

---

# Lausunto: Ohjelmistojen samankaltaisuus, VF Partner vs. Systemek

8.8.1997  
Prof. Ilkka Haikala,  
TTKK/Ohjelmistotekniikka  
PL 553  
33101 Tampere

(03)3652911  
ijh@cs.tut.fi

Olen jo aikaisemmin ilmaissut kahdesti käsitykseni VF Partnerin ja Systemekin välisestä maksupääteohjelman mahdollista kopiointia koskevasta kiistassa. Ensimmäisessä lausunnossani 8.12.1996 käsittelen syitä, joiden seurauksena saman ohjelmoijan uudelleen ohjelmoima ohjelma saattaa muistuttaa hänen aikaisemmin toteuttamaansa saman asian tekevää ohjelmaa. Lisäksi kommentoin Systemekin ja VF Partnerin ohjelmaversioiden välisiä samankaltaisuuksia niiltä osin, kun niitä oli käsitelty VTT:n Unto Pulkkinen asiasta antamassa lausunnossa. Johtopäätöksenä totesin, ettei VTT:n raportissa esitettyjen seikkojen perusteella voi osoittaa VF Partnerin ohjelmaa kopioksi. Tämän jälkeen minulta pyydettiin uutta lausuntoa, koska esiin oli otettu uusia kopioituiksi väitettyjä muistipaikkoja. Lausunnossani 24.2.1997 käyn ko. muistipaikkojen sisällöt verrattain seikkaperäisesti läpi ja totean, etteivät nekään muuta aikaisempaa käsitystäni. Muistipaikkojen lukumäärä oli tässä vaiheessa n. 40, kun koko ohjelmassa ohjelmakoodia on n. 300 muistipaikassa. Nyt olen saanut VTT:n uuden raportin, jossa tarkasteltujen muistipaikkojen määrä on jo n. 80 ja otan tästä syystä asiaan uudelleen kantaa.

Koska kaiken TCL-koodin perusteellinen läpikäyminen on tavattoman työläs urakka, laadin ohjelmien samankaltaisuuden selvittämiseksi muutaman yksinkertaisen ohjelman. Ensimmäinen ohjelma etsii ohjelmarivien lopussa olevia tabulointi- ja välilyöntimerkkejä sekä laskee tiedostoista ohjelmarivien ja kommentoitujen ohjelmarivien määrät. Toinen ohjelma tutkii ohjelmatiedostojen samankaltaisuutta vertailemalla muistipaikkoja keskenään. Ohjelman avulla pyrin saamaan kuvan ohjelmien samankaltaisuuden lisäksi myös niiden erilaisuudesta.

Käytin uuteen selvitystyöhön noin 8 työpäivää, yhteensä noin 80 tuntia. Tästä ajasta noin 4 päivää kului vertailussa käyttämäni ohjelmien tekemiseen ja noin 3 päivää kokeiden tekemiseen ja raportin kirjoittamiseen. Lopuksi käytin yhden päivän raportin viimeistelyyn ja tulosten tarkastamiseen.

## 1. Johdanto

### 1.1. Vertailun lähtökohdat

Aikaisemmissa vertailuissa johtopäätösten lähtökohtana olivat seuraavat seikat (suora lainaus ensimmäisestä lausunnosta):

---

mäisen lausuntoni kohdasta *Yhteenveto*):

- Koska oletan, että Pulkkisen analyysi ohjelmistoista on pääosin paikkansa pitävä, olen itse tutustunut vain osaan koko ohjelman koodista. Pulkkisen lausunnon mukaan vain pieni osa ohjelmistosta sisältää samankaltaisuutta (n. 10% ohjelmasta).
- Juselius ja Salojää ovat toteuttaneet Systekin ohjelman kopioiduiksi väitetyt osat tai ovat ainakin ylläpitäneet niitä ollessaan työsuhteessa Systekiin, joten he ovat tunteneet Systekin ohjelmassa käytetyt ratkaisut hyvin toteuttaessaan VF Partnerin ohjelmistoa.

Nyt kuitenkin VTT (tosin eri tutkijoiden toimesta) on monilta osin muuttanut kantaansa asiassa, joten aikaisemmilta lausunnoiltani putoaa osittain pohja pois. Tästä syystä en tässä launnossa oletta enää mitään VTT:n lausuntojen oikeellisuudesta. Oletuksen Juseliuksen ja Salojään monivuotisesta työskentelystä ko. ohjelmakoodin parissa oletan edelleen pitävän paikkansa.

## 1.2. Vertailtavat tiedostot

Vertailtavien ohjelmien yhteisiä ominaisuuksia on koottu taulukkoon 1. Sarakkeiden merkitykset ovat seuraavat.

- Tiedoston koko: tiedoston koko tavuina.
- Rivejä: ohjelman kaikkien rivien lukumäärä.
- Kommentoituja rivejä: rivejä, joilla esiintyy vähintään yksi puolipiste<sup>1</sup>. Puolipisteitä voi esiintyä joissain vakioimerkkijonoissa. Tämä saattaa aiheuttaa muutaman rivin virheen kommenttien määrässä.
- Ohjelmakoodimuistipaikkoja: niiden rivien lukumäärä, jotka alkavat \$-merkkiin päättyvällä muistipaikan numerolla. Muistipaikan numero voi alkaa myös &-merkillä. Tiedostossa VFPARTSO on 333 muistipaikkaa. Pankkikohtaisten erojen hoitamiseksi 38 muistipaikkaa esiintyy kuitenkin kahdesti, eli toimivassa ohjelmassa eri muistipaikkoja on todellisuudessa 295. Käsiteltäessä jatkossa ko. tiedostoa, vertailuissa otetaan mukaan vain tiedostossa ensimmäisenä oleva versio ko. muistipaikasta.
- Käskyjen kokonaismäärä: analyysiohjelman koodista löytämien konekäskyjen määrä. Tiedoston VFPARTSO osalta mukaan on laskettu vain ensimmäinen muistipaikka, jos sama muistipaikka esiintyy moneen kertaan.
- Komentoitamattomat hännät: niiden rivien lukumäärä, joilla ei ole kommenttia, mutta rivin lopussa on ylimääräisiä tabulointi- tai välilyöntimerkkejä.
- Kommentoidut hännät: niiden rivien lukumäärä, joilla on kommentti ja kommentin lopussa on ylimääräisiä tabulointi- tai välilyöntimerkkejä.

Tiedosto SYSTEKSO sisältää Systekin version maksupääteohjelmasta. VFPARTSO sisältää VF Partnerin version ohjelmasta. Tiedosto TL sisältää VF Partnerin tietoliikenneosuuden muistipaikat. Ne ovat mukana myös tiedostossa VFPARTSO lähes samassa muodossa kuin tiedostossa TL. Em. tiedostot on saatu postitse levykkeellä Vantaan nimismiespiiristä kesäkuussa 1997. VFPUUSI on VF Partnerin edustajan minulle toimittama versio VF Partnerin nykyisestä ohjelmasta. Halusin sen vertailuun mukaan, koska ko. ohjelma on jatkokehitetty tiedoston VFPARTSO ohjelmasta. Ohjelmassa on VF Partnerin edustajilta saamieni tietojen mukaan kirjoitettu tietoliikenneosuus kokonaan uudestaan ja myös muihin osiin on tehty runsaasti ylläpitotyötä<sup>2</sup>. Vertaamalla tiedostoja VFPARTSO ja VFPUUSI

---

1. TCL-ohjelmassa riville puolipisteen jälkeen jäävä osa on ns. ohjelmakommentti. Kommentti ei vaikuta mitenkään ohjelman toimintaan. Kommentteja lisätään ohjelmiin, jotta niiden toimintaa olisi helpompi ymmärtää esimerkiksi ylläpitotöiden yhteydessä.

2. Ylläpidoksi sanotaan ohjelmaan sen valmistumisen jälkeen tehtäviä muutoksia. Yleisesti arvioidaan, että ohjelmistotyöstä vain noin 1/3 on uusien ohjelmistojen kehitystyötä ja 2/3 vanhojen ohjelmien ylläpitoa.

	SYSTEKSO	VFPARTSO	TL	VFPUUSI
Tiedoston koko (tavuina)	152779	175793	29471	165966
Rivejä	6125	7801	1178	9631
Kommentoituja rivejä	2826	3128	528	3868
Ohjelmakoodimuistipaikkoja	285	333/295 *)	54	400/329
Käskyjen kokonaismäärä	5871	5812 **)	926	6442
Komentoimattomat hännät	224 (4%)	586 (8%)	101 (9%)	47 (0%)
Kommentoidut hännät	153 (3%)	308 (4%)	56 (5%)	35 (0%)

\*) osa muistipaikoista esiintyy useaan kertaan

\*\*\*) VFPARTSO:n samannumeroisten muistipaikkojen käskyt on laskettu vain kertaalleen)

**Taulukko 1:** Ohjelmien yleisiä ominaisuuksia .

keskenään saa kuvan siitä, millaista samankaltaisuutta tiedostoissa esiintyy, kun ohjelman jatkokehitetään toisesta ohjelmasta.

### 1.3. Merkintätavoista

Viittaa eri tiedostoissa oleviin muistipaikkoihin seuraavasti.

- v123: muistipaikka 123 VF Partnerin tiedostossa VFPARTSO.
- t123: muistipaikka 123 VF Partnerin tiedostossa TL.
- s123: muistipaikka 123 Systemin tiedostossa SYSTEKSO.

VTT:n uuden raportin sivuihin viittaa seuraavasti.

- VTTs5: VTT:n raportin sivu 5.
- VTTI4s5: VTT:n raportin liitteen 4 sivu 5 (tietoliikenneosuuden muistipaikkojen vertailu).
- VTTI5s5: VTT:n raportin liitteen 5 sivu 5 (muiden muistipaikkojen vertailu).

### 1.4. Raportin rakenne ja keskeisimmät havainnot

Raporttini loppuosan rakenne on seuraava.

