.3.2 Eesti Grid

Eesti Grid on pikaajaline projekt Eestis asuvate suuremate arvutusvõimsuste sidumiseks ühtsesse süsteemi ning tulemi integreerimiseks sarnaste rahvusvaheliste projektidega, et tagada Eesti teadusele juurdepääs jagatud probleemilahendusele ja globaalsetele andmebaasidele [32].

Eesti GRID projekti juhtivaks arendajaks ja administreerijaks on haridus- ja teadusvõrke haldav EENet. Aastal 2004 loodi huvitatud asutuste esindajatest algatusgrupp, millest koos lisanduvate asutustega projekti käivitumisel moodustub Eesti Gridi konsortsium. Algatusgruppi kuuluvad:

- Tartu Ülikool (TÜ),
- Tallinna Tehnikaülikool (TTÜ),
- Eesti Hariduse ja Teaduse Andmesidevõrk (EENet),
- Keemilise ja Bioloogilise Füüsika Instituut (KBFI, i.k. *National Institute of Chemical Physics and Biophysics, NICPB*),
- Eesti Biokeskus,
- Microlink AS,
- EGeen AS,
- BioData OÜ.

Organisatsiooniline struktuur on avatud kõikidele teadus- ja arenduskeskustele, sealhulgas ka ettevõtetele.

"Eesti Grid" integreerub Euroopa GRID projektidesse ja arendab sellealast rahvusvahelist koostööd ning haakub projekti "Eesti edu 2014" ideekavandis väljatoodud mõtetega. EENet ja KBFI kuuluvad BalticGridi projekti[24] partnerite hulka ning esimene neist kuulub ka EGEE[25] mittelepinguliste partnerite, st. partnerite, kes ei saa (rahalist)toetust Euroopa Liidult, aga on huvitatud tööprogrammist ja osalevad mõnes EGEE tegevuses, sekka.

Eesti Gridi projekti loomisel otsustati kasutada, järgides CERN'i ja naaberriikide tendentsi, vahevara, mis baseeruks *Globus Toolkit*'il. Valiti NorduGrid projekti raames arendatud ARC (*Advance Resource Connector*) vahevara. Põhjuseks eelnimetatud tendets ning piisav toetus ja funktsionaalsus. 2006. a. algul mindi üle LCG-2 vahevarale seoses koostööga BalticGridi ja EGEE projektides.

2.3.3 LCG-2

LCG sündis vajadusest valmistada ette arvutusinfrastruktuur LHC (*Large Hadron Collider*) eksperimentide andmete simulatsiooniks, töötlemiseks ja analüüsiks. LHC-d konstrueeritakse CERN'is ja aastaks 2007 saab sellest maailma suurim ja jõudsaim osakeste kiirendi. LHC andmete käsitsuse vajadused arvutusjõudluse, andmemahu, andmete juurdepääsu tagamise suhtes on väga suured(15 PetaBaiti andmeid aastas). Kaalutlused näitavad, et kõikide ressursside rahastamine ühes asukohas selliste vajaduste rahuldamiseks ei ole teostatav ja nõustuti, et LCG arvutusteenus implementeeritakse geograafiliselt hajutatud Arvutusliku Andme Grid'ina (*Computational Data Grid*).

LCG projekt on ka üks kahest pilootrakendusest, mis on valitud areneva Euroopa Gridi infrastruktuuri implementatsiooni juhtimiseks ning jõudluse ja funktsionaalsuse kinnitamiseks. Seda teostatakse EGEE projekti raames.

LCG-2 vahevara tuleneb mitmetest Gridi arendusprojektidest nagu DataGrid, DataTag, Globus, GriPhyN, iVDGL ja EGEE projekt.

Enamikele LCG-2 pakutavatele teenustele on juurdepääs läbi liideste, käsurea liidese(*Command Line Interface*, CLI) ja graafilise kasutajaliidese (*Graphical User Interface*, GUI), või rakenduste vahendusel, mis kasutavad erinevaid API-sid [26].

2.3.3.1 LCG-2 arhitektuur

LCG on geograafiliselt hajusalt asetsevate ressursside ja teenuste kollektsioon, mis on organiseeritud Virtuaalsetesse Organisatsioonidesse (*Virtual Organisaton*, VO). LCG sisaldab töökoormuse haldamise süsteemi (*Workload Management System*, WMS), andme haldamise süsteemi (*Data Management System*, DMS), informatsiooni süsteemi (*Information System*, IS), autoriseerimise ja autentimise süsteemi, arvepidamissüsteemi, erinevaid järelvaatamise (*monitoring*) teenuseid ja installeerimisteenuseid.

WMS on vastutav kasutajate sisestatud tööde haldamise eest. Ta seab töö nõuded vastavusse kättesaadavate ressurssidega ja planeerib/lisab järjekorda töö sooritamise sobival klastril. Seejärel jälitab töö staatust ja võimaldab kasutajal omandada töö lõppedes selle väljundit.

DMS võimaldab kasutajatel faile lokaliseerida, kopeerida mitmete asukohtade vahel ning liigutada neid sisse ja välja Grid'ist. See on saavutatud andmete ülekandmisega erinevate protokollide vahendusel (enamkasutatud on GridFTP) ja koostöös tsentraalse failikataloogiga (*Replica Location Service*, RLS).

IS varustab informatsiooniga LCG-2 ressursside ja nende staatuste kohta. Informatsioon antakse välja individuaalsete ressursside poolt ja kopeeritakse tsentraalsetesse andmebaasidesse. Seda kasutab WMS, et seada ressursse vastavusse tööde nõuetega ja klassifitseerida ressursse. Informatsiooni kasutavad veel jälgimisteenused ning DMS, et valida salvestusressursse.

Autentimis- ja autoriseerimissüsteem sisaldab nimekirja VO-desse jagatud inimestest, kes on autoriseeritud LCG-2 kasutamiseks. Nimekiri on tavaliselt igasse masinasse, mis pakub Gridi teenuseid, alla laaditud, et kaardistada LCG-2 kasutajad kohalike masina kasutajatega.

Lisana on LCG-2's mitmeid jälgimissüsteeme: GridICE jälgib Gridi ressursside kasutust, R-GMA võimaldab kasutajatel omandada erinevat informatsiooni jooksvatelt töödelt ja salvestada seda relatsioonilises andmebaasis ning lõpuks jälgimissüsteemid Gridi teenuste staatuse ja funktsionaalsuse kontrolliks.

Mõningates sõlmedes on kasutusel alusvara haldamisele pühendatud teenused nagu LCFGng, et hallata kohalike Gridi teenuste installeerimist, *upgrade*'i ehk "omaduste tõstmist" ja hooldust.

Andmete ja asukohtade (*sites*) suhtes jagatakse LCG ressursid kolme kihti(*tier*): Kihi 0 keskus – eksperimentide andmete allikas, CERN; ülejäänud asukohad salvestavad ja töötlevad osa nendest andmetest. Kihi 1 keskused on asukohad, mis omavad märkimisväärset kogust salvestusmahtu kuivõrd Kihi 2 keskused võivad omada seda, kuid ei pruugi.

Põhjalikum ülevaade LCG arhitektuurist on saadaval viites [27].

3 Realisatsioon

Antud peatükis täpsustatakse loodava testimisvahendi nõuded, kirjeldatakse disain ja programmiline lahendus. Vahendi arendus toimub iteratiivse tarkvaraarenduse kontseptsioonil ja kirjeldus kasutab UML komponente. Testimisvahend integreeritakse programmipaketi DOUG arendusse.

