## TLÜ HAAPSALU KOLLEDŽ

Infotehnoloogia osakond

## Rene Saarsoo

# **VEEBILEHTEDE KODEERIMISPRAKTIKAD**

Bakalaureusetöö

Juhendaja: mag. Jaagup Kippar

## **TÄNUSÕNAD**

Kõigepealt sooviksin ma tänada oma juhendajat, kes andis mulle selle töö tegemisel vägagi vabad käed, kuid oli sellegipoolest alati abivalmis, kui ma tema nõuandeid vajasin. Kuid kindlasti pole võimalik üle hinnata tema rolli silmapaistva moosekandi ja rõõmsa tuju loojana pikkadel õhtutel õhtutel ühes tillukeses ent rahvarohkes saunas. See andis nii palju juurde!

Minu tänud kuuluvad ka Haapsalu Kolledži arvutipargi administraatorile Siim Kobinile, kes lubas minu käsutusse terve klassitäie arvuteid ja siis veel ühe, ning tuli läbi pimeda öö Kolledžisse, et sulgeda uks, kui mina kell 2 öösel arvutite konfigureerimise lõpetasin.

Ma sooviksin tänada ka kõiki neid, kes on loonud kogu seda vaba tarkvara, mida ma oma töö käigus pruukisin. Minu eriline tänu kuulub Gisle Aas'le, kes on Perl'i mooduli HTML::Parser autor ning Björn Höhrmann'le kelle loodud on CSS::SAC – ilma nende kahe suurepärase komponendita oleks see töö võinud jääda teostamata.

Ning viimaks tahaksin ma tänada ka Steve Ferguson'i, kes avaldas W3C CSS-i validaatori binaarversiooni just siis, kui ma seda kõige enam vajasin.

## **SISUKORD**

KASUTATAVAD LÜHENDID	6
SISSEJUHATUS	9
Töö struktuur	10
1. NÕUDED VEEBILEHTI ANALÜÜSIVALE PROGR	AMMILE12
1.1. Programmi põhiloogika	12
1.2. Nõuded HTTP protokolli käitlemisel	12
1.3. Nõuded (X)HTML-i analüüsimisel	14
1.4. Nõuded CSS-i analüüsimisel	17
1.5. Nõuded JavaScript'i analüüsimisel	23
2. VARASEMAD UURINGUD	27
2.1. Parnas, Dagfinn. (2001). How to cope with incorr	rect HTML27
2.2. Karppinen, Marko. (2002). State of the Validation	200227
2.3. Allsopp, John. (2005). Semantics in the wild	28
2.4. Saarsoo, Rene. (2005-2006). Web Standards in Es	stonia28
2.5. Web Authoring Statistics. (2005). Google Inc	29
3. VALIM	30
4. PROGRAMMI IMPLEMENTEERIMINE	33
4.1. Keele valik	33
4.2. Kasutatud moodulid	33
4.3. Kasutatud eraldiseisvad programmid	34
4.3.1. Andmebaas	34
4.3.2. Validaatorid	34
4.3.3. GNU Wget	35
4.4. Andmebaasistruktuur	
4.5. Programmi ülesehitus	37
4.6. Implementeerimise käigus ette tulnud probleemid	38
5. UURIMUSE LÄBIVIIMINE	
5.1. Valimi töötlemine	40
5.2. Arvutipargi ettevalmistamine	

5.3. Programmi jooksutamine	42
6. TULEMUSED	45
6.1. Ebasobilike lehekülgede eemaldamine valimist enne analüüsi	45
6.2. HTTP	46
6.2.1. Päised	47
6.2.1.1. Content-Type päis	48
6.2.1.2. Server päis	48
6.2.1.3. Connection päis	50
6.3. (X)HTML	50
6.3.1. Veebilehtede maht	50
6.3.2. Teksti osakaal veebilehtedel	51
6.3.3. Kommentaaride osakaal	51
6.3.4. Elemendid	52
6.3.5. Dokumenditüübid	56
6.3.6. XHTML	57
6.3.7. Kodeeringud	58
6.3.8. Koodi korrektsus	60
6.4. CSS	64
6.4.1. Atribuudid	65
6.4.1.1. Atribuut color	67
6.4.1.2. Atribuut font-size	67
6.4.1.3. Atribuut font-family	68
6.4.1.4. Atribuut text-decoration	70
6.4.1.5. Atribuut font-weight	71
6.4.1.6. Atribuut width	71
6.4.1.7. Atribuut text-align	71
6.4.2. Mõõtühikud	72
6.4.3. Värvid	73
6.4.4. Pildiformaadid	73
6.4.5. Selektorid	73
6.4.5.1. Selektorite tüübid	73

6.4.5.2. Elementide selektorid	75
6.4.5.3. Klassi selektorid.	77
6.4.5.4. ID selektorid	77
6.4.6. Ät-reeglid	80
6.4.7. Koodi korrektsus	80
6.5. JavaScript	83
6.5.1. AJAX	86
KOKKUVÕTE	87
Kokkuvõte tulemustest	87
Järeldused	88
Hinnang kasutatud metoodikale	89
ALLIKAD	90
LISA 1. ANDMEBAASISTRUKTUUR	92
LISA 2. PROGRAMMI WEBSTAT PEAMOODULI LÄHTEKOOD	96
LISA 3. CD-ROM JA VEEBIST LEITAVAD MATERJALID	112

## **KASUTATAVAD LÜHENDID**

- AJAX Asynchronous JavaScript And XML (ingl. k. "asünkroonne JavaScript ja XML") on interaktiivsete veebrirakenduste loomise tehnika. Selle eesmärk on parandada veebilehtede reageerimisaega seeläbi, et serveriga suhtlemine toimub taustal, kasutaja tegevusega samaaegselt. XML-i roll seisneb selles, et enamasti toimub serveriga suhtlus XML formaadis.
- CSS Cascading Style Sheets (ingl. k. "kaskaaduvad stiililehed" või "kaskaadlaadistik") on keel, mis eeskätt mõeldud (X)HTML vormingus dokumentide kujundamiseks.
- **DOCTYPE** *Document Type Declaration* (ingl. k. "dokumenditüübi deklaratsioon") on deklaratsioon SGML (sh. (X)HTML) faili alguses, mis näitab, millisele DTD-le antud dokument vastab.
- **DTD** *Document Type Definition* (ingl. k. "dokumenditüübi definitsioon") on kirjeldus, mis määrab SGML faili süntaksireeglid.
- **FPI** Formal Public Identifier (ingl. k. "avalik identifikaator") on DOCTYPE'i osa ja viitab DTD ametlikule nimele.
- **HTML** *Hypertext Markup Language* (ingl. k. "hüperteksti ülesmärkimise keel") on märgendkeel, milles on kirjutatud praktiliselt kõik veebilehed.
- **HTTP** *Hypertext Transfer Protocol* (ingl. k. "hüperteksti ülekandmise protokoll") on protokoll, läbi mille toimub veebis infovahetus.
- IE Microsoft Internet Explorer, Microsofti poolt toodetav ja levitatav veebilehitseja.

- **ODP** *Open Directory Project* (ingl. k. "avatud kataloogi projekt") on lingikogu, mille koostamise ja haldamisega tegelevad tuhanded vabatahtlikud moderaatorid. Ühtlasi on kõik kokkukogutud lingid allalaetavad RDF formaadis.
- **RDF** *Resource Description Framework* (ingl. k. "ressursi kirjeldamise raamistik") on perekond spetsifikatsioone metaandmete salvestamiseks kasutades XML formaati.
- **SGML** *Standard Generalized Markup Language* (ingl. k. "Standardne generaliseeritud märgendkeel") on märgendkeelte klass, millel osaliselt baseerub ka HTML.
- SI System Identifier (ingl. k. "süsteemi identifikaator") on DOCTYPE'i osa, mis määrab DTD faili aadressi (URI).
- URI Uniform Resource Identifier (ingl. k. "ühelaadne ressursi identifikaator") on identifikaator, mida internetiprotokollid kasutavad ressursile viitamiseks. URI-d on näiteks ftp://ftp.example.org, mailto:foo@example.com, isbn:0-306-40615-2.
- UTF-8 8-bitine Unicode'i kodeering ehk Unicode'i transformatsiooni formaat (*Unicode Transformation Format*). Unicode omakorda on kooditabel, mis sisaldab pea kõiki sümboleid kirjutamaks kõigis maailma keeltes.
- W3C World Wide Web Consortium (ingl. k. "Veebi konsortsium") on rahvusvaheline organisatsioon, mille eesmärgiks on juhtida veebi tema täie potensiaali suunas. W3C töötab välja erinevaid soovitusi (mitteametlikult tuntud ka kui veebistandardid) (X)HTML-i, CSS-i ja paljude-paljude teiste veebitehnoloogiate osas.
- **WDG** *Web Design Group* (ingl. k. "veebi disaini grupp") on organisatsioon, mis loodud eesmärgiga brauserist ja resolutsioonist sõltumatute kõigile kasutajagruppidele ligipääsetavate veebilehtede loomist populariseerida. Selleks tarbeks pakutakse veebilehel http://www.htmlhelp.com/ mitmesuguseid tööriistu ja õpetusi.

**XHTML** - *Extensible Hypertext Markup Language* (ingl. k. "laiendatav hüperteksti ülesmärkimise keel") on XML-i reeglitega vastavusse viidud HTML.

**XML** – *Extensible Markup Language* (ingl. k. "laiendatav märgendkeel") on range süntaksiga märgendkeelte klass, mis on ühtviisi kergesti loetav inimesele ja masinale.

#### **SISSEJUHATUS**

Viimased aastad on toonud veebis kaasa palju muutusi. Ligikaudu viie aasta jooksul on kanda kinnitanud mitmed värsked tehnoloogiad nagu RSS¹ uudisvood ja AJAX. Staatilistest veebilehed on muutunud järjest harvemaks. Ajaveebe ja vikisid on tekkinud nagu seeni pärast vihma. Turule on ilmunud koguni uued veebilehitsejad: Mozilla Firefox ning Safari.

Kuid kogu selle tehnoloogilise pillerkaari südames on endiselt "vana hea" HTML. Kõik veebilehed on kirjutatud kasutades just seda keelt (isegi kui HTML koodi ainsaks sisuks on viide pildifailile või Flash'i-rakendusele), aga HTML-i ümber on koondunud veel terve rida teisi tehnoloogiaid, mille kohta pole eriti täpselt teada, kui palju neid kasutatakse. Nende hulgas on eeskätt veebilehtede kujundamiseks mõeldud CSS, programmeerimiseks kasutatav JavaScript ning dokumendi struktuuri modifitseerimist võimaldav DOM. Kuid mitte vähem olulised pole ka lehekülgede sisse paigutatavad erinevates formaatides pildid ning Flash'i ja Java rakendused.

Välja selgitada, kui palju üht või teist tehnoloogiat veebis kasutatakse, ongi käesoleva töö esimene eesmärk. Ent veelgi suurem küsimus, kui erinevate tehnoloogiate kasutamise osakaal, on see, *kuidas* neid kasutatakse.

Kui võtta ette suvaline lehekülg ning kontrollida selle süntaksi korrektsust mõne validaatoriga, võime enam kui kindlad olla, et tulemus on negatiivne – lehekülg ei valideeru. Hoolimata mitmesuguste organisatsioonide pingutustest pole veebis *de facto* standardiks mitte HTML 4.01 või HTML 3.2 või isegi HTML 2.0, vaid hoopis vigane HTML. Kui palju (õigemini peaksime ütlema "kui vähe") korrektselt koostatud lehekülgi kõigi nende vigastega võrreldes on, me jällegi täpselt ütelda ei oska.

<sup>1</sup> Siin ja edaspidi kogu sissejuhatuse vältel on kõik tehnoloogilised lühendid jäetud lahti kirjutamata. See on seetõttu, et kõigi nende seletamine siin muudaks sissejuhatuse kordades pikemaks misläbi jutu enda mõte võib minna hoopistükkis kaotsi. Kõik lühendid on pikemalt lahti seletatud töö järgnevates peatükkides, lühiseletused on aga ära toodud leheküljel 6.

Analüüsida veebilehtedel kasutatava HTML-i ja CSS-i vastavust standarditele (ehk kui paljude lehtede HTML ja CSS valideerub), on seega käesoleva töö teine eesmärk. Ent jällegi, üksnes süntaksi korrektsuse kontroll ei anna meile päris täpselt aimu sellest, kui kõrge on lehekülje kvaliteet.

On üldteada fakt, et arvutiprogramm võib olla süntaksi poolest perfektne, aga sellegipoolest vigadest pungil – samamoodi on lugu ka veebilehtedega. Üksnes inimene on võimeline lõplikult otsustama, kas lehekülg on koostatud hästi või halvasti. Kuid süntaksi kontrolli ja inimjõul läbiviidava koodi ülevaatuse vahele jääb veel üks võte, mille alusel saab lehekülgede kvaliteeti hinnata. Selleks on heade ja halbade praktikate kasutamise mõõtmine.

Programmeerimisest on teada mitmeid halbu praktikaid, mille ohtral esinemisel koodis võib kaunis kindlalt väita, et tegu on madala kvaliteediga. Goto-lausete ja globaalsete muutujate ohter kasutamine on üldtuntud näited. Teisalt head praktikad, nagu tugev kapseldamine ning palju iseseisvalt testitavaid mooduleid, on märgid kvaliteetsest koodist.

Samamoodi ka veebilehtedega: hulk tabeleid ja pilte leheküljel viitab halva praktikana tuntud HTML-i kasutamisele kujunduskeelena. Teiselt poolt, üksnes väliste stiililehtede kasutamine ning presentatiivsete elementide vältimine, on märgid kvaliteetsest koodist.

Erinevate veebilehtede kodeerimispraktikate kasutamise uurimine on käesoleva töö kolmandaks ning ühtlasi kõige olulisemaks eesmärgiks.

### Töö struktuur

Peatükk 1 "Nõuded veebilehti analüüsivale programmile" kirjeldab nõudmisi koostatavale programmile, mida uuringu läbiviimisel kasutatakse. Ühtlasi antakse selle käigus kiire ülevaade erinevatest veebitehnoloogiatest, mida uuritakse.

Peatükk 2 "Varasemad uuringud" räägib olulisematest ja huvitavamatest uuringutest, mida on selles valdkonnas varem läbi viidud.

Peatükk 3 "Valim" käsitletab valimi koostamiseks kasutatud metoodikat.

Peatükk 4 "Programmi implementeerimine" annab ülevaade sellest, kuidas uuringu läbiviimiseks kasutatud programm reaalselt kirjutati ja milliseid probleeme selle tegevuse käigus lahendada tuli.

Peatükk 5 "Uurimuse läbiviimine" kirjeldab uurimuse enda läbiviimist hulga TLÜ Haapsalu Kolledži arvutite peal.

Peatükk 6 "Tulemused" toob lugeja ette ülevaate andmetest, mis uuringu käigus koguti. Ühtlasi on tegemist töö kõige olulisema peatükiga – neil kel kiire, soovitab autor lugemist alustada just sealt; seletused tundmatutele lühenditele leiab lugeja leheküljelt 6.

Lõpetuseks tehakse kokkuvõte uuringu olulisimatest tulemustest ja hinnatakse läbiviidud uuringu tulemuslikkust.

Tööle on lisatud ka andmebaasistruktuuri kirjeldavad joonised (lisa 1) ning loodud programmi peamooduli lähtekood (lisa 2). Töö käigus kogutud andmestik ja koostatud programm on paigutatud CD-le ja kättesaadavad läbi veebi; neid allikaid kirjeldab lisa 3.

## 1. NÕUDED VEEBILEHTI ANALÜÜSIVALE PROGRAMMILE

Selleks, et eelpool püstitatud eesmärke täita oli tarvis läbi uurida suur hulk veebilehti. Kuna veebis on miljardeid lehekülgi, siis vähegi esindusliku valimi käsitsi läbitöötlemine ei tulnud kõne alla. Oli tarvis programmi, mis talle antava lehekülgede nimekirja iseseisvalt läbi töötleks.

## 1.1. Programmi põhiloogika

Programmi põhiloogika pidi olema sarnane järgnevale pseudokoodile:

```
while ( aadress = võta_järgmine_aadress() ) {
   if ( lähtekood = lae_alla( aadress ) ) {
       tulemused = analüüsi( lähtekood );
       salvesta_andmebaasi( aadress, tulemused );
   }
   else {
       salvesta andmebaasi(
           aadress,
           VIGA_ALLALAADIMINE_EBAÕNNESTUS
      );
   }
}
```

Ehk siis, programm pidi läbi käima kõik talle ette antud aadressid, alla laadima seal asuvad veebilehed, analüüsima neid ning salevastama tulemused andmebaasi. Allalaadimise ebaõnnestumise puhul tuli salvestada üksnes veateade.

Muidugi, päris nii lihtne lõplik programm ei saanud olla, sestap ka järgnev ülevaade sellest, mida iga lehekülje puhul konkreetselt analüüsida tuli.

## 1.2. Nõuded HTTP protokolli käitlemisel

HTTP ehk *Hypertext Transfer Protocol*, inglise keeles "hüperteksti ülekandmise protokoll", on protokoll, läbi mille toimub veebis infovahetus. HTTP töötab päring-vastus stiilis. Klient saadab serverile päringu ning server vastab sellele omaltpoolt saates staatuse

rea (näiteks "HTTP/1.1 200 OK") ning selle järel päised ja vastuse sisu. Üks tüüpiline HTTP vastus on järgmine:

```
HTTP/1.1 404 Not Found
Date: Thu, 20 Apr 2006 21:19:26 GMT
Server: Apache/2.0.40 (Red Hat Linux)
Accept-Ranges: bytes
X-Powered-By: PHP/4.3.11
Connection: close
Content-Type: text/plain; charset=iso-8859-1

Error 404: Not Found
The page you requested, was not found from this server.
```

Kõigepealt tuli tuvastada, millise HTTP staatus-koodiga server päringule vastas ning siis sobivalt tegutseda. Näiteks teate peale, et lehekülg asub teisel aadressil, pidi programm proovima lehekülge alla laadida tollelt teiselt aadressilt. Vaid siis, kui päring oli edukas (koodiga 200) võis programm oma tegevust allalaetud lehekülje kallal jätkata.

Esimese tegevusena peale allalaadimist tuli salvestada informatsioon erinevate HTTP päiste kohta, mis allalaetud lehega ühes saadeti. Eriti oluline oli salvestada Content-Type nimelise päise väärtused, et oleks võimalik hiljem tuvastada, kas tegu oli üldse HTML² või XHTML³ vormingus leheküljega. HTML lehekülgi identifitseerib väärtus text/html aga mõeldev on ka text/sgml, XHTML-i jaoks on mõeldud application/xhtml+xml, aga mõeldavad veel ka application/xml ja text/xml. Muidugi, enamik lehekülgi, mis oma sisu XHTML-na serveerivad (saates sobivad päised), reeglina väljastavad hoopis HTML-i, kui nende poole just sobivalt ei pöörduta ja esmajärjekorras XHTML-i ei küsita. W3C⁴ soovitab XHTML-i serveerimisel HTTP spetsifikatsioonis kirjeldatud *Content-Negotiation* 

<sup>2</sup> *Hypertext Markup Language* (ingl. k. "hüperteksti ülesmärkimise keel") – selles märgendkeeles on kirjutatud praktiliselt kõik veebilehed. <a href="http://www.w3.org/MarkUp/">http://www.w3.org/MarkUp/</a>

<sup>3</sup> Extensible Hypertext Markup Language (ingl. k. "laiendatav hüperteksti ülesmärkimise keel") – XML-i reeglitega vastavusse viidud HTML. XML (Extensible Markup Language) on range süntaksiga märgendkeel, mis on ühtviisi kergesti loetav inimesele ja masinale, erinevalt näiteks SGML-ist (Standard Generalized Markup Language), millel baseerub HTML, mis on samuti inimese poolt kergesti loetav, ent masina jaoks küllaltki keerukas.

W3C ehk *World Wide Web Consortium* on rahvusvaheline organisatsioon, mille eesmärgiks on juhtida veebi tema täie potensiaali suunas. W3C töötab välja erinevaid soovitusi (mitteametlikult tuntud ka kui veebistandardid) (X)HTML-i, XML-i ja paljude-paljude teiste tehnoloogiate osas. <a href="http://www.w3.org/">http://www.w3.org/</a>

tehnikat, pakkumaks brauseritele, mis application/xhtml+xml tüübist aru ei saa sisu text/html kujul (Hazaël-Massieux 2003). Seetõttu pidi programm alustuseks omalt poolt päringut esitades panema kaasa sobiva Accept-päise, mis nägi välja järgmine:

```
Accept: application/xhtml+xml;q=0.9, application/xml;q=0.8, text/html;q=0.7, text/plain;q=0.6, text/sgml;q=0.6, text/xml;q=0.6, */*;q=0.5
```

Parameetri q väärtused formaadi tüüpide järel ütlevad serverile, millist formaati lehte külastav klient rohkem või vähem ihaldab. Korrektselt toimiv server saab selle põhjal valida kliendi poolt kõige eelistatuma (kõrgeima q väärtusega) tüübi, mida ta pakkuda suudab, ning siis selles vormingus kliendile lehekülje serveerida.

### 1.3. Nõuded (X)HTML-i analüüsimisel

Ühe kaasaegse HTML dokumendi põhistruktuur on järgmine<sup>5</sup>:

HTML dokumendi põhiosa koosneb elementidest. Elemendid koosnevad algusmärgendist ja lõpumärgendist (mis võib ka puududa). Elemendi algusmärgendi küljes võivad olla atribuudid ning atribuutidel väärtused. Elemendi sees (algus ja lõpumärgendi vahel) võib olla

<sup>5</sup> Siin ja edaspidi on kõik (X)HTML-i koodinäited toodud HTML kujul, kui just pole teisiti märgitud.

tekst ja/või teised elemendid.

Ülaltoodud näites on elemendi html algusmärgendiks <html lang="et"> ja lõpumärgendiks </html>. Sellel elemendil on üks parameeter lang, mille väärtuseks on et. html-elemendi sees on elemendid head ja body, mis omakorda sisaldavad veel teisi elemente, teksti ja kommentaare.

Korrektse HTML dokumendi alguses on dokumenditüübi deklaratsioon (*Document Type Declaration*) ehk DOCTYPE, mis viitab dokumenditüübi definitsioonile (*Document Type Definition*) ehk DTD-le. Oma olemuselt pole tegu millegi enamaga, kui lihtsalt märkega selle kohta, millist HTML-i versiooni dokument kasutab. Järgnevalt vaatame, millistest osadest DOCTYPE koosneb alloleva näite põhjal.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```

- <! DOCTYPE deklaratsiooni algus.
- HTML juurelemendi nimi. XHTML faili puhul peab see olema kirjutatud väiketähtedega, HTML-i puhul pole tähesuurus oluline.
- PUBLIC siinkohal märgib, et järgneva sõne (tekst jutumärkide vahel) on FPI;
   SYSTEM sama koha peal tähendaks, et koheselt järgneb SI.
- "-//W3C//DTD HTML 4.01//EN" FPI ehk *Formal Public Identifier* (ingl. k. ,avalik identifikaator") on viide DTD ametlikule nimele.
- "http://www.w3.org/TR/html4/strict.dtd" SI ehk *System Identifier* (ingl. k. "süsteemi identifikaator") annab DTD faili aadressi. SI määramine pole kohustuslik, kuid näiteks paljud brauserid lülituvad SI määranguta dokumendi puhul nn. *quirks mode*'i, mis peamiselt tähendab seda, et dokumendi renderdamisel ei järgita täpselt W3C soovitusi.

Ülaltoodud näites on kasutusel HTML-i versioon 4.01, mille täpse kirjelduse DTD kujul võib leida aadressilt <a href="http://www.w3.org/TR/html4/strict.dtd">http://www.w3.org/TR/html4/strict.dtd</a>.

Informatsioon HTML faili kodeeringu kohta antakse edasi HTTP päiste kaudu, lisades Content-Type päise väärtusele charset atribuudi (vaata HTTP päiste näidet leheküljel 13). Kuid kodeeringu võib määrata ka kasutades HTML-i sisest HTTP päiste ekvivalenti, näiteks järgnev kood määrab dokumendi kodeeringuks UTF-8<sup>6</sup>:

```
<meta http-equiv="content-type" content="text/html; charset=utf-8">
```

HTML dokumendi sees sedasi määratud kodeering on madalama prioriteediga, kui HTTP päises määratud kodeering. Seega kehtestub meta-elemendiga määratud kodeering vaid juhul kui HTTP päises *pole* kodeeringut määratud, vastasel juhul kasutatakse dokumendi kuvamisel päistest pärit kodeeringut.