- Luvussa 2. esitän yksityiskohtaisesti muistipaikkojen vertailussa käyttämäni menetelmän, ns. "samuusmitan". Menetelmän avulla pystyy etsimään vertailtavista ohjelmatiedostoista samankaltaisia muistipaikkoja. Menetelmää voi ainakin siinä mielessä pitää onnistuneena, että sen avulla löytää valtaosan VTT:n raportin liitteissä 4 ja 5 esitetyistä muistipaikkapareista.
- Luvussa 3. esitän graafisen esitystavan, jonka avulla kahden tiedoston samankaltaisista käskysekvensseistä saa helposti yleiskuvan (ns. "samuusprofiili"). Samalla tulee myös osoitettua, että

---

tiedostoissa saattaa olla yllättävänkin paljon samankaltaisuutta, vaikka ne eivät olisi kopioita toisistaan. Osoittaminen perustuu tiedoston SYSTEKSO vertaamiseen itseensä.

- Luvussa 4. vertaan em. menetelmän avulla tiedostoja VF PARTSO ja SYSTEKSO. Lopputulos vertailusta on, että samankaltaisuutta on verraten vähän ja että tiedostot ovat suurimmalta osin erilaisia. Samankaltaisuus näyttää olevan suurempaa tietoliikenneosuudessa.
- Luvussa 5. vertaan tiedostoja TL ja SYSTEKSO. Tässä tapauksessa samankaltaisuutta löytyy huomattavasti enemmän kuin luvun 4. vertailussa.
- Luvussa 6. tarkastelen tiedostoissa olevia “häntiä”, ts. rivien lopussa olevia näkymättömiä merkkejä. Häntiä voi käyttää todisteena sekä ns. tiedostokopioinnin tapahtumisesta, että myös todisteena siitä, että kopiointia ei ole tapahtunut. Häntävertailun perusteella näyttää siltä, että tämä ns. tiedostokopiointi on jossain vaiheessa tapahtunut joidenkin muistipaikkojen osalta, mutta ei systemaattisesti.
- Luvussa 7. esittelen systemaattisen tavan vertailla tiedostojen häntiä ja kommentteja toisiinsa. Teen menetelmän avulla vertailut eri tiedostojen välillä. Vertailut vahvistavat käsitystä, että samankaltaisuutta on pääasiassa tietoliikenneosuudessa.
- Luvussa 8. kommentoin VTT:n uuden raportin sisältöä. Raportti päättyy johtopäätökseen, jonka mukaan VF Partnerin ohjelma on jonkinasteinen kopio Systekin ohjelmasta. Osoitan VTT:n raportin päättyvän väärään johtopäätökseen. Johtopäätökset on mielestäni pääosin tehty asiallisesti, mutta yksi keskeinen lähtökohta on väärä: raportissa todetaan toistuvasti perusteluna, että “toisistaan riippumattomat” ohjelmoiijat eivät ole voineet päätyä näin samanlaisiin ratkaisuihin. Tämä lähtökohta on saamieni tietojen mukaan täysin väärä.
- Luvussa 9. selvitän vielä, miten lähdekieliseen ohjelmaan — tosin ilman kommentteja — voi perehtyä VeriFonen maksupäätteestä.

Yhteenveto havainnoista on, että verrattuna asiasta aikaisemmin näkemiini selvityksiin VTT:n uusi raportti ei juurikaan tuo lisävalaistusta käsillä olevaan oikeustapaukseen. Ainoa oleellinen lisä on tiedostokopioinnin havaitseminen joidenkin muistipaikkojen kohdalla. Vain hieman enemmän vaivaa nähden muistipaikkojen käskyt olisi voitu kopioida käsin paperilta tai saattaa skannerin ja merkkientunnistusohjelman avulla paperilta sähköiseen muotoon.

Esittämieni johtopäätösten yksi perusedellytys on, että vertailua varten laatimani ohjelmistot toimivat virheettömästi. Käytettävissä olevan ajan puitteissa ohjelmia ei voinut testata perusteellisesti ja ei ole tavatonta, että huolellakin tehdyistä ja testatuista ohjelmista löytyy pitkän käytön jälkeen virheitä. Ohjelmilla saadut tulokset näyttävät kuitenkin johdonmukaisilta ja ristiriidattomilta. Olen lisäksi suorittanut ristiintarkistuksia ja tarkastanut tuloksia käsin, vaikka se onkin monessa kohdassa erittäin työlästä. Tehtyjä tarkistuksia on kuvattu liitteessä 2. Pyrin myös suorittamaan tiedon siirron ohjelmien tulostiedostoista mahdollisimman mekaanisesti ja aina sähköisessä muodossa, jotta välttyisin “näppihäiriöiltä”. Käytetyt ohjelmat annan tarvittaessa mielelläni viranomaisen tai muun tutkijan tarkastettavaksi. Sama pätee tässä raportissa esitettyjen tulosten pohjana olleisiin ohjelmien tuottamiin numeerisia arvoja sisältäviin tiedostoihin.

## **2. Muistipaikkojen vertailumenetelmä**

Esitän tässä kohdassa käyttämäni muistipaikkojen vertailumenetelmän jolla voidaan mitata kahden muistipaikan samankaltaisuutta. Vertailun tuloksena saadaan samuusprosentti, joka vaihtelee nollan ja 100 prosentin välillä.

---

Maksupäätteen ohjelmat toteutetaan TCL-kielellä<sup>3</sup>. Ohjelmat jakautuvat muistipaikkoihin, joissa kussakin on joukko yhteenkuuluvia TCL-käskyjä (tyypillisesti muutamia kymmeniä). Muistipaikka vastaa lähinnä perinteisen ohjelmointikielen aliohjelmaa. Yhdessä muistipaikassa oleva koodi voi esimerkiksi laskea päivän järjestysnumeron vuoden alusta lukien tai lähettää sanoman tietoliikennelinjalle jne. Kahden muistipaikan samankaltaisuuden havaitsemista vaikeuttavat mm. seuraavat seikat

- samalle ohjelmariville saa kirjoittaa useita käskyjä,
- käskyt saa kirjoittaa joko pienillä tai isoilla kirjaimilla,
- ohjelmaan lisätyt välilyönnit sekä tabulointimerkit eivät vaikuta sen toimintaan,
- kommentit eivät vaikuta ohjelman toimintaan,
- muuttujien numeroita voidaan vaihtaa ohjelman merkityksen muuttumatta ja
- käytettyjä puskureita voidaan vaihtaa tietyin rajoituksin ohjelman merkityksen muuttumatta.

Näin muistipaikkojen sisällöt voivat olla samankaltaiset, vaikka ne ensi silmäyksellä näyttäisivät hyvinkin erilaisilta. Tästä on hyvänä esimerkki kuvan 1 muistipaikat, jotka tarkemman tarkastelun perusteella osoittautuvat hyvinkin samankaltaisiksi. Samankaltaisuuksien huomaamiseksi muistipaikat voi ensin muuttaa "luurankomuotoon", jossa mukaan otetaan vain kunkin käskyn operaatiokoodi, joka lisäksi ilmaistaan aina isolla kirjaimella. Käskyn muut osat, välilyönnit ja tabuloinnit sekä kommentit poistetaan. Näin saadaan kuvan 2 luurankomuodossa olevat ohjelmanpätkät, joista samankaltaisuutta voi jo silmämääräisesti löytää melko helposti: Systekin muistipaikan (vasemmalla) ensimmäisestä käskystä alkava käskysarja käskyyn 22 on sama kuin VF Partnerin käskysarja käskystä 2 käskyyn 24. Samaten Systekin käskysarja käskystä 27 käskyyn 33 on sama kuin VF Partnerin käskysarja käskystä 26 käskyyn 32.

Tällaisessa vertailussa on omat ongelmansa. Koska tarkastelemme vain käskyjen operaatiokoodeja, voivat samalta näyttävät käskysarjat tehdä aivan eri asioita. Jos käskysarjat ovat kuitenkin riittävän pitkiä, tämä on melko epätodennäköistä. Toisaalta käskyjen järjestystä vaihtamalla saa käskysarjat näyttämään erilaisilta. Käskyjen järjestyksen sekoittaminen on kuitenkin erittäin virhealtista puuhaa, varsinkin kun ottaa huomioon TCL-kielen hyppykäskymekanismiin, jossa on aina kerrottava, kuinka monen käskyn yli aiotaan hypätä. Luultavasti olisi helpompi tehdä koko ohjelman osa alusta alkaen uudelleen kuin sekoittaa sen käskyt radikaalisti uuteen järjestykseen.

Käytämäni muistipaikkojen vertailumenetelmä perustuu edellä kuvattujen yhteisten käskysarjojen etsimiseen. Ensin etsitään pisin yhteinen käskysarja (kuvan 2 esimerkissä Systekin käskyt 0-22), tämän jälkeen etsitään seuraavaksi pisin sarja (käskyt 27-33) jne. Lopuksi lasketaan löydettyjen yhteisten käskysarjojen pituudet yhteen. Muistipaikkojen samuusprosentiksi määritellään tämän summan prosenttiosuus muistipaikan pituudesta. Koska vertailtavat muistipaikat voivat olla eri pituisia, lasketaan prosenttiosuus aina pitemmän muistipaikan pituudesta. (Jos prosenttiosuus laskettaisiin lyhyemmästä muistipaikasta, voisi samuus olla 100%, vaikka muistipaikat eivät sisältäisikään täsmälleen samoja käskyjonoja.)

Kuinka pitkät samanlaiset käskysarjat sitten pitäisi ottaa huomioon? Hyvin lyhyillä sarjoilla satunnaiset samankaltaisuudet tulevat voimakkaasti mukaan ja pitkillä taas pienetkin muutokset koodissa estävät samankaltaisuuden havaitsemisen. Kutsun seuraavassa lyhimmän mukaan kelpuutettavan sarjan pituutta *samuusparametriksi* ja merkitsen sen arvoa kirjaimella D. Jos kaikki aivan lyhyetkin sarjat otettaisiin huomioon (ääritapauksessa yhden mittainen sarja, samuusparametri D=1), muistipaikat saattaisivat näyttää hyvinkin samanlaisilta vain siitä syystä, että niissä sattumalta tai tietyn ohjelmointitavan seurauksena esiintyy samoja käskyjä. Jos taas mukaan hyväksytään vain verraten pitkiä sarjoja (D on esimerkiksi 10), jo pienet muutokset koodissa saisivat muistipaikat näyttämään erilaisilta ja kovin lyhyet

---

3. Maksupäätteen ohjelmoinnin yksityiskohdista kiinnostunutta lukijaa varten olen liittänyt tämän raportin liitteeksi ensimmäisen lausuntoni yhteyteen tekemäni lyhyen kuvauksen TCL-ohjelmoinnista.