3.1 Nõuete spetsifikatsioon

3.1.1 Sissejuhatus

3.1.1.1 Süsteemi eesmärk ja vajadus.

Süsteemi eesmärgiks on lihtsustada ja kiirendada programmipaketi DOUG arendust automatiseerides selle regressioontestimist. Kuna tegemist on tarkvaraga, mis opereerib mahukatel sisend- ja väljundandmetel ning nõuab suurt arvutusressurssi, seega ka aega töötamiseks, on just regressioontestimine üks aeganõudvamaid ja üksluisemaid tegevusi selle arendusel. Regressioontestimine on ühetaoliste tegevuste kordamine mitmeid kordi ja just see omadus võimaldab protsessi automatiseerimist. Hetkel toimub paketi testimine nö. käsitsi iga arendaja poolt. Kuna arendus on hajus ja koodi salvestamine toimub hoidlas, kuhu iga arendaja saab omal vastutusel muudatusi programmikoodi teha, siis loodav testimissüsteem lisab kindlust, et lisatud muudatus ei kahandanud paketi kvaliteeti ja standardsed testijuhtumid läbivad testimise.

3.1.1.2 Side organisatsiooniga

DOUG-i arendus toimub TÜ Suuremahuliste paralleelarvutuste ja GRIDi uurimisgrupi poolt Eero Vainikko juhendamisel ning loodav testimisvahend integreeritakse selle arendustegevusse. Uurimisgrupil on vaba juurdepääs Eesti Gridi ja Baltic Gridi ressurssidele, mida testimissüsteem võimalusel kasutama hakkab.

3.1.2 Nõuete analüüs

Nõuded on täpsustatud mitteformaalses stiilis ja paika pandud koostöös DOUG-i arendajatega. Üldiselt jagunevad nad funktsionaalseteks ja mittefunktsionaalseteks.

3.1.2.1 Funktsionaalsed nõuded

Üldine funktsionaalsus

Peale koodi sisestust arendaja poolt koodihoidlasse käivitatakse automaatselt testimise programm, mis laeb hoidlast alla lähtekoodi ning testib koodi kompileeritavust ja käitumist etteantud testijuhtumite korral. Tulemused salvestatakse, analüüsitakse ja nendest teavitatakse arendajaid.

Vahendit saab kasutada üldiseks ja individuaalseks testimiseks. Individuaalset testimist kasutavad üksikud arendajad koodi kompileeritavuse kontrollimiseks ja üksikute vähemnõudlike testijuhtumite käitamiseks enne muudatuse koodihoidlasse salvestamist. Selleks peab olema võimalused valida erinevaid testijuhtumeid ja vältida täpsustatud üldiste testijuhtumite käivitamist. Samuti võimalus täpsustada kompileeritava lähtekoodi asukoht kohalikus masinas ja selle põhjal testimist teha.

Üldine testimine toimub automaatselt peale koodi salvestamist koodihoidlasse. See hõlmab endas koodi allalaadimist hoidlast, selle kompileeritavuse testi ning üldisemate ja mahukamate testijuhtumite jooksutamist. Kuna üldisemad testijuhtumid on mahukad ja ajakulukad, tuleb võimalusel kasutada nendega testimisel Gridi ressurssi. See etendab kolme eesmärki: DOUG-i testimist, kaudselt GRID-i testimist ja kahe eelmise koostööd ja ühilduvust. Nende eesmärkide põhjuseks on arendajate tihe koostöö projektiga Eesti Grid.

DOUG-i lähtekood asub svn hoidlas ja tuleb implementeerida liides või moodul, mis suhtleks hoidlaga läbi svn kasutajaliidese. Autentimistunnused pole vajalikud, kuna kood on vabalt allalaetav. Neid on vaja juhul, kui salvestatakse lähtekoodi muudatusi hoidlasse, mida antud testimisvahendis pole ette nähtud.

Kompileerimise tulemusi peab hiljem olema võimalik vaadelda. Individuaalsel testimisel peab vahend soovitavalt kompileerimise vead ka ekraanile kuvama. Üldisel testimisel salvestatakse veaväljundid kas kettale või SE-sse.

Testijuhtumite läbimisel salvestatakse erinevate juhtumite väljundid ja testiprogrammi hetkeline konfiguratsioon, millega teste läbi viidi, kettale, et hiljem oleks võimalik nende andmefailide põhjal analüüse teha. Üldise testimise korral toimub võimalusel salvestus SE-sse.

Analüüse on kahte sorti: esialgne lühike kokkuvõte testimisest ja üldise testimise põhjalikum analüüs.

Esialgne analüüs sisaldab infot kompileeritavuse, testide läbimise/läbikukkumise kohta ja üldist ajakulu. Kindlasti sisaldab infot, kas testide käitamiseni üldse jõuti. Esialgset analüüsi kasutatakse individuaalsel testimisel, kus see ka ekraanile kuvatakse, ja üldisel testimisel, mille korral antud analüüs sisaldub arendajate teavituses.

Põhjalikumad analüüsid väljundfailide põhjal hõlmavad andmeid väljundite võrdluste tulemusest oodatavatega, testide ja testijuhtumite läbiviimiseks kulunud aja kohta, kokkuvõtted vigadest ja testimisparameetritest. Analüüsid on kuvatavad veebilehel.
Antud funktsionaalsus implementeeritakse sisuliselt vahendist eraldi ja sõltub vahendi väljundite formaadist.

Üldise testimise lõppedes saadetakse arendajatele E-posti teel teavitus, mis sisaldab lühikokkuvõtet testimise tulemusest. See on eriti tähtis juhul, kui testijuhtumite läbimiseni ei jõuta, kas siis operatsioonisüsteemi, võrgu- või kompileerimisvigade tõttu.

Testimine peab toimuma iga etteantud testijuhtumi korral ka erineval arvul paralleelsetel protsessidel(vt. Süsteemi nõuded). Näiteks võib ühte testijuhtumit jooksutada 3 korda: 1 protsessina, 4 paralleelse protsessina ja 9 paralleelse protsessina. Protsesside arvud on täpsustatavad testija poolt.

Testimisprogrammi lähtekood hakkab asuma svn-i hoidlas.

3.1.2.2 Mittefunktsionaalsed nõuded

Individuaalseks ja üldiseks testimiseks peab vahend olema piisavalt paindlik (*flexible*).

Vahend peab olema porditav erinevatele arhitektuuridele, kuna Gridi infrastruktuuris olevad arvutuselemendid põhinevad erinevatel platvormidel.

Graafilist kasutajaliidest pole ette nähtud, sest vahend ei ole mõeldud laiale tarbijaskonnale, vaid DOUG-i arendajatele, kes üldiselt kasutavad käsurea liidest.

Platvorm
Kindlasti peab vahend töötama Linuxi operatsioonisüsteemil, sest see ühtib arendajate kasutatava keskkonnaga. Samuti on Gridi arvutuselementide platvormid enamasti Linuxi põhised.