XHTML dokumendi puhul on reeglid üldjoontes samad, mis HTML-i puhul. Kõige olulisemaks erinevuseks just käesoleva töö seisukohast on see, et XHTML dokument algab XML-i deklaratsiooniga, näiteks nõnda:

```
<?xml version="1.0" encoding="big5"?>
```

Erinevalt HTML-ist on XHTML-le (õigupoolest XML-le) sisse ehitatud mehhanism dokumendi kodeeringu määratlemiseks. Ent sarnaselt HTML-ga on dokumendi sees määratud kodeering madalama prioriteediga, kui HTTP päistes antu.

Pöördudes tagasi programmi nõudmiste juurde, tulenesid kõigest eelpoolkirjutatust järgmised nõuded:

<sup>6</sup> UTF-8 on 8-bitine Unicode'i kodeering ehk Unicode'i transformatsiooni formaat (*Unicode Transformation Format*). Unicode omakorda on kooditabel, mis sisaldab pea kõiki sümboleid kirjutamaks kõigis maailma keeltes. <a href="http://www.unicode.org/">http://www.unicode.org/</a>

- Programm pidi salvestama kõigi (X)HTML dokumendis leiduvate elementide nimed ning samuti ka nende elementide atribuutide nimed.
- Kõigi atribuutide väärtuste salvestamine polnud tarvilik eeskätt selliste atribuutide puhul, mille võimalike väärtuste skaala on väga lai. Näiteks kõigi alt atribuutide väärtuste talletamine poleks andnud muud kui hulga raisatud kõvakettapinda, kuid kindlasti tuli salvestada sedasorti atribuutide nagu type, rel ja lang
  väärtused.
- Programm pidi mõõtma ja salvestama (X)HTML faili suuruse ja selles sisalduva teksti ning kommentaaride mahu.
- Salvestada tuli kõik leitud dokumenditüüpide deklaratsioonid ning XML-i deklaratsioonid.
- Kui HTTP päises polnud kodeeringut määratud, siis tuli salvestada ka meta elemendiga või XML-i deklaratsiooniga määratud kodeering, vastasel juhul HTTP päisest pärit kodeering.

Kõige lõpuks tuli läbi viia ka (X)HTML-i valideerimine, ehk kontrollida (X)HTML dokumendi süntaksi vastavust deklareeritud DTD-le. Kuna oli teada, et kaugeltki mitte kõik (X)HTML dokumendid ei sisalda DOCTYPE määrangut, siis nende valideerimiseks tuli kasutada versiooni "HTML 4.01 Transitional", mis on ühelt poolt kõige uuem, teisalt aga kõige vabama süntaksiga HTML-i versioon. (Näiteks *W3C Markup Validation Service*<sup>7</sup> käitubki DOCTYPE määrangu puudumisel just nõnda.)

#### 1.4. Nõuded CSS-i analüüsimisel

CSS ehk *Cascading Style Sheets* (ingl. k. "kaskaadlaadistik" või "astmelised stiililehed") on keel, mis eeskätt mõeldud (X)HTML vormingus dokumentide kujundamiseks. CSS koodi saab veebilehega siduda kolmel erineval moel. Esimene variant (mis on kõige vähem soovitatav) on lisada lisada stiilideklaratsioonid läbi style atribuudi selle elemendi külge, mille välimust soovitakse muuta:

<sup>7</sup> Tuntud kui W3C Validaator ja kättesaadav aadressilt <a href="http://validator.w3.org/">http://validator.w3.org/</a>

```
Punane rasvane tekst.
```

Teine võimalus, on paigutada CSS (X)HTML dokumendi päisesse (head elemendi sisse) kasutades elementi style:

```
<style type="text/css">
p { color: red; font-weight: bold }
</style>
```

Kolmas ja soosituim võimalus on viidata eraldiseisvale CSS failile kasutades link elementi:

```
<link rel="stylesheet" type="text/css" href="stiilid.css">
```

Lisaks on võimalik viidata ka alternatiivsetele stiililehtedele, mida kasutaja saab soovi korral kasutada vaikimisi stiililehe asemel:

CSS-i stiilimäärangud esitatakse reeglitena (*rule*). Näiteks järgmine reegel:

```
p {
    color: red;
    font-weight: bold;
}
```

Reeglid koosnevad selektorist ning talle järgnevatest deklaratsioonidest. Selektor määrab, millistele elementidele veebilehel stiili rakendatakse. Ülaltoodud näites on selektoriks p, mis tähendab, et stiil rakendatakse kõigile p elementidele. Loogeliste sulgude vahele jäävad kaks deklaratsiooni: color: red; ning font-weight: bold;. Iga deklaratsioon

koosneb kahest osast: atribuut ja väärtus. Toodud näites on atribuutideks color ja fontweight, väärtusteks aga red ja bold.

Erinevaid selektoreid on CSS-is palju, näiteks:

Nagu näitest näha, võib hulga erinevaid selektorid koguda ühte reeglisse, eraldades selektorid üksteisest komadega. Lisada võib ka kommentaarid, mis kirjutatakse /\* ja \*/ vahele.

Ka atribuutide väärtused võivad olla väga mitmesugused, nagu kajastab järgnev koodilõik:

```
text-decoration: underline; /* võtmesõnad */
font-family: "Courier New", monospace; /* alternatiivide loend */
line-height: 1.5; /* arvud */
margin: lem 2ex 15px 6%; /* suhtelised ühikud */
padding: 5cm 40mm 2in 28pt; /* absoluutsed ühikud */
background: url("lilled.png"); /* aadressid */
```

Eriliselt võimalusterikas on värvuste määratlemine. Järgnevas näites määravad kõik viis deklaratsiooni ühe ja sama värvi – punase:

```
color: red;
color: #ff0000;
color: #f00;
color: rgb(255,0,0);
color: rgb(100%,0%,0%);
```

Peale selle võib atribuudi väärtuse lõpus olla võtmesõna !important, mis tähendab, et antud deklaratsioon on teistest sama atribuudi väärtustest kõrgema prioriteediga. Eriti suur roll on !important võtmesõnal kasutajapoolsete stiililehtede juures – seal teeb selle kasutamine võimalikuks kirjutada üle veebilehe autori poolt määratud stiilid. Näiteks järgnev kasutaja stiilileht määrab, et kõigi lehekülgede lingid peavad olema sinised ja allajoonitud, hoolimata sellest, mida lehekülgede autorid on oma CSS-ga määranud:

```
a:link {
   color: blue !important;
   text-decoration: underline !important;
}
```

Lisaks harilikele CSS-i reeglitele on kasutusel ka ät-reeglid (nimi tuleneb sellest, et kõik need reeglid algavad "@" märgiga).

@media ät-reegel määrab, millisele meediatüübile on tema sisse paigutatud reeglid mõeldud. Näiteks võib seeläbi stiililehega määrata, et ekraanil näitamiseks kasutataks "Arial" fonti, aga prinditud veebileht oleks fondiga "Times New Roman". Mõned näited:

```
@media screen { /* arvutiekraanile mõeldud CSS-i reeglid */ }
@media print { /* printimisel kasutatavad reeglid */ }
@media tty, handheld { /* konsooli ja pihuarvuti reeglid */ }
```

CSS-i meediatüübi määramiseks on veel teisigi mooduseid peale ät-reegli. Meediatüübi saab määrata ka kasutades (X)HTML-i media atribuuti, seda nii link, kui ka style elemendis:

```
<link rel="stylesheet" type="text/css" media="aural" href="style.css">
<style type="text/css" media="projection"></style>
```

Viimaks võib meediatüübi määrata ka @import ät-reegliga, mida vaatame järgnevalt.

@import ät-reegel on mõeldud väliste stiililehtede kaasamiseks stiililehte, kusjuures kaasatud stiililehed võivad omakorda sisaldada @import reegleid, mis kaasavad veel teisi stiililehti jne. Lisaks võib importimisel määratleda, millise meediatüübi jaoks kaasatav CSS fail mõeldud on. Näiteks:

```
@import "esileht.css";
@import url(lihtsustatud.css) tty, handheld, braille, aural;
```

Vähemkasutatavatest ät-reeglitest võib ära tuua veel järgmised:

Pöördudes taas tagasi programmi nõudmiste juurde, tulenesid kogu eelnevast jutust CSS-i analüüsimiseks järgmised nõudmised:

- Programm pidi (X)HTML failist eraldama atribuudi style väärtused, elemendi style sisu ning lingitud CSS failide aadressid.
- Lingitud CSS failid tuli alla laadida ja kui CSS-ist leiti @import reegli abil lingitud stiililehti, siis tuli ka need viidatud CSS failid alla laadida jne. Stiililehed tuli alla laadida hoolimata meediatüübist, küll aga tuli ignoreerida alternatiivseid stiililehti, mis nagu nimigi ütleb on üksnes alternatiiviks vaikimisi kasutatavale CSS-le.
- Mõõta ja salvestada tuli kogu CSS-i maht, seda nii style atribuutide, style elementide kui lingitud stiililehtede lõikes. Ühtlasi tuli salvestada lingitud stiililehtede arv.
- Mõõta ja salvestada tuli ka CSS-ist leitud kommentaaride maht.

- Salvestada tuli kõik CSS-i atribuudid.
- Atribuutide väärtustest tuli salvestada kõik võtmesõnad, sõned (näiteks fontide nimed), kasutatavad mõõtühikud (näiteks kui väärtus oli 17px, tuli salvestada mõõtühik px) ja taustapiltides kasutatavad pildiformaadid (näiteks väärtus url (/pildid/bg.jpg) tuli salvestada kui url (jpg)).
- Kui väärtuseks oli värv, siis tuli salvestada, kuidas värv oli määratud (kas #rgb, #rrggbb, rgb(r,g,b) või rgb(r%,g%,b%)). Samuti ära märkida, kui kasutati mõnda 17-st põhivärvist, mis CSS-is võtmesõnaga defineeritud.
- Salvestada tuli kõik erinevad CSS-i selektorite tüübid.
- Eraldi tuli salvestada atribuudi-, klassi-, id- ja elemendiselektorite konkreetsed väärtused ning kõik tuntud ja tundmatud pseudoelemendid ja -klassid.
- Salvestada tuli kõik erinevad ät-reeglid ja nende parameetrid.

Kõige lõpuks tuli läbi viia CSS-i valideerimine, kontrollimaks koodi vastavust W3C CSS-i soovitustele. Kuna erinevalt (X)HTML-ist ei sisaldu CSS koodis versiooniinfot, siis tuli valida üks CSS-i versioon, mille alusel kõiki lehti valideerida. Kuna W3C enda CSS-i validaator kasutab vaikimisi valideerimiseks versiooni 2, mis on ühtlasi ka uusima ametlik CSS-i soovitus, siis oli loogiline valida just see versioon ka siinse uurimuse käigus läbiviidava valideerimise aluseks.

Selle otsuse negatiivseks küljeks on, et leheküljed, mis on koostatud veebistandardite osas teadlike isikute poolt võivad kasutada juba ka osakesi CSS 2.1 ja CSS 3 spetsifikatsioonidest, mis aga pole veel ametlikult soovitusteks saanud, ja seetõttu CSS 2 reeglite kohaselt mitte valideeruda. Samas, kui valideerida CSS 2.1 või CSS 3 alusel, võivad vigaseks osutuda CSS failid, mis on täiesti korrektsed CSS 2 tingimustes, aga kasutavad atribuute või väärtusi, mis on uuematest spetsifikatsioonidest välja jäetud. (Näiteks atribuudil display puudub CSS 2.1 spetsifikatsioonis väärtus compact, mis aga CSS 2 puhul on lubatud.)

Lõppkokkuvõttes sai otsustatud pigem tunnistada vigaseks CSS, mis kasutab võimalusi, mida kõige moodsam W3C soovitus ei kirjelda, kui CSS, mis järgib täpselt kõige uuemat soovitust, ent pole korrektne uuemate spetsifikatsioonide valguses, mis aga pole veel soovituse staatusesse jõudnud.

### 1.5. Nõuded JavaScript'i analüüsimisel

JavaScript ehk ECMAScript<sup>8</sup> on programmeerimiskeel, mida peamiselt kasutatakse kliendi arvutisse laetud veebilehel jooksvate programmide loomiseks. ECMAScript on iseenesest laiem standard, defineerides programmeerimiskeele, mille kasutamine pole kaugeltki piiratud üksnes veebilehtedega. Näiteks vastab ECMAScript'i standardile Flash'is kasutusel olev ActionScript. Kuna käesolev uurimus käsitleb üksnes veebilehtede skriptimiseks mõeldud ECMAScript'il baseeruvat keelt, siis räägime siin ja edaspidi JavaScript'ist, mis ühtlasi on ka üldlevinud termin veebilehitsejates kasutatavale skriptimiskeelele viitamiseks.

Sarnaselt CSS-ga on ka JavaScript'i puhul kolm moodust, kuidas JavaScript'i kood (X)HTML failiga siduda. Esiteks saab kasutada onclick, onmouseover, onmouseout jpt on-eesliitega atribuute:

```
Tervitus.
```

Teiseks saab koodi kirjutada script elemendi sisse, mis erinevalt style elemendist võib asuda nii dokumendi päises kui kehaosas:

```
<script type="text/javascript">
alert("Hello, world!");
</script>
```

<sup>8</sup> ECMA ehk *ECMA International* on euroopa standardiorganisatsioon, mis on keskendunud info- ja kommunikatsioonisüsteemide standardiseerimisele. <a href="http://www.ecma-international.org/">http://www.ecma-international.org/</a>

Kolmandaks (ja kõige soovituslikumaks) võimaluseks on eraldiseisva JavaScript'i faili linkimine, mis toimub kasutades script elemendi atribuuti src:

```
<script type="text/javascript" src="hello.js">
```

Erinevalt CSS-st ei saa JavaScript'i fail ise teisi JavaScript'i faile linkida, kuid see-eest toetavad paljud brauserid javascript'i URI<sup>9</sup>-sid:

```
<a href="javascript: alert('Hello, world!');">Tervitus</a>
```

Selline võimalus on küll väga mugav veebilehtede autoritele, ent sedasorti URI-d on täiesti kasutud kõigile neile kasutajatele, kelle veebilehitseja JavaScript'i ei toeta.

JavaScript'i kohta on teada mitmeid erinevaid häid ja halbu praktikaid, mille kasutamist on koodis kaunis kerge tuvastada. Näiteks on üldteada halvaks praktikaks JavaScript'i abil otse (X)HTML lehe sisse teksti kirjutamine kasutades document.write() meetodit. Samuti on ei peeta heaks praktikaks kasutada mitmeid järgnevas koodinäites kasutatud võimalusi (mis paljuski on asendunud uute ja moodsamate meetoditega):

```
if ( document.layers ) {
    document.forms[0].username="Mallory";
    window.status = "foo";
    if ( navigator.appVersion < 6.0 ) {
        window.location = "http://www.example.com";
    }
}</pre>
```

<sup>9</sup> URI ehk *Uniform Resource Identifier* (ingl. k. ühelaadne ressursi identifikaator) on identifikaator, mida internetiprotokollid kasutavad ressursile viitamiseks. URI alaliikidena eristatakse URL-i ja URN-i. URL (*Uniform Resource Locator*) on URI, mis määrab ressursi asukoha. URN (*Uniform Resource Name*) on URI, mis määrab ressursi nime, kuid ei pruugi määrata asukohta. Näiteks URI mailto:juku@example.com määrab nii e-posti konto nime kui aadressi, samas isbn:0-306-40615-2 määrab küll raamatu identifikaatori, ent ei määra asukohta. Kuna üldjuhul pole oluline eristada, kes URI on URL või URN (rääkimata sellest, et tihtipeale on tegu mõlemaga korraga), siis on selles töös kasutatud üksnes URI-d. URI-ga samas tähenduses kasutatakse tekstis ka sõnu "aadress" ja "veebiaadress". <a href="http://www.ietf.org/rfc/rfc3986.txt">http://www.ietf.org/rfc/rfc3986.txt</a>

Teisalt heaks praktikaks loetakse JavaScript'i puhul (X)HTML-i *Document Object Model*'i ehk DOM-i<sup>10</sup> kasutamist. Levinud on järgmises koodinäites kasutatud konstruktsioonid:

```
if ( document.selectElementById && document.getElementsByTagName ) {
   var errors = document.selectElementById( "errors" );
   var errorList = errors.getElementsByTagName( "li" );
   for ( var i in errorList ) {
      if ( linkList[i].className="selected" ) {
            linkList[i].className="disabled";
            }
    }
}
```

Lisaks otseselt headele ja halbadele praktikatele on ka praktikaid, mis annavad aimu, mis liiki veebirakendusega on tegemist. Näiteks, kui JavaScript'i koodis leidub identifikaator XMLHttpRequest, siis võib suure kindlusega väita, et lehekülg kasutab viimasel ajal populaarseks saanud AJAX<sup>11</sup> tehnoloogiat.

Kõige eelpoolkirjutatu põhjal tulenesid JavaScript'i analüüsimiseks järgmised nõudmised:

- Programm pidi (X)HTML failist eraldama on-eesliitega atribuutide väärtused, elemendi script sisu ning lingitud JavaScript'i failide aadressid.
- Kõik välised JavaScript'i failid tuli alla laadida.
- Mõõta ja salvestada tuli kogu JavaScript'i maht, seda nii atribuutide, script elementide kui lingitud failide lõikes. Ühtlasi tuli salvestada lingitud skriptide arv.
- Mõõta ja salvestada tuli ka JavaScript'is sisalduvate kommentaaride hulk.
- Koodis tuli tuvastada võimalikult palju erinevaid konstruktsioone, alustades võtmesõnadega (while, for, var, ...), seejärel erinevad meetodite nimed (alert, prompt, document.write, window.open, document.getElementByTagName, ...)

<sup>10</sup> DOM on mehhanism, mille abil programmeerimiskeel saab manipuleerida XML või SGML faile.

<sup>11</sup> AJAX ehk *Asynchronous JavaScript And XML* on interaktiivsete veebrirakenduste loomise tehnika. Selle eesmärk on parandada veebilehtede reageerimisaega seeläbi, et serveriga suhtlemine toimub taustal, kasutaja tegevusega samaaegselt. XML-i roll seisneb selles, et enamasti toimub serveriga suhtlus XML formaadis.

ning lõpetades XMLHttpRequest'ga.

### 2. VARASEMAD UURINGUD

Selles valdkonnas on tehtud mitmeid varasemaid uuringuid. Enamus neist on on olnud väikesemahulised, kuid on ka mõningaid päris suuri. Järgnevalt tahakski lühidalt peatuda mitmetel suurematel ja/või huvitavamatel uuringutel.

## 2.1. Parnas, Dagfinn. (2001). How to cope with incorrect HTML

Oma magistritöö raames korraldas Dagfinn Parnas 2001. aastal uuringu, selgitamaks välja, kui suur on vigaste lehekülgede osakaal veebis. Ta vaatas läbi ligikaudu 2,4 miljonit URI-d ning leidis, et vaid 0.71% kõigust lehtedest, mida õnnestus alla laadida, valideerus.

Kasutatav valim koosnes URI-dest, mis olid hangitud *Open Directory Project* nimelisest veebikataloogist. *Open Directory Project* on lingikogu, mille koostamise ja haldamisega tegelevad tuhanded vabatahtlikud moderaatorid. Ühtlasi on kõik kokkukogutud lingid allalaetavad RDF<sup>12</sup> formaadis.

Parnas'e uuringu näol on tegemist seni teadaolevatest suurima veebilehtede valideerumise hindamisega. Käesoleva töö autor pole teadlik ühestki varasemast ega hilisemast vähegi võrreldava mahuga uuringust.

#### 2.2. Karppinen, Marko. (2002). State of the Validation 2002

Marko Karppinen korraldas 2002. aasta veebruaris uuringu W3C liikmesorganisatsioonide lehekülgedel. Võis arvata, et W3C liikmed on esirinnas veebistandardite rakendamise osas, kuid selgus, et üksnes 18 lehekülge 506-st (ehk 4%) kasutas korrektset (X)HTML-i.

Käesoleva aasta märtsikuus kordasin ma Karppinen'i uuringut ja leidsin, et nelja aastaga on olukord tublisti paranenud: 286-st liikmesorganisatsioonist omas valideeruvat lehekülge tervelt 61 (ehk 17%). Sellegipoolest pole liikmesorganisatsioonide olukord just kiita, sest

<sup>12</sup> RDF ehk *Resource Description Framework* on perekond spetsifikatsioone metaandmete salvestamiseks kasutades XML formaati. <a href="http://www.w3.org/RDF/">http://www.w3.org/RDF/</a>

vigast (X)HTML-i kasutavad enamus. Uuringu tulemused on avaldatud aadressil <a href="http://www.triin.net/2006/03/05/Validating sites of W3C members">http://www.triin.net/2006/03/05/Validating sites of W3C members</a> .

## 2.3. Allsopp, John. (2005). Semantics in the wild

John Allsopp'i 2005. aasta novembrikuus läbi viidud uuring 1315. leheküljel vaatles (X)HTML-i atribuutide class ja id väärtusi. Uuring tuvastas, millised on populaarseimad klassi ja ID väärtused, mida (X)HTML-is kasutatakse, või vähemasti nendel 1315 lehel, sest valimi moodustamiseks kasutas ta robotit, mis alustas <a href="http://www.stopdesign.com">http://www.stopdesign.com</a> aadressilt ning liikus edasi sealt viidatud lehekülgedele.

Kuna uuringu alguspunktiks oli veebistandardeid populariseeriv lehekülg, siis on huvitav, et sellegipoolest leiti uuringu käigus lähendastelt lehekülgedelt väga palju presentatiivseid klassi ja ID nimesid (nagu "red", "top" ja "bottom").

### 2.4. Saarsoo, Rene. (2005-2006). Web Standards in Estonia

Minu enda läbi viidud uuringud Eesti veebilehtedel on seni olnud minu teada ainsad, kus on sarnaseid valimeid on uuritud kindlate ajavahemike tagant korduvalt sama metoodikat kasutades.

Oma proseminaritöös vaatlesin ma ligikaudu 22 tuhandet URI-d, mis olid võetud Neti.ee Eesti serverite nimekirjast (<a href="http://www.neti.ee/cgi-bin/serverid">http://www.neti.ee/cgi-bin/serverid</a>). 2005. aasta veebruarikuu seisuga valideerus nendest vaid 2,22%. Kui ma kordasin oma uuringut sama aasta augustis, selgus, et valideeruvate lehekülgede osakaal oli langenud 2,17% peale. Kuid kolmandal korral, 2006. aasta märtsis, taas uuringut kordasin oli valideeruvate lehtede osakaal hoopis tõusnud – pea poole võrra – 3,02% peale.

Lisaks valideeruvate lehekülgede arvu muutustele täheldas uuring pidevat XHTML dokumenditüübi deklaratsioonide osakaalu tõusu. Samuti UTF-8 osakaalu tõusu lehekülgedel kasutatavate kodeeringute seas.

Käesolev töö toetub suuresti nimetatud kolme uuringu läbiviimisel saadud kogemustele ning üritab vältida tehtud vigu.

## 2.5. Web Authoring Statistics. (2005). Google Inc

2006. aasta alguses avaldas Google 2005. aasta detsembris veidi üle miljardil lehekülje läbi viidud uuringu tulemused. Uuring vaatles (X)HTML-i klasside- elementide ja atribuutide nimesid ning seotud metaandmeid, nagu meta-elemendi atribuutide väärtused ja rel ning rev atribuutide väärtused.

Uuringu peamine kaal seisneb just valimi suuruses, Google ei avalda infot selle kohta, kuidas valim täpselt koostati, kuid võib arvata, et vaadeldavad leheküljed valiti juhuslikult Google'i enda andmebaasist. Kuna Google'i andmebaasi struktuur on kindlasti ärisaladus, siis on mõneti arusaadav, miks valimi koostamise kohta täpsem info puudub. Uuringu õigsuses siiski vast kahelda pole põhjust, sest tulemused langevad suuresti kokku nii minu kui ka teiste autorite varasemate uuringutega.

#### 3. VALIM

Esindusliku valimi leidmine oli käesoleva uuringu üks suuremaid probleeme. Varem Eesti lehekülgedel läbi viidud uuringud kannatasid suuresti just selle all, et valim polnud piisavalt representatiivne. Kasutatud Neti.ee Eesti serverite nimekiri<sup>13</sup> oli küll suur, sisaldades enamikku .ee domeeninimesid, kuid kannatas selle all, et paljudel aadressidel polnud üldse veebilehte või oli lühike leht, mis teatas et lehekülg on kas valmimisel või suletud – ning selliseid lehekülgi oli päris palju, ja mitmed neist lihtsatest lehtedest juhtusid ka valideeruma, mistõttu valideeruvate lehekülgede arv Eestis võis olla üksjagu ülehinnatud.

Kuna ma tol hetkel polnud teadlik ühestki suurest ülemaailmsest veebilehtede nimekirjast, siis sai valimi koostamiseks kaalutud järgmisi variante:

- Domeenide nimekirja ammutamine DNS<sup>14</sup> serveritest.
- Mõne suure avaliku veebikataloogi, nagu näiteks <a href="http://directory.google.com/">http://directory.google.com/</a> kõikide linkide allalaadimine, kasutades robotit, mis külastab kataloogi kõiki kategooriaid ja seejärel kõiki alamkategooriaid jne.