393\$		820\$	
I3.4.49	; if 0/1 tietue	G	
X1U73	; kanta/p,iv	I32.3.48	;JOS MUU KUIN 0-9 NIIN PRINT
I30		I3.4.49	
I4.2.50	; if 2 tietue	X1	
O1X3U77	; 3 ens.	U351	;KANTA/P_IVITYS
I25		I26	
I4.2.51	; if 3 tietue	I4.2.50	
O1X8U790	; kortin alku	O	
I20	; j,,nn"s+seur. tietue	X3	
I15.2.52	; if 4 tietue	U333	;3 EKAA
O1X1*Q#5	; lis,ys/poisto	I22	
GA790	; alku	I4.2.51	;3-TIETUE
X8U727	; loppu	O	
GA791		X8	
X1U791	; sarjasulku	U77	;ALKU
GXU90	; j,,nn"s	I17	
L359	; kielto.talletus	I12.2.52	;4-TIETUE
I3.2.57	; if 9 tietue	O	
O1X5T80	; korttti.lkm	X1	
GXU90	; j,,nn"s	*Q#5	;LIS_YS/POISTO
L392	; seur. tietue	G	
		A77	;ALKU
		X8	
		U399	;LOPPU = KOKO NUMERO 16
		G	
		O	
		X1	
		U40	;SARJASULKUKOODI
		L838	;KIELLON TALLETUS
		I4.2.57	;9-TIETUE
		O1	
		X5	
		T384	;KORTTIEN LUKUM__R_
		*M	
		G	
		X	
		*L79	

**Kuva 1:** Muistipaikat S393 (vasemmalla) ja V820 (VTTI5s30).

muistipaikat putoaisivat kokonaan pois vertailusta. Jos esimerkiksi kuvan 2 vertailussa asetettaisiin D=4, mukaan kelpuutettaisiin edellä esitetyt kaksi sarjaa. Jos asetettaisiin D=2, mukaan tulisi vielä kahden mittainen sarja, joka molemmissa muistipaikoissa alkaa kohdasta 34 (G, X).

Jotta saisin jonkinlaisen kuvan siitä, mitä D:n arvon valinta vaikuttaa, vertasin kokeeksi tiedoston SYSTEKSO muistipaikkaa s393 kaikkiin muihin saman tiedoston muistipaikkoihin. Tällöinhän pitäisi löytyä täsmälleen yksi muistipaikka, jonka samuusprosentti on 100% (eli siis tietenkin muistipaikka s393 itse). Muiden muistipaikkojen pitäisi vertailussa näyttää "riittävän" erilaisilta. Tämän kokeen tulos on esitetty kuvassa 3. Kuvasta näkee, että pienillä samuusparametrin D arvoilla monet muistipaikat näyttävät samankaltaisilta kuin muistipaikka s393. D:n arvolla 4 satunnaisuudesta seuraava samuus on yleensä parinkymmenen prosentin luokkaa. Tein vastaavan kokeen muutamilla muilla muistipaikoilla, ja tulokset olivat samansuuntaisia, joten päätin käyttää muissa kokeissa D:n arvoa 4. Esimerkiksi kuvan 2

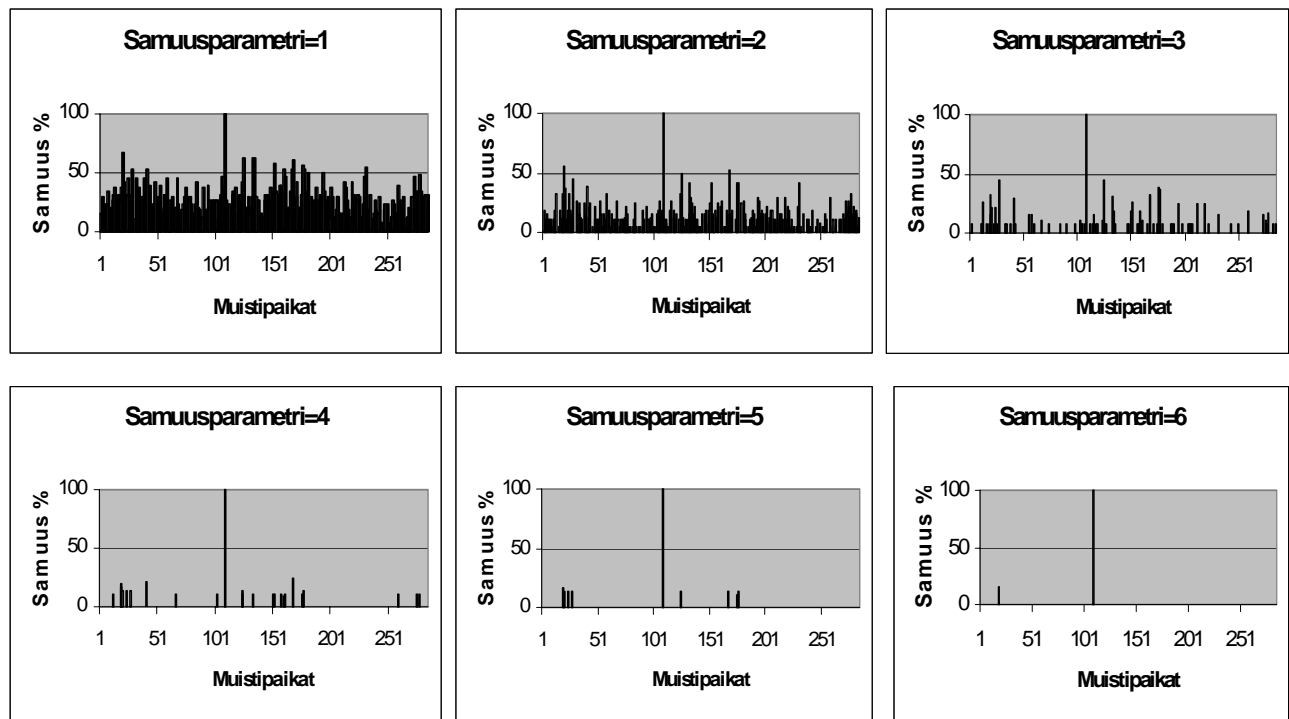
0:I	0:G
1:X	1:I
2:U	2:I
3:I	3:X
4:I	4:U
5:O	5:I
6:X	6:I
7:U	7:O
8:I	8:X
9:I	9:U
10:O	10:I
11:X	11:I
12:U	12:O
13:I	13:X
14:I	14:U
15:O	15:I
16:X	16:I
17:*Q	17:O
18:G	18:X
19:A	19:*Q
20:X	20:G
21:U	21:A
22:G	22:X
23:A	23:U
24:X	24:G
25:U	25:O
26:G	26:X
27:X	27:U
28:U	28:L
29:L	29:I
30:I	30:O
31:O	31:X
32:X	32:T
33:T	33:*M
34:G	34:G
35:X	35:X
36:U	36:*L
37:L	

**Kuva 2:** Muistipaikat S393 (vasemmalla) ja V820 luurankomuodossa.

muistipaikkojen s393 ja v820 samuusprosentiksi saadaan D:n arvolla 10 61%. D:n arvo 4 tuottaa samuusprosentin 79%. D:n arvolla 3 päädyttäisiin samaan prosenttilukuun. D:n arvolla 2 saataisiin prosenttiluvuksi 84% ja D:n arvolla 1 87%.

Tiedostossa SYSTEKSO on vain 3 muistipaikkaa, jotka ovat lyhyempiä kuin 4 käskyä, ja niiden osalta samuusprosentiksi tulee näin määriteltynä aina 0%. Vain kolme muistipaikkaparia on samuusparametrin arvolla D=4 100% samanlaisia — niiden luurankomuodot ovat täsmälleen samat.

Edellä esitetty samuusmitta on osoittautunut ainakin siinä mielessä järkeväksi, että sen avulla pystyy tiedostoista etsimään mekaanisesti muistipaikat, jotka ihmisenkin suorittaman vertailun perusteella näyttävät samoilta. Esimerkiksi VTT:n raportin liitteissä 4 ja 5 esitetyt muistipaikkaparit löytyvät muutamaa poikkeusta lukuunottamatta automaattisesti, kun annetaan samuusmitan laskevalle ohjelmalle toinen muistipaikoista ja annetaan sen etsiä toisesta tiedostosta muistipaikka, jonka kohdalla samuusmitta on suurin.



**Kuva 3:** Samuusparametrin vaikutus vertailtaessa muistipaikkaa S393 tiedoston Systekso muistipaikkoihin.

### 3. Samuusmitan ominaisuuksia, “samuusprofiili”

Samuusmitan ominaisuuksien arvioimiseksi olisi suotavaa, että käytössä olisi:

1. Kaksi ohjelmatiedostoa, joista kiistatta tiedetään, että vaikka ohjelmat tekevät samoja asioita, kummankaan kehityksessä ei ole käytetty toista lähtökohtana.
2. Kaksi ohjelmatiedostoa, joista toinen kiistatta tiedetään toisesta jatkokehitetyn versioksi.

Näitä tiedostopareja vertailemalla voitaisiin tutkia, miten samuusmitta käyttäytyy kun

- verrataan tiedostoja, joissa samankaltaisuus aiheutuu siitä, että samat ohjelmoijat tekevät saman ohjelman uudelleen ja
- verrataan tiedostoja, joista toinen on kopio, jota on hämäysmielessä yritetty muuttaa eri näköiseksi.