3.1.2.3 Süsteemi nõuded

Testitavaks süsteemiks on programmipakett DOUG. Pakett vajab käimiseks ja kompileerumiseks erinevaid keskkondi ja lisateeke. Kõige tähtsamaks süsteemi nõudeks on LAM MPI keskkonna olemasolu. LAM (*Local Area Multicomputer*) on teateedastusteegi MPI (*Message Passing Interface*) standardi avatud lähtekoodiga implementatsioon paralleelsete rakenduste jaoks. See keskkond pakub mähisvahendid nagu kompilaator ja linker selles keskkonnas töötavate rakenduste kompileerimiseks. Võimaldab paralleelrakendustel käia mitmel varem täpsustatud paralleelsel protsessil.
Nõutavad teegid ja keskkonnad: lam-mpi 7.1.2, umfpack4.4, metis4.0, blas ja kompilaatoritest: intel c++ v9, intel c v9, intel f95 v9.
Paketi kompileerimise hõlbustamiseks kasutatakse GNU make vahendit.
Selleks, et oleks võimalik lähtekoodi svn hoidlast alla laadida, peab svn-i kliendiliides olema installeeritud masinas, kus testimisvahend töötab.

3.1.3 Süsteemi mudel

3.1.3.1 Kasutusmalli mudel

Joonisel 5 on toodud vahendi kasutusloo diagramm. Väliste süsteemidega on seotud 6 tegevust ja välisteks süsteemideks on 3 rolli: Arendaja, Koodihoidla ja Gridi ressurss. Järgnevalt kirjeldatakse iga rolli lähemalt.

Arendaja

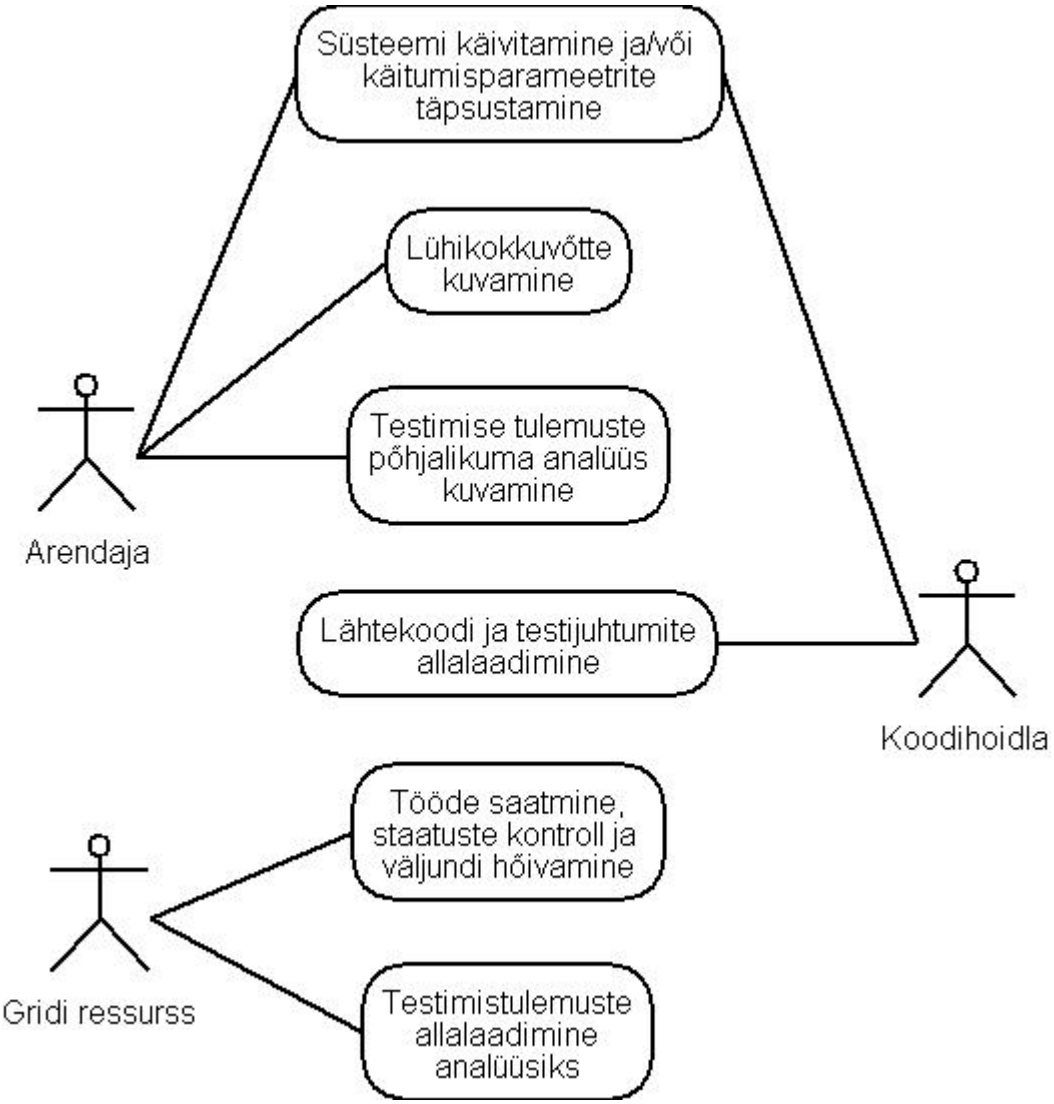
Süsteem võimaldab arendajal käivitada vahendit ja käivitusel täpsustada parameetrid, millest sõltub vahendi töö. Samuti tehakse arendajale kättesaadavaks nii testimise lühikokkuvõte kui ka põhjalikumad testimise analüüsi tulemused.

Koodihoidla

Koodihoidla käivitab otseselt ja automaatselt testimisvahendi peale lähtekoodi muutust hoidlas. Kaudselt algatab selle sündmuse arendaja. Teiseks koodihoidlaga seotud tegevuseks on lähtekoodi ja testijuhtumite andmete allalaadimine süsteemi poolt kasutades hoidla kasutajaliidest.

Gridi ressurss

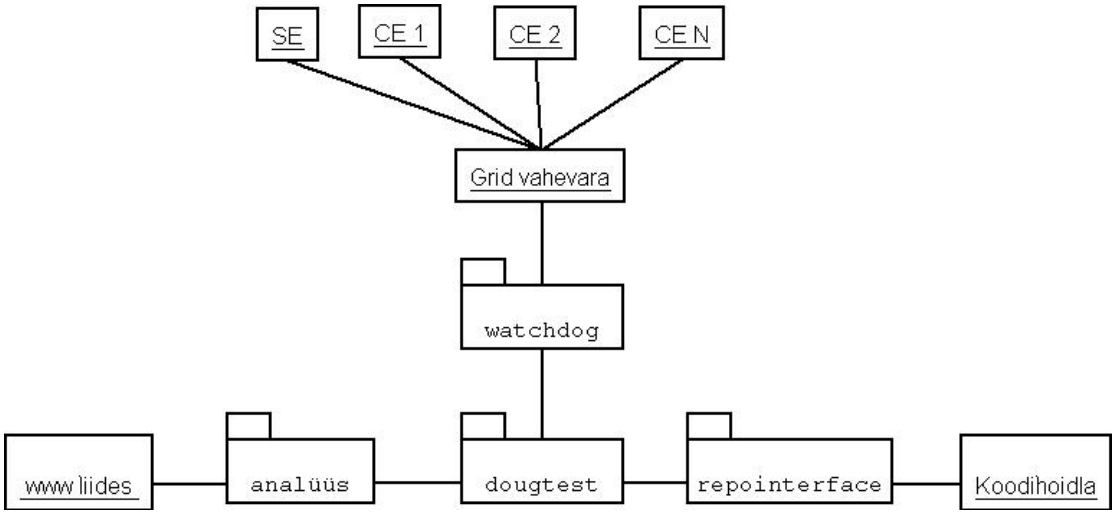
Süsteem kasutab Gridi ressurssi selliste tegevuste nagu tööde ehk testijuhtumite saatmise Gridi, tööde staatuste kontrolli ja väljundi hõivamise ning testitulemuste allalaadimine kaudu. Viimane tegevus toimub juhul, kui testitulemuste salvestamine toimub Gridi ressursil.