Õnneks ei tulnud kumbagi neist variantidest pikemalt kaaluda, sest leidsin internetiavarustest leheküljel 27 kirjeldatud Dagfinn Parnas'e uuringu, kus ta kasutas *Open Directory Project* (ODP) leheküljelt alla laetud URI-de nimekirja. Kiire kontroll näitas, et ODP on endiselt elujõus ning pakub ka kogu oma kataloogi sisu RDF formaadis allalaadimiseks. Nüüd, 5 aastat hiljem, sisaldas see juba pea kaks korda rohkem aadresse, tervelt 4,7 miljonit.

<sup>13</sup> Aadressil <a href="http://www.neti.ee/cgi-bin/serverid">http://www.neti.ee/cgi-bin/serverid</a>

<sup>14</sup> DNS ehk *Domain Name System*, on domeeninimede süsteem, mis seab omavahel vastavusse serverite IP aadressid ja domeeninimed. <a href="http://www.ietf.org/rfc/rfc1034.txt">http://www.ietf.org/rfc/rfc1034.txt</a> ja <a href="http://www.ietf.org/rfc/rfc1035.txt">http://www.ietf.org/rfc/rfc1035.txt</a>

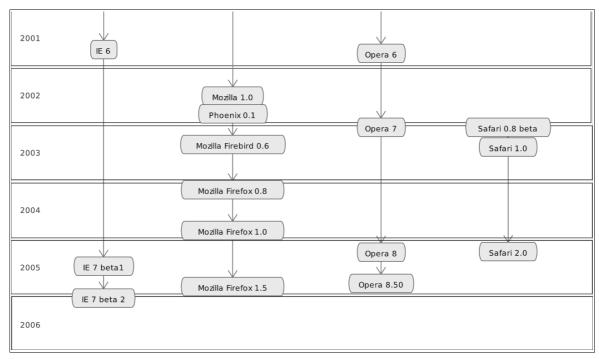
Lisaks sellele, et nimetatud URI-de nimekiri oli esinduslik (hõlmates kõiki populaarsemaid lehekülgi, mida vabatahtlikud sinna lisanud olid), mahukas ja kergesti kättesaadav, tekkis seeläbi ka võimalus teada saada, kuidas on olukord korrektse koodiga veebilehtede osas Parnas'e läbi viidud uuringust saati muutunud.

Nagu juba sissejuhatuseski sai räägitud, siis on olukord veebis viimase viie aasta jooksul paljuski muutunud. Viis aastat tagasi oli olukord veebilehitsejate maastikul hoopis teistsugune, kui see on täna. Microsoft Internet Explorer (IE) oli võitnud nn. brauserite sõja ning praktiliselt ainuvalitses turgu. Kuid, nagu illustreerib joonis 1 leheküljel 32, hakkasid järgnevate aastate jooksul turule ilmuma tugevad konkurendid. Mozilla jõudis oma arengus viimaks 1.0 versioonini, peale mille ilmumist tekkis varsti ka tänaseks palju populaarsust kogunud Mozilla Firefox. Opera vahetas oma endise renderdusmootori välja uue ja parema standardite toega Presto vastu ning 2005. aasta lõpuks muutus kogunisti täiesti tasuta jagatavaks. 2003. aasta alguses tuli Apple välja omaenda brauseriga Safari, mis põhines KHTML mootoril ning, mis sai OS X-is vaikimisi kasutusel olevaks brauseriks.

Kogu selle aja, mis teised brauserid jõudsalt arenesid ja järjest paremini veebistandardeid toetasid, püsis IE täpselt samas seisus, kuhu ta oli jõudnud 2001. aastaks. Alles 2006. aasta alguses ilmus IE seitsmenda versiooni esimene avalik beeta, mis aga endiselt on standardite toe osas konkurentidest mitu sammu tagapool.

Kuid just veebistandardite populaarsus on vahepealsete aastatega jõudsasti kasvanud. Üheks suuremaks tõukejõuks sellele oli 2003. aastal Jeffrey Zeldman'i sulest ilmunud "Designing with Web Standards", mis on veebistandardite kommuunis nüüdseks tuntuks saanud kui "oranž raamat"<sup>15</sup>. 2005. aasta sügisel Sydney's toimunud suuresti veebistandarditele pühendatud konverentsil Web Essentials 05 oli osaleda soovijaid juba nõnda palju, et kõik, kes juba varakult ei registreerunud, pidid ukse taha jääma.

<sup>15</sup> Raamat algab peatükiga "99.9% of Websites Are Obsolete" (ingl. k. "99,9% veebilehtedest on iganenud"), kuid olles pärast raamatu ilmumist lugenud Parnas'e uurimust, teatas Zeldman oma kodulehel, et eksis: "It turns out only 99.29% of websites are obsolete, according to a thesis that examined 2.4 million web pages. Kindly ignore our earlier estimate." (ing. k. "Selgus, et üksnes 99,29% veebilehtedest on iganenud, vastavalt teadustööle, mis uuris 2,4 miljonit lehekülge. Palun teid lahkesti ignoreerida meie varasemat hinnangut."). (Zeldman 2003).



Joonis 1: Nelja levinuima brauseri olulisimad väljalasked alates 2001. aastast. Selgelt on näha, et alates viimasest Microsoft Internet Explorer'i versioonist on konkureerivad brauserid jõudsasti edasi arenenud. (Wikipedia erinevate artiklite põhjal.)

Kõigi nende muutuste taustal jääb aga õhku küsimus, kas olukord veebilehtede kvaliteedi osas on tõesti paremaks muutunud, või on lihtsalt palju veebilehti juurde tekkinud (teiste hulgas ka neid, mis käsitlevad veebistandardeid)?

Püüdes võimalikult täpselt jäljendada Parnas'e uuringut, peaks käesolev töö pakkuma sellele küsimusele vastuse.

#### 4. PROGRAMMI IMPLEMENTEERIMINE

#### 4.1. Keele valik

Peatükis 1 kirjeldatud programmi loomiseks oli tarvis valida võimalikult sobilik programmeerimiskeel. Valitud keeles pidi olema saadaval võimalikult palju lisamooduleid/teeke, mis lihtsustaksid mitmeid programmile seatud ülesandeid – eeskätt HTML-i, CSS-i ja JavaScript'i parsimist. Lisaks oli tarvis ka häid tekstitöötlusvõimalusi, sest sisse lugeda ja töödelda tuli nii (X)HTML-i kui CSS-i validaatori väljund. Ja viimaks, kuna andmed oli tarvis salvestada andmebaasi, läks tarvis ka andmebaasiliidest.

Mõningase uurimise ja otsingute tulemusena selgus, et peaaegu kõik seda mida tarvis, võimaldab Perl. Perl on üldtuntud oma mitmekülgsete tekstitöötlusfunktsioonide poolest ning *Comprehensive Perl Archive Network* ehk CPAN sisaldab tohutul hulgal erinevaid avatud lähtekoodiga mooduleid, milledest mõned osutusid suurepäraselt sobilikeks just selle projekti jaoks.

#### 4.2. Kasutatud moodulid

Programmis sai kasutatud järgmisi Perl'i mooduleid:

- **HTML::Parser** parser, mis mõeldud just reaalse internetis leiduva (X)HTML-i analüüsimiseks.
- CSS::SAC Simple API<sup>16</sup> for CSS ehk SAC<sup>17</sup> implementatsioon Perlis.
- DBI Perli üldine andmebaasiliides, konkreetseks suhtluseks MySQL<sup>18</sup> andmebaasiga läks veel lisaks tarvis DBD::mysql moodulit.

<sup>16</sup> Application Programming Interface (ingl. k. rakenduse programmeerimise liides).

<sup>17</sup> SAC on samasugune lihtne API CSS-i failide analüüsimiseks kui SAX (*Simple API for XML*) on XML failide jaoks. Kuna CSS on XML-ist keerukam keel, siis on SAC-is rohkem erinevaid sündmusi, millele reageerida, kuid üldjoontes toimib kõik sarnaselt SAX-ga. <a href="http://www.w3.org/TR/SAC/">http://www.w3.org/TR/SAC/</a>

<sup>18</sup> http://www.mysql.com/

- URI URI-dega ümberkäimine, näiteks URI-de võrdlemiseks ja suhteliste URI-de absoluutseks muutmiseks.
- **Getopt::Declare** käsurea parameetrite analüsaator.
- Date::Simple ja Time::Simple kuupäeva ja kellaajaga ümberkäimine.

## 4.3. Kasutatud eraldiseisvad programmid

Lisaks erinevate moodulite kasutamisele pidi loodav programm suhtlema ka teiste iseseisvate programmidega. Nendeks olid erinevad validaatorid, andmebaas ja GNU Wget.

#### 4.3.1. Andmebaas

Oma eelnevates Eesti veebilehtedel läbiviidud uuringutes olin kogutud andmed salvestanud harilikesse tekstifailidesse. Kuna käesoleva töö käigus kogutav andmestik oli kordades keerukam ja mahukam kui minu varasemate uuringute puhul, siis oli selge, et andmed tuleb salvestada relatsioonilisse andmebaasi. (Õigupoolest olin ka Eesti veebilehtedelt kogutud andmed viimases augustikuus korraldatud uuringus juba tekstifailidest välja filtreerinud ja andmebaasi paigutanud, et andmetöötlust vähegi lihtustada.)

Kasutatavaks andmebaasimootoriks sai valitud MySQL. Üheltpoolt seetõttu, et töö autor oli tolle andmebaasiga juba tuttav, mistõttu langes ära vajadus suuremat sorti juurdeõppimiseks, teisalt piisas MySQL-i pakutavatest võimalustest antud rakenduse jaoks täiesti, hoopis olulisem külg oli lihtsast veebilehest märksa suuremate (miljonid kirjed) andmemahtudega hakkama saamine, mille osas MySQL-il on samuti head näitajad.

#### 4.3.2. Validaatorid

(X)HTML-i valideerimiseks sai kasutatud WDG<sup>19</sup> Validaatorit<sup>20</sup>. Selleks oli kaks head põhjust:

<sup>19</sup> WDG ehk *Web Design Group* on organisatsioon, mis loodud eesmärgiga brauserist ja resolutsioonist sõltumatute kõigile kasutajagruppidele ligipääsetavate veebilehtede loomist populariseerida. Oma veebilehel <a href="http://www.htmlhelp.com/">http://www.htmlhelp.com/</a> pakutakse selleks tarbeks mitmesuguseid tööriistu ja õpetusi.

<sup>20</sup> http://www.htmlhelp.com/tools/validator/offline/index.html.en

- 1. Just seda validaatorit kasutas oma uuringus Dagfinn Parnas, ning kuna eesmäriks oli vähemasti (X)HTML-i valideerimise osas jäljendada Parnase uuringut võimalikult lähedaselt, siis oli loogiline kasutada valideerimiseks ka sama programmi.
- 2. Kõige populaarsem (X)HTML-i validaator, *W3C Markup Validation Service*, on küll vabalt veebist allalaetav, ent selle installeerimine ja kasutamine on märksa keerukam kui WDG Validaatori puhul.

Õigupoolest koosneb WDG Validaator kolmest komponendist:

- wdg-sgml-lib kollektsioon erinevaid DTD-sid, mille põhjal valideerimist teostada.
- **Iq-nsgmls** valideeriv SGML-i parser, kirjutatud C++ keeles.
- validate Perl'i skript, mis võimaldab (X)HTML faile valideerida, kasutades eeltoodud parserit ja DTD-sid.

WDG poolt oli saadaval ka CSS-i süntaksi kontrollimise vahend CSSCheck<sup>21</sup>, ent kuna see ei toetanud paljusid CSS2 konstruktsioone, siis jäi ainukese mõeldava variandina alles W3C CSS Validation Service<sup>22</sup>.

### 4.3.3. GNU Wget

GNU Wget on programm veebist failide automatiseeritud allalaadimiseks. Seda programmi kasutas oma töös ka Dagfinn Parnas, mistõttu sai see ka käesoleva töö puhul veebilehtede allalaadimist teostama valitud.

Alternatiivina sai kaalutud ka omaenda allalaadimis-skripti loomist, kasutades näiteks Perli libwww-perl moodulit, kuid sellest sai loobutud, sest Wget pakkus juba palju sisse-ehitatud funktsionaalsust, eeskätt HTTP protokolliga ümberkäimise osas (näiteks automaatne reageerimine HTTP ümbersuunamistele), mis ise loodud koodi puhul oleks tulnud kõik ka ise implementeerida ning ühtlasi oleks seeläbi suurenenud võimalike vigade arv

<sup>21 &</sup>lt;a href="http://www.htmlhelp.com/tools/csscheck/">http://www.htmlhelp.com/tools/csscheck/</a>

<sup>22</sup> Tuntud kui W3C CSS-i Validaator, kättesaadav aadressilt http://jigsaw.w3.org/css-validator/

programmis.

Ühtlasi on Wget'i sisse ehitatud ka võimalus allalaadimine lõpetada, kui teatud aja jooksul pole serverist vastust saabunud. Parnas seadistas Wget'i nõnda, et too ei oodanud ühegi lehekülje taga rohkem kui kümme sekundit – samamoodi sai toimitud ka selles töös.

#### 4.4. Andmebaasistruktuur

Enne, kui võis koostama asuda programmi ennast, oli tarvis paika panna andmebaasistruktuur. Sealjuures oli oluline, et struktuur võimaldaks salvestada just nii palju infot, kui hilisemal andmete analüüsil tarvis. Kuna andmemahud olid suured, siis oli oluline vältida liiga paljude andmete salvestamist, et jääda siiski kasutada oleva kõvakettaruumi piiridesse. (Näiteks arvutused esialgse andmebaasimudeli põhjal näitasid, et 4,7 miljoni lehekülje läbivaatamise tulemusena kasvaks üksnes (X)HTML-i infot säilitava andmebaasi maht mitmesaja gigabaidini – seetõttu sai välja jäetud hulk väheolulist informatsiooni ning vähendatud vajaminevat andmemahtu drastiliselt.)

Üldistatult võttes oli loodud andmebaasistruktuur järgmine. Andmebaasis oli üks keskne tabel, mis sisaldas kõigi läbivaadatud lehekülgede URI-sid. Selle tabeliga olid otseselt või kaudselt seotud pea kõik teised tabelid. Andmestruktuuri perifeerias (seoste kaudu kõige kaugemal kesksest tabelist) asusid tabelid, mis hoidsid mitmesugust tekstilist informatsiooni: näiteks lehekülgedelt leitud (X)HTML-i elementide ja atribuutide nimed ning validaatorite veateated. Need tabelid olid seotud keskse tabeliga peamiselt läbi mitumitmele seoste. Näiteks ühendas peatabelit ja (X)HTML-i elementide nimesid sisaldavat tabelit vahetabel, kus oli kirjas millisel veebilehel milline element esines ning kui mitukorda.

Kõik andmebaasi 38 tabelit ja enamik nendevahelisi seoseid on esitatud joonistena lisas 1. Andmebaas on lisatud ka tööga kaasas olevale CD plaadile (vaata lisa 3).

# 4.5. Programmi ülesehitus

Perl'is kirjutatud programm, mis sai nimeks webstat, koosnes järgmistest peamistest osadest:

- andmebaasiliides sisaldas kõiki päringuid andmebaasiga suhtlemiseks,
- andmebaasipuhver säilitas sagedasti kasutatavate andmebaasiväljade sisu, puhver oli vajalik selleks, et vähendada andmebaasile tehtavate päringute hulka ning ühtlasi seeläbi programmi tööd kiirendada,
- Wget'i liides toimetas veebist failide allalaadimist GNU Wget programmi abil,
- HTTP päiste analüsaator analüüsis HTTP päiseid, mis koguti kokku Wget-i väljundist,
- **(X)HTML-i analüsaator** analüüsis veebilehtede struktuuri, filtreerides muuhulgas välja lehekülgedel sisalduva CSS-i ja JavaScripti,
- (X)HTML-i validaatori liides toimetas (X)HTML-i valideerimist WDG Validaatori abil,
- CSS-i analüsaator analüüsis CSS-i struktuuri,
- CSS-i validaatori liides toimetas CSS-i valideerimist W3C CSS-i Validaatoriga,
- JavaScript'i analüsaator analüüsis JavaScripti struktuuri.

Andmebaasi puhverdamine toimus nõnda, et kui näiteks HTML failist leiti uus element, siis kõigepealt kontrolliti, kes sellele vastav ID on juba puhvris olemas. Kui jah, siis kasutati olemasolevat ID-d, kui aga element puhvris puudus, üritati lisada see uus element andmebaasi. Kui andmebaasis vastava nimega elementi veel polnud, siis lisamine õnnestus ja andmebaas tagastas lisatud elemendi ID. Kui aga element oli juba andmebaasis olemas, siis lisamine ebaõnnestus ning programm sooritas andmebaasi päringu vastava nimega elemendi ID teadasaamiseks.

Taoline puhverdamise algoritm oli tarvilik seetõttu, et iga programmi eksemplar pidi

arvestama ka teiste samaaegselt töötavate eksemplaridega, sest töökiiruse suurendamiseks oli tarvis jooksutada programmi korraga mitmete arvutite peal (täpsemalt on põhjuseid mitme programmieksemplari üheaegseks jooksutamiseks kirjeldatud peatükis 5 "Uurimuse läbiviimine" leheküljel 40). Juhul kui elemendi andmebaasis esinemist oleks kontrollitud tavalise SELECT päringuga, siis oleks võimalik olnud järgmine stsenaarium:

Programmi eksemplar A leiab HTML-ist elemendi foo, ta teeb päringu andmebaasi, tuvastamaks, kas element foo on seal juba olemas – andmebaas vastab eitavalt. Vahepeal avastab ka programmi eksemplar B HTML-ist elemendi foo ja teeb samuti andmebaasile päringu, kus saab teada, et elementi foo seal veel pole. Seejärel lisab programmi eksemplar A andmebaasi elemendi foo, aga eksemplar B on endiselt arvamusel, et elementi foo andmebaasis pole ning üritab samuti elementi foo lisada, misläbi element foo saab andmebaasis olema topelt. Muidugi, kui andmebaasi väli on defineeritud unikaalsena, siis elementi topelt ei sisestata ja lõpptulemusena jõutakse sama funktsionaalsuseni, mida pakub eespool kirjeldatud algoritm, ent kuna tulemus on sama, siis on mõistlik valida lihtsamini teostatav algoritm, mis antud juhul on see, kus koheselt proovitakse andmebaasi uut elementi sisestada.

Programmi peamooduli lähtekood on ära toodud lisas 2. Kogu programmi lähtekood on leitav tööga kaasas olev CD plaadi pealt (vaata lisa 3).

## 4.6. Implementeerimise käigus ette tulnud probleemid

Kui ma olin üle saanud oma isiklikust probleemist, milleks oli vähene Perl'i tundmine<sup>23</sup>, põrkasin ma programmi loomise juures järgnevalt kirjeldatud probleemide otsa.

Esiteks selgus, et kasutatav CSS-i parser CSS::SAC sisaldas mõningaid olulisi vigu: alates korrektsete CSS-failide vigaseks tunnistamisest kuni programmi täieliku hangumiseni mõningate vigade peale CSS-is. Kuid CSS-i parseri lähtekood oli piisavalt lihtne ja arusaadav ning mõningase pingutuse järel õnnestus kõik leitud probleemid kas täiesti

<sup>23</sup> Siinkohal oli palju abi Greg London'i suurepärasest raamatust *Impatient Perl*, mis on kättesaadav aadressilt <a href="http://www.perl.org/books/impatient-perl/">http://www.perl.org/books/impatient-perl/</a>

parandada või pakkuda neile osaline lahendus, mis rahuldas programmi webstat vajadusi.

Järgmised suuremad probleemid tekkisid W3C CSS Validation Service'i kompileerimisel. Selgus, et W3C lehel olnud hoiatus, et neil, kes soovivad validaatorit kompileerida, tuleb iseseisvalt tublisti vaeva näha, oli kõigiti tõsine. Kui validaatori lähtekood oli CVS<sup>24</sup>-i kaudu alla laetud ja seda kompileerida üritatud, selgus, et puudu oli hulk vajaminevaid komponente. Seejärel sai W3C CVS-i puust otsitud neid vajaminevaid osi ja need alla laetud, mispeale Java kompilaator jällegi teavitas, et vajab midagi, mida tal pole. Seega tuli taas W3C CVS-i puusse tagasi pöörduda ja puuduolevaid detaile otsida, jne, jne jne.

Kuna minu kogemused suuremate Java programmide kompileerimisel olid praktiliselt olematud, siis siinkohal suur tänu minu juhendajale, kes mind selles protsessis tublisti aitas, olgugi, et kogu see suur kompileerimine lõppes sellega, et www-validator-css@w3.org listi kirjutas Steve Ferguson ning teatas, et on loonud CSS-i validaatorist valmiskompileeritud versiooni ning teinud selle vabalt kättesaadavaks aadressil <a href="http://www.illumit.com/css-validator/">http://www.illumit.com/css-validator/</a>

<sup>24</sup> Concurrent Versioning System – vabavaraline versioonihaldussüsteem.

# 5. UURIMUSE LÄBIVIIMINE

### 5.1. Valimi töötlemine

Open Directory Project poolt kokku kogutud aadresside nimistu sai RDF kujul ODP lehelt alla laetud ning URI-d sellest välja filtreeritud – kokku 4 749 132 tükki.

Nende hulgast sai eemaldatud URI-d, mis ei osutanud veebi (mille alguses polnud http: või https:). Eemaldatud sai ka duplikaadid, sh. loeti duplikaatideks ka URI-d mis erinesid üksnes parameetrite poolest (näiteks URI-d http://example.com/?foo=bar ja http://example.com/?baz=baf). Lisaks sai üritatud eemaldada veel võimalikult palju URI-sid, mis selgelt viitasid HTML-ist erinevas formaadis dokumendile (näiteks lõppesid faililaiendiga jpeg, wav, zip, pdf, doc, ...). Peale kirjeldatud töötlust jäi järgi 4 361 674 URI-d.

Ent ka kõigi nende aadresside järjestikune läbivaatamine oleks programmil võtnud üle poole aasta (arvestades ühe lehekülje jaoks ääretult optimistlikud 4 sekundit). Muidugi, kõik need arvestused olid läbi viidud juba enne programmi kirjutamise juurde asumist ja ka lahendusega oli loodud programmis arvestatud.

Kui Dagfinn Parnas 2001. aastal oma uuringut läbi viis, siis jaotas ta valimi mitmeks võrdse pikkusega lõiguks ja jooksutas paralleelselt valideerimise skripti 30 arvuti peal korraga (igas masinas kaks programmi) – viies nõndaviisi vajamineva aja 6 öö peale (igal ööl töötas programm 6 tundi). Samamoodi sai toimitud ka käesoleva töö puhul.

Kuna minu kasutada oli 17 arvutit (üks arvutilabor), siis sai valim jaotatud 85-ks umbes võrdse suurusega osaks (arvestusega jooksutada programmi igas masinas viies eksemplaris). Eesmärgiks oli valimist läbi vaadata nii suur osa kui võimalik – arvutused näitasid, et kogu valimi läbitöötamine oleks vajanud rohkem aega kui nädal, mida mul aga enam polnud, samuti polnud võimalik rakendada töösse ka rohkem masinaid.

# 5.2. Arvutipargi ettevalmistamine

Kuna loodud programm vajas töötamiseks Linux'it<sup>25</sup>, siis oli tarvis kõigisse 17-sse labori arvutisse lisaks olemasolevale Microsoft Windows XP-le ka üks Linux'i distributsioon installeerida. Valik langes Debian GNU/Linux'ile, mis oli antud ettevõtmise jaoks sobilik mitmel põhjusel. Kuna programmi jooksutamiseks oli tarvilik vaid käsurida ning mõned levinud UNIX-i programmid, siis oli ebaotstarbekas üles seada X-i ja aknahaldureid, mida osad disributsioonid vaikimisi teevad, Debian'i vaikimisi installatsioon paigutab aga üksnes tõeliselt minimaalse komplekti tööriistu<sup>26</sup>. Kuid ehkki olles minimaalne, tegi Debian'i installer suurepärast tööd riistvara tuvastamisel ja kogu süsteemi konfigureerimisel (algselt sai kaalutud ka Gentoo Linux'i kasutamist, ent loobutud just aeganõudva installeerimisprotseduuri pärast). Kolmandaks Debian'i plussiks oli mugav käsurealt kasutatav paketihaldussüsteem, sest peale baassüsteemi installeerimist oli tarvis lisada veel mõned programmid, nagu näiteks *Java Runtime Environment* (JRE), et jooksutada W3C CSS-i validaatorit.

Lisaks nimetatud 17-le masinale oli veel 18. arvuti, kuhu sai samuti Debian Linux installeeritud. Too masin talitles keskse MySQL-i andmebaasiserverina, millega kõik 85 programmi 17-st masinast ühendust võtsid, et sinna läbivaadatud lehekülgedelt kogutud andmed salvestada.