Kohdan 2 mukainen vertailupari saadaan tiedostoista VFPARTSO ja VFPUUSI, koska VFPUUSI tiedetään jatkokehitetyn versioksi tiedostosta VFPARTSO. Kohdan 1 mukaista vertailuparia ei kuitenkaan ole käytettävissä. Asiaa voi simuloida joko jakamalla jonkin tiedoston kahteen osaan tai vertailemalla tiedostoa itsensä kanssa. Molempia tapoja on sovellettu jatkossa, joten kuvaan ne lyhyesti seuraavassa.

Tiedoston voi jakaa esimerkiksi kahteen yhtäsuureen osaan ja verrata sitten tiedoston alkuosaa sen loppuosaan. Nyt tiedetään varmasti, etteivät tiedostot ole kopioita toisistaan. Ongelmaksi muodostuu kuitenkin se, että ohjelmat tekevät eri asioita, ja se että vertailtavassa tiedostossa on vähemmän

---

muistipaikkoja kuin alkuperäisessä. Vähäisemmän muistipaikkamäärän seurauksena “satunnaista samuutta” esiintyy vähemmän kuin todellisessa tilanteessa, koska todennäköisyys, että samanlaisia kohtia löytyy pienenee muistipaikkojen määrän pienentyessä.

Vertailtaessa tiedostoa itseensä kullekin muistipaikalle etsitään samasta tiedostosta sitä eniten muistuttava muistipaikka. Näin löydetty muistipaikka ei ole kopio vertailtavasta muistipaikasta, koska ei olisi järkevää laittaa ohjelmaan kahta samansisältöistä muistipaikkaa (periaatteessahan tämä tietysti on mahdollista). Samuusprosentti verrattuna löydettyyn muistipaikkaan kuvaa tällöin, millaista “satunnaista samanlaisuutta” aiheuttaa se, että samat ohjelmoijat käyttävät toistuvasti eri tilanteissa samantapaisia ratkaisuja. Tämäkään ei täysin vastaa kohdassa 1 esitettyä vertailuparia, koska vertailtavasta tiedostosta nyt puuttuu muistipaikka, joka tekee saman asian kuin vertailtava muistipaikka itse. Löydetty eniten muistuttava muistipaikka saattaa siis usein olla itse asiassa toiseksi eniten muistuttava muistipaikka, jolloin “satunnaista samuutta” siis saattaa esiintyä vähemmän kuin todellisessa kohdan 1 mukaisessa tilanteessa.

Molemmilla yllä kuvatuilla tavoilla saadut samuusprosentit antavat joka tapauksessa jonkinlaisen kuvan siitä, miten “satunnaista samuutta” esiintyy muistipaikkojen välillä, kun samat ohjelmoijat kirjoittavat uutta ohjelmaa TCL-kielellä. Tuloksia tarkasteltaessa on kuitenkin pidettävä menetelmän heikkoudet mielessä.

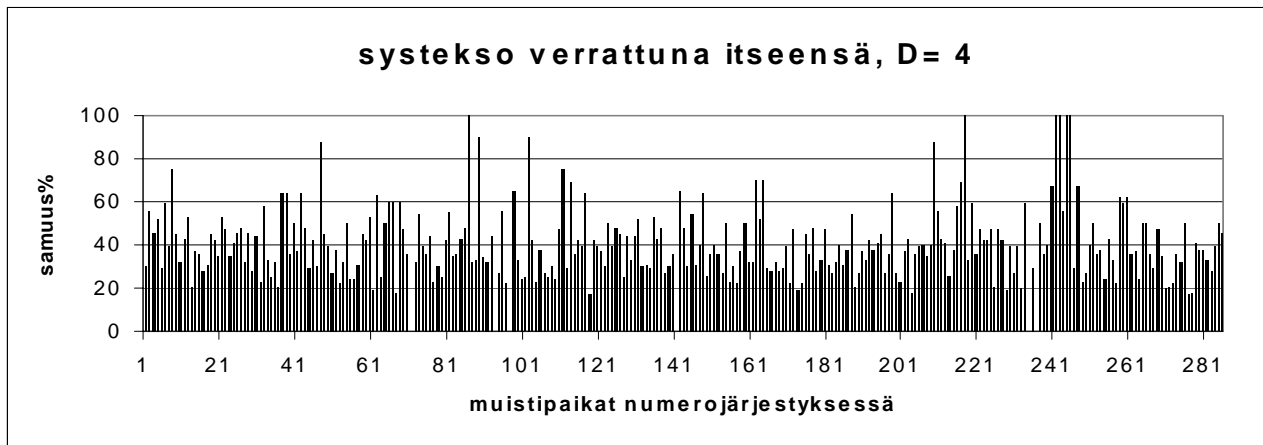
“Satunnaisen samuuden” määrän arvioimiseksi vertasin ensin tiedostoa SYSTEKSO itseensä edellä kuvatulla tavalla. Tulos on esitetty kuvassa 4A. Muistipaikat, yhteensä lähes 300, on sijoitettu vaaka-akselille ja kunkin muistipaikan samuusprosentti sitä eniten muistuttavan muistipaikan kanssa esitetään kuvassa pylväänä. Kuvasta näkee esimerkiksi, että ensimmäiselle muistipaikalle löytyy pari, jonka samuusprosentti on 30% prosenttien paikkeilla.

Koska muistipaikkoja on paljon, kasvavat vierekkäiset pylväät toisiinsa kiinni ja kuvasta on muutenkin vaikea muodostaa yleiskäsitystä. Kuva muuttuu selkeämmäksi, kun lajitellaan pylväät suuruusjärjestykseen samuusprosentin perusteella. Tämä on tehty kuvassa 4B. Jatkossa esittämässäni kuvissa onkin systemaattisesti sovellettu pylväiden lajittelua.

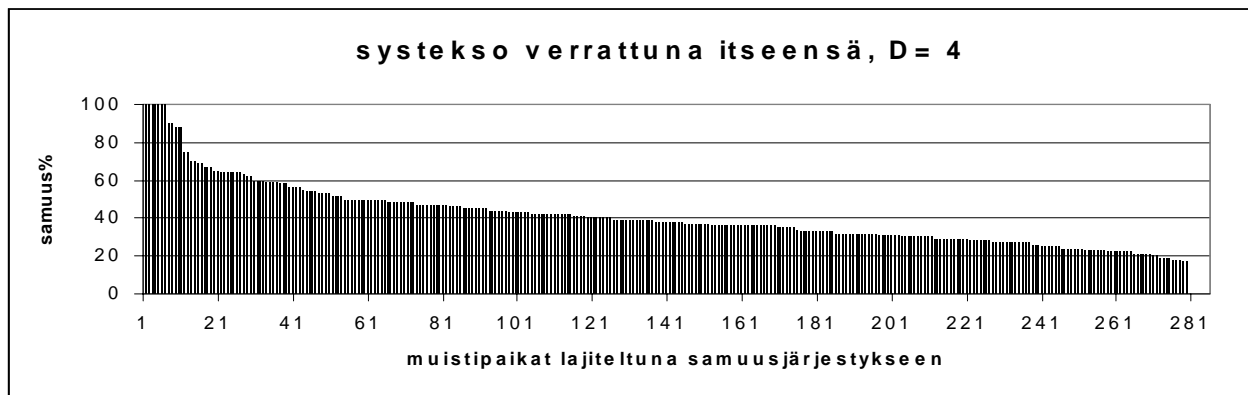
Kuvasta 4 näkee, että satunnaista samanlaisuutta esiintyy ehkä yllättävänkin paljon. Pylväiden korkeus kuvassa on keskimäärin noin 40%. Huomiota herättää se, että kolmelle muistipaikalle löytyy 100% kopio. Tämä näkyy kuutena 100 prosenttiin yltävänä pylväänä kuvissa 4A ja 4B. Tarkempi tarkastelu osoittaa, että kyseessä ei ole aivan satunnainen samankaltaisuus, vaan muistipaikoissa olevat ohjelmat tekevät lähes samoja asioita. Esimerkiksi yhdessä tapauksessa parin toinen muistipaikka kasvattaa sanomanumeroa ja toinen vähentää sitä. Luurankomuodossa ohjelmakoodi on täsmälleen sama.

Seuraavaksi tein kokeen, jossa jaoin tiedoston SYSTEKSO kahteen yhtäsuureen osaan: OSA1 ja OSA2. Nyt kyseessä on kaksi tiedostoa, jotka eivät missään tapauksessa ole kopioita toisistaan, ja niiden sisältämä koodi lisäksi tekee aivan eri asioita. Suoritin vertailun hakemalla jokaiselle alkuosan muistipaikalle sitä eniten muistuttavan muistipaikan tiedoston loppuosasta ja piirsin siitä kuvaa 4B vastaavan kuvan. Vertailun tulos on nähtävissä kuvassa 5. Kuvasta on häivytetty yksittäiset pylväät yhdistämällä ne yhtenäiseksi vaalean harmaaksi alueeksi. Alueella on sama luonteenomainen muoto kuin kuvan 4B tummalla alueella. Pylväiden keskimääräinen korkeus on nyt noin 30%, joka on noin 10 prosenttiyksikköä pienempi kuin kuvan 4 vertailussa. Pienempi prosenttiluku selittyy osittain sillä, että nyt vertailtavat ohjelmasosat tekevät aivan eri asioita. Enimmältään tämä johtuu kuitenkin siitä, että nyt vertailtavia muistipaikkoja on puolet vähemmän. Kun muistipaikkoja on vähemmän, pienenee todennäköisyys, että niistä sattumalta löytyy samanlaisia kysymyksiä.