Joonis 5. Kasutusloo diagramm

3.1.3.2 Mudeli kirjeldus

Joonisel 6 on toodud testimissüsteemi sise- ja väliskomponendid. Süsteemi moodustavad moodulid dougtest, repointerface, watchdog, analüüs ja veebiliides. Välisteks komponentideks on Koodihoidla ja Grid vahevara. Järgnevalt nendest täpsemalt.



Joonis 6. Süsteemi sise- ja väliskomponendid

Välised komponendid

Koodihoidlas salvestatakse testitava paketi lähtekood ja testijuhtumid. See hoolitseb muutuste salvestamise eest koodis. Lähtekoodi muutusi saavad teha arendajad, kellele on antud juurdepääsuks autentimistunnused.

Grid vahevara ülesandeks on saadetud tööde vastuvõtmine ning töö määramine ja edasisaatmine sobivasse arvutuselementi(CE). See toimub juhul, kui sisestajal on luba Gridi ressursi kasutamiseks. Samuti toimub vahevara abil ühendus salvestuselemendiga (SE).

Sisesed komponendid

Moodul *dougtest* on süsteemi peamoodul, mille kaudu käivitatakse testimissüsteem. Siin toimub testimise juhtimine ja kompileerimine. Peamoodulis initsialiseeritakse moodulid watchdog ja repointerface.

Mooduli repointerface ülesandeks on suhtluse tagamine koodihoidlaga. Ta tunneb kasutatavaid protokolle ning asukoha edastab talle dougtesti moodul.

Moodul *watchdog* realiseerib suhtluse Gridi vahevaraga, milleks on LCG. Siin toimub testijuhtumite edastamine Gridi töödena ja nende jälgimine. Lühikokkuvõtte tegemiseks hõivab vastavad andmed.

Analüüsi mooduli töö toimub sisuliselt eraldiseisvana peamoodulist ja see käivitatakse veebiliideselt. Antud moodul ei suhtle otse peamooduliga, vaid kasutab selle salvestatud tulemusi.

Vahendi arendus toimub iteratiivse ja inkrementaalse tarkvaraarenduse kontseptsioonil ja on jagatud nelja etappi, millest iga etapp lisab süsteemile funktsionaalsust:

1) Vahendi põhistruktuur ja individuaalse testimise funktsionaalsus.
2) Lisatud funktsionaalsus automaatseks käivituseks peale koodi sisestust koodihoidlasse ja arendajate teavitamiseks. Samuti laetakse alla koodihoidlast paketi DOUG lähtekood. Kogu testimine toimub programmi asukoha suhtes lokaalsel masinal.
3) Lisatud funktsionaalsus üldiseks testimiseks, sh. Gridi ressursi kasutus.
4) Täiendav funktsionaalsus mahukamate analüüside teostamiseks ja veebilehel kuvamiseks, mis töötab sisuliselt põhivahendist eraldiseisvana, kuid kasutab testimisvahendi salvestatud tulemusi.

3.2 Disain

Selles peatükis esitatakse programmi disain, mis rahuldaks eelmises osas toodud nõudeid. Eesmärgiks on komponentide detailsem kirjeldus etappide kaupa.

3.2.1 Keel

Testimisvahendi implementatsiooniks on valitud programmeerimiskeel *Python* just selle porditavuse tõttu. Python on objekt-orienteeritud programmeerimiskeel, millel on hea tugi erinevate keelte ja vahenditega integreerimiseks ning on platvormist praktiliselt sõltumatu. Jookseb operatsioonisüsteemidel Windows, Linux/Unix, Mac OS X, OS/2 ja mitmetel muudel platvormidel. Eesti Gridi arvutuselementidel on kõigil olemas Pythoni teegid.

3.2.2 DOUG-i kaustade struktuur

DOUG-i kompileerimiseks ja testimiseks on vajalik teada selle kaustade ja failide struktuuri. Paketi põhikaustas testimiseks vajalikud failid ja kaustad:

- *Makefile* – *make* vahendi sisendfail, mis defineerib loogilised sammud paketi kompileerimiseks.
- *bin* kaust – kompileerimine paigutab DOUG-i käivitatavad (*executable*) failid sellesse kausta. Failinimedeks on *DOUG_main*, mis pakub paketi

põhifunktsionaalsust ja *aggr_DOUG*, mis implementeerib eksperimentaalset funktsionaalsust.

- *src* kaust – paketi lähtekood. Siin asub ka fail *Make.def*, mis täpsustab parameetrid kompileerimiseks. Iga erineva arhitektuuri jaoks tuleb täpsustada antud arhitektuurile omane *Make.def* fail.

3.2.3 Etapp 1

Esimeses etapis realiseeritakse testimisvahendi põhistruktuur ja individuaalse testimise funktsionaalsus. Sel tasemel on süsteemil 1 roll: arendaja. Süsteemi eesmärk on realiseerida Pythoni skriptis testitava paketi kompileerimine ja täpsustatud testijuhtumitel testimine. Mõlemad tegevused toimuvad arendaja kohalikul masinal. Selleks realiseeritakse osalised funktsionaalsused moodulitest *dougtest* ja *watchdog*.

3.2.3.1 Testide tulemuskaustade ülesehitus

Testitulemused salvestatakse sel etapil testimisvahendi suhtes kohalikul masinal ja samas kaustas, kus käivitatakse testimine. Salvestuskaustade struktuur jääb kõigis etappides samaks, järgmistes etappides võivad juurde lisanduda mõned andmefailid ning 3. etapis realiseeritakse võimalus andmete SE-sse salvestamiseks. Et identifitseerida erinevaid testi käivitusi, siis iga testi tulemused salvestatakse kausta, mille nimi vastab järgmisele formaadile:

YYYY-mm-dd-hh-MM-ss,

kus tähed tähendavad:

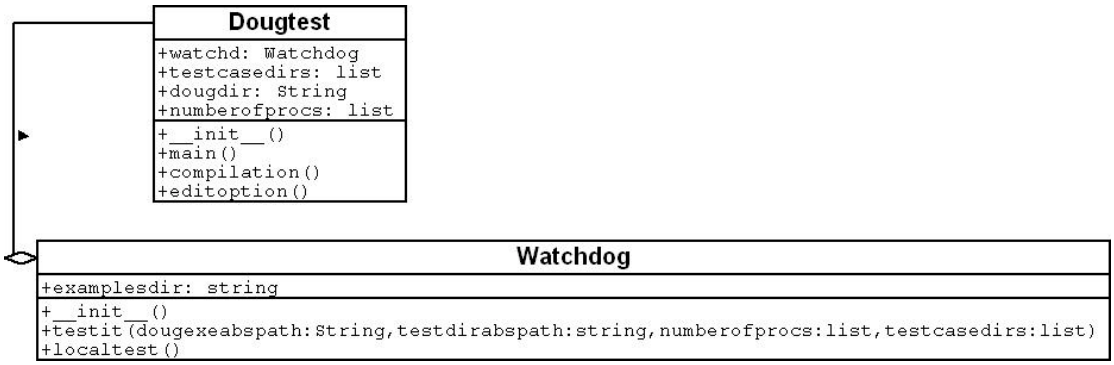
Y – aasta numbrid,
m – kuu numbrid aastas,
d – päeva numbrid kuus,
hh – tunnid ööpäevas,
MM – minutid tunnis ja
ss – sekundid minutis.