Nagu plaanitud, nõnda tehtud. Kõigisse kirjeldatud arvutitesse sai installeeritud Linux, Kaffe nimeline JRE, libc6-dev (et kompileerida HTML::Parser), g++ (et kompileerida WDG Validaator), libmysqlclient14-dev (et kompileerida DBD::mysql), WDG (X)HTML-i validaator, W3C CSS-i validaator ning CPAN-i kaudu Perl'i moodulid DBI, DBD::mysql, HTML::Parser, URI, CSS::SAC ja Getopt::Declare.

Mingil (nüüdseks ununenud) põhjusel ei õnnestunud aga installeerida kahte väikest Perl'i moodulit Date::Simple ja Time::Simple. Neid kasutas programm selleks, et ennast peale öö

<sup>25</sup> Arvatavasti oleks sobinud ka mõni teine UNIX-i laadne operatsioonisüsteem, aga kuna programm sai arendatud just Linux'i platvormil, siis oli Linux'i kasutamine kõige loomulikum ja kindlam variant.

<sup>26</sup> Autori üllatuseks puudusid Debian'i värskelt installeeritud süsteemist isegi sellised tillukesed programmid nagu less ja vim – küll olid olemas more ning vi.

otsa töötamist välja lülitada, et arvutilaboris võiksid alata loengud. Kuna neid mooduleid tarvitati vaid ühes kitsas programmi osas, siis sai nendest lihtsalt loobutud ja vajalik funktsionaalsus implementeeritud kasutades Perl'i standardseid vahendeid.

Teine probleem ilmnes programmi esmakordsel käivitamisel – selgus, et Debian'i installatsioon sisaldas paar numbrit vanemat Wget-i versiooni, kui see, mis oli kasutusel programmi loomiseks kasutatud arvutis, ning nende väljundite formaat erines üksjagu. Wget-i kasutamine lehekülgede allalaadimiseks oli osutunud veaks, kuid õnneks selliseks, mis sai kergesti parandatud väikse muudatusega regulaaravaldises, millega väljundit töödeldi.

# 5.3. Programmi jooksutamine

Algselt oli plaanis programmi jooksutada aprillikuu esimesel nädalal (3. – 9. aprill), ent programmi valmimine venis ning uuringu läbiviimine nihkus järgmisesse nädalasse, mis tegelikult oli õnnelik kokkusattumus, sest selgus, et 5. aprill oli kuulutatud CSS-i vabaks päevaks<sup>27</sup> ning, kui uurimus oleks läbi viidud just sel ajal, võinuks tulemused oluliselt mõjutatud olla veebimeistritest, kes otsustasid tol päeval oma lehekülgedelt kogu CSS-i eemaldada.

Programmi esimene jooksutamine ööl vastu 11. aprilli tõi välja mitmeid vigu ja probleeme.

Esiteks jooksid paljud programmi eksemplarid kokku CSS-i analüüsimise käigus. Probleemi lähemalt uurides selgus, et CSS::SAC moodulile olid lisamata jäänud eelnevalt loodud parandused. Peale mooduli paikamist, probleem lahenes.

Teine probleem oli, et osad programmi eksemplarid katkestasid töö JavaScript'i analüüsimise käigus, andes veateateks mitte just eriti informatiivse "Segmentation fault". Kuna vea põhjust ei õnnestunud esmase kiire otsimise käigus tuvastada, sai langetatud otsus, et JavaScript'i koodi analüüsimisest loobutakse hoopis ja salvestatakse üksnes info erineval

<sup>27</sup> *CSS Naked Day* pealkirja all kutsus Dustin Diaz veebimeistreid 5. aprillil eemaldama oma lehekülgedelt kogu CSS, näitamaks maailmale, kui oluline on struktuurselt kirjutatud (X)HTML-i kasutamine. http://naked.dustindiaz.com/

teel (X)HTML-ga seotud JavaScript'i mahu kohta. Ainukese koodi analüüsimise meetmena sai sisse võetud identifikaatori XMLHttpRequest tuvastamine.

Peale loetletud ja mõningate teiste pisemate probleemide parandamist sai andmebaasist kõik eelnevalt kogutud andmed eemaldatud ning alustatud programmi jooksutamisega uuesti puhtalt lehelt. Järgneval ööl (vastu 12. aprilli) töötas programm ilma oluliste tõrgeteta.

Kuid olles saanud piisavalt kindlalt tööle programmi, vedas alt võrk. Vähemalt paar tundi enne, kui programm hommikul oma töö lõpetas, kadus arvutilaborist ja vististi ka kogu TLÜ Haapsalu Kolledžist internetiühendus, mis kajastus kenasti ka läbitöötatud lehekülgede statistikas – üle sajatuhande URI oli kantud andmebaasi ebaõnnestunud allalaadimise sildi all. Uurimise tulemust see ei mõjutanud, üksnes tol öösel edukalt analüüsitud lehekülgede arvu.

Ilmnes ka üksjagu tõsisem probleem ning taas seoses JavaScript'iga. Eelmisel päeval tehtud muudatus JavaScript'i koodi analüüsimisse oli hõlmanud ka tillukest muudatust andmebaasistruktuuris, kuid ununenud oli muuta ka andmebaasi info salvestamiseks kasutatud päringut. Kuna programmi andmebaasiliideses olid veateated peale arendusfaasi lõppu maha keeratud (vältimaks ekraanide täitumist hunniku teadetega selle kohta, et üks või teine päring ebaõnnestus, mis kuulus programmi normaalse töö juurde), siis jäi koodi parandamise käigus tekkinud probleem avastamata. Praktikas tähendas see kõik seda, et kogu info, mis sellel ööl JavaScript'i kohta kogutud jäi andmebaasi salvestamata.

Sellest kõigest hoolimata sai siiski otsustatud peale vea parandamist uuringut jätkata poolelijäänud seisust, sest JavaScript'i kasutamise osakaal oli võimalik välja arvestada ka üksnes (X)HTML-i elementide ja atribuutide statistika põhjal ning JavaScript'i failide suuruse ning muu säärase osas polnud oluline, et uuritaks absoluutselt kõigilt valimis olnud lehekülgedelt nopitud JavaScript'i koodi.

Kuid sellega veel probleemid ei piirnenud. Programmi teistkordsel käivitamisel lõpetas operatsioonisüsteem peagi tema tegevuse veateatega "*Out of memory*". Lähemal uurimisel

selgus, et põhjuseks oli liiga suure hulga andmete lugemine andmebaasipuhvrisse.

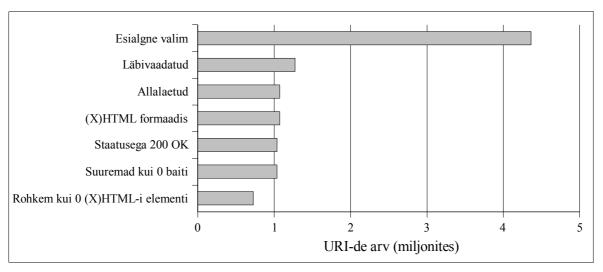
Nimelt oli webstat kirjutatud nõnda, et programm luges käivitamisel andmebaasist kõik sinna kogunenud (X)HTML-i elementide, atribuutide, CSS-i atribuutide jms nimed puhvrisse. Programmi esmakordsel käivitamisel see probleeme ei tekitanud, sest andmebaas oli tühi, samuti ei tekkinud probleeme programmi testimise käigus, sest andmemahud olid väikesed. Kuid esimese ööga oli programm läbi vaadanud oma 300 000 lehekülge ning avastanud neilt tervelt 20 000 erinevat (X)HTML-i elemendi nime – kuid need olid üksnes elementide nimed – puhvrisse loeti ka atribuudid, atribuutide ja elementide kombinatsioonid ning elementide, atribuutide ja nende väärtuste kombinatsioonid. Polnud imestada, et ühe arvuti mälu (500MB) ei mahutanud kogu seda infot, mida 85 programmi olid suutnud kokku koguda.

Probleem sai lahendatud puhvri eellaadimise eemaldamisega programmist (selleks oli tarvis üksnes paar rida paaris moodulis välja kommenteerida).

Järgnevalt sai programmi jooksutatud ööl vastu 13. aprilli ning seejärel neljapäeva (13. aprill) õhtust esmaspäeva (17. aprill) hommikuni. Just nõnda pikk mitmepäevane jooksutamine oli võimalik seetõttu, et reede 14. aprill oli Suur Reede ja seega õppetööst vaba päev.

### 6. TULEMUSED

# 6.1. Ebasobilike lehekülgede eemaldamine valimist enne analüüsi



Joonis 2: Kui palju lehekülgi kogu esialgsest valimist jõuti läbi vaadata, ning kui paljud viimaks vastasid kõigile kriteeriumitele: alla laetud, (X)HTML formaadis, staatusega 200 OK, suuremad kui null baiti ja vähemalt üks element.

Programm jõudis ette antud 4 361 674 URI-st läbi vaadata 1 272 764 ehk 29%. (Vaata joonis 2.)

Nendest 199 877 URI puhul allalaadimine ebaõnnestus – peamiseks põhjuseks nii suure hulga lehekülgede kättesaamata jäämises oli uurimuse esimesel ööl toimunud internetiühenduse katkestus. Reaalselt alla laetud lehekülgi oli seega 1 072 887.

Kuna uuring käsitles just veebilehti, siis tuli eemaldada URI-d, kus Content-Type päis oli midagi muud kui text/html, application/xhtml+xml, application/xml või text/xml – vaid need võisid osutada (X)HTML vormingus dokumentidele. Selgus, et allalaetud URI-dest olid (X)HTML vormingus 1 072 146.

Seejärel tuli eemaldada ka leheküljed, kus HTTP staatuskood polnud 200 OK, sest uuringu seisukohalt olid olulised vaid harilikud leheküljed, mitte näiteks veateate lehed, mis ütlesid

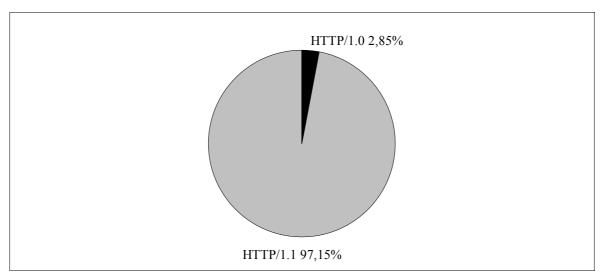
oma külastajatele vaid "404 Not Found". Peale nimetatud toimingu sooritamist jäi veebilehti alles 1 037 832.

Lisaks tuli valimist välja praakida veel ka täiesti tühjad lehed: mahuga null baiti. Alles jäi 1 036 685 lehekülge, mille suurus oli üle nulli baidi.

Viimaks tuli eemaldada ka leheküljed, mis ei sisaldanud ühtki (X)HTML-i elementi. Selgus, et lehekülgi, mis tõesti ka (X)HTML-i sisaldasid, oli vaid 725 984. Kuid enamik lehtedest, mis ühtki (X)HTML-i elementi uuringu andmetel ei sisaldanud, olid täiesti korralikud veebilehed, seega oli tegemist veaga veebilehti analüüsinud programmis – miks programm ei suutnud neil lehtedelt aga ühtegi elementi leida, jääb autorile mõistatuseks.

Kõigis järgnevates analüüsides on kasutatud vaid neilt 725 984-lt leheküljelt kogutud andmeid.

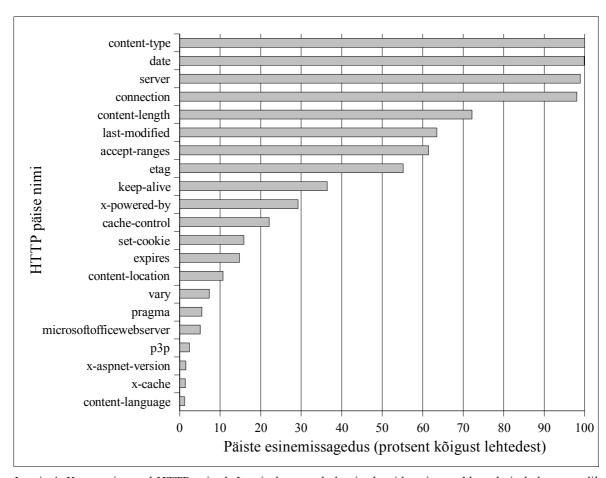
#### 6.2. HTTP



Joonis 3: HTTP 1.0 ja 1.1 kasutatavuse võrdlus – enamik veebiservereid kasutavad HTTP/1.1 protokolli.

97,15% läbivaadatud lehekülgedest jookseb HTTP/1.1 protokolli kasutava serveri peal, mis on ühtlasi kõige uuem HTTP protokolli standard. Vaid 2,85% kasutab veel HTTP/1.0 protokolli. (Vaata joonis 3.)

## 6.2.1. Päised



Joonis 4: Kasutataivamad HTTP päised. Joonisel on toodud päised, mida esines rohkem kui ühel protsendil lehtedest.

Levinuimate HTTP päiste nimed on toodud joonisel 4. Loomulikult esines kõigil lehtedel content-type päis, sest leheküljed, millel see puudus olid eelnevalt valimist valja praagitud. Populaarsuselt järgmised kolm – date, server ja connection – esinesid praktiliselt kõigil lehtedel. Märksa vähem leidus aga content-length päist. Arvatavasti on põhjus selles, et dünaamiliselt genereeritud lehtede puhul ei oska server öelda, kui suur loodav lehekülg saab olema, eriti juhul kui lehekülge hakatakse edastama varem, kui kogu kood on valmis genereeritud.

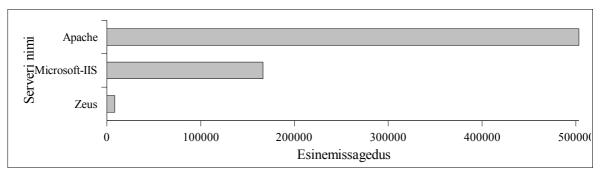
Vähemalt 15% lehtedest kasutab set-cookie päist, kuid küpsiseid kasutavate lehtede arv

on kindlasti suurem, sest levinud praktikaks on küpsise seadmine leheküljel A ning ümbersuunamine leheküljele B, kus kontrollitakse küpsise olemasolu. Käesolev uurimus aga kogus kokku vaid päised lehtedelt, mis olid ümbersuunamiste lõplikuks sihtkohaks.

## 6.2.1.1. Content-Type päis

Vaid 112 lehekülge (0,015%) kasutas application/xhtml+xml content-type'i, application/xml oli kasutuses neljal ning text/xml kahel lehel. Kõiki ülejäänuid lehti serveeriti text/html sisutüübiga.

## 6.2.1.2. Server päis



Joonis 5: Apache ja Microsoft Internet Information Services on HTTP päise server põhjal ülekaalukalt kõige levinumad serverid.

HTTP päist nimega server kasutab veebiserveri tarkvara reeglina oma nime teadvustamiseks. Selle info põhjal on populaarseim veebiserver Apache<sup>28</sup>, teisel kohal Microsoft Internet Information Services<sup>29</sup>. Kolmandal kohal oleva Zeus'i<sup>30</sup> kasutamise osakaal on nende kõrval praktiliselt olematu. (Vaata joonis 5.)

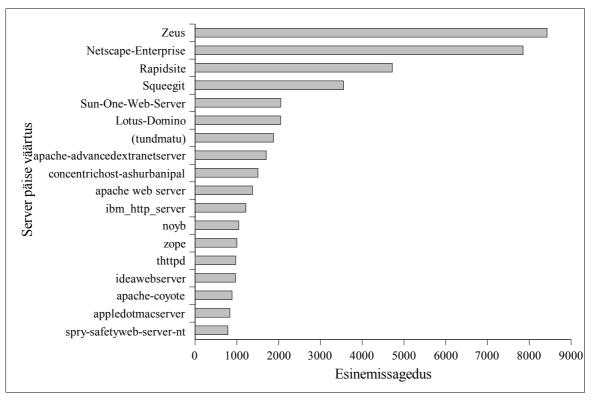
Joonisel 6 leheküljel 49 on toodud 18 populaarsemat veebiserverit, mis jäävad kahest mainitud gigandist tahapoole. Äramärkimist väärib, et näiteks RapidSite<sup>31</sup> näol pole tegemist mitte veebiserveritarkvaraga vaid hoopis virtuaalserveri teenusepakkujaga.

<sup>28</sup> http://httpd.apache.org

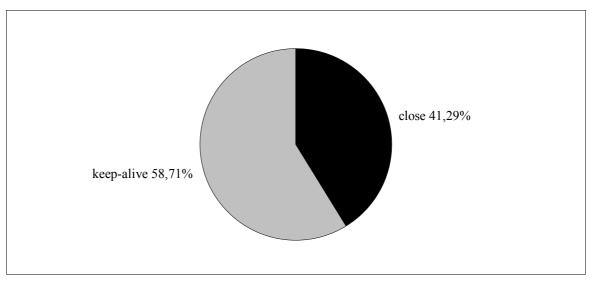
<sup>29</sup> http://www.microsoft.com/WindowsServer2003/iis/default.mspx

<sup>30</sup> http://www.zeus.com/

<sup>31</sup> Oma kodulehe <a href="http://www.rapidsite.net/about.shtml">http://www.rapidsite.net/about.shtml</a> andmetel on RapidSite maailma suurim hostingupakkuja, tagades veebimajutuse enam kui 100 000-le kodulehele.



Joonis 6: Populaarseimad server päise väärtused Apache ja Microsoft-IIS järel.



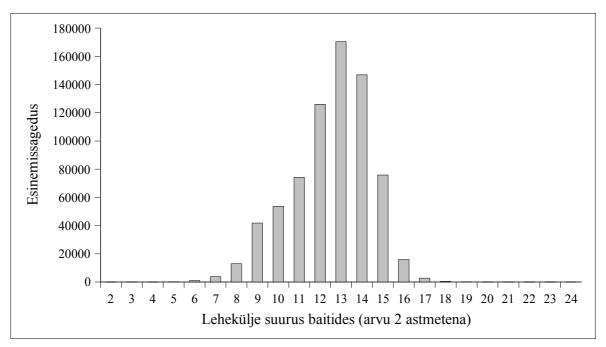
Joonis 7: Connection päise väärtused. Lisaks joonisel toodutele leidus ka teisi Connection väärtusi, ent kõik need kokku moodustasid alla 0,01% ning polnud seega mõtet joonisele märkida.

## 6.2.1.3. Connection päis

Neljas levinuim HTTP päis oli connection. Selle päise puhul kasutati peaasjalikult kahte väärtust: keep-alive ning close. Nagu näha jooniselt 7 leheküljel 49, oli ülekaalus väärtus keep-alive.

## 6.3. (X)HTML

## 6.3.1. Veebilehtede maht



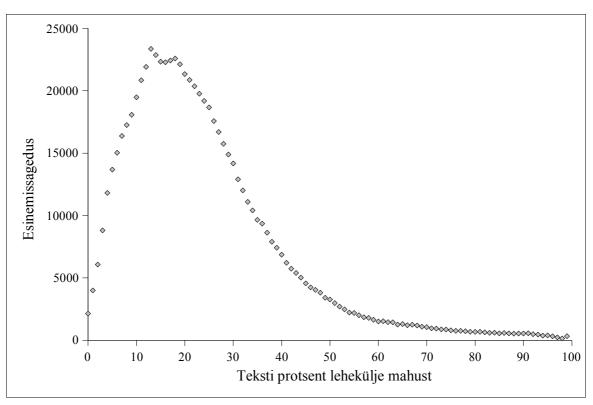
Joonis 8: Veebilehed (X)HTML-i suuruste lõikes. Joonise horisontaalteljel on antud lehekülje suuruse logaritm alusel 2. Iga tulp kohal n näitab seda, kui palju on veebilehti suurusega x, kui x on vahemikus  $2^n \le x < 2^{n+1}$ .

Nagu näitab joonis 8, jääb enamuse uuringus osalenud veebilehtede suurus ((X)HTML faili suurus) vahemikku 256 baiti kuni 128 KB (ehk 2<sup>8</sup> kuni 2<sup>17</sup> baiti). Kõige enam on lehti vahemikus 8 KB kuni 16 KB (ehk 2<sup>13</sup> kuni 2<sup>14</sup> baiti). Keskmine lehekülje suurus on 16KB.

Ülaltoodud veebilehtede (X)HTML-i maht on arvestatud vaid ühe allalaetud (X)HTML faili järgi, seda ka raame (*frames*) kasutavate lehtede puhul. Kuna raame kasutavad

joonise 13 (leheküljel 55) andmetel tervelt 17% lehtedest<sup>32</sup>, siis on suure tõenäosusega joonis 8 nihutatud väiksemate mahtude suunas.

### 6.3.2. Teksti osakaal veebilehtedel



Joonis 9: Veebilehtede jaotus teksti osakaalu põhjal. Enamikul veebilehtedest moodustab tekst alla 40% (X)HTML faili suurusest.

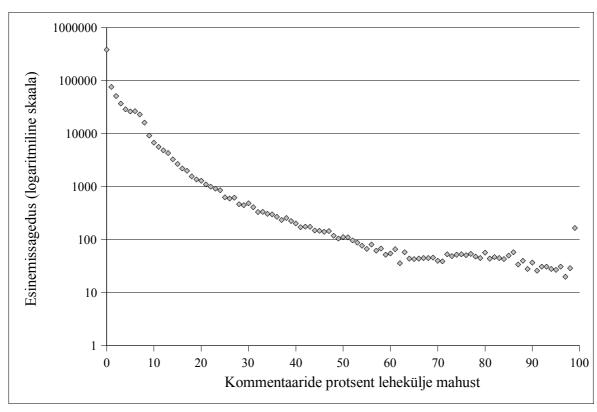
Enamikel lehtedest on (X)HTML märgendeid rohkem kui lehekülje sisu (teksti) ennast. Keskmisel leheküljel on teksti osakaal 26%. (Vaata joonis 9.)

### 6.3.3. Kommentaaride osakaal

Ligikaudu pooltel lehtedest (53%) ei kasutata kommentaare üldse. Leheküljed, kus kommentaare leidub, jääb nende maht enamasti alla 10%. Keskmine kommentaaride maht kõigi lehtede lõikes on 3,20%. Keskmine kommentaaride maht lehekülgedel, kus kommentaare üldse esineb on 5,19%. Joonis 10 leheküljel 52 näitab, kuidas leheküljed jaotuvad

<sup>32</sup> Elementide frame ja iframe osakaalu summa.

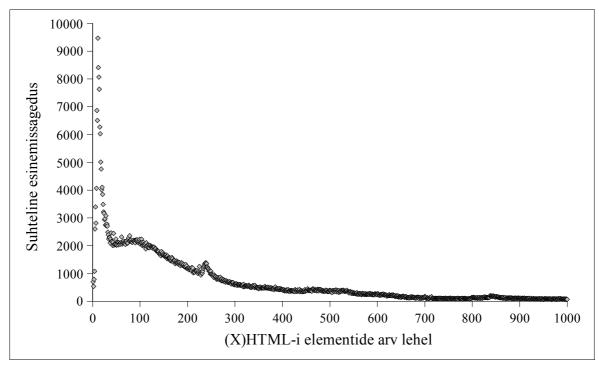
neil olevate kommentaaride osakaalu põhjal.



Joonis 10: Lehekülgede jaotus (X)HTML kommentaaride osakaalu põhjal. Kuna lehekülgi, kus kommentaare on väga vähe on kümneid kordi rohkem kui neid, kus kommentaarid moodustavad 10%, 20% jne, on kasutatud logaritmilist skaalat, et tuua välja, mis toimub skaala alumises otsas.

### 6.3.4. Elemendid

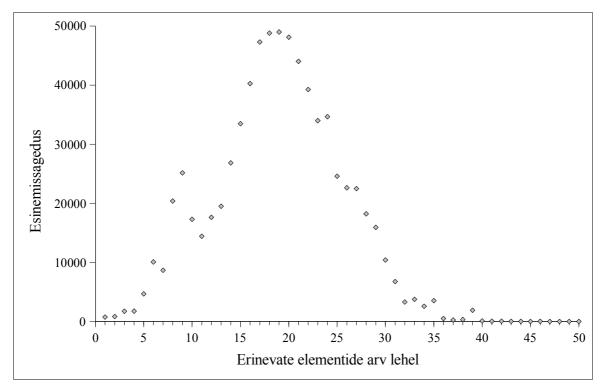
Järgnev peatükk vaatleb kuidas leheküljed kasutavad (X)HTML elemente. Avaldatud tulemused langevad suuresti kokku Google Inc. poolt läbi viidud uuringuga, mida sai kirjeldatud peatükis 2.5 leheküljel 29. Kuna Google'i uurimuses kasutatud valim oli ligikaudu tuhat korda suurem kui käesoleva uurimuse oma, siis võib nende tulemusi lugeda märksa tõepärastemaks kui käesoleva töö omi. Sellegipoolest on esitatud mitmed Google'i uurimusega praktiliselt üks-ühele võrreldavad joonised ning ära toodud ka viited Google'i vastavasisulistele joonistele ning kommentaarid erinevuste kohta. Seda just seetõttu, et lugejal oleks mugav neid kahte uuringut kõrvutada.