Jos joku nyt haluaisi väittää — tosin virheellisesti — tiedostoa OSA1 tiedoston OSA2 kopioksi, asiaa olisi hankala perustella puoleen tai toiseen kuvan 5 avulla. Tämän ongelman ratkaisemiseksi kokeilin seuraavaa ideaa. Kun muistipaikkaa eniten muistuttava muistipaikka on löydetty, etsitään sitä *toiseksi eniten* muistuttava muistipaikka. Jos eniten muistuttava muistipaikka nimittäin oletetaan kopioksi, ei sitä

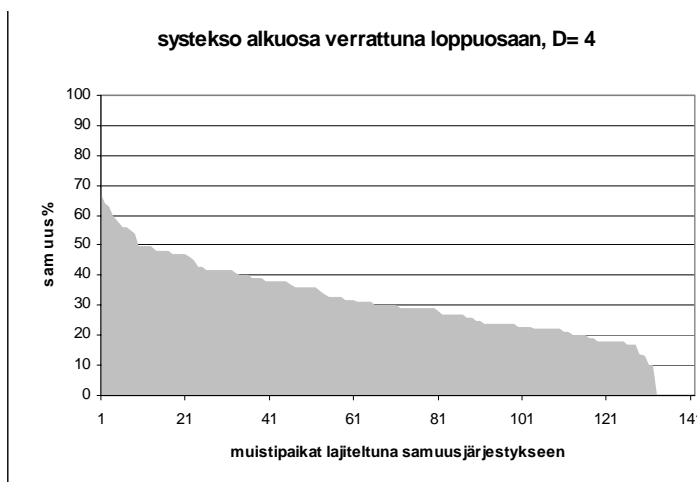


A) Muistipaikat esiintymisjärjestyksessä

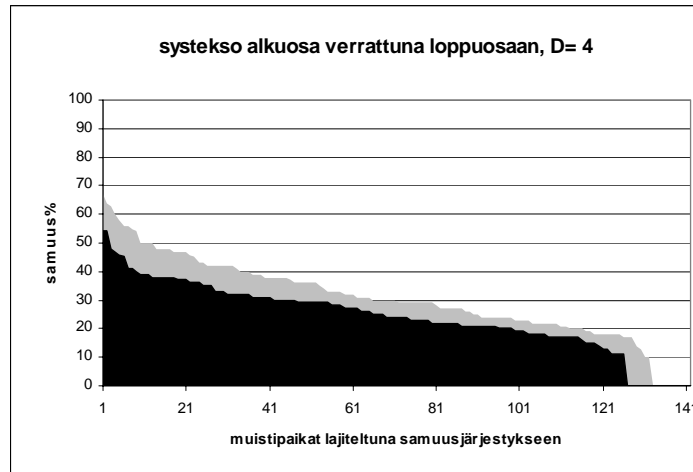


B) Muistipaikat samuusjärjestyksessä

Kuva 4: Tiedosto SYSTEKSO verrattuna itseensä.



Kuva 5: Tiedoston SYSTEKSO alkupuoliskon muistipaikat verrattuna loppuosan muistipaikkoihin.



**Kuva 6:** Tiedoston SYSTEKSO alkuosa verrattuna loppuosaan. Harmaa alue = eniten muistuttavat muistipaikat, musta alue = toiseksi eniten muistuttavat muistipaikat.

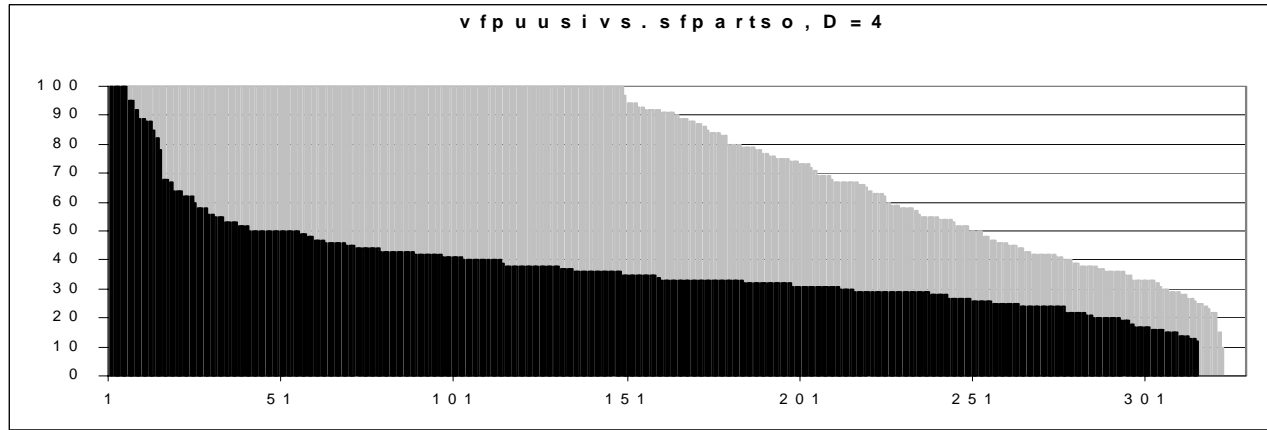
toiseksi eniten muistuttava muistipaikka käytännössä voi olla kopio. (Sillä eihän muistipaikka voi olla kahden eri muistipaikan kopio, kuin vain siinä tapauksessa, että samansisältöinen muistipaikka esiintyy ohjelmassa kahdesti. Tämä on erittäin harvinaista.) Piirtämällä samaan kuvaan näkyviin sekä muistipaikkaa eniten muistuttavat muistipaikat ja niitä toiseksi eniten muistuttavat muistipaikat, saa kuvan käytetyn vertailumenetelmän aiheuttamasta “satunnaisesta” samuudesta. Tämä on tehty tiedostojen OSA1 ja OSA2 vertailun osalta kuvassa 6. Ylempi vaalea alue esittää jo kuvassa 5 esitettyjä eniten muistuttavia muistipaikkoja ja alempi tumma alue toiseksi eniten muistuttavia muistipaikkoja.

Kuvan 6 voi katsoa olevan muodoltaan luoneenomainen, kun verrataan kahta eri ohjelmaa (tosin mahdollisesti samojen ohjelmoijien tekemää) keskenään. Alueiden väliin jäävän alueen korkeus on keskimäärin n. 6%. Jos vertailun kohteena oleva tiedosto on isompi ja ohjelmat lisäksi tekevät samantapaisia asoita, ovat käyrät keskimäärin korkeammalla ja niiden välinen ero myös kasvaa. Kuvan 4 perusteella voi arvioida, että kaksi kertaa isommassa tiedostossa käyrät kulkevat todennäköisesti n. 10% korkeammalla ja niiden välinen ero on 10% luokkaa. Ero voi tosin ehkä olla jonkin verran suurempikin ilman, että sitä voi pitää todisteena kopioinnista, koska kuvassa 4 on vertailtu tiedostoa itseensä. Tällöin vertailusta puuttuu vertailtavan muistipaikan kanssa saman asian tekevä muistipaikka. Jos sama ohjelmoija toteuttaa samansisältöisen muistipaikan uudelleen, syntyy joissain tapauksissa pitkiäkin samansisältöisiä sekvenssejä (esimerkiksi pankkistandardin määrittelemän sanoman kokoaminen).

Millainen olisi tulos, jos vertailtavana olisi kaksi tiedostoa, joista toinen on jatkokehitetty versio toisesta? Tutkin tätä vertailemalla tiedostoa VFPUUSI tiedostoon VFPARTSO. VFPUUSI on jatkokehitetty versio tiedostosta VFPARTSO.

Vertailun tulos on kuvassa 7. Kuvaan on kuvan 6 tapaan piirretty sekä eniten että toiseksi eniten muistuttavat muistipaikat. Kun kuvaa vertaa kuvaan 6, huomaa vaalean alueen täysin erilaisen käyttäytymisen. Vaalean ja tumman alueen reunat muodostavien käyrien ero on nyt keskimäärin 43%, kun edellisen kuvan perusteella tekemieni johtopäätösten perusteella voisi odottaa n. 10% suuruusluokkaa olevaa eroa. Noin puolet muistipaikoista osoittautuu 100% samoiksi. Vaalea alue on aivan eri muotoinen kuin aikaisemmissa kuvissa. Tumma alue on suunnilleen saman muotoinen kuin kuvassa 6 ja myös odotusten mukaisesti n. 10 prosenttiyksikköä korkeammalla.

Kuva 7 osoittaa havainnollisesti tiedostojen läheisen sukulaisuuden: vaalean ja tumman alueen reunojen muoto ja etäisyys toisistaan toimii eräänlaisena “samuusprofiilina”, josta ohjelmien läheisen sukulais-



**Kuva 7:** Tiedostojen VFPUUSI ja VFPARTSO vertailu.

uuden erottaa ainakin selvissä tapauksissa yhdellä silmäyksellä. Käytänkin tätä kuvatyyppeä seuraavissa tiedostojen samuusvertailuissa.

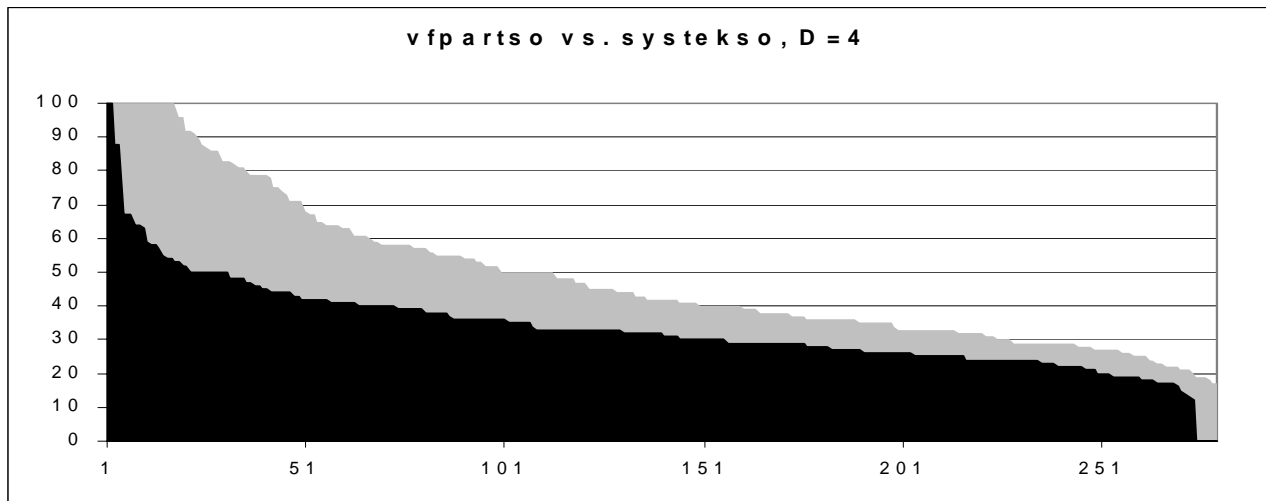
Edellä tekemäni samuusvertailut, joissa tiedoston SYSTEKSO muistipaikkoja on vertailtu toisiinsa, osoittavat mielestäni kiistattomasti, että samojen ohjelmoijien toteuttamat, jopa eri asioita tekevät muistipaikat, voivat muistuttaa toisiaan hämmästyttävän paljon.