Antud tähistamisviis garanteerib testi identsuse, sest praktiliselt pole võimalik võrgu latentsuse tõttu ühes sekundis kahte koodi sisestust teha.

Järgnevalt luuakse iga testijuhtumi jaoks oma kaust nimega *testcase<n>*, kus <n> tähistab testijuhtumi järjekorranumbrit antud testis. Et eristada samade testandmete testimise väljundeid erinevate protsesside arvu korral, luuakse iga protsesside arvu jaoks omakorda kaust *proc<n>*, kus <n> tähistab kasutatud protsesside arvu. Paketi DOUG väljundiks on iga testijuhtumi korral *log.<n>* fail, milles <n> tähistab protsessi järjekorraarvu. Näiteks 3 protsessiga paketi ühe testijuhtumi testimisel tekivad kaustad *YYYY-mm-dd-hh-MM-ss/testcase1/proc3/* ja viimasesse märgitud kausta DOUG-i poolt genereeritud väljundfailid: *log.0*, *log.1* ja *log.2*. Samuti salvestatakse testi peakausta kompileerimise väljundid, LAM-MPI hetkelised keskkonna parameetrid ja testivahendi parameetrid.

3.2.3.2 Klassidiagramm

Selles etapis on realiseeritud 2 klassi: Dougtest ja Watchdog (Joonis 7). Peaklassi Dougtest käivitamisel täpsustatakse programmi parameetrid käsureal. Käsurea valikute tõlgendamiseks kasutatakse pythoni teeki *getopt*, mis võimaldab hõlpsalt käsurea valikuid objektimuutujatega siduda.



Joonis 7. Etapi 1 klassidiagramm.

Klass Dougtest omab muutujaid täpsustatud testijuhtumite kaustade kohta (*testcasedirs*), kataloogi kohta, kus asub DOUG-i kood (*dougdir*) ja numbrite listi, mis sisaldab protsesside arve, millega tuleks testid läbi viia(*numberofprocs*). Need andmed on täpsustatavad käsurealt.
Meetodid mida see klass implementeerib on: __init__() – initsialiseerimiseks, main() – peameetod, compilation() – kompileerimiseks ja edgoption() – protsesside arvu käsurealt tõlgendamiseks.

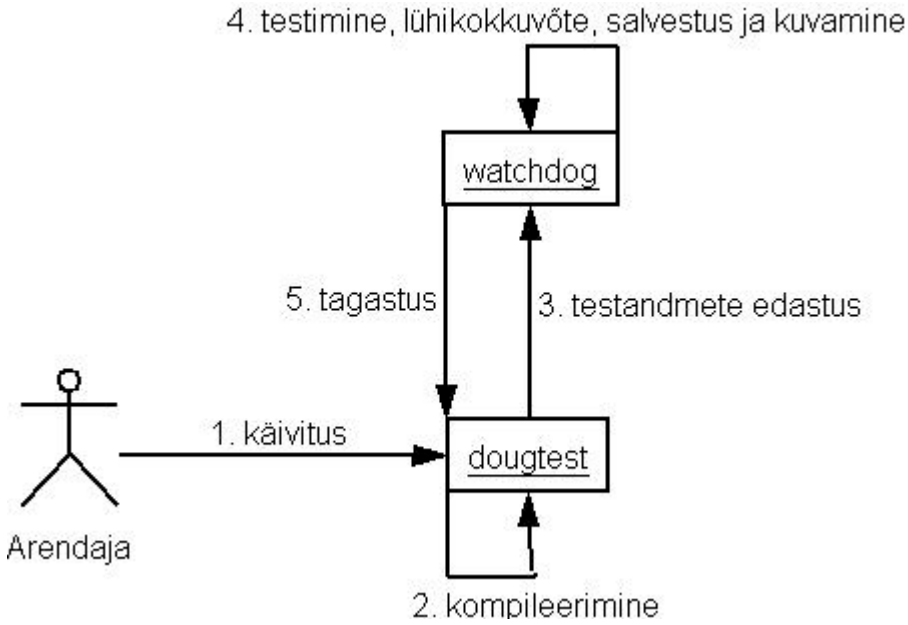
Klass *Watchdog* implementeerib testijuhtumite käitamise. Andmed testijuhtumite kohta edastatakse meetodi parameetritena. Testijuhtumid käivitatakse kohalikul masinal jadamisi. Salvestab testi väljundfailid kirjeldatud failistruktuuri ja kuvab iga testijuhtumi kohta lõpetamistulemused väljumiskoodi alusel.

3.2.3.3 Koostöödiagramm

Joonisel 8 on toodud sel etapil põhiline koostöö diagramm.

Arendaja käivitab testivahendi kutsudes välja peamooduli dougtest. Kompileerimiseks on vajalik teada paketi peakausta asukohta. Vaikeväärtusena on selleks samas kaustas olev *doug* alamkaust. Järgnevalt püüab vahend kompileerida lähtekoodi vastavalt Makefile ja Make.def failides toodud andmetele. Kui sel sammul viga tekib, siis teavitatakse kasutajat ja väljutakse. Kompileerimise õnnestumise korral edastatakse testimise andmed watchdogi moodulile, viib testid läbi ükshaaval.

Kõik mahukamad süsteemsed käsud realiseeritakse teegi *popen2* klassi *Popen3* vahendite abil. Antud klass käivitab käsud operatsioonisüsteemile alamprotsessina. Selleks, et alamprotsessi seiskumine ei mõjutaks testimisvahendi tööd, realiseeritakse alamprotsesside kasutus ja kontroll järgmiselt:



Joonis 8. Etapi 1 koostöödiagramm.

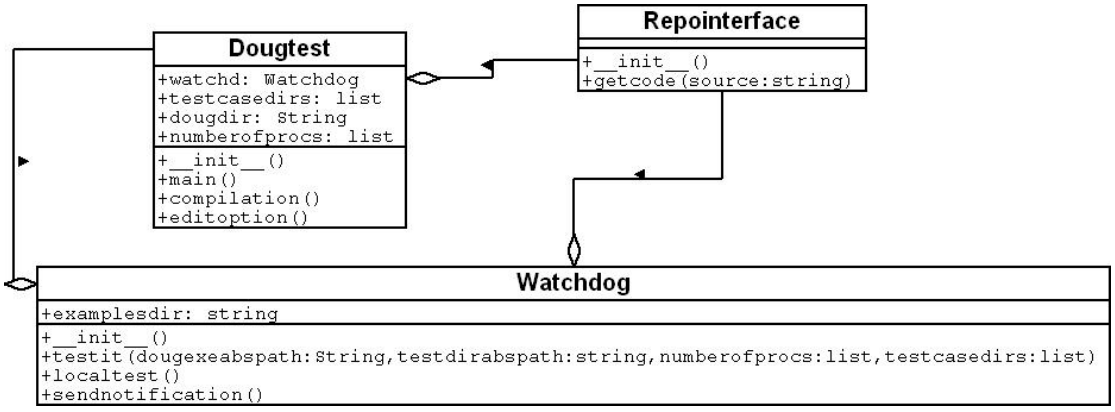
```
p=popen2.Popen3(comm)      #loob alamprotsessi, mis täidab käsku comm
processid=p.pid            #salvestab loodud protsessi id
i=timelimit                #protsessi käimise ajaline limiit
while p.poll()==-1:        #kui protsess pole veel lõpetanud
        if(i<0):           #kui i<0, järelikult antud aeg läbi
                print "Error“
                os.kill(processid, signal.SIG_DFL) #lõpeta alamprotsessi töö
                sys.exit(3)    #ja välju süsteemist(tegemist on kriitilise käsuga)
        i=i-checkinterval  #aeg ei ole veel läbi, lahuta “magamisaeg”
        sleep(checkinterval)   #peata peaprotsessi käitamine etteantud ajaks
```