Joonis 11: (X)HTML Elementide koguarv lehel. Olgugi, et joonise X-telg algab nullist, on joonisele kantud siiski vaid leheküljed, millel leidus rohkem kui 0 elementi (arvu 0 on skaala alguses kasutatud tarkvara tehniliste piirangute tõttu). Sarnane joonis Google'i uuringus: <a href="http://code.google.com/webstats/2005-12/pages.html">http://code.google.com/webstats/2005-12/pages.html</a>

Joonis 11 näitab, kui palju elemente veebilehtedel enamasti kasutatakse. Kõrvutades seda Google'i joonisega võib öelda, et erinevad jõnksud tollel joonisel tulenevad kasutatud valimi omapärast ja väiksusest – Google'i avaldatud graafikul on lehekülgede arvu langus elementide arvu kasvades tunduvalt sujuvam. Kuid mõlemad tulemused näitavad, et üks keskmine veebileht kasutab alla 500 elemendi.

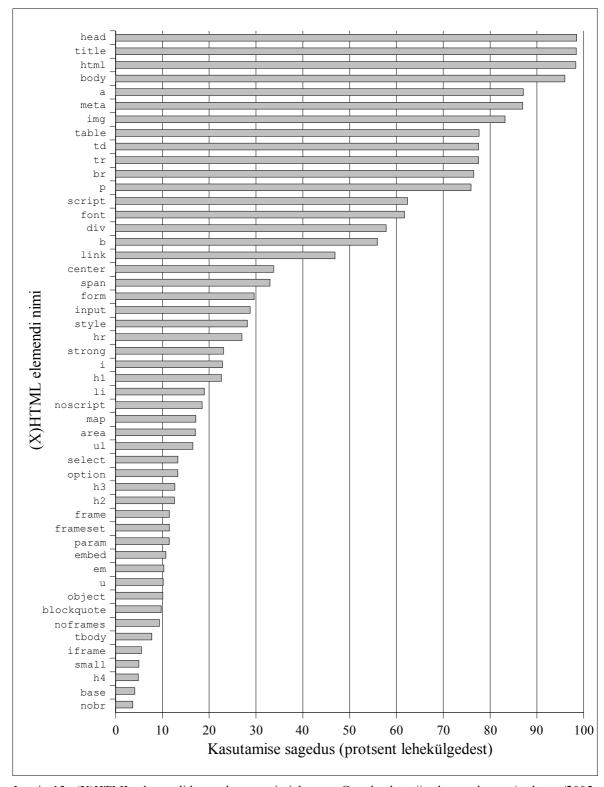
Märksa ilmekamad on tulemused, kui me vaatame kui palju *erinevaid* elemente lehe-külgedel kasutatakse. Joonis 12 leheküljel 54 näitab sarnaselt Google'i uuringuga, et enamike lehtedel on 13-27 erinevat elementi. Kõige sagedasemini 19 erinevat elementi. Nagu jooniseltki selgesti näha (veelgi ilmekamalt näitab seda Google'i joonis), jaotuvad leheküljed erinevate elementide arvu alusel normaaljaotuse reeglite kohaselt – nii väga vähe kui ka väga palju elemente sisaldavad leheküljed on haruldased.



Joonis 12: Erinevate elementide arv lehel. Google: http://code.google.com/webstats/2005-12/pages.html

Millised siis on need elemendid, mida kasutatakse? 50 populaarsemat on ära toodud joonisel 13 leheküljel 55. Üldjoontes langeb joonis kokku Google Inc. poolt avaldatud uuringu tulemustega (seda suuremal või vähemal määral esimese 19 elemendi puhul, sest Google piirdub vaid 19 populaarseima elemendi nimetamisega).

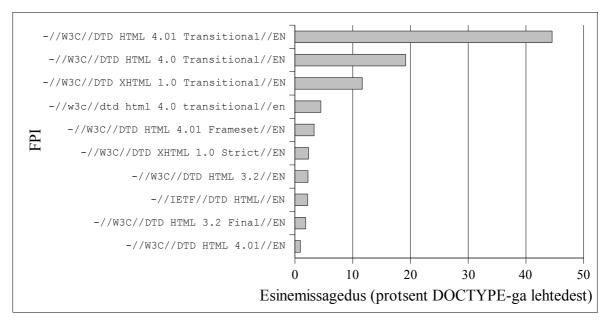
Kuna Google'i uurimus, millele see peatükk viitamast ei väsi, käsitleb põhjalikult levinumaid (X)HTML-i elemente, siis ei hakka käesolev töö seal kirjutatut kordama, vaid soovitab huvitatud lugejal suunduda iseseisvalt aadressile http://code.google.com/webstats/ning sealsete materjalidega tutvuda. Küll aga proovib käesolev töö järgnevalt puudutada (X)HTML-i kasutamise neid tahke, mis Google'i uuringutulemustes katmata jäävad, näiteks saab joonise 13 põhjal öelda, et Flash tehnoloogiat kasutavad vähemalt 10% lehtedest (elemendid embed, object ja param).



Joonis 13: (X)HTML elemendid populaarsuse järjekorras. Google: <a href="http://code.google.com/webstats/2005-12/pages.html">http://code.google.com/webstats/2005-12/pages.html</a> ja <a href="http://code.google.com/webstats/2005-12/elements.html">http://code.google.com/webstats/2005-12/elements.html</a>

#### 6.3.5. Dokumenditüübid

Vaid 39,08% lehtedest kasutab dokumenditüübi deklaratsiooni (DOCTYPE). 66,47% DOCTYPE'i kasutajatest jätab aga määramata DTD-le viitava URL-i (SI).



Joonis 14: Levinuimad FPI-d. Selge ülekaal on Transitional perekonna dokumenditüüpidel.

Levinuimaks dokumenditüübiks on ülekaalukalt HTML 4.01 Transitional, mida kasutab üle 40% lehtedest, sellele järgneb HTML 4.0 Transitional (ligikaudu 19%) ning kolmandal kohal on XHTML 1.0 Transitional. (Vaata joonis 14.) Selline dokumenditüüpide järjekord langeb täpselt kokku minu Eesti lehekülgedel läbi viidud uuringutega (vrd. Saarsoo, 2006b).

Joonisel 14 toodud FPI-de seas on neljandal kohal väiketähtedega kirjutatud HTML 4.0 Transitional'i FPI – mis seetõttu pole ka korrektne. Autor ei oska öelda, mis võiks olla põhjuseks, et nõnda paljud lehed sellist vigast DOCTYPE'i kasutavad.

Kasutades Henri Sivonen'i veebilehel olevat tabelit (Sivonen 2005), sai uuritud, kui paljude lehekülgede puhul kasutavad brauserid nn. *Quirks Mode* režiimi ja kui paljudel juhtudel *Standards Mode* režiimi. Aluseks sai võetud IE 6, kui maailma enimkasutatava

brauseri käitumine. (Sivonen'i tabelis on küll eristatud *Standards Mode* ja *Almost Standards Mode*, kuid kuna IE kasutab vaid *Almost Standards Mode* režiimi, siis viitame sellele siinkohal kui *Standards Mode*'le.)<sup>33</sup>

Selgus, et lisaks 60% lehtedest, mida kuvatakse *Quirks Mode*'is seetõttu, et neil pole üldse DOCTYPE'i määratud, näidatakse ka DOCTYPE'i määranud lehtedest ligikaudu 70% *Quirks Mode* režiimis. Seega vaid ligikaudu 10% lehtedest on kuvatavad *Standards Mode* režiimis.

#### 6.3.6. XHTML

15,24% DOCTYPE'i määravatest lehtedest kasutab XHTML dokumenditüüpi. Tervelt 22,40% nendest lehtedest kasutavad ka XML-i proloogi (palju on seda seetõttu, et selle kasutamine tähendab IE puhul automaatset *Quirks Mode*'i lülitumist). Kuid ka HTML dokumenditüübiga lehtedest kasutab XML-i proloogi tervelt 0,29%.

W3C soovituse kohaselt (XHTML Media Types 2002) tuleks XHTML dokumente serveerida kasutades HTTP päise content-type väärtusena application/xhtml+xml, kuid võib kasutada ka application/xml või text/xml. Erandina: XHTML 1.0 dokumente võib serveerida ka text/html kujul, ent teisi XHTML perekonna liikmeid ei tohiks.

XHTML-i kasutavatest lehtedest vaid 0,24% on serveeritud application/xhtml+xml, application/xml või text/xml kujul. Kõik ülejäänud text/html-na. Muidugi, XHTML 1.0 puhul on see lubatud, aga kuidas on lood siis, kui jätame XHTML 1.0 dokumendid vaatluse alt välja?

Dokumente, mis kasutavad teisi XHTML DOCTYPE'e oli valimis 1930 ehk 4,46% kõigist XHTML-i kasutajatest. Kuid ka nendel lehtedel pole olukord palju parem – vaid 2,80% (valimis 54 lehekülge) neist on serveeritud appliction/xhtml+xml kujul (kõik ülejäänud

<sup>33</sup> Täpsem jutt Standards Mode, Almost Standards Mode ja Quirks Mode erinevustest on teadlikult käesolevast tööst välja jäetud. Lugeja võib leida lisainfot sellestsamast Sivonen'i tööst. Lühidalt öeldes järgivad brauserid Standards Mode puhul täpselt W3C standardeid, Quirks Mode on aga tagaspidi ühilduv (bacwards compatible) režiim, mille puhul näiteks mõningad CSS-i atribuudid käituvad erinevalt.

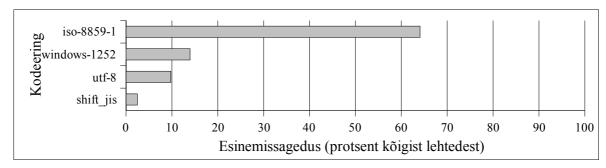
olid text/html kujul).

Lehekülgedel, mis appliction/xhtml+xml kujul serveeritud, on populaarseimaks DOCTYPE'iks XHTML 1.1, millele järgnevad XHTML 1.0 Strict ja Transitional.

## 6.3.7. Kodeeringud

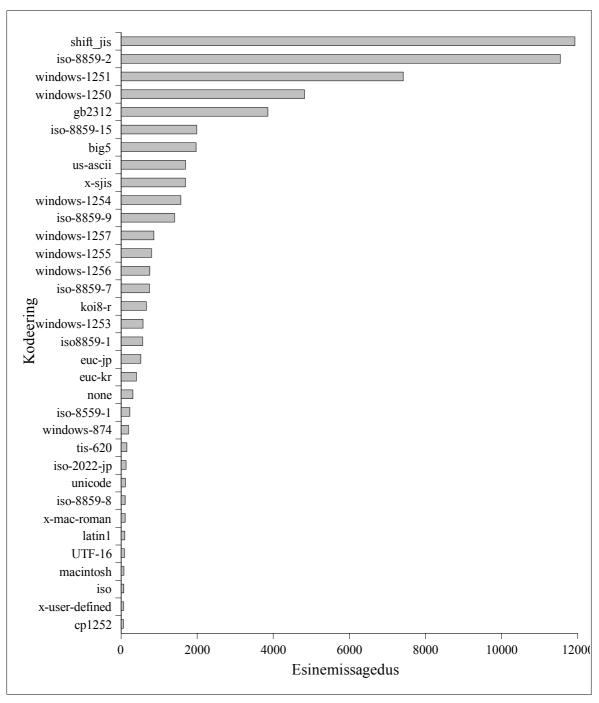
Vaid 66% veebilehtedest määrab oma kodeeringu ühel või teisel moel, olgugi et HTML 4.01 spetsifikatsioon ütleb "user agents must not assume any default value for the "charset" parameter" (lihtsustatult: "brauserid ei tohi eeldada mingit vaikimisi kodeeringut"). Erinevalt spetsifikatsioonist enamik brausereid seda aga siiski teeb, kasutades reeglina Windows-1252 kodeeringut.

Vaid 15% kõigist lehtedest kasutab kodeeringu määramiseks HTTP content-type päise charset atribuuti, 51% lehtedest kasutab selleks aga (X)HTML-i meta elementi. Seejuures nendest lehtedest, mis määravad kodeeringu HTTP päises, 61% määrab selle ka (X)HTML-i meta elemendiga.



Joonis 15: Levinuimad kodeeringud.

Joonis 15 annab ülevaate levinuimatest kodeeringutest, joonis 16 leheküljel 59 loetleb vähemlevinud. Kodeeringute statistika on koostatud nõnda, et kõrgeima prioriteediga oli HTTP content-type päisega määratud kodeering, sellest madalam prioriteet oli (X)HTML-i meta elemendis esitatud kodeeringul ja kõige madalam XML-i proloogis oleval kodeeringul.



Joonis 16: Vähemlevinud kodeeringud

Ülekaalukalt levinuimaks kodeeringuks on ISO 8859-1, mille asemel aga enamik brausereid kasutab Windows-1252 kodeeringut, mis aga pole midagi muud, kui ISO 8859-1 ülemhulk (Wikipedia, Windows-1252). Sageduselt teisel kohal ongi Windows-1252.

Populaarsuselt kolmas on Unicode'i kodeering UTF-8 ning neljas Shift\_JIS – jaapani keele jaoks kasutatav kodeering.

Viiendal kohal olev ISO 8859-2 (vaata joonis 16 leheküljel 59) on kasutatav paljude Euroopa keelte puhul. Kuues, Windows-1251 on populaarseim spetsiaalselt kirillitsa jaoks mõeldud kodeering. Järgmine, Windows-1250 on ISO 8859-2 ülemhulk, kuid mõned mõlemas kodeeringus leiduvad tähed ei vasta samadele koodipunktidele (Wikipedia, Windows-1250). GB 2312 on kodeering hiina keele tarbeks. ISO 8859-15 on sarnane kodeering ISO 8859-1'ga asendades viimases mõned sümbolid. Big5 on jällegi kasutusel hiina keele puhul.

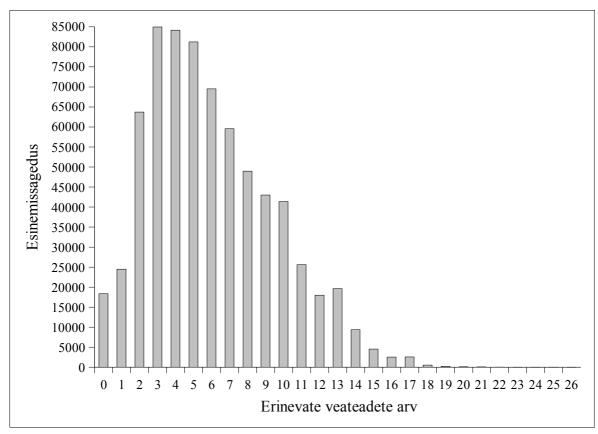
#### 6.3.8. Koodi korrektsus

Kõik (X)HTML lehed, mis uuringu käigus alla laeti ning mille HTTP staatuskood oli 200 OK valideeriti kasutades valideerivat SGML-i parserit. 3,50% lehekülgede valideerimine ebaõnnestus veateate "unrecognized DOCTYPE; unable to check document" tõttu, mille põhjuseks võis olla kas vigane dokumenditüübi deklaratsioon või mõne spetsiifilise DTD kasutamine.

Dagfinn Parnas otsustas oma uuringus (2001) vastava veateatega dokumendid arvata välja nii valideeruvate kui ka mitte valideeruvate hulgast, sest pole võimalik kergesti eristada omaloodud dokumenditüüpide kasutamist vigastest DOCTYPE'dest, ning kuna SGML parser lõpetas peale tundmatu DOCTYPE'i leidmist dokumendist vigade otsimise, siis mõjutanuks see ka vigade arvu statistikat. Samamoodi sai toimitud ka käesolevas uuringus.

Selgus, et 1 002 350-st allalaetud leheküljest, mille staatuskood oli 200, valideerus vaid 25 890 ehk 2,58%. Seda on küll vähe, ent siiski mitu korda rohkem kui 0,71%, mis Parnas sai analoogse metoodikaga tulemuseks 2001. aastal. (Siinkohal toodud protsendi arvutamisel sai võimalikult täpselt püütud Parnas'e metoodikat jäljendada.) Parnas'e toodud protsenti võib lugeda täpsemaks, sest tolles uuringus vaadati läbi ligikaudu kaks korda rohkem URI-sid kui käesoleva uuringu käigus. Teisalt oli aga Parnas'e metoodikas ka üks olulisem kitsaskoht.

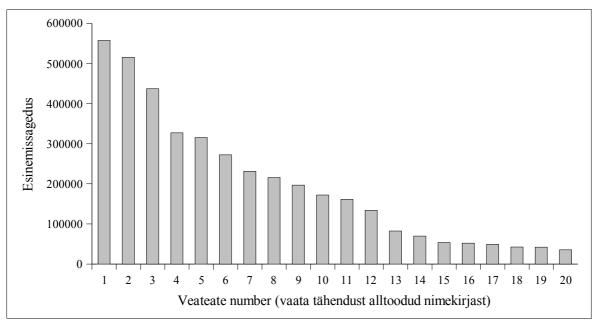
Nimelt ei vaadeldud Parnas'e läbiviidud uuringus, millise HTTP Content-Type päisega lehekülge koos serveeriti. Seega oli võimalik, et server saatis (X)HTML faili asemel hoopis pildi koos päisega Content-Type: image/jpeg, mida ükski brauser (X)HTML-na ei tõlgendaks, kuid tolles uuringus seda ei arvestatud ning kõiki URI-sid, mis õnnestus alla laadida, prooviti valideerida – kui aga tegu polnud SGML formaadis failiga, siis see kindlasti valideeruda ei saanud, tõstes seeläbi mittevalideeruvate lehtede osakaalu.



Joonis 17: Erinevate veateadete arv veebilehtedel. Kõige sagedasemad on kolme erinevat tüüpi veaga lehed.

Seetõttu sai uuritud, kui suur on valideeruvate lehtede osakaal siis, kui valimist eemaldada URI-d, mis ennast (X)HTML formaadile kohase Content-Type päisega ei tutvustanud, või mis ei sisaldanud ühtegi (X)HTML-i elementi (näiteks harilikud tekstifailid). Tulemused olid paremad, ent mitte oluliselt – 702 723 URI-st valideerusid 18 373 ehk 2,61% (vrd. eeltoodud 2,58% ning Parnas'e 0,71%).

Nagu on näha jooniselt 17 leheküljel 61, on sagedaseimad 3-5 veateatega lehed. Keskmine erinevate vigade arv lehel on 6,02 ning mediaan on 6. Parnas'e uuringus oli keskmine vigade arv 5,02 ning mediaan 5.



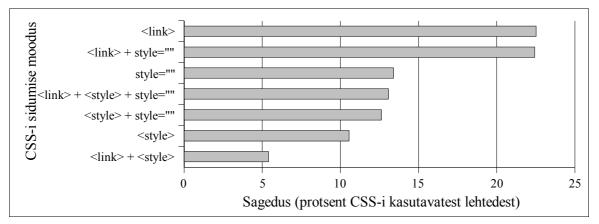
Joonis 18: Levinuimad veateated.

Joonisel 18 antud veateadete numbritele vastavad järgmised veateated:

- 1. Kasutatakse Ebastandardset atribuuti. (*there is no attribute x for this element (in this HTML version*))
- 2. Puudub kohustuslik atribuut. (*required attribute x not specified*)
- 3. Puudub dokumenditüübi deklaratsioon. (missing document type declaration)
- 4. Lõpumärgend elemendile, mida pole alustatud. (*end tag for element x which is not open; try removing the end tag or check for improper nesting of elements*)
- 5. Element asub keelatud kontekstis. (element x not allowed here; check which elements this element may be contained within)
- 6. Puudub lõpumärgend. (end tag for x omitted; possible causes include a missing end tag, improper nesting of elements, or use of an element where it is not allowed)

- 7. Element asub keelatud kontekstis blokitasandi element reatasandi elemendi sees. (*element x not allowed here; possible cause is an inline element containing a block-level element*)
- 8. Tundmatu olem (näiteks &foo;) (unknown entity x)
- 9. Tundmatu element. (*element x not defined in this HTML version*)
- 10. Atribuudi väärtus pole ümbritsetud jutumärkidega. (an attribute value must be quoted if it contains any character other than letters (A-Za-z), digits, hyphens, and periods; use quotes if in doubt)
- 11. Tundmatu atribuudi väärtus. (*value of attribute x cannot be y; must be one of a, b, c* ... )
- 12. Puudub kohustuslik alamelement. (missing a required sub-element of x)
- 13. Väärtus ei kuulu ühegi atribuudi poolt määratud gruppi. (*x is not a member of a group specified for any attribute*)
- 14. Keelatud sümbol. (*illegal character number x*)
- 15. Tekst pole selles kontekstis lubatud. (text is not allowed here; try wrapping the text in a more descriptive container)
- 16. Atribuudi kahekordne määramine. (duplicate specification of attribute x)
- 17. Element asub keelatud kontekstis puudub vajalik ülemelement. (*element x not allowed here; assuming missing y start-tag*)
- 18. Vigane kommentaar. (invalid comment declaration; check your comment syntax)
- 19. Atribuudi väärtus peab koosnema ühest *token*'ist. (*value of attribute x must be a single token*)
- 20. Atribuudi nimi peab algama nime või *token*'iga. (*an attribute specification must start with a name or name token*)

### 6.4. CSS

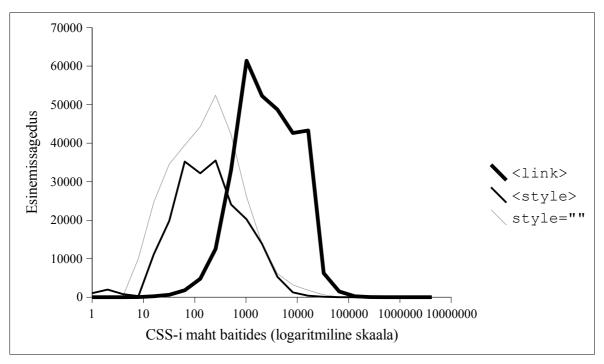


Joonis 19: Moodused CSS-i sidumiseks veebilehega. Joonisel olev tekst link> tähistab väliste stiililehtede
kasutamist (mitte üksnes läbi link elemendi vaid ka style elemendi sees @import ät-reeglit kasutades).

67,20% vaadeldud lehtedest kasutab CSS-i ühel või teisel moel. Kuid *kuidas* seda kasutatakse? Eeskätt, kuivõrd lingitakse väliseid stiililehti (elemendi link abil), kuivõrd tarvitatakse elementi style ja kui paljudel juhtudel lisatakse stiilideklaratsioonid style atribuudi läbi? Vastused nendele küsimustele annab joonis 19, mille pealt on näha, et praktiliselt sama palju, kui on lehekülgi, mis kasutavad üksnes väliseid stiililehti on lehekülgi, mis lisaks välistele kasutavad ka style atribuuti. Leheküljed, mis ühel või teisel moel kasutavad väliseid stiililehti (parimat praktikat), moodustavad 63,43% kõigist CSS-i tarvitavatest lehtedest, ent pea samapalju (61,51%) on lehti mis kasutavad style atribuudiga stiilide määramist (halvimat praktikat).

Rääkides välistest CSS-i failidest, siis 72,2% lehtedest, mis välist CSS-i tarvitavad, lingivaid vaid ühte välist stiililehte, ent tervelt 17,9% kasutavad kahte CSS-faili, 4,0% kolme ning 3,9% nelja. Ülejäänud ligikaudu 2% kasutavad rohkem kui nelja CSS-i faili.

Nagu selgub jooniselt 20 leheküljel 65, on väliste CSS failide maht enamasti suurem kui (X)HTML faili sisse paigutatud CSS-i maht. Jooniselt on näha, kuidas CSS-i mahu osas nii style elemendi kui style atribuudi kasutamine järgib sarnast mustrit, kuid välise CSS-i tarvitamine kaldub suuremate mahtude poole.



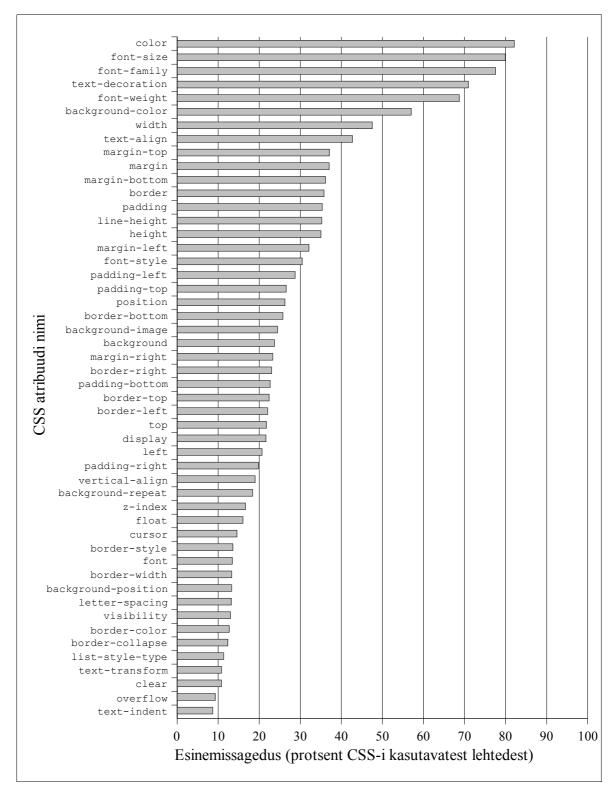
Joonis 20: Erineval moel leheküljega seotud CSS-i maht. Väliste stiililehtede maht on enamasti suurem kui (X)HTML faili sisse paigutatud CSS-i puhul.