#### 4. Tiedoston VFPARTSO analysointi

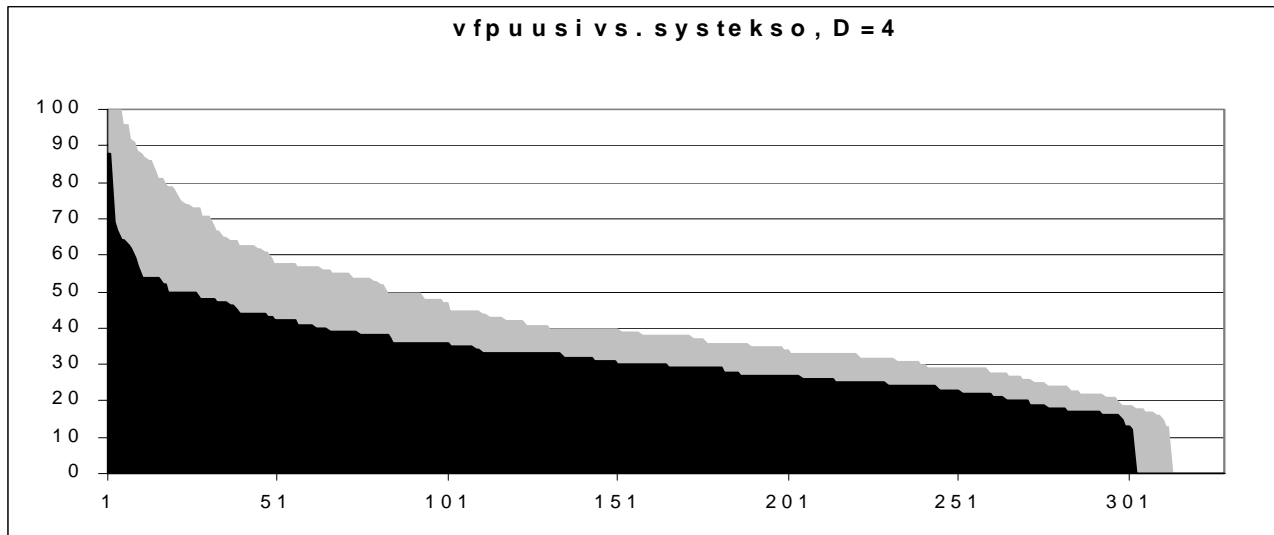
Edellisen kohdan kuvien 6 ja 7 tapaan laadin samuusprofiilin vertailemalla tiedostoa VFPARTSO tiedostoon SYSTEKSO. Vertailussa on mukana myös tiedostossa TL olevat muistipaikat, joita tarkastellaan vielä erikseen seuraavassa kohdassa. Saman muistipaikan esiintyessä useaan kertaan tiedostossa VFPARTSO, on vertailuun otettu mukaan vain sen ensimmäinen esiintymä. Samuusparametrin D arvo oli 4 (kuten kaikissa muissakin kokeissa jatkossa).

Vertailun lopputulos on nähtävissä kuvassa 8. Kuvassa pystyakselilla on samuusprosentti ja vaakakselille on sijoitettu löydetty samuusprosentit lajiteltuna laskevaan suuruusjärjestykseen. Ylempänä oleva vaaleampi alue kuvaa eniten muistuttavia ja alempi tumma alue toiseksi eniten muistuttavia muistipaikkoja. Kuvasta voi lukea mm. että noin sadan muistipaikan osalta samuus on enemmän kuin 50%. Toisaalta samaan aikaan toiseksikin parhaiten muistuttavien samuus on vajaat 40%. Käyrien väliin jäävän alueen voi ajatella esittävän samankaltaisuutta, joka on seurausta esimerkiksi siitä, että sama ohjelmoija koodaa saman asian tekevän ohjelmakohdan sattumalta samalla tavalla, on käytetty samaa lähdemateriaalia tai standardit pakottavat tekemään tietyn asian samalla tavalla. Sen keskimääräinen koko on noin 15%, joka on hieman suurempi, kuin mitä voisi odottaa pelkän sattuman tuloksena edellisessä kohdassa tekemieni päätelmien nojalla (n. 10%). Muilta osin ohjelmat kuitenkin ovat erilaiset ainakin tässä käytetyn samuuskriteerin perusteella.

Vertasin myös VF Partnerin ohjelmiston nykyistä versiota VFPUUSI Systemin ohjelmaan. Tulos on kuvassa 9. Nyt käyrien väliin jäävä alue on supistunut jo keskimäärin noin 11 prosenttiin. Jos hyväksytään ajatus, että nykyinen versio, jossa merkittäviä osia ohjelmasta on kirjoitettu kokonaan uudelleen, on VF Partnerin omaa tuotantoa, eikä kopio, voi kuvien 8 ja 9 samankaltaisuuden perusteella päätellä, että Systemin ja VF Partnerin ohjelmat ovat vain pieneltä osin samanlaisia.



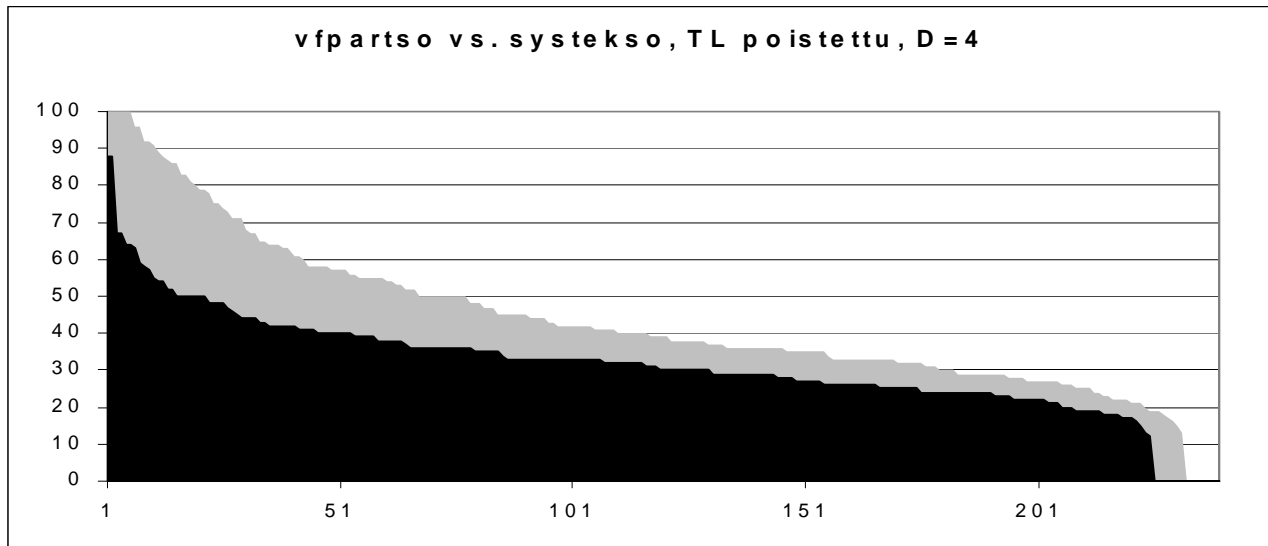
**Kuva 8:** Tiedostojen VFPARTSO ja SYSTEKSO vertailu.



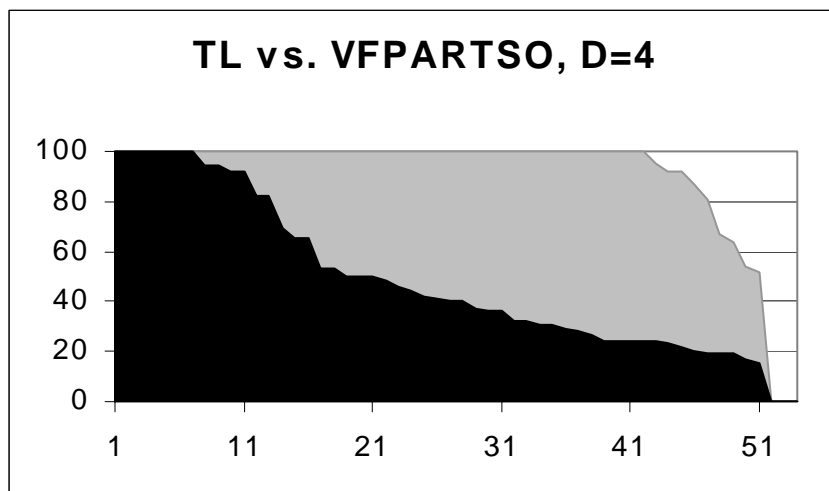
**Kuva 9:** Tiedoston VFPUUSI muistipaikat verrattuna tiedoston SYSTEKSO muistipaikkoihin.

Koska on otaksuttavaa, että merkittävä osuus samankaltaisuudesta tulee tietoliikenneosuuden muistipaikoista (54 kappaletta), suoritin myös mittauksen, jossa tietoliikenneosuuden muistipaikat on poistettu tiedostosta VFPARTSO (V400-V499). Tulos on kuvassa 10. Käyrien väliin jäävän alueen koko on nyt keskimäärin noin 12%, mikä edelleen tukee olettamusta, että samankaltaisuutta on pääasiassa vain tietoliikenneosuuden muistipaikoissa.