3.2.4 Etapp 2

Teises etapis lisandub funktsionaalsus, mis võimaldab teste automaatselt käivitada peale koodi muutust koodihoidlas. Lisatakse moodul *repointerface*, mis sisaldab klassi *Repointerface* realiseerides meetodi DOUG-i lähtekoodi allalaadimiseks.

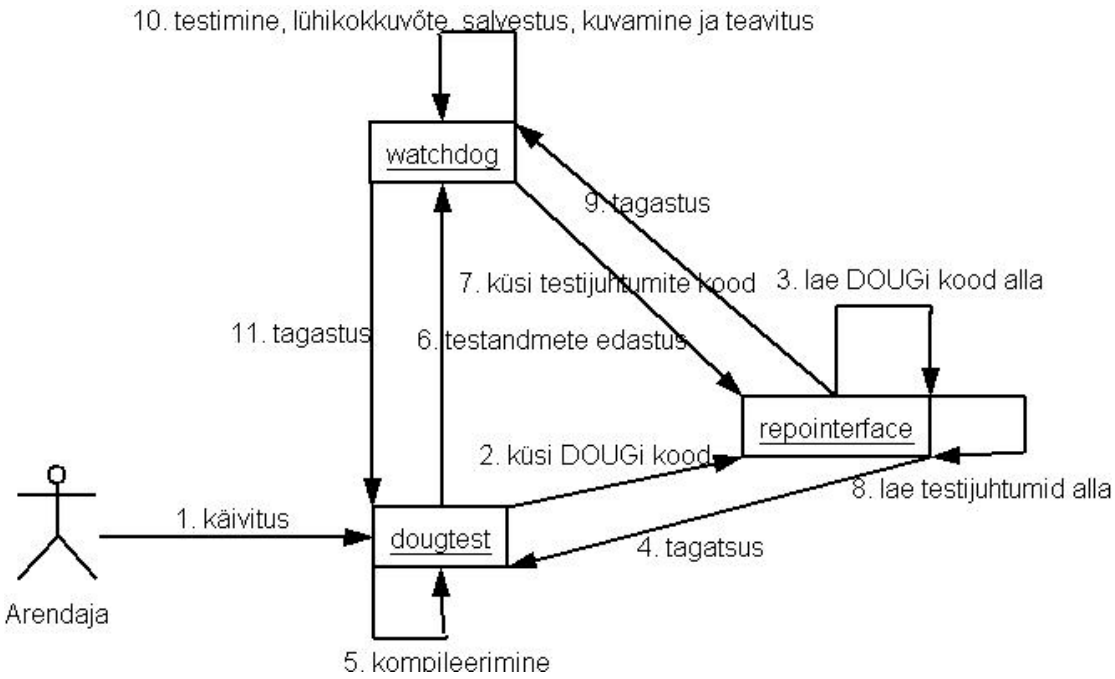
3.2.4.1 Klassidiagramm

Nagu Joonisel 9 on näha, on lisandunud *Repointerface* klass meetodiga *getcode()*. Meetod saab parameetrina kaasa koodi asukoha, mis tuleb alla laadida. Meetodi väljakutse toimub Dougtesti klassist DOUG-i koodi allalaadimiseks ja Watchdogi klassist testijuhtumite allalaadimiseks. Samuti nagu esimeses etapis, toimub ka siin kompileerimine ja testimine lokaalsel masinal ning kasutatakse sama kaustastruktuuri. Meetod *sendnotification()* kasutab pythoni e-posti teeke arendajate teavitamiseks testide läbimisest ja teavitus sisaldab endas lühikokkuvõtet testimisest.



Joonis 9. Etapi 2 klassidiagramm.

3.2.4.2 Koostöödiagramm



Joonis 10. Etapi 2 koostöödiagramm.

Joonisel 10 on näidatud etapi 2 realisatsiooni peamine koostöödiagramm. Eelmise etapiga võrreldes on lisandunud 6 tegevust, mis on kõik seotud lisatud mooduli *repointerface*'ga. Samuti on lisandunud tegevus *teavitus* mooduli *watchdog*i tegevuste hulka.

Kokkuvõte

Tarkvara mahu ja funktsionaalsuse suurenemisel suureneb ka regressioontestimise testide hulk. Mingil hetkel kasvab testide hulk nii suureks, et testimist ei ole enam võimalik manuaalselt läbi viia ega testandmeid manuaalselt läbi vaadata – selleks kuluv aeg läheb lihtsalt liiga pikaks. Seetõttu on ainus praktiline viis regressioontestimist hallata see automatiseerida.

Töö eesmärgiks oligi programmipaketi DOUG regressioontestimise automatiseerimine ja lisavõimalusena kasutada Gridi ressurssi testide läbiviimiseks. Esitati ülevaated üldisest testimisest ja pikemalt olemasolevatest testimisvahenditest, tehti kokkuvõte erinevate allikate testimisega seotud mõistete erinevustest.

Tänapäeval on testimisvahendite valik väga lai, nii sortimendilt kui pakkujate arvult. Kommertstooted pakuvad küll laia funktsionaalsust, kuid see eest on mõeldud tarkvaraarendusorganisatsioonidele integreerimiseks tarkvaraarendusprotsessi ning hinnalt kallid. Esitati populaarsemate kommertstoodete võrdlus, mis näitas põhiliselt võrdseid üldtulemusi. Vabavaraliste testimisvahendite funktsionaalsus on veel liialt limiteeritud või realiseeritud mingi kindla keele põhiselt.

Testimisega seotud mõistete erinevused näitasid, et testimine metoodikana ei ole veel oma küpsuse taset saavutanud ning areng toimub.

Teises peatükis kirjeldati programmipaketti DOUG kui testitavat süsteemi ja sellega seotud tarkvarakomponente.

Lõpuks esitati loodava testimisvahendi realisatsioon: nõuete spetsifikatsioon ja disain. Esimeses täpsustati põhiliselt vahendi funktsionaalsed, mittefunktsionaalsed ja süsteemi nõuded ning visualiseeriti süsteemi üldised moodulid ja seotud väliskomponendid. Implementeerimine jagati nelja etappi. Disaini alapunktis kirjeldati vahendi detailsem ülesehitus etappide kaupa kasutades UML komponente. Vahendi kahe esimese etapi funktsionaalsus ka kodeeriti, mis tähendab automaatset koodihoidla poolt käivitatavat süsteemi testijuhtumite jooksutamiseks lokaalsel masinal.