### 6.4.1. Atribuudid

Joonise 21 (leheküljel 66) põhjal võib öelda, et kõige rohkem tarvitatakse CSS-i tema tüpograafiliste vahendite pärast. Kõige levinum on teksti värvi, -suuruse, -fondi, -allajoonimise ja -rasvasuse määramine (atribuudud color, font-size, font-family, text-decoration ja font-weight). Kuid atribuut font, mis lubab korraga määrata mitmeid teksti omadusi, nagu esitatud allolevas koodinäites, asetseb alles 39. kohal.

```
font: italic bold 12pt/1.5em "Verdana", "Arial", sans-serif;
```

Atribuudi font vähese populaarsuse põhjuseid on kindlasti mitmeid. Esiteks on see kindlasti kõige keerukama süntaksiga CSS 2 atribuut. Teiseks ei hõlma see lühendatud atribuut kaht väga populaarset tüpograafilist omadust: teksti värvi ning allajoonimist; samas sisaldades atribuuti font-variant, mis asetub populaarsuselt alles 61. kohale.



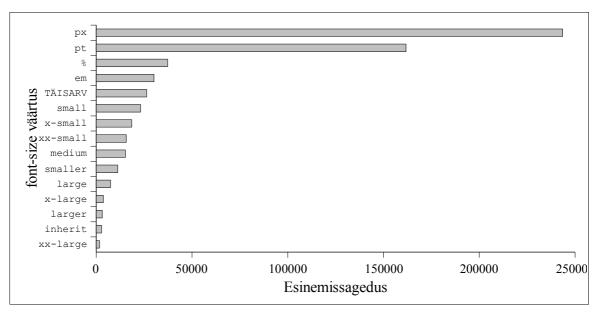
Joonis 21: Levinuimad CSS-i atribuudid.

### 6.4.1.1. Atribuut color

Nähtavasti pole nimi "color" olnud just kõige parem valik teksti värvi kirjeldava atribuudi nimetamiseks – märksa loogilisem oleks font-color või text-color, sest hoolimata sellest, mida kirjutavad W3C spetsifikatsioonid, kasutavad tuhanded leheküljed just neid kahte olematut atribuuti (sealjuures kirjutavad paljud veel ka inglisepäraselt text-colour). Katsetused levinuimate brauseritega (IE, Opera, Mozilla Firefox) näitasid, et nende atribuutide kasutamine ei anna vähimatki reaalset tulemust.

### 6.4.1.2. Atribuut font-size

Nagu ilmekalt esitab joonis 22, on kõige levinumaks mooduseks tekstisuuruse määramisel pikselite ( px ) või punktide ( pt ) kasutamine. Kusjuures mõlemad tehnikad on tuntud kui halvad praktikad, keelates osades brauserites (IE versioon ≤ 6) kasutajal tekstisuuruse muutmise.



Joonis 22: Atribuudi font-size levinuimad väärtused.

Heade praktikatena tuntud tekstisuuruse määramine protsendi (%), tähesuuruse (em) või võtmesõnaga (xx-small, x-small, small, medium, ...) on tunduvalt vähem levinud.

Vaadates kasutatud võtmesõnu, võib järeldada, et teksti muutmine väiksemaks on märksa levinum kui suuremaks muutmine.

Hulk lehti kasutab tekstisuuruse määramiseks lihtsalt täisarvu (näiteks font-size: 12;), mis pole CSS-i spetsifikatsioonide kohaselt korrektne, aga mida enamik brausereid tõlgendab pikselitena (ehk nagu eelmises näites oleks kirjutatud font-size: 12px;).

## 6.4.1.3. Atribuut font-family

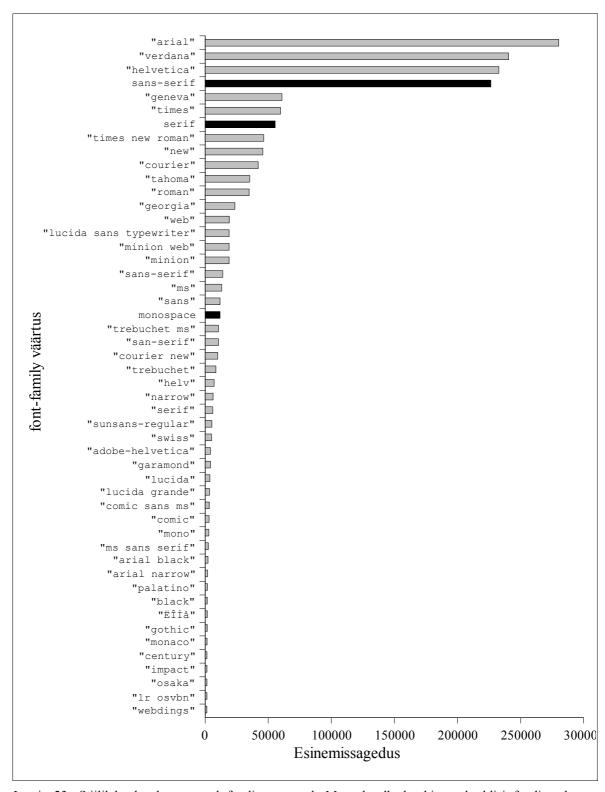
Joonis 23 leheküljel 69 loetleb 50 populaarseimat fondi nime, mida atribuudi font-style väärtusena kasutatakse. Neli ülekaalukalt populaarseimat esinevad CSS-is sageli koos, näiteks nõnda:

```
font-family: Verdana, Arial, Helvetica, sans-serif;
```

Üldistest fondiperekondadest (*generic font family*) kasutatakse peaasjalikult kolme: sansserif, serif ja monospace. Perekonnad cursive ja fantasy on haruldased.

Joonise järgi on väga populaarsed ka sellised fondid nagu "Times", "New" ja "Roman" – enamasti on autor siiski soovinud kasutada fonti "Times New Roman", ent jätnud jutumärgid kirjutamata, mistõttu käesolevas töös kasutatud CSS-i parser need erinevateks fontideks luges. Analoogne probleem on kindlasti ka "Courier New" puhul. Näide:

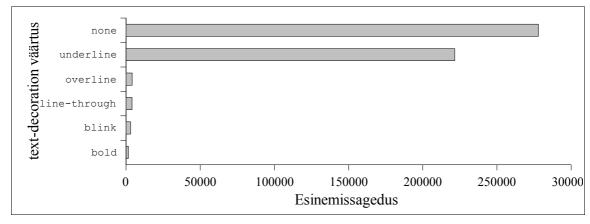
```
font-family: "Times New Roman", sans-serif; /* korrektne */
font-family: Times New Roman, sans-serif; /* vigane */
```



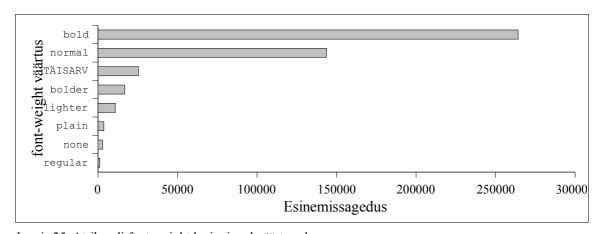
Joonis 23: Stiililehtedes kasutatavad fondimäärangud. Mutsad tulbad tähistavad üldisi fondiperekonna nimesid (*generic font family*).

### 6.4.1.4. Atribuut text-decoration

Nagu nähtub jooniselt 24, on selle atribuudi peamine kasutusala linkidelt allajoonimise eemaldamine (lingid on peamised elemendid, mida brauserid vaikimisi alla joonivad). Kuid ka alljoone lisamine on väga levinud, eriti võrreldes ülajoone, läbikriipsutuse ja vilkumisega, mida kasutavad vähesed lehed ("vähe" on muidugi siinkohal väga suhteline mõiste – maailmas on miljoneid veebilehti, mis just neid text-decoration väärtusi kasutavad).



Joonis 24: Atribuudi text-decoration levinuimad väärtused.



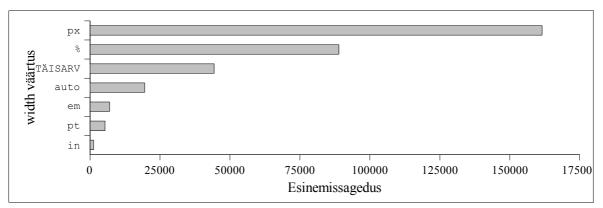
Joonis 25: Atribuudi font-weight levinuimad väärtused.

## 6.4.1.5. Atribuut font-weight

Enimkasutatavaks väärtuseks sellel atribuudil on loomulikult bold, kuid ka normal on väga sage. Vähem kasutatakse täisarvulisi väärtusi, mis aga erinevates brauserites tihtipeale erineva tulemuse annavad. Joonisel 25 leheküljel 70 olevad väärtused plain, none ja regular W3C CSS-i spetsifikatsioonides ei eksisteeri.

#### 6.4.1.6. Atribuut width

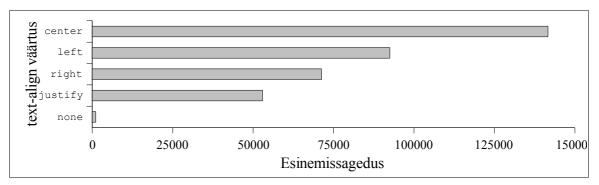
Nagu näha jooniselt 26, määratakse laiust enamasti pikselites, kuid kaunis sage on ka protsentide kasutamine. Sarnasel font-size atribuudiga on levinud täisarvuliste väärtuste kasutamine, mida brauserid tõlgendavad pikselitena olgugi, et W3C CSS-i spetsifikatsioonid seda ei teha luba.



Joonis 26: Atribuudi width levinuimad väärtused

### 6.4.1.7. Atribuut text-align

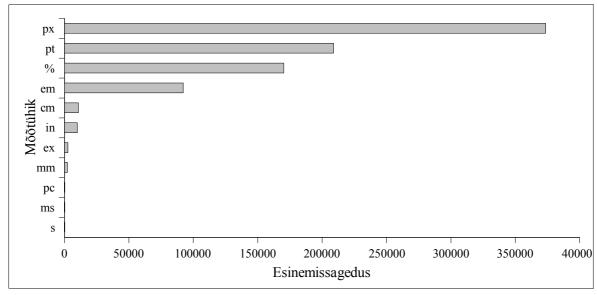
Selle atribuudi puhul üllatusi pole. Levinuim on teksti joondamine keskele, seejärel vasakule, siis paremale ja lõpuks rööbiti. Joonisel 27 leheküljel 72 olev väärtus none pole sellele atribuudile W3C CSS-i spetsifikatsioonide järgi lubatud.



Joonis 27: Atribuudi text-align levinuimad väärtused.

### 6.4.2. Mõõtühikud

Kõige levinuimaks CSS-is kasutatavaks ühikuks on pikselid (joonis 28). Populaarsuselt teisel kohal on punktide kasutamine, mis aga mõeldud peaasjalikult trükimeedia tarbeks. Veidi vähem kasutatakse protsenti ning veel vähem tekstikõrgust ( em ). Ülejäänud ühikud leiavad vaid marginaalset kasutust.



Joonis 28: Levinuimad CSS-is kasutatavad mõõtühikud.

Mõõtühikute kasutamisel üldiselt on suured sarnasused atribuudi font-size kasutamisega (vaata joonis 22 leheküljel 67) – see tuleneb küllap sellest, et font-size on niivõrd levinud atribuut, et mõjutab tugevasti ka üldist statistikat.

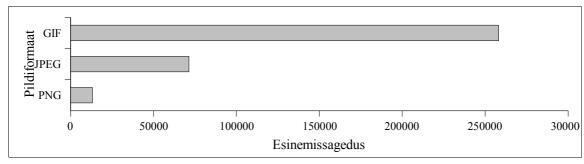
#### 6.4.3. Värvid

Ülekaalukalt levinumaks mooduseks värvide määramisel on kasutada kuju #rrggbb. Populaarsuselt järgmine lühendatud variandi #rgb kasutamine ning sellega ligikaudu sama populaarne on võtmesõnade black, white, green jne kasutamine. Kõige ebapopulaarsem on rgb() funktsiooni kasutamine.

Populaarseim värv, mida CSS-iga #rrggbb kujul määratakse on valge. Talle järgnevad must, punane, sinine, kollane, helehall, tumehall, tumesinine, tumepunane ja roheline.

## 6.4.4. Pildiformaadid

Nagu näitab joonis 29, on levinuimaks taustapildi formaadiks GIF. Märksa vähem kasutatakse JPEG ning hoopis vähe PNG formaati.

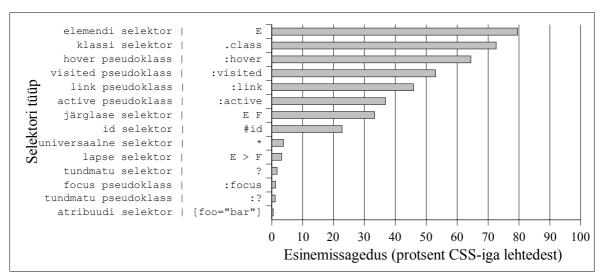


Joonis 29: Levinuimad pildiformaadid taustapiltides.

#### 6.4.5. Selektorid

# 6.4.5.1. Selektorite tüübid

Levinuimad CSS-i selektorite tüübid on toodud joonisel 30 leheküljel 74, ülejäänud uuringu käigus leitud selektorite tüübid on näidatud joonisel 31 leheküljel 75.



Joonis 30: Levinuimad CSS-i selektorite tüübid. Esimeses tulbas selektori nimi, teises näide.

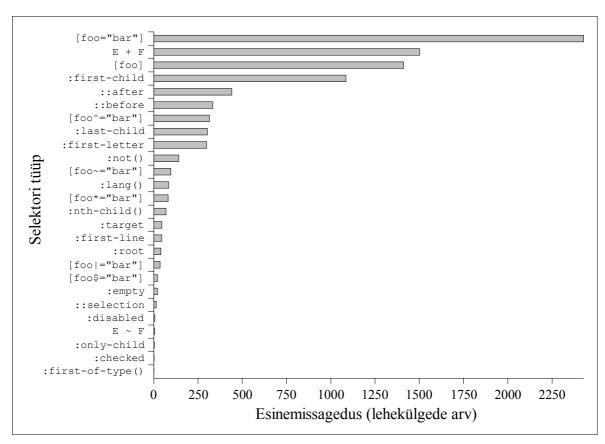
Nagu joonistelt näha, on levinuimateks selektoriteks mitmesugused CSS 1 selektorid. Üksnes pseudoklassid :first-letter ja :first-line on nõnda vähe kasutatud, et ei mahu esimesele joonisele. Levinuimaks selektoriks on igati õigustatult elemendi selektor, mida kasutab pea 80% CSS-i kasutavatest lehtedest. Klassi selektor on aga pea kolm korda populaarsem kui analoogset funktsionaalsust pakkuv ID selektor. Igati ootuspäraselt on väga sagedased linkide olekutega seotud pseudoklassid, sest (X)HTML-i presentatiivsed elemendid ei võimalda näiteks külastatud linke kujundada. Isiklikust kogemusest võib autor öelda, et paljude lehtede CSS piirdubki vaid järgmise näite sarnase koodiga.

```
a:link, a:visited, a:hover, a:active {
   text-decoration: none;
   color: #f2c0c0;
}
```

Vaid 33% CSS-iga lehtedes kasutab järglase selektorit, mis viitab sellele, et ülejäänud 67% lehekülgede autorid teavad CSS-i pakutavatest võimalustest väga vähe.

Kõige populaarsem CSS 2 seletor on universaalne selektor ( \* ), seda kindlasti seetõttu, et tegu on ainsa CSS 2 selektoriga, mida ka IE 6 toetab. Ühtlasi kasutatakse seda selektorit ka

ühes IE versioon ≤ 6 brauserile suunatud nn. häkis³⁴. Analoogselt kasutatakse ka lapse selektorit (E > F) seadmaks stiile CSS 2-te toetavatele brauseritele, mida IE ≤ 6 peab ignoreerima.

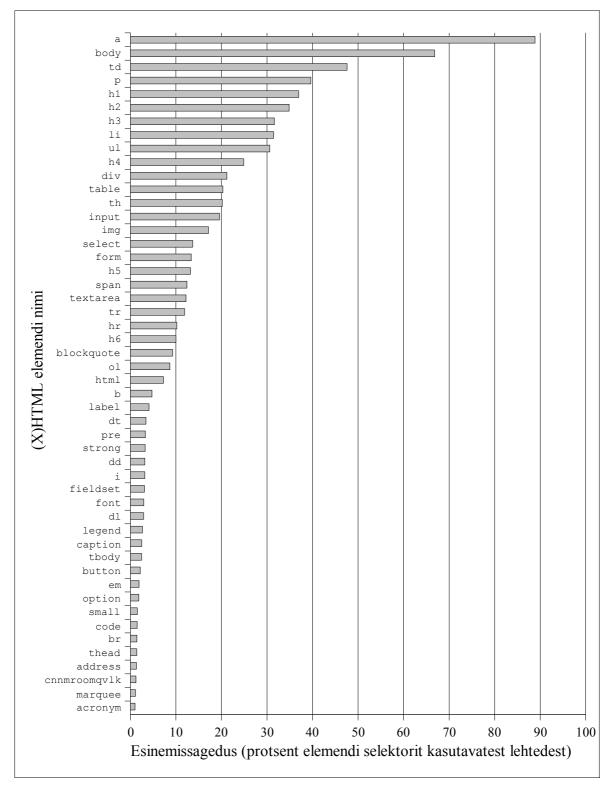


Joonis 31: Vähemlevinud CSS-i selektorid. (Joonise 30 jätk.)

## 6.4.5.2. Elementide selektorid

Joonis 32 leheküljel 76 annab ülevaate (X)HTML-i elementidest, mida CSS-i kaudu enim selekteeritakse. Esimesel kohal on ootuspäraselt element a – linkide välimuse muutmine on üks populaarsemaid tegevusi veebilehe kujundamisel. Elemendile body stiilide määramine võrdsustub kogu lehekülje kujunduse üldiste parameetrite seadmisega – seega igati ootuspärane (aga üksjagu kasutatakse ka elementi html). Kolmandal kohal olev td viitab aga sellele, et enamik lehekülgi kasutavad lehekülje paigutuse loomiseks tabeleid.

<sup>34</sup> Tuntud kui *star html hack*, kuna kasutatakse ära IE ≤ 6 viga, kus selektor \* html selekteerib dokumendi html elemendi, olgugi, et html element tegelikult ühegi teise elemendi sees ei sisaldu.



Joonis 32: Elementide selektorite esinemissagedus.

Üllataval sage on ka elemendi p välimuse muutmine, see on isegi populaarsem kui pealkirjade h1, h2, h3, ... välimuse muutmine. Arvatavasti on see seetõttu, et paljud leheküljed lihtsalt ei kasutagi pealkirju. Sama lugu on ka loenditega – elemendid u1 ja 1i.

#### 6.4.5.3. Klassi selektorid

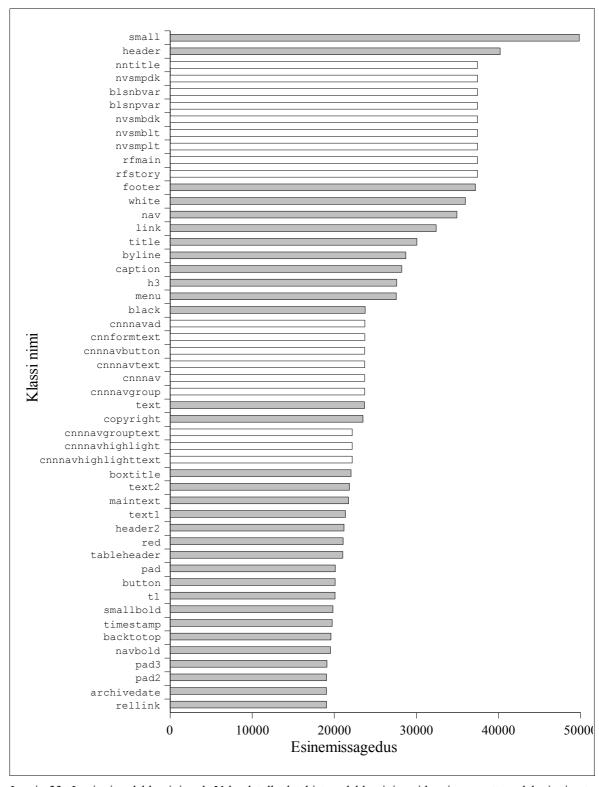
Klassiselektorid olid elemendi selektorite järel levinuimad. Õnnetuseks sattus valimisse hulk lehekülgi, mis kasutasid üht ja sama CSS-i faili, näiteks <a href="http://www.cnn.com/">http://www.cnn.com/</a> alamlehed. Sellised suured lehed tihtipeale kasutavad palju samu klassinimesid, seega mõjutades kõvasti joonisel 33 leheküljel 78 toodud klassiselektorite statistikat.

Ma olen joonisel märgistanud klassinimed, mis pärinevad mõnest konkreetsest leheküljestikust (*website*), valge värviga, kuid sellegipoolest ei tohiks toodud statistikat eriti usaldada. Üldjoontes langevad populaarsemad klassinimed kokku Google'i 2005. aasta uurimuses (X)HTML-i class atribuutide väärtuste statistikaga, kuid seda vaid väga üldjoontes (vrd. <a href="http://code.google.com/webstats/2005-12/classes.html">http://code.google.com/webstats/2005-12/classes.html</a>).

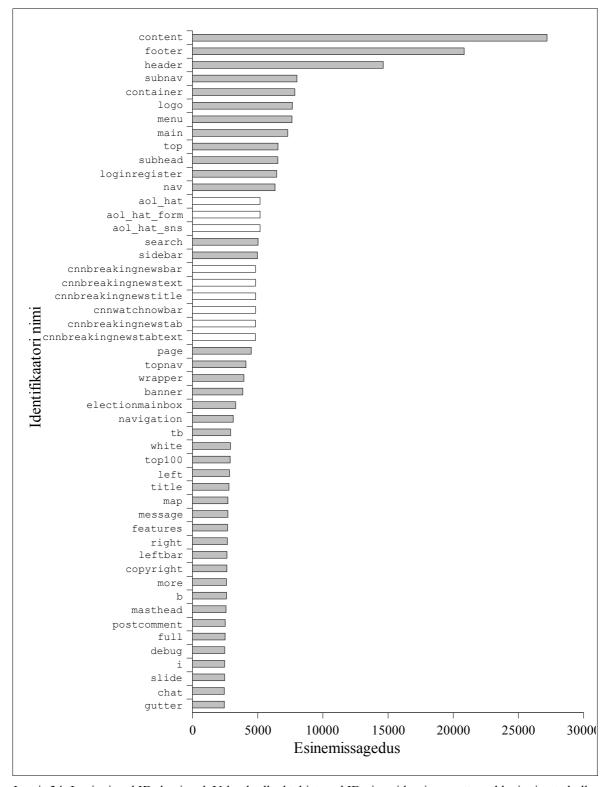
Võiks arvata, et ka ülejäänud käesoleva uurimuse käigus kogutud andmed tuleks kahtluse alla panna, sest valim sisaldas hulka väga sarnaseid lehekülgi, kuid see oleks liialt ennatlik. Kui vaadata näiteks sellesama joonise 33 X-teljel olevaid numbreid, siis on näha, et tegelikult ei moodusta sellised sarnased lehed kogu valimist just eriti suurt osa. Joonise skaala ulatub küll tervelt 50 000 leheküljeni, kuid kogu valimi 700 000-st on see vaid 7%, ning tegelikult peakski üks esinduslik valik veebilehti sisaldama natuke rohkem lehekülgi nendest veebilehestikest, mis on suuremad ja populaarsemad (ehk siis näiteks Juku isiklikku kodulehte ei tohiks arvestada võrdväärsena Microsoft'i kodulehe kõrval).

### 6.4.5.4. ID selektorid

ID selektorite statistika (joonis 34 leheküljel 79) on märksa usaldusväärsem kui ülalkirjeldatud klassiselektorite oma, sest suured leheküljestikud ei paista ID-sid eriti kasutavat.