Lopuksi vertasin — lähinnä tarkistusmielessä — tiedostossa TL olevia muistipaikkoja tiedoston VFPARTSO muistipaikkoihin (kuva 11). Kuvasta näkee, että muistipaikkojen sisällöt ovat lähes samat molemmissa tiedostoissa, eli tiedostossa VFPARTSO olevat tietoliikennemuistipaikat poikkeavat tiedoston TL muistipaikoista vain muutamien muistipaikkojen (9) osalta. (Vaaka-akselin oikeassa reunassa näyttäisi olevan 3 muistipaikkaa, joiden osalta samuusprosentti on nolla. Kyseessä ovat kuitenkin ne



**Kuva 10:** Tiedostojen VFPARTSO ja SYSTEKSO vertailu, tietoliikenneosuus poistettu aineistosta.

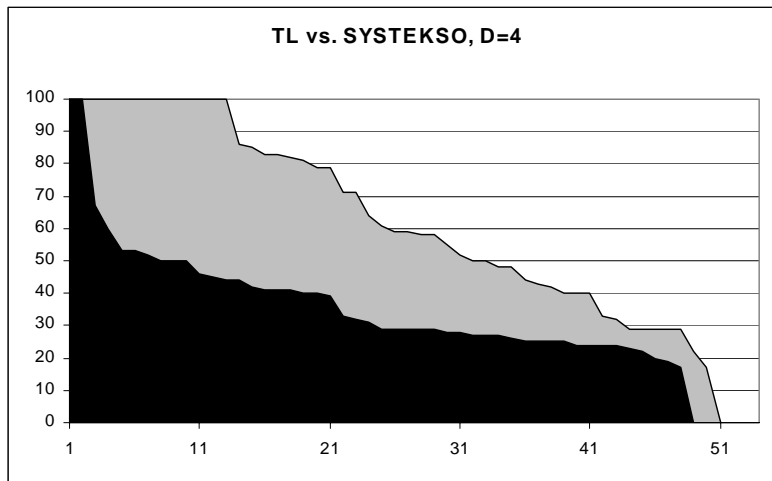


**Kuva 11:** Tiedoston TL muistipaikat verrattuna tiedoston VFPARTSO muistipaikkoihin.

kolme muistipaikkaa, joiden pituus on alle neljä käskyä, joten niiden osalta samuusprosentti jää nollassi, vaikka ne ovatkin sisällöltään identtisiä.)

## 5. TL vs. SYSTEKSO

Tiedoston TL muistipaikkojen vertailu tiedoston SYSTEKSO muistipaikkoihin on esitetty kuvassa 12. Kuva paljastaa, että nyt samankaltaisuutta on huomattavasti enemmän kuin aikaisemmissa vertailuissa. Käyrien väliin jäävä alue on nyt keskimäärin 27%. Valtaosa tästä samankaltaisuudesta on selitettävissä



**Kuva 12:** Tiedostojen SYSTEKSO ja TL vertailu

sillä, että tietoliikenneosuuden algoritmit määräytyvät suurelta osin standardien perusteella (Kermit, pankkistandardit). Tätä samankaltaisuutta olen kommentoinut muistipaikkakohtaisesti jo edellisessä lausunnossani. Yhteenvedona voi todeta, että ohjelman tässä osassa standardit ohjaavat ohjelmakoodia erittäin tarkasti. Toisaalta ohjelmakoodi on yksinkertaista, eikä mielestäni ole hämmästyttävää, että ohjelmoija uudelleenikäyttää samaa ohjelmaa toteuttaessaan kerran koodattuja standardipalasia. Tyypillisiä esimerkkejä tällaisista ohjelman paloista ovat esimerkiksi BCD-luvun pakkaus, tarkistussumman laskeminen, tietyn sanoman muodostaminen linjalle lähettämistä varten jne.

## 6. Kommentit ja “hännät”

Vertailtaessa muistipaikkoja voidaan samankaltaisuutta yrittää löytää myös ohjelman kirjoitusasusta. VTT:n uudemman raportin liitteissä 4 ja 5 on pyritty analysoimaan kommentteja ja erityisesti ns. häntiä. Hännällä tarkoitan tässä rivin lopussa olevia näkymättömiä turhia merkkejä (välilyötejä ja/tai tabulointimerkkejä).

Vertailun idea on, että jos ohjelmaa muutetaan siten, että rivin lopussa oleva kommentti poistetaan, saattaa riville kuitenkin jäädä vahingossa loppuun näkymättömiä merkkejä. Jos tällaisia häntiä on systemaattisesti sellaisissa kohdissa, joissa alkuperäisessä ohjelmassa on kommentteja, voidaan suurella todennäköisyydellä olettaa kopioinnin tiedostosta toiseen tapahtuneen. Erityisen selvältä tilanne tuntuu, jos alkuperäisessä ohjelmassa on häntä ja kopioituksi väitetyssä ohjelmassa on häntä samassa kohdassa, koska häntää ei todennäköisesti huomaisi poistaa kopioinnin yhteydessä. (Kopioinnilla tarkoitetaan tässä ohjelmakoodin osan siirtämistä suoraan sähköisessä muodossa tiedostosta toiseen. Ohjelmoija voi esimerkiksi tekstieditorin avulla “leikata” palasen toisesta tiedostosta “leikkuulaudalle” ja “liimata” sen editorin komennolla toiseen tiedostoon.)

Toisaalta häntien avulla voi mahdollisesti osoittaa myös päinvastaista. Jos alkuperäisessä tiedostossa on häntä ja kopioksi väitetyssä kohdassa sitä ei ole, tuntuu todennäköiseltä, että ainakaan suoraa tiedostokopiointia ei ole tapahtunut, koska häntää ei todennäköisesti olisi huomannut poistaa. Jos alkuperäisessä tiedostossa on kommentti, mutta kopioksi väitetyssä ei ole kommenttia eikä häntää, voisi sen myös jossain määrin ajatella todistavan, ettei kopiointia ole tapahtunut: jos häntiä on joissain tapauksessa jätetty, niin miksi ei sitten lähes poikkeuksetta?

---

VTT:n raportin logiikan mukaan jo häntien olemassaolo on epäilyttävää (VTTs 9, kohta 6: "... Ainoa selitys tällaisille..."), koska ohjelmoijat eivät sen mukaan laita ohjelmiinsa edellä kuvattuja häntiä. Tämä väite on kuitenkin omituinen, koska taulukon 1 mukaan häntiä on myös Systekin ohjelmassa useita satoja. Tämän logiikan mukaanhan siis myös Systekin ohjelman pitäisi olla plagiaatti jostain aikaisemmasta ohjelmasta. Häntä syntyy helposti, kun ohjelmoija jää miettimään seuraavaa käskyä ja mahdollisesti samalla pohtii, pitäisikö kommentti kirjoittaa. Kommentille tuleva sisennys tulee mahdollisesti tehtyä, mutta itse kommentti jää kirjoittamatta.

Häntiä esiintyy kaikissa tarkastelemissani tiedostoissa. Suhteellisesti eniten niitä on tiedostossa TL, jonka riveistä 9% on ilman kommenttia ja hännällisiä. 5% riveistä on sellaisia, että niillä olevan kommentin lopussa on häntä. Yhteensä siis n. 14% riveistä päättyy häntään. Tiedostossa VFPARTSO vastaavat luvut ovat 8% ja 4%, eli yhteensä 12% ja tiedostossa SYSTEKSO 4% ja 3% eli yhteensä 7%. Sekä Systekin että VF Partnerin ohjelmat tuottaneista ohjelmoijista yksi tai useampi tuntuu olleen poikkeuksellisen "raskassorminen", koska häntiä on kaikissa tiedostoissa näinkin paljon.

En käynyt tiedostoja systemaattisesti läpi häntävertailun osalta, mutta jo pintapuolisenkin tarkastelun perusteella huomaa, että tarkasteltavissa tiedostoissa esiintyy kaikenlaisia tapauksia: Vertailtaessa vastinmuistipaikkoja hännät ovat toisinaan samoissa kohdissa, toisinaan taas hännät ovat eri kohdissa, joskus Systekin koodissa on kommentti ja VF Partnerin koodissa vastaavalla kohdalla häntä, toisinaan taas häntää ei ole jne. Todisteita sekä kopioinnin puolesta että sitä vastaan siis löytyy.

VTT:n raportin liitteistä 4 ja 5 löytyy joitain muistipaikkoja, joissa hännät osuvat silmiinpistävästi sopivasti kohdalleen ja kommentitkin ovat samankaltaisia. Näiden muistipaikkojen osalta on helppo uskoa tiedostokopioinnin jossain vaiheessa tapahtuneen. Tyypillinen esimerkki on esimerkiksi muistipaikkapari s717&t436 (VTT14s2). Kyseessä on osa maksupäättevarmennusstandardin määrittelemän BCD-koodin pakkauksen tekevistä koodista. Samalla on luultavasti kopioitu samaan algoritmiin liittyvät muistipaikat t435, t434 ja t432, jotka kaikki löytyvätkin tietoliikenneosuuden kopioituksi väitettyjen muistipaikkojen listan kärjestä. Kopioinnin varmistane se, että kaikkien neljän muistipaikan osalta samuusprosentti Systekin muistipaikkojen kanssa on 100%.

Edellisessä kohdassa tehtyjen samuusprosenttiin perustuvien vertailujen perusteella näyttää joka tapauksessa siltä, että kopiointia ei kuitenkaan ole tehty systemaattisesti, vaan se koskee vain verraten pientä ohjelman osaa. Kun lisäksi kopioitujen osien toiminnan määräävät pankki- ja tietoliikennestandardit, ei niitä voi pitää mitenkään innovatiivisina, vaan aivan tavallisena yksikertaisena rutiinikoodina.