Edasiarendusena jätkub 3. ja 4. etapi realiseerimine, milleni antud töö maht ei ulatunud. Need hõlmavad testijuhtumite saatmist Gridi ressursile, võimalikku andmete salvestust Gridi SE-s ning vahendit mahukamaks analüüsiks. Koodi kompileerimiseks arvutussõlmel peab iga arhitektuuri jaoks olema kirjeldatud vastav *Make.def* fail, mis võib kujuneda mahukaks ülesandeks. Põhjalikuma analüüsi kuvamiseks tuleb üles seada veebileht.

Software test tools in GRID

Bachelor thesis
Anti Kivi

Abstract

When software capacity and functoinality increases number of tests cases in regression testing increases as well. At some point the number of test cases grows so large that it is not possible to manually test and review outputs – time spent will be too long. Therefore the only way to manage regression testing is to automate it.

The goal of this research was to automate regression testing of DOUG software package and as an extra possibility to use GRID resources to execute tests. Overviews of general testing and existing test tools were given, summary was made of different sources' definitions related to testing.

Nowadays test tools selection is very wide. Commercial products offer wide functionality, but are meant to be used by software development companys for integrating to a software development process and they are expensive. Comparison of most popular commercial products showed that general results are basically equal. Free tools' functionality is yet too limited or implemented to support only some specific language.

The differences of testing related definitions showed that testing as a methodology is not yet achieved its maturity level and is evolving.

In the second chapter DOUG were described as system under testing and were given descriptions of its related components.

Finally test tool implementation for DOUG were presented: requirements specification and design. In the former one were specified main functional, non-functional and system requirements and visualized system modules and external components. Implementation of software were divided into four phases. In design section were described more detailed architecture phase by phase using UML components. Tool's two first phases were coded which means that were produced system that is automatically executable by repository and testcases are executed locally.

As a future development implementation of 3rd and 4th phases will continue. This is the point where given research capacity didnt reach. Future developments include sending testcases to Grid resource, possible data saving to SE and tool for analysis. For DOUG code compilation on CE there must be a *Make.def* file for every architecture. This may be a difficult task. For printing more essential analysis there is need for a website.

Viited*

[1] LaQuSo – joint activity of Technische Universiteit Eindhoven and Radboud University Nijmegen.
http://www.laquso.com.

[2] R. Robinson. *Automation test tools.* 2001.
http://www.aptest.com/pub/toolcomparison.doc.

[3] E. Dustin. *Automated Testing Tool Evaluation Matrix.* Quality Web Systems, Addison Wesley, lk. 243-381, 2001.
http://www.stickyminds.com.

[4] E. Hendrickson. *Evaluating Tools*, 1999.
http://www.stickyminds.com/.

[5] H. Nestra. *Tarkvaratehnika loengumaterjal*, 2002.
http://www.cs.ut.ee/~kiho/TVTkonspekt/HNestraTestimine.

[6] T. Sildeberg. *Tarkvara testimine ja süsteemide monitooring*. Arvutimaailm, 2003.
http://vana.am.ee/7017.

[7] J. Heyes. *Software Engineering Handbook*. Boca Raton, 874 lk., 2002.

[8] B. Boehm. *Object-Oriented and Classical Software Engineering*, 1984.

[9] Wikipedia
http://www.wikipedia.org.

[10] A. Hartman, AGEDIS Consortium. *Model Based Test Generation Tools*.
http://www.agedis.de/documents/ModelBasedTestGenerationTools_cs.pdf.

[11] R. A. DeMillo, W. M. McCracken, R. J. Martin, J. F. Passafiume. *Software Testing and Evaluation*. The Benjamin/Cummings Publishing Company, California, USA, 537 lk., 1987.

[12] C. Ghezzi, M. Jazayeri, D. Mandrioli. *Fundamentals of Software Engineering*. International Second Edition, 2003.

[13] E. Kit. *Software Testing In the Real World: Improving the Process*, 1995.

[14] K. Kaljula. *Tarkvara testimine nõuete formuleerimise, analüüsi ja disaini etapis*. Magistritöö. Tartu Ülikool.
www.utlib.ee/ekollekt/diss/mag/2004/b16698368/Kaljula.pdf.

[15] AdventNet QEngine. *Product Comparison Document*.
http://www.adventnet.com/products/qengine/product-comparison.html.

[16] E. Vainikko. *GTLA loengumaterjal*.

http://math.ut.ee/~eero/GTLA/slaidid.pdf.

[17] GridCafe.
http://gridcafe.web.cern.ch/gridcafe/.

[18] A. Hektor, L. Anton, M. Kadastik, K. Skaburskas, H. Teder. *First scientific results from the Estonian Grid*. 2004.
http://grid.eenet.ee/failid/2004-11-02_TA_artikkel/0411024_TA_artikkel.pdf.

[19] R. Berlich. *Grid Basics*.
http://public.eu-egee.org/faq/grid_computing_basics.pdf.

[20] I. Foster, C. Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann, 2003.

[21] F. Berman, G. Fox, A. J. G. Hey. *Grid Computing: Making The Global Infrastructure a Reality.* John Wiley & Sons, 2003.

[22] Global Grid Forum.
http://www.ggf.org.

[23] K. Kägo. *Grid-tööde haldamise vahend bioinformaatika rakenduse näitel*. Bakalaureusetöö, 2005.
http://www.egeen.ee/u/vilo/edu/Students/Kauri_Kago/kauri_kago_bak_010605.pdf.

[24] *BalticGrid* projekt.
http://www.balticgrid.org.

[25] EGEE projekt.
http://www.eu-egee.org.

[26] A. D. Peris, P. M. Lorenzo, F. Donno, A. Sciabà, S. Campana, R. Santinelli. *LCG User Developer Guide*.
http://grid-deployment.web.cern.ch/grid-deployment/cgi-bin/index.cgi?var=eis/docs.

[27] S. Campana, M. Litmaath, A. Sciabà. *LCG-2 Middleware Overview*, 2004.
https://edms.cern.ch/file/498079//LCG-mw.pdf.

[28] Tartu Ülikooli Tehnoloogiainstituut.
http://www.tuit.ut.ee.

[29] Subversion.
http://subversion.tigris.org/.

[30] D. A. Wheeler. *Comments on Open Source Software / Free Software (OSS/FS) Software Configuration Management (SCM) Systems*, 2005.
http://www.dwheeler.com/essays/scm.html.

[31] S. Fish. *The New Breed of Version Control Systems*.

http://www.onlamp.com/pub/a/onlamp/2004/01/29/scm_overview.html.

[32] A. Aabloo, P. Burk, J. Einasto, I. Koppel, V. Kotkas, M. Kraav, E. Lippmaa, R. Rahu, M. Raidal, H. Teder, E. Vainikko, J. Vilo. *Eesti GRID*. http://grid.eenet.ee/failid/Eesti-GRID.6.04.pdf.

[33] A. Aabloo, P. Burk, I. Koppel, M. Kraav, R. Rahu, M. Raidal, H. Teder, E. Vainikko, J. Vilo. *Eesti teadusvaldkonnad, mis vajaksid GRIDi*, 2003. http://grid.eenet.ee/failid/Vajadused.v1.1.pdf.

*Kõikide viitade viimane külastusaeg in 28.05.2006.