Joonis 33: Levinuimad klassinimed. Valged tulbad tähistavad klassinimesid, mis on sattunud levinuimate hulka vaid seetõttu, et mõne suurema veebilehestiku (näiteks CNN) alamlehti esines valimis hulgim.

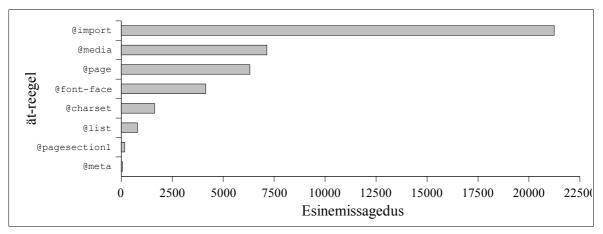


Joonis 34: Levinuimad ID-de nimed. Valged tulbad tähistavad ID nimesid, mis on sattunud levinuimate hulka vaid seetõttu, et mõne suurema veebilehestiku (näiteks AOL) alamlehti esines valimis hulgim.

Kolm ülekaalukalt kõige sagedasemat ID väärtust on content, footer ja header, mis langevad kokku ka John Allsopp'i 2005. aasta novembrikuus läbiviidud uuringuga. Allsopp'i uuringu ja käesoleva vahele ei saa küll päris otseseid paralleele tõmmata, sest Allsopp kogus ID nimed (X)HTML-ist, käesolev töö aga CSS-ist. Lisaks hõlmas Allsopp'i uuring vaid 1315 lehekülge.

Erinevalt klasside statistikast, mis algas klassiga small, ei leidu ID-de esikümnes otseselt kujundusliku tähendusega nimetusi. Huvitav on veel märkida, et väärtus subnav on millegipärast üksjagu populaarsem kui nav.

# 6.4.6. Ät-reeglid



Joonis 35: CSS-i ät-reeglid reastatuna esinemissageduste põhjal.

Vaieldamatul sagedaseim ät-reegel on @import, mida leidub pea sama tihti, kui kõiki teisi ät-reegleid kokku (joonis 35), ent ka seda kasutab alla 5% CSS-iga lehtedest. Järgnevad @media, @page ning @font-face. @charset on kasutuses kaunis vähestel lehtedel, kuid tänasel päeval pole selle ät-reegli kasutamise järele ka erilist vajadust – CSS failid sisaldavad enamasti vaid ASCII kooditabeli piiresse jäävaid sümboleid. Kõik ülejäänud joonisel 35 esitatud ät-reeglid pole W3C CSS-i standardites kirjeldatud.

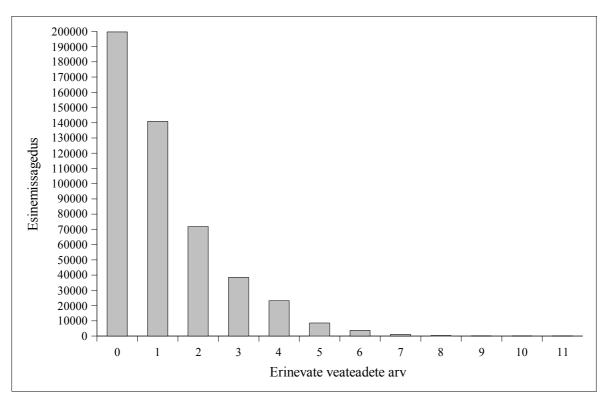
#### 6.4.7. Koodi korrektsus

Vaid 40,92% CSS-i kasutavate lehtede CSS vastab W3C CSS 2 soovitusele (see ei

tähenda, et kõigi ülejäänud lehtede CSS oleks ilmtingimata vigane – näiteks võivad lehed kasutada võimalusi CSS 2.1 või CSS 3 spetsifikatsioonidest, kui enamasti pole siiski see mittevalideerumise põhjuseks, sest nagu näitab eelpooltoodud CSS-i keelekonstruktsioonide statistika, kasutatakse peamiselt vaid CSS 1 võimalusi).

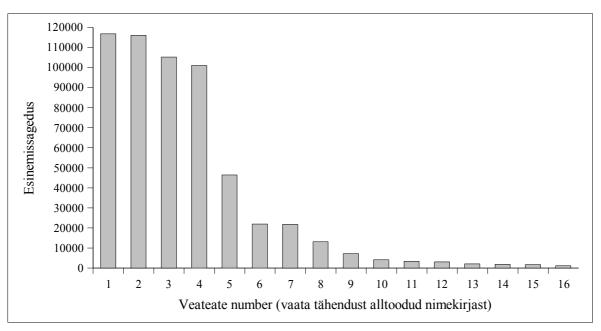
Võiks arvata, et leheküljed, mille (X)HTML valideerub, kasutavad ka korrektset CSS-i, kuid päris nii see pole. Valideeruva (X)HTML-ga lehtedest vaid 51.28% kasutab stiililehti, mis CSS 2 soovitusele vastavad (vaid 10% rohkem).

Joonis 36 annab ülevaate erinevate CSS-i valideerimise käigus saadud veateadete arvude esinemissagedustest. Erinevalt (X)HTML-i valideerimise vigadest, on CSS puhul kõige sagedasemad ilma vigadeta stiililehed.



Joonis 36: Erinevate veateadete arv stiililehtedes. Kõige sagedasem on ilma vigadeta CSS.

Kuid milliseid vigu peamiselt CSS-is tehakse? 16 levinuimat veateadet on toodud joonisel 37 leheküljel 82. Joonisel antud numbritele vastavad järgmised alltoodud veateated:



Joonis 37: Levinuimad CSS-i validaatori veateated.

- 1. Atribuuti x ei eksisteeri (*Property x doesn't exist*)
- 2. Atribuudi x väärtus ei saa olla y (y is not a x value)
- 3. Tundmatu viga (*Unrecognized*)
- 4. Numbri järel peab olema mõõtühik (*You must put an unit after your number*)
- 5. Viga parsimisel (*Parse error Unrecognized*)
- 6. Proovi leida semikoolon enne atribuudi nime lisa see. (*Attempt to find a semi-colon before the property name. add it*)
- 7. Liiga palju väärtusi või tundmatu väärtus (*Too many values or values are not recognized*)
- 8. Tundmatu ühik *x* (*Unknown dimension : x*)
- 9. Vigane värv (x is not a valid color 3 or 6 hexadecimals numbers)
- 10. Üldiste fondi-perekondade nimesid ei panda jutumärkidesse (*Generic family names are keywords, and therefore must not be quoted.*)
- 11. Liiga vähe väärtusi atribuudile *x* (*Too few values for the property x*)

- 12. ID või klassi nimi ei saa alata numbriga (In CSS1, a id/class name could start with a digit (".55ft"), unless it was a dimension (".55in"). In CSS2, such ids/classes are parsed as unknown dimensions (to allow for future additions of new units))
- 13. Atribuudile *x* pole negatiivsed väärtused lubatud (*x negative values are not allowed*)
- 14. Vigane eraldaja kujundi definitsioonis. Peab olema koma. (Invalid separator in shape definition. It must be a comma.)
- 15. x on vigane operator (x is an incorrect operator)
- 16. Tundmatu veateade (*Unknown error*)

# 6.5. JavaScript

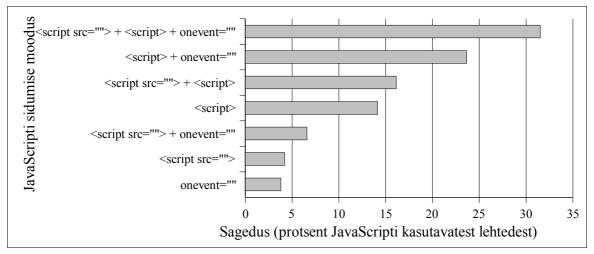
JavaScript'i kasutavaid lehti on vaid veidi vähem kui CSS-i kasutavaid – 65,09% (vrd. CSS-i kasutas 67,20%). Sealjuures JavaScript'i URI-sid kasutavad tervelt 25% lehtedest, olgugi sellel moel javascripti (X)HTML-ga sidumine ei kajastu üheski W3C soovituses. JavaScript'i URI-d on näiteks selliseid:

```
<a href="javascript: alert('Hello!');">JavaScript'i link</a>
```

Teisalt on aga selliste URI-de kasutamine üksi (ilma teiste JavaScript'i sidumise meetoditeta) vähelevinud – kasutuses alla poolel protsendil JavaScript'i kasutavatest lehtedest. Nimetatud praktika on väga sarnane on-algusega atribuutide kasutamisele, ning seda tehnikat kasutatakse peamiselt *koos* teiste JavaScript'i sidumise meetoditega, eraldiseisvana jääb selle kasutus alla viie protsendi (joonis 38 leheküljel 84).

Nagu kirjeldatud leheküljel 23 peatükis "1.5 Nõuded JavaScript'i analüüsimisel", võib JavaScript olla (X)HTML failiga seotud kolmel erineval moel:

Kuid erinevalt CSS-ist pole JavaScript'i puhul üksnes väliste failide kasutamine kuigivõrd levinud. Jooniselt 38 on näha, et kõige levinum on hoopis kõigi kolme meetodi koos kasutamine. Leheküljed, kus kasutatakse vaid väliseid JavaScript'i faile kuuluvad vähemusse, neid on ligikaudu kolm korda vähem kui lehti mis kasutavad script elemendi sisse paigutatud JavaScript'i.



Joonis 38: Erinevad meetodid JavaScript'i sidumiseks (X)HTML-ga. onevent märgib joonisel ükskõik millise on-eesliitega atribuudi kasutamist. Joonis ei kajasta JavaScript'i URI-de kasutamist, mis pole standardne meetod JavaScript'i sidumiseks.

Vaid väliste JavaScript'i failide kasutamine on tuntud nime *unobtrusive JavaScript*<sup>35</sup> all, ning tunnustatud paljude poolt parima praktikana, kuid selle tehnika kasutamist õpetavad vaid vähesed Internetis leiduvad JavaScript'i õpetused – enamik õpetusi soovitab funktsioonide väljakutsed paigutada on-eesliitega atribuutidesse. Järgnevalt on kirjeldatud on-atribuute kasutav JavaScripti kood *Hello world*'i näitel. HTML fail:

<sup>35</sup> Suurepärase ülevaate sellest tehnikast annab Christian Heilmann'i loodud õpetus "*Unobtrusive Javascript*" aadressil: <a href="http://www.onlinetools.org/articles/unobtrusivejavascript/">http://www.onlinetools.org/articles/unobtrusivejavascript/</a>

```
<title>Tervitus</title>
<script type="text/javascript" src="tervitus.js">
<a href="tervitus.html" onclick="tervitus();">Tervitus</a>
```

## JavaScript'i fail:

```
function tervitus() {
   alert("Tere, maailm!");
}
```

Sama funktsionaalsus *unobtrusive JavaScript*'i kasutades. HTML fail:

```
<title>Tervitus</title>
<script type="text/javascript" src="tervitus.js">
<a href="tervitus.html">Tervitus</a>
```

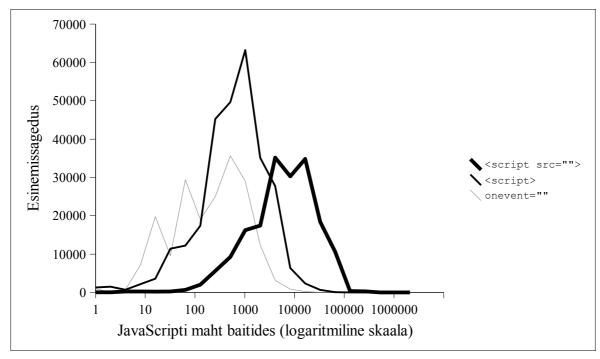
## JavaScript'i fail:

```
window.onload = function() {
    document.getElementsByTagName("a")[0].onclick = function() {
        alert("Tere, maailm!");
    }
}
```

Tulles tagasi JavaScript'i statistika juurde – võiks ju arvata, et viimati esitatud tehnika on liialt keerukas ja enamik JavaScript'iga lehti kasutab esimest, kui ei, väline JavaScript'i fail koos tekstisiseste funktsiooniväljakutsetega on vaid veidi populaarsem kui eraldiseisva JavaScript'i faili üksinda kasutamine. Üllataval kombel on aga kaunis levinud kasutada koodi sisaldavat script elementi kas siis üksi või koos väliste JavaScript'i failidega. Tekib küsimus, miks mitte siis JavaScript juba tervenisti eraldi faili tõsta? Autor jääb siinkohal vastuse võlgu.

Joonisel 39 leheküljel 86 on toodud erineval viisil (X)HTML-ga seotud JavaScript'i võrd-

lus koodi suuruse alusel. Selgelt on näha, et script elemendi sisse paigutatud JavaScript on kõige sagedasem, kuid kui tegemist on suurte JavaScripti kogustega, siis eelistatakse see paigutada siiski eraldi faili.



Joonis 39: Erineval moel leheküljega seotud JavaScript'i maht. Väliste JavaScript'i failide maht on enamasti suurem kui (X)HTML faili sisse paigutatud JavaScript'i puhul (analoogselt CSS-ga, vrd. vaata joonis 20 leheküljel 65).

## 6.5.1. AJAX

Probleemide tõttu programmi jooksutamisel (vaata lehekülg 42 peatükk 5.3) oli ainuke JavaScript'i sisulisem analüüs sõne XMLHttpRequest otsimine koodist. Leiti 6125 veebilehte, mille JavaScript'i kood seda märksõna sisaldas – seega kasutab AJAX laadset tehnoloogiat 1,90% JavaScript'i kastavatest lehtedest.

# **KOKKUVÕTE**

#### Kokkuvõte tulemustest

Uuringu käigus sai läbi vaadatud ligikaudu veerand ODP kataloogis olevatest URI-dest. Ümmarguselt 700 000 lehekülje koodi analüüsimise põhjal sai koostatud statistika, mis peaks pakkuma vastused järgmistele sissejuhatuses esitatud küsimustele.

Milliseid tehnoloogiaid veebilehtedel kasutatakse?

- 67% lehtedest kasutas CSS-i.
- 65% lehtedest kasutas JavaScript'i.
  - 1,9% JavaScript'iga lehtedest kasutas AJAX-it.
- Vähemalt 10% lehtedest kasutas Flash'i.

Kui korrektset koodi veebilehed kasutavad?

- 2,6% lehekülgede (X)HTML vastas dokumenditüübi definitsioonile (2001. aastal oli see Dagfinn Parnas'e andmetel 0,7%).
- 41% CSS-i kasutavate lehtede CSS vastas W3C CSS 2 soovitusele.
- Korrektse (X)HTML-ga lehtedest vaid 51% kasutas korrektset CSS 2-te.

Millised head ja halvad praktikad on veebilehtede loomisel levinud?

- 97% veebilehtedest edastati HTTP/1.1 protokolliga, mis on moodsaim HTTP protokolli standard (HTTP/1.0 oli vähelevinud).
- (X)HTML:
  - 39% lehtedest deklareeris oma dokumenditüübi. 3 levinuimat dokumenditüüpi olid "HTML 4.01 Transitional", "HTML 4.0 Transitional" ja "XHTML 1.0

Transitional".

- Ülekaalukalt levinuimaks kodeeringuks oli ISO 8859-1. UTF-8 kodeeringut kasutas 10% kodeeringu määranud lehtedest.
- Ligikaudu 80% lehtedest kasutas tabeleid ja 17% raame (*frames*).

#### CSS:

- Levinuimad atribuudid olid seotud tekstiga: color, font-size, font-family.
- Levinuimad mõõtühikud olid pikselid (px, seejärel pt, % ja em).
- Sagedaseimaks taustapildiformaadiks oli GIF, millele järgnesid JPEG ja PNG.
- Kõige rohkem kasutati elemendiselektoreid, millega enim selekteeriti elemente a, body ja td.
- CSS-i sidumiseks kasutati kõige sagedasemini link elementi.
- JavaScript'i levinuimaks veebilehega sidumise mooduseks oli koodi script elemendi sisse paigutamine.

#### Järeldused

Liiga palju on lehekülgi, mis kasutavad skriptimist ja liialt vähe neid, mis kujundatud stiililehti kasutades. Kuid ka stiililehtede kasutuselevõtt pole enamike lehtede puhul kaasa toonud suurt hüpet paremuse poole. Veebis, kus ei saa kunagi kindel olla, milliste parameetritega riist- või tarkvara lehekülje külastaja võib kasutada, defineeritakse kujundusdetailide suurus enamasti pikselites ning tekstikõrgus punktides.

Kuid mitte see teadmine pole käesoleva töö tulemuseks – see kõik oli teada juba varemgi. Käesoleva töö eesmärgiks oli iseloomustada olukorda veebis, tuues välja konkreetsed arvud, protsendid, pingeread. See eesmärk ka saavutati. Viidi läbi ka võrdlused varasemate uuringutega, misläbi oli võimalik teha järeldusi valitsevate suundade kohta.

Kui võtta kriteeriumiks valideeruvate lehtede osakaal, siis on viimase viie aasta jooksul

veebilehtede kvaliteet paranenud. Kuid korrektse koodiga lehekülg on ka tänasel veebimaastikul üpris harv nähtus. Halvad praktikad on visad taanduma.

Samamoodi nagu Dagfinn Parnas'e uuring 2001. aastal oli võrdluseks sellele tööle, on see töö võrdlusmaterjaliks tulevastele uuringutele. Tänu kokkukogutud materjali suurele hulgale ja detailide rohkusele, saab seega võimalikuks hulga enamate parameetrite võrdlemine.

Olen oma proseminaritöös läbiviidud uuringut korranud poole-aastaste vahedega ning jälginud muutusi Eesti veebimaastikul. Loodan ka seda uurimust võimalusel edaspidi korrata. Eeldused selleks on loodud läbi töö käigus kirjutatud programmi, mis on tehtud ka veebi kaudu vabalt kättesaadavaks kõigile, kes sooviksid samalaadseid uuringuid läbi viia.

Kuid järgnevate uuringute käigus ei tohiks kindlasti seisma jääda samale tehnoloogilisele aluspinnasele. Lisaks vigade parandamisele programmis tuleks keskenduda ka metoodika ülevaatamisele ja täiustamisele.

## Hinnang kasutatud metoodikale

Enamik probleeme läbiviidud uuringu metoodikas tulenesid kasutatud valimist. Esiteks oli valim liialt väike, mistõttu tulemustes kippusid siin ja seal tooni andma mõningad suuremad leheküljed. See võis olla ka konkreetselt kasutatud ODP kataloogi probleem, kuid kuna ei uuritud teiste veebikataloogide sisu, siis on raske öelda, kas nood oleksid olnud representatiivsemad. Hoopis iseküsimus siinkohal on see, mille alusel võrrelda erinevate veebikataloogi representatiivsust. Kuid ükskõik millist veebikataloogi ka valimina kasutada – nihutab see vältimatult valimit natuke "parema" veebi poole, sest igasugusesse veebikataloogi kantakse peamiselt ikka neid lehekülgi, mis natukenegi väärtust omavad, jättes seeläbi valimist välja hulga lehekülgi, mis on kaunikesti sisutühjad.

Kui üks mis kindel – just ODP kataloogist pärit lehekülgede kasutamine tegi võimalikuks tulemuste kõrvutamise viie aasta taguste andmetega.

## **ALLIKAD**

- Allsopp, J. (2005). *Semantics in the wild*. [2006, aprill 21]. http://westciv.typepad.com/dog or higher/2005/11/real world sema.html
- Hazaël-Massieux, D. (2003) Content-Negotiation Techniques to serve XHTML 1.0 as text/html and application/xhtml+xml. (2003). W3C. [2006, aprill 29]. http://www.w3.org/2003/01/xhtml-mimetype/content-negotiation
- HTML 4.01 Specification. (1999). W3C. (Eds.) Raggett, D., Hors, A. L., Jacobs, I. [2006, aprill 28]. <a href="http://www.w3.org/TR/html4/">http://www.w3.org/TR/html4/</a>
- Karppinen, M. (2002). *State of the Validation 2002*. [2006, aprill 21]. http://www.markokarppinen.com/20020222.html
- Parnas, D. (2001). *How to cope with incorrect HTML*. [Magistritöö]. Bergen: University of Bergen. [2005, oktoober 16]. http://www.ub.uib.no/elpub/2001/h/413001/Hovedoppgave.pdf
- Saarsoo, R. (2006). *Validating sites of W3C members*. [2006, aprill 21]. http://www.triin.net/2006/03/05/Validating sites of W3C members
- Saarsoo, R. (2006). *Web Standards in Estonia vol 3*. [2006, aprill 21]. http://www.triin.net/2006/03/11/Web Standards in Estonia vol 3
- Sivonen, H. (2005). *Activating the Right Layout Mode Using the Doctype Declaration*. [2006, aprill 28]. <a href="http://hsivonen.iki.fi/doctype/">http://hsivonen.iki.fi/doctype/</a>
- Web Authoring Statistics. (2005). Google Inc. [2006, aprill 21]. http://code.google.com/webstats/index.html

Windows-1252. Wikipedia. [2006, aprill 28]. http://en.wikipedia.org/wiki/Windows-1252

Windows-1250. Wikipedia. [2006, aprill 28]. http://en.wikipedia.org/wiki/Windows-1250

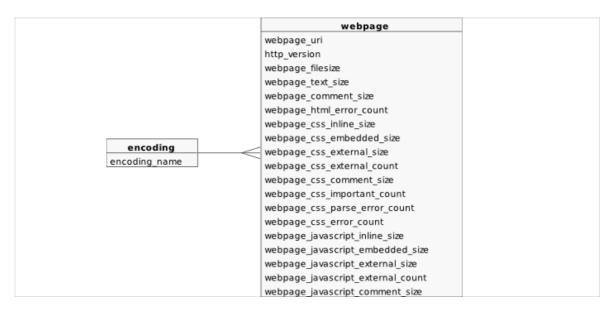
XHTML Media Types. (2002). W3C. (Ed.) Masayasu, I. [2006, aprill 28]. http://www.w3.org/TR/xhtml-media-types/

Zeldman, J. (2003, juuni 12). We stand corrected. *The Daily Report*. [2006, aprill 21]. http://www.zeldman.com/daily/0603a.shtml#westand

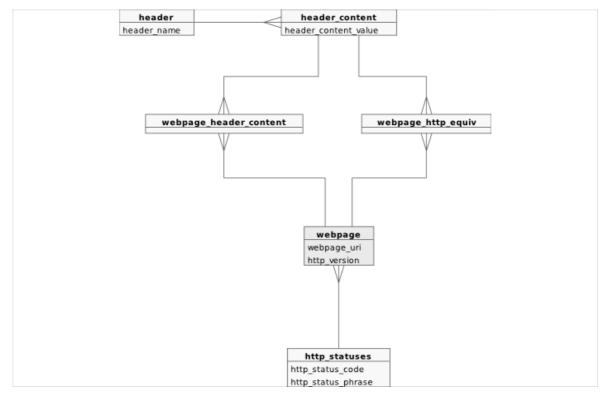
# LISA 1. ANDMEBAASISTRUKTUUR

Järgnevalt on esitatud programmi webstat poolt kasutatud andmebaasi struktuur. Kuna andmebaas koosneb 38 tabelist, siis polnud mõttekas neid kõiki ühele joonisele paigutada. Suur hulk tabeleid on seotud tabeliga webpages, kuid kuna see on andmebaasi suurim tabel, siis polnud ka seda mõtet kõigil joonistel koos kõigi väljadega kujutada – seega on kõigil teistel joonistel peale esimese toodud ära vaid webpages tabeli paar esimest välja. Lisaks on mitmete tabelite puhul lühendatud väljade nimetusi kirjutades nime algusesse kolm punkti – näiteks tabelis webpage\_html\_elements on väli webpage\_html\_element\_count, kuid joonisel on kirjutatud vaid ...count.

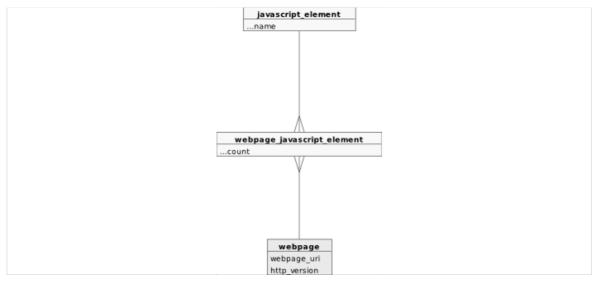
Joonistel on andmebaas kujutatud olem-seos diagrammina, kus olemite nimed on kirjutatud ainsuses. Tegelikus andmebaasis vastavad olemitele aga tabelid, millede nimed on kirjutatud mitmuses. Näiteks esimese joonise olemile encoding vastab tabel encodings.

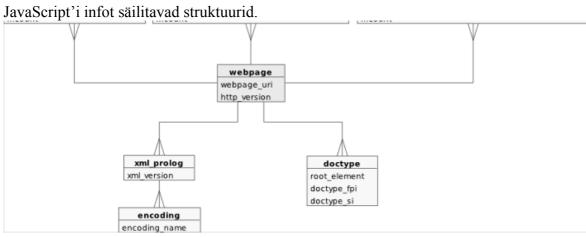


Keskne olem webpage ning kodeeringud.

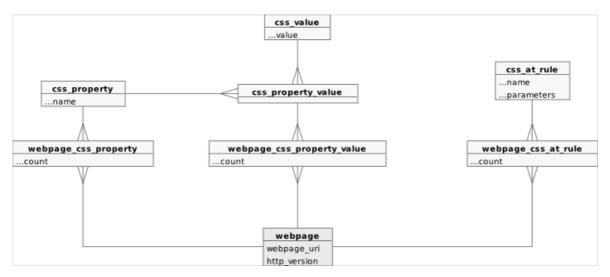


HTTP staatus-koode ja päiste infot säilitavad struktuurid.

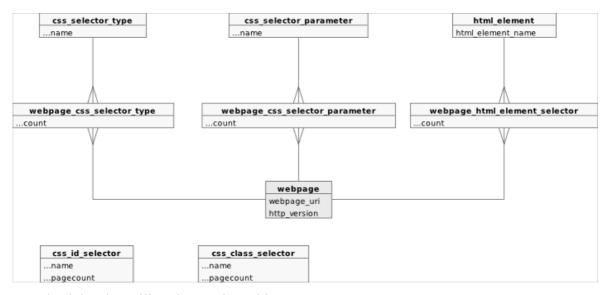




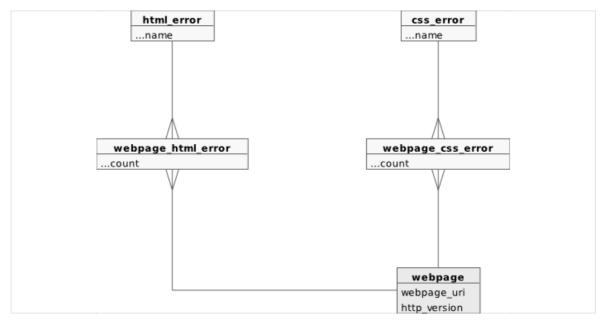
(X)HTML-i info säilitamise struktuurid.



CSS-i atribuutine ja ät-reeglite salvestamise struktuurid.



CSS-i selektorite säilitamise struktuurid.



(X)HTML-i ja CSS-i valideerimise tulemuste hoidmiseks mõeldud struktuurid.

# LISA 2. PROGRAMMI WEBSTAT PEAMOODULI LÄHTEKOOD

```
package WebStatistics::WebStatistics;
use warnings;
use strict;
# wget
use WebStatistics::Wget;
# URIs
use URI;
# Filesystem info
use File::stat;
# database and cache
use DBI;
use WebStatistics::SQL::General;
use WebStatistics::Cache::General;
# HTTP Headers
use HTTP::HeadersParser;
use WebStatistics::HTTPHeaders;
# HTML
use HTML::Parser;
use WebStatistics::HTMLElements;
use WebStatistics::HTMLValidator;
# CSS
use CSS::SAC;
use WebStatistics::CSSElements;
use WebStatistics::SACErrorHandler;
use WebStatistics::CSSValidator;
# JavaScript
use WebStatistics::JavaScriptParser;
# Utilities
use Data::Dumper;
```

```
=head1 NAME
WebStatistics::WebStatistics - the main module of WebStatistics package
=head1 SYNOPSIS
    use WebStatistics::WebStatistics;
    $database = "webstat";
    $host = "localhost";
    $user = "bob";
    $pass = "UnBR32K2b13P2ssW0rD";
   my $webstat = new WebStatistics::WebStatistics(
        $database,
       $host,
       $user,
        $pass
    );
    # analyze the URI and save results to database
    $webstat->parse_address( "www.example.com" );
    $webstat->cleanup;
=cut
=head1 METHODS
=head2 new( $db name, $db host, $db user, $db pass )
Constructor. Takes database name, host, username and password
as parameters, to establish the connection to the MySQL database.
=cut
sub new {
   my $type = shift;
   my $db_name = shift;
   my $db_host = shift;
   my $db_user = shift;
   my $db pass = shift;
   my $self = bless {}, $type;
    $self->init( $db name, $db host, $db user, $db pass );
   return $self;
}
```

```
# initialization
sub init {
   my $self = shift;
   my $db name = shift;
   my $db host = shift;
   my $db user = shift;
   my $db pass = shift;
    # establish database connection
   my $database = "webstat";
    $self->{db} = DBI->connect(
        "DBI:mysql:database=$db name;host=$db host",
        $db user,
        $db pass,
        {PrintError => 0},
    );
    # SQL statements
    $self->{sql} = new WebStatistics::SQL::General( $self->{db} );
    # create cache
    $self->{cache} = new WebStatistics::Cache::General();
    # Headers handler
    $self->{headers handler} = new WebStatistics::HTTPHeaders( {
        SQLStatements => $self->{sql},
        DatabaseCache => $self->{cache},
    } );
    # Headers parser
    $self->{headers parser} = new HTTP::HeadersParser( {
        DocumentHandler => $self->{headers handler},
    } );
    # The same things for Http-Equiv's
    $self->{http_equiv_handler} = new WebStatistics::HTTPHeaders( {
        SQLStatements => $self->{sql},
        DatabaseCache => $self->{cache},
       IsHTTPEquiv => 1,
    $self->{http equiv parser} = new HTTP::HeadersParser( {
        DocumentHandler => $self->{http equiv handler},
    } );
    # HTML parser
    $self->{html parser} = new WebStatistics::HTMLElements( {
        SQLStatements => $self->{sql},
        DatabaseCache => $self->{cache},
    # recognyze empty XML tags like <br/>
    $self->{html parser}->empty element tags( 1 );
    # HTML Validator
    $self->{html validator} = new WebStatistics::HTMLValidator( {
        SQLStatements => $self->{sql},
```

```
DatabaseCache => $self->{cache},
    } );
    # CSS handler
    $self->{css handler} = new WebStatistics::CSSElements( {
        SQLStatements => $self->{sql},
        DatabaseCache => $self->{cache},
    } );
    # CSS parser
    $self->{css parser} = CSS::SAC->new( {
        DocumentHandler => $self->{css handler},
       ErrorHandler => new WebStatistics::SACErrorHandler(),
    } );
    # CSS Validator
    $self->{css validator} = new WebStatistics::CSSValidator( {
        SQLStatements => $self->{sql},
        DatabaseCache => $self->{cache},
    } );
    # JavaScript parser
    $self->{js_parser} = WebStatistics::JavaScriptParser->new( {
        SQLStatements => $self->{sql},
        DatabaseCache => $self->{cache},
    } );
    # set verbosity on
    $self->{verbose} = 1;
    # names of files where to save downloaded HTML, CSS and JavaScript
    $self->{downloaded html filename} =
        "webstat-html-download.html";
    $self->{downloaded css filename} =
        "webstat-css-download.css";
    $self->{downloaded_javascript_filename} =
        "webstat-javascript-download.js";
}
=head2 cleanup()
Should be called before exiting the script.
At the moment does nothing more, then closes the database connection.
=cut
sub cleanup {
   my $self = shift;
    # disconnect from database
    $self->{db}->disconnect();
}
```

TLÜ Haapsalu Kolledž Rene Saarsoo

```
=head2 parse file( $filename )
Reads URI's from file $filename and passes to parse address() method.
=cut
sub parse file {
   my $self = shift;
   my $filename = shift;
    # Read webpage addresses from inputfile
    open ( my $input file, $filename ) || die "Unable to open
'$filename'.";
   while ( my $address = <$input file> ) {
        $self->parse address( $address);
}
=head2 parse address( $URI )
Analyzes a single URI and stores results to database.
=cut
sub parse address {
   my $self = shift;
   my $address = shift;
    $address = $self->normalize uri( $address );
   my $redirect count=0;
   BEGINNING OF PARSE ADDRESS:
    # It might happen, that two sites redirect to one-another,
    # if no redirect_count exists, then the script would just start
    # endlessly jumping back and forward between the sites.
    # An example: www.globaling.org <--> www.tobaccopedia.com
    $redirect count++;
    if ( $redirect count > 10 ) {
        print "No more than 10 redirects allowed. Skipping.\n"
            if ($self->{verbose});
        # Just insert address into database, but nothing more.
        $self->insert webpage uri( $address );
        return;
    }
   print \frac{n}{033}[32m\address\033[0m\n" if ( \$self->\{verbose} );
    if ( my $page = wget($address, $self->{downloaded html filename}) ){
```

TLÜ Haapsalu Kolledž
Rene Saarsoo

```
# downloading the page succeeded
# the eventual address downloaded might have been different
# than the initial address supplied, so we overwrite our
# initial address with the actually downloaded address.
$address = $page->{downloaded address};
print "Successfully downloaded: $address\n"
    if ( $self->{verbose} );
# parse the HTTP headers of the page
$self->{headers parser}->parse header list( $page->{headers} );
if ( $self->{headers handler}->get http status code() == 200 ) {
    # Now, the HTTP status code suggests, that we have reached
    # an normal webpage. This might be the case, but it might
    # also be, that the page does not use a real HTTP redirect,
    # but instead has an http-equiv=refresh meta-element,
    # which directs to another location.
    # First parse our downloaded HTML
    print "Parsing HTML.\n" if ( $self->{verbose} );
    $self->{html parser}->parse( $page->{content} );
    # Next parse the http-equiv headers found from HTML
   my $http equiv list = $self->{html parser}->get http equiv();
    $self->{http_equiv_parser}->parse header list(
        $http equiv list
    );
    # get the refresh-address
   my $refresh address =
        $self->{http equiv handler}->get refresh url();
    if ( !$refresh address ) {
        # as there is no refresh-address, this means there
        # can be no redirection (well, of course there is a
        # chance of JavaScript based redirection, but this is
        # a way out of the leage of this small program).
        $self->parse html( $address );
    else {
        # Yes, indeed, we have a http-quiv=refresh redirection.
        # But wait! First we have to ensure it doesn't
        # redirect to the same page we're in - some pages use
        # http-refresh to simply refresh themselves after
        # a while.
        # Ensure, that we are dealing with absolute URI
        $refresh address = $self->convert to absolute uri(
            $refresh address,
            $address
        );
        # compare the URIs
        if ( URI::eq( $refresh address, $address ) ) {
```

```
# countinue...
                    $self->parse html( $address );
                }
                else {
                    # well, it's a redirect after all...
                    # So, we just replace our address with the
                    # redirect one
                    print "Found redirect from meta-element.\n"
                        if ( $self->{verbose} );
                    $address = $refresh address;
                    # clean all counters and other
                    # statistical attributes,
                    # because we don't need the data gathered
                    # from a HTML file,
                    # which simply redirects to another page
                    $self->{html parser}->init statistical attributes;
                    $self->{html_parser}->clean_count_cache;
                    $self->{http_equiv_handler}->empty_cache();
                    # jump into the beginning of this routine with...
                    # emm... goto.
                    goto BEGINNING OF PARSE ADDRESS;
                }
            }
        }
        else {
            # If the HTTP status code is different than "200 OK",
            # then conclude, that we have reached into an error page.
            my $status =
                $self->{headers handler}->get http status code();
            print "Failure: HTTP Status code $status\n"
                if ( $self->{verbose} );
            # Just write headers to database and
            # continue with next address.
            if ( my $webpage id = $self->insert webpage uri($address) ) {
                $self->{headers handler}->set current page id(
                    $webpage id
                );
                $self->{headers handler}->commit();
            }
        }
   }
   else {
        # Download failed. Most likely because of timeout, which is
        # only 10 seconds ( to be compatible with research of Parnas. )
       print "Download failed.\n" if ( $self->{verbose} );
        # Just insert address into database, but nothing more.
        $self->insert webpage uri( $address );
   }
}
```

# as the page just refreshes himself,

```
sub parse html {
   my $self = shift;
   my $address = shift;
    # insert webpage address into database
   my $webpage id = $self->insert webpage uri( $address );
    # if webpage address already existed in database,
    # exit and continue with next address.
    if ( !$webpage id ) {
        print "This address is already in database. Skipping.\n"
            if ( $self->{verbose} );
        $self->{html parser}->init statistical attributes;
        $self->{html parser}->clean count cache;
        $self->{http equiv handler}->empty cache();
        return;
    }
    # add HTTP headers to database
    $self->{headers handler}->set current page id( $webpage id );
    $self->{headers_handler}->commit();
    $self->{headers_handler}->empty_cache();
    # add Http-equivs to database
    $self->{http equiv handler}->set current page id( $webpage id );
    $self->{http equiv handler}->commit();
    $self->{http equiv handler}->empty cache();
    # add HTML data to database
    $self->{html parser}->set current page id( $webpage id );
    $self->{html parser}->commit();
    # Validate HTML
    $self->validate html( $webpage id );
    # save file-, text- and comment length and
    # validation errors of html file
    $self->{sql}->{webpages}->{set html sizes}->execute(
        stat( $self->{downloaded html filename} )->size,
        $self->{html_parser}->get_text_length,
        $self->{html_parser}->get_comment_length,
        $self->{html validator}->get error count,
        $webpage id
    );
    # Parse CSS and JavaScript
    $self->parse css( $address, $webpage id );
    $self->parse javascript( $address, $webpage id );
    # clean statistical attributes of HTML
    $self->{html parser}->init statistical attributes();
    $self->{html validator}->init statistical attributes();
    # remove HTML file
   unlink( $self->{downloaded html filename} );
}
```

```
# Validates downloaded HTML file.
# Commits results to database and cleans validator up after done.
sub validate html {
   my $self = shift;
   my $webpage id = shift;
    $self->{html validator}->validate(
        $self->{downloaded html filename}
    $self->{html validator}->set current page id( $webpage id );
    $self->{html validator}->commit();
    if ( $self->{verbose} ) {
        if ( $self->{html validator}->is valid ) {
            print "\033[32mVALID\033[0m HTML :)\n";
        }
        else {
            print
                "\033[31mINVALID\033[0m HTML :( " .
                $self->{html_validator}->get_error_count .
                " errors.\n"
        }
    }
}
# Parses inline, embedded and external CSS.
# commits results to database and cleans all up.
sub parse css {
   my $self = shift;
   my $html_address = shift;
   my $webpage id = shift;
   my parse error count = 0;
    # Parse and validate inline CSS
   my $inline css = $self->{html parser}->get inline css;
   my $inline css length = length($inline css);
    if ($inline css length > 0) {
        if ( $self->parse_inline_css( $inline_css ) ) {
            $parse_error_count ++;
        $self->validate inline css( $inline css );
    }
    # Parse and validate embedded CSS
   my $embedded css = $self->{html parser}->get embedded css;
   my $embedded css length = length($embedded css);
    if ( \$embedded css length > 0 ) {
        if ( $self->parse embedded css( $embedded css ) ) {
            $parse error count ++;
        }
```

```
$self->validate embedded css( $embedded css);
    }
    # get external css URIs,
    # first from <link> elements, and then from @import rules.
   my $external css files = $self->{html parser}->get external css;
        @$external css files,
        @{ $self->{css handler}->get imported stylesheets }
    # Parse and validate external CSS
   my $parse result =
        $self->parse_external_css( $external css files, $html address );
   my $external css count += $parse result->{count};
   my $external css length += $parse result->{length};
    $parse error count += $parse result->{parse errors};
    # commit all kind of CSS counts and sizes to database
    $self->{sql}->{webpages}->{set css sizes}->execute(
        $inline_css_length,
        $embedded css length,
        $external css length,
        $external css count,
        $self->{css handler}->get comment length,
        $self->{css handler}->get important count,
        $parse error count,
        $self->{css_validator}->get_error_count,
        $webpage id
    );
    # commit css statistics
    $self->{css handler}->set current page id( $webpage id );
    $self->{css handler}->commit();
    $self->commit_css_validation_results( $webpage id );
    # clean up
    $self->{css handler}->init statistical attributes();
sub parse inline css {
   my $self = shift;
   my $inline css = shift;
   print "Parsing inline CSS\n" if ( $self->{verbose} );
    eval {
        $self->{css parser}->parse style declaration( \$inline css );
    if ($@) {
        # if parsing fails with fatal error,
```

TLÜ Haapsalu Kolledž
Rene Saarsoo

```
print $0 if ( $self->{verbose} );
        return; # false
   else {
       return 1; # true
}
sub validate inline css {
   my $self = shift;
   my $inline css = shift;
   print "Validating inline CSS\n" if ( $self->{verbose} );
    # we close the style declarations inside simple selector "x",
    # as otherwise validator would certainly report it as invalid css.
    $inline css = "x{$inline css}";
    # write styles into temporary file
   open( my $file, '>', $self->{downloaded css filename} );
   print $file $inline css;
   close( $file );
    # validate the file
    $self->{css validator}->validate( $self->{downloaded css filename} );
    # after done, delete the file
   unlink( $self->{downloaded css filename} );
}
sub parse embedded css {
   my $self = shift;
   my $embedded css = shift;
   print "Parsing embedded CSS\n" if ( $self->{verbose} );
    #print $embedded css . "\n"; die "stop";
    eval {
        $self->{css parser}->parse( { string => $embedded css } );
    };
    if ($@) {
        # if parsing fails with fatal error,
       print $0 if ( $self->{verbose} );
        return; # false
    }
    else {
        return 1; # true
}
```

```
sub validate embedded css {
   my $self = shift;
   my $embedded css = shift;
   print "Validating embedded CSS\n" if ( $self->{verbose} );
    # write styles into temporary file
   open( my $file, '>', $self->{downloaded css filename});
   print $file $embedded css;
   close( $file );
    # validate the file
    $self->{css validator}->validate( $self->{downloaded css filename} );
    # when done, delete the file
   unlink( $self->{downloaded_css_filename} );
}
sub parse_external_css {
   my $self = shift;
   my $external_css_files = shift;
   my $base address = shift;
   my $external css count = 0;
   my $external css length = 0;
   my $parse error count = 0;
    foreach my $css address ( @$external css files ) {
        # resolve absolute address
        $css address = $self->convert to absolute uri(
            $css address,
            $base address
        );
        # download file with wget
        print "Downloading external CSS file $css address\n"
            if ( $self->{verbose} );
        if ( my $external css = wget(
               $css address,
               $self->{downloaded css filename}
        ) ) {
            # only accept downloads with 200 OK status code
            if ( \$external css->{headers}->[0] =~ m\{^\s*HTTP/.*200\}i ) {
                # parse css with CSS::SAC
                print "Parsing external CSS\n" if ( $self->{verbose} );
                eval {
                    $self->{css parser}->parse(
                        { string => $external css->{content} }
                };
                # if parsing fails with fatal error,
```

```
print $0 if ( $self->{verbose} );
                    $parse error count++;
                # Validate stylesheet
                print "Validating external CSS\n"
                    if ( $self->{verbose} );
                $self->{css validator}->validate(
                    $self->{downloaded css filename}
                # erase downloaded file
                unlink( $self->{downloaded css filename} );
                # parse all imported stylesheets
                my $parse result = $self->parse external css(
                    $self->{css handler}->get imported stylesheets,
                    $external css->{downloaded address}
                );
                # increment the count and length of external css files
                $external css count += $parse result->{count};
                $external css length += $parse result->{length};
                $parse error count += $parse result->{parse errors};
                # add the size of current file too
                $external_css_length += length(
                    $external css->{content}
                );
            }
        }
        # the count is incremented even if download failed
        $external css count++;
    # return the count of css files
    return {
        count => $external css count,
        length => $external css length,
        parse errors => $parse error count,
    };
}
# Validates downloaded CSS file.
# Commits results to database and cleans validator up after done.
sub commit css validation results {
   my $self = shift;
   my $webpage id = shift;
    $self->{css validator}->set current page id( $webpage id );
    $self->{css validator}->commit();
```

```
if ( $self->{verbose} ) {
        if ( $self->{css validator}->is valid ) {
            print "\033[32mVALID\033[0m CSS :) \n";
        else {
            print
                "\033[31mINVALID\033[0m CSS :( " .
                $self->{css validator}->get error count .
                " errors.\n"
        }
   }
   $self->{css validator}->init statistical attributes();
# Parses inline, embedded and external JavaScript.
# commits results to database and cleans all up.
sub parse javascript {
   my $self = shift;
   my $html address = shift;
   my $webpage_id = shift;
   # Gather all javascript together
    # get inline JS
   my $inline js = $self->{html parser}->get inline javascript;
   my $inline js length = length($inline js);
    # get embedded JS
   my $embedded js = $self->{html parser}->get embedded javascript;
   my $embedded js length = length($embedded js);
    # get external JS filenames
   my $external js files =
        $self->{html parser}->get external javascript;
    # Download all external JS
   my $external_js = "";
   my $external_js_count = 0;
   foreach my $js_address ( @$external_js files ) {
        # resolve absolute address
        $js address =
            $self->convert to absolute uri( $js address, $html address );
        # download file contents with wget (erase file after download)
        print "Downloading external JavaScript file $js address\n"
            if ( $self->{verbose} );
        if ( my $downloaded js = wget($js address, undef, 1) ) {
            # only accept downloads with 200 OK status code
            if ( \$downloaded js - > \{beaders\} - > [0] = m{^\s*HTTP/.*200}i ) {
                $external js .= $downloaded js->{content};
```

```
}
        $external js count++;
   my $external js length = length($external js);
    # Concatenate all JavaScript together
   my $js = $inline js . $embedded js . $external js;
    # search for XMLHttpRequest
   my $xml http request = 'N';
   if (\$ j s = \sqrt{XMLHttpRequest/s}) {
       print "Found XMLHttpRequest.\n" if ( $self->{verbose} );
        $xml_http request = 'Y';
    }
# for some unknown reason we get Segmentation fault
# when parsing javascript. so we just skip the whole thing.
#
     # Parse JavaScript
#
    if (length(\$js) > 0) {
        print "Parsing JavaScript\n" if ( $self->{verbose} );
#
#
         $self->{js parser}->parse( $js );
    # commit all kind of JS counts and sizes to database
    $self->{sql}->{webpages}->{set javascript sizes}->execute(
        $inline_js_length,
        $embedded_js_length,
        $external_js_length,
$external_js_count,
        $xml_http_request, # $self->{js parser}->get comment length,
        $webpage_id
    );
    # commit JS statistics
     $self->{js parser}->set current page id( $webpage id );
    $self->{js parser}->commit();
#
#
    # clean up
#
     $self->{js parser}->init statistical attributes();
```

```
# insert supplied webpage address into database and return associated ID
# if the address already exists in database, return false
sub insert webpage uri {
   my $self = shift;
   my $address = shift;
    if ( $self->{sql}->{webpages}->{insert}->execute( $address ) ) {
       return $self->{db}->{mysql insertid};
   else {
       return;
}
sub convert to absolute uri {
   my $self = shift;
   my $uri = shift;
   my $base_uri = shift;
   return URI->new abs( $uri, $base uri )->canonical->as string;
}
sub normalize_uri {
   my $self = shift;
   my $uri = shift;
   return URI->new( $uri )->canonical->as string;
=head1 AUTHOR
Rene Saarsoo <nene@triin.net>
This module is licensed under the same terms as Perl itself.
=cut
1;
```

# LISA 3. CD-ROM JA VEEBIST LEITAVAD MATERJALID

Selle tööga on kaasas CD-ROM, millele on salvestatud kõik uurimuse käigus kogutud andmed, kasutatud programmi lähtekood ning töö ise digitaalsel kujul. CD-ROM-i täpne sisu on järgmine:

webstat.tar.gz – sisaldab programmi webstat lähteteksti ja andmebaasistruktuuri, WDG HTML-i validaatorit, W3C CSS-i validaatorit, Perl'i moodulit CSS::SAC koos töö raames tehtud parandustega ja programmi installeerimisjuhiseid (failis INSTALL).

database/ – sisaldab kugu uurimuse käigus veebilehtedelt kogutud andmetest moodustatud andmebaasi. Failis main.sql on andmebaasistruktuur ning \*.txt.gz failides tabelite sisu. Failis INSTALL on toodud juhised andmete lahtipakkimiseks ja üleviimiseks MySQL andmebaasi.

veebipraktikad.pdf, veebipraktikad.odt, annotatsioon.pdf ja abstract.pdf – käesoleva töö digitaalne koopia PDF ning OpenDocument Text formaadis, ning eesti ja inglise keelne annotatsioon PDF kujul.

Lisaks on pea kogu CD-l leiduv materjal kättesaadav ka veebist:

http://www.triin.net/2006/05/10/webstat.tar.gz – programm webstat.

http://www.triin.net/2006/05/10/database.tar – uurimuse käigus koostatud andmebaas.

<u>http://www.triin.net/2006/05/10/veebipraktikad.pdf</u> – töö digitaalne koopia PDF formaadis.

http://www.triin.net/2006/05/10/veebipraktikad-annotatsioon.pdf ja http://www.triin.net/2006/05/10/webpractices-abstract.pdf – annotatsioonid.