Lisad

Lisa 1

Definitsioonid

Puuk ehk tarkvaraviga on tarkvara omadus, mida tal ei tohiks olla[5].
Verifitseerimine on protsess eesmärgiga kindlustada (tarkvara)toote vastavus spetsifikatsioonile [5].
Valideerimine on protsess eesmärgiga kindlustada (tarkvara)toote vastavus kasutaja nõudmistele [5].
Petri võrk (Petri net, Place/Transition net) – üks süsteemide matemaatilisi esitusi; modelleerimiskeel, mis graafiliselt kujutab süsteemi struktuuri suunatud kahealuselise graafina (bipartite graph). Koosneb koha sõlmedest (place node), ülemineku sõlmedest (*transition node*) ja suunatud kaartest, mis ühendavad kahte eelmist [9].
Refactoring – tehnika, mis võimaldab restruktureerida koodi distsiplineeritud viisil. Püüab parandada tarkvara loetavust, hallatavust ja laiendatavust. Aitab ka paljastada peidetud ebaõigeid kodeerimispraktikaid [11].
Draiver(Driver) – tõlge H. Nestra [5] – abikood või -vahend, mis rajab sobiva keskkonna ja kutsub välja testitava mooduli juhtides selle tööd; kasutatakse peamiselt individuaalse mooduli testimisel [11]. Alternatiivne tõlge (T. Külaots, 2002): ajur.
Tupik (stub) – tõlge H. Nestra [5] – abikood või -vahend, mis on mõeldud asendamaks moodulit või alamprogrammi, mida testitav moodul kutsub või kasutab; kasutatakse peamiselt moodultestimises [11]. Alternatiivne tõlge (T. Külaots, 2002): tüügas.
(Toote) Konfiguratsioon – identifitseerib toote komponendid ja ka täpsed komponentide versioonid [12].
Konfiguratsiooni haldus – distsipliin tarkvara arenduse koordineerimiseks ja tarkvara toodete ning komponentide muutuse ja evolutsiooni kontrollimiseks [12].
Pareto 80/20 printsiip – 80 % tagajärgedest pärinevad 20 % põhjustest [9].
OSS/FS – Open Source Software/Free Software – avatud lähtekoodiga tasuta tarkvara.
EGEE – Enabling Grid for E-sciencE – Euroopa komisjoni finantseeritav projekt, mille eesmärgiks on üles ehitada, toetudes hiljutistele grid tehnoloogia saavutustele, ja arendada gridi teenuse infrastruktuur Euroopas, mis on teadlastele kättesaadav 24 tundi ööpäevas[25].
BalticGrid (BG) – projekt, mille eesmärk on laiendada Euroopa Gridi integreerides sinna uued partnerid Balti riikidest ning arendada Gridi infrastruktuuri nendes riikides[24].
LHC – Large Hadron Collider – osakeste kiirendi, mis sondeerib sügavamale aine sisemusse kui kunagi varem. Konstrueeritakse CERN'is.
LCG – LHC Computing Project – projekt, mille eesmärk on ehitada ja ülal pidada andmehoidlat ja analüüsi infrastruktuuri kogu kõrge energia füüsika kogukonna jaoks, mis kasutab LHC-d.
CLI – Command Line Interface – käsurea liides.
GUI – Graphical User Interface – graafiline kasutajaliides.
API – Application Programming Interface – rakenduse programmeerimise liides.

Lisa2

CD/DVD struktuur

Selles alapunktis esitatakse ja kirjeldatakse antud tööga kaasas oleva CD või DVD struktuur. Programmi tööks vajalikud failid on *kood/dougtest.py*, *kood/repointerface.py* ja *kood/watchdog.py* ning testimissüsteem käivitub failist *kood/dougtest.py*. Kindlasti on vajalik osas *3.1.2.3 Süsteemi nõuded* kirjeldatud tingimuste täitmine. Infokandjal on esitatud ka kahe testi väljundi tulemused. Järgnevalt CD/DVD sisu loend:

kood – peakaust, kus asuvad testimise süsteemi käivitatavad failid ja kuhu toimub testimisandmete kohalik salvestus;

kood/dougtest.py – fail, mis esindab testimissüsteemi peamoodulit ja millest toimub programmi käivitus;

kood/repointerface.py – fail, mis on mooduli repointerface realisatsioon. Vajalik testimissüsteemi tööks ja realiseerib ühenduse koodihoidlaga;

kood/watchdog.py – fail, mis on mooduli watchdog realisatsioon. Vajalik testimissüsteemi tööks ja realiseerib testijuhtumite haldamise (käivitamine, staatuse kontroll, tulemuste haldamine) praegu kohalikul masinal;

kood/repointerface.pyc ja *kood/watchdog.pyc* – pythoni loodud kompileeritud versioonid üleval kirjeldatud failidest;

kood/doug – kaust, kus asub paketi DOUG kood. Kausta struktuur vastab hoidlas kasutatavale struktuurile; on alla laetud testimissüsteemi poolt;

kood/doug_examples – kaust, kus asub paketi testimisandmed (sisendandmefailid ja kontrollfailid); alla laetud hoidlast testimissüsteemi poolt;

kood/standard-output-example.txt ja *kood/standard-output-example_2.txt* – antud failidesse on salvestatud testimissüsteemi standardväljundid vastavalt esimese ja teise testi kohta;

kood/2006-05-29-16-46-50 ja *kood/2006-05-29-16-51-31* – testimissüsteemi poolt loodud kaustad vastavalt 1. testi ja 2. testi kohta, nagu on kirjeldatud osas 3.2.3.1;

kood/lamboot.err ja *kood/lamboot.out* – failid, kuhu on salvestatud käsu *lamboot* standardväljund ja veaväljund; *lamboot* käsku kasutatakse LAM MPI keskkonna ülesseadmiseks ja sellest sõltub DOUG-i käivitus.

Järgnevalt ühe testimise salvestatud tulemused. Vaadeldakse 1. testimise tulemust kaustas *kood/2006-05-29-16-46-50*. Selle struktuur peegeldab osas 3.2.3.1 kirjeldatud struktuuri:

kood/2006-05-29-16-46-50/dougtestmake.out ja *kood/2006-05-29-16-46-50/dougtestmake.err* – failidesse on salvestatud vastavalt kompileerimise (käsk *make*) standardväljund ja veaväljund;

kood/2006-05-29-16-46-50/laminfo.out ja *kood/2006-05-29-16-46-50/laminfo.err* – failid sisaldavad käsu *laminfo* standardväljundit ja veaväljundit; käsk *laminfo* tagastab info LAM MPI keskkonna kohta;

kood/2006-05-29-16-46-50/testcase<n> – kaustad (<n> on antud juhul arv 1 kuni 7), kuhu on salvestatud erinevate testijuhtumite väljundid; testijuhtumeid oli kokku 7;

kood/2006-05-29-16-46-50/testcase<n>/procs<m> – mingi testijuhtumi <m> protsessiga käivitamise juhtum. On näha, et käivitati 1, 4 ja 7 protsessiga, mis on testimissüsteemi vaikeväärtuseks;

kood/2006-05-29-16-46-50/testcase<n>/procs<m>/log.<k> – paketi DOUG enda väljund, <k> tähistab protsessi järjekorra numbrit; näiteks testijuhtumi 2 korral, 4 protsessiga käivitamisel tekkis vastavasse kausta DOUG-i väljundfailid *log.0*, *log.1*, *log.2* ja *log.3*. Samas kaustas olevad *log.err* ja *log.out* on vastavalt veaväljund ja standardväljund käsule *mpirun*. Käsku kasutab testimissüsteem ja käsuga käivitatakse paralleelrakendus, milleks praegusel juhul on DOUG ja märgitakse ära protsesside arv, millel soovitakse antud rakendust kasutada.