Ohjelmointityössä on tavallista, että ohjelmaa kirjoitettaessa uudelleenkäytetään aikaisemmin kirjoitettua koodia. Esimerkiksi kirjoittaessani tämän raportin laadinnassa tarvitsemiani ohjelmia, hyödynsin osia kahdesta aikaisemmin kirjoittamastani ohjelmasta ja lisäksi kopioin useissa yhteyksissä koodinpätkiä käyttämäni ohjelmointikielen (C++) kääntäjän mukana tulleista esimerkkitiedostoista. Tämä on yleinen käytäntö ohjelmia tehtäessä. Eräässä toisessa tuntemassani maksupäätteen toteutusprojektissa tietoliikenneosuus saatiin sovittamalla ohjelmaan julkisohjelmana saatavissa oleva C-kielinen Kermit-protokolla<sup>4</sup>. Käytännössä ohjelmoijalle kertyy vuosien varrella koodia, jota hän hyödyntää uudelleen ja uudelleen eri yhteyksissä — toisinaan myös työnantajan välillä vaihduttua. Ohjelmoija ei tule näin tehdessään helposti ajatelleeksi, että asiaan saattaa jonkun tulkinnan mukaan liittyä tekijänoikeuden loukkaus, varsinkin jos kyseessä ovat yksikertaiset ja verraten helposti uudelleen tuotettavat koodinpätkät.

---

4. Lauri Pesonen, *Nokia2000 -maksupäättejärjestelmä*. Diplomityö, TTKK (1990) 100s.

---

## 7. Häntien ja kommenttien vertailu tilastollisesti

Häntien vertailu käsityönä on erittäin työlästä puuhaa ja johtaa helposti vääriin johtopäätöksiin. Yritän seuraavassa kehittää vertailuun systematiikkaa, vaikka se tässä tapauksessa onkin hankalampaa kuin käskysekvenssejä etsittäessä.

Vertailtaessa keskenään kahta tiedostoa kommenttien ja häntien osalta voidaan yrittää etsiä tiedostoista vastinkohtia ja tämän jälkeen verrata rivejä keskenään. Vertailun suorittamiseksi kukin rivi voidaan luokitella yhteen seuraavista luokista:

- Luokka E: rivillä ei ole kommenttia eikä häntää (valtaosa käskyistä kuuluu tähän luokkaan).
- Luokka H: rivi päättyy häntään, mutta siinä ei ole kommenttia.
- Luokka K: rivillä on kommentti, mutta ei häntää.
- Luokka KH: rivillä on ensin komentti ja sen perässä häntä.

Vertailtaessa kahden tiedoston vastinrivejä syntyy 16 mahdollista kombinaatiota: kummallakaan rivillä ei ole kommenttia (luokka E&E), ensimmäisellä rivillä ei ole kommenttia kun toisella on häntä ilman kommenttia (E&H) jne. Tilanteen havainnollistamiseksi olen kirjannut mahdolliset vaihtoehdot taulukkoon 2. Esimerkiksi taulukon vasemman alanurkan ruudun merkintä E&KH tarkoittaa, että tiedoston 2 rivillä ei ole kommenttia eikä häntää, mutta tiedoston 1 vastaavalla rivillä on komentti jonka lopussa on häntä.

Kun tarkastellaan tilannetta, jossa tiedostoa 1 epäillään tiedoston 2 kopioksi, voi taulukkoa lukea seuraavasti. Jokainen pystysarake kertoo, mitä ko. luokan riveille riville on mahdollisessa kopioinnissa tapahtunut. Esimerkiksi kolmas pystysarake (K) sisältää vaihtoehdot:

- K&E: rivillä aikaisemmin ollut kommentti on kadonnut,
- K&H: kommentti on muuttunut hännäksi,
- K&K: komentti on säilynyt kommenttina ja
- K&KH: komentti on muuttunut hännälliseksi kommentiksi.

Taulukon 2 ruutuihin on sijoitettu '+'- ja '-'-merkkejä. Niiden tulkinta on seuraava:

- '--': vahvasti kopioinnin puolesta,
- '-': lievästi kopioinnin puolesta,
- 'o': en ota kantaa,
- '+': lievästi kopiointia vastaan ja
- '++': vahvasti kopiointia vastaan.

Perustelut näiden päätelmien tekoon ovat seuraavat.

- Pystysarake E, alkuperäisessä tiedostossa ei ole kommenttia.
  - E&E, 'o': tämän perusteella ei juurikaan voi tehdä johtopäätöksiä, koska suuri osa riveistä on joka tapauksessa ilman kommentteja.
  - E&H, E&K, E&KH, '+': nämä puoltaisivat kaikki lievästi sitä, että kopiointia ei ole tapahtunut. Joka tapauksessa kommentti ja/tai häntä on lisätty riville. Kopioitaessa orjallisesti ei tarvitsisi miettiä kommenttia.
- Pystysarake H, alkuperäisessä tiedostossa on häntä.
  - H&E, '++': vahvasti kopiointia vastaan, koska häntää ei todennäköisesti olisi huomattu poistaa kopioinnissa.
  - H&H, '--': vahvasti kopioinnin puolesta, koska häntä todennäköisesti jäisi vahingossa tiedostoon.

		tiedosto 2			
		E	H	K	KH
tiedosto 1	E: ei kommenttia, ei häntää	<b>E&amp;E</b> o	<b>H&amp;E</b> ++	<b>K&amp;E</b> +	<b>KH&amp;E</b> +
	H: häntä, ei kommenttia	<b>E&amp;H</b> +	<b>H&amp;H</b> --	<b>K&amp;H</b> --	<b>KH&amp;H</b> --
	K: kommentti, ei häntää	<b>E&amp;K</b> +	<b>H&amp;K</b> +	<b>K&amp;K</b> -	<b>KH&amp;K</b> o
	<b>KH</b> : kommentti ja häntä	<b>E&amp;KH</b> +	<b>H&amp;KH</b> -	<b>H&amp;KH</b> -	<b>KH&amp;KH</b> --

**Taulukko 2:** Vastinrivien mahdolliset kombinaatiot vertailtaessa kahta tiedostoa keskenään.

- H&K ‘+’: lievästi kopiointia vastaan (vrt. E&H yllä).
- H&KH ‘-’: lievästi kopioinnin puolesta, koska häntä on jäänyt riville vahingossa kommenttia lisättäessä.
- Pystysarake K, alkuperäisessä tiedostossa kommentti.
  - K&E, ‘+’: lievästi kopiointia vastaan, koska kommentti olisi pitänyt poistaa huolella niin, ettei häntää jää.
  - K&H, ‘--’: vahvasti kopioinnin puolesta, koska häntä jää helposti vahingossa, kun kommentin poistaa.
  - K&K, ‘-’: lievästi kopioinnin puolesta, koska kommentti on samassa kohdassa (toisaalta on otettava huomioon, että komentoitujakin rivejä on aika paljon).
  - K&KH, ‘-’: lievästi kopioinnin puolesta, häntä tullut vahingossa.
- Pystysarake KH, kommentti jonka perässä on häntä.
  - KH&E, ‘+’: lievästi kopiointia vastaan (vrt. K&E yllä).
  - KH&H, ‘--’: vahvasti kopioinnin puolesta (vrt. K&H yllä)
  - KH&K, ‘o’: lievästi kopioinnin puolesta puhuu se, että komentti on samassa kohdassa, kopiointia vastaan taas puhuu hännän häviäminen (tämä tosin saattaa riippua käytettävästä tekstieditorista) — kaiken kaikkiaan vaikea ottaa kantaa.
  - KH&KH, ‘--’: vahvasti kopioinnin puolesta.

Ylläoleva päättely on melko mielivaltainen ja sisältää monia virheiden mahdollisuuksia. Varsinkin “lievien” todisteiden tapauksessa voisi joskus vetää vastakkaisenkin johtopäätöksen, ja voihan esimerkiksi jopa kopiointia vahvasti indikoiva tilanne H&H tulla vertailussa vastaan aivan sattumaltakin, koska häntiä joka tapauksessa on aika paljon. Osittain johtopäätöksiin voivat vaikuttaa jopa mahdollisessa kopiointityössä käytetyn tekstieditorin ominaisuudet. Päättelyn epävarmuutta voi jossain määrin eliminoida tekemällä vertailukokeita tiedostoilla, joista tiedetään varmuudella, että ne eivät ole kopioita toisistaan. Päättelyyn voisi vielä ottaa mukaan sisennykset ja näkymättömien merkkien määrät, mutta ne on tässä jätetty tarkastelun ulkopuolelle monestakin syystä (ajanpuute, monimutkaisuus, sisennykset joka tapauksessa erilaisia, sisennyskäytäntö ohjaa tietynlaisiin sisennyksiin).

0:I ; if 0/1 tietue	2:I
1:X	3:X
2:U ; kanta/p„iv	4:U ;KANTA/P_IVITYS
3:I	5:I
4:I; if 2 tietue	6:I
5:O	7:O
6:X	8:X
7:U ; 3 ens.	9:U ;3 EKAA
8:I	10:I
9:I ; if 3 tietue	11:I ;3-TIETUE
10:O	12:O
11:X	13:X
12:U ; kortin alku	14:U ;ALKU
13:I ; j„„nn”s+seur. tietue	15:I
14:I ; if 4 tietue	16:I ;4-TIETUE
15:O	17:O
16:X	18:X
17:*Q ; lis„ys/poisto	19:*Q ;LIS_YS/POISTO
18:G	20:G
19:A ; alku	21:A ;ALKU
20:X	22:X
21:U ; loppu	23:U ;LOPPU = KOKO NUMERO 16
22:G	24:G

**Kuva 13:** Kommenttien ja häntien liittäminen muistipaikkoihin (vastinosat muistipaikoista s393 (vasemmalla) ja v820 luurankomuodossa).

Vertailua tehtäessä ongelmaksi jää vielä, miten keskenään vertailtavat rivit tiedostoista etsitään. Tässä voi käyttää apuna kohdassa 2. esittämäni muistipaikkojen vertailumenetelmää, jolla tiedostoista voi yrittää etsiä toisiaan vastaavia käskysarjoja. Tiedostot muutetaan ensin ko. kohdassa kuvattuun luurankomuotoon ja kommentit liitetään aina siihen käskyyn, joka on ko. rivillä viimeisenä. Esimerkiksi kuvan 1 muistipaikkojen yhtenevän alkuosan kommenttien katsotaan sijoittuvan luurankomuotoiseen ohjelmaan kuvan 13 esittämällä tavalla. Rivit, joilla ei ole käskyä, jäävät vertailun ulkopuolelle.

Vertailu tapahtuu nyt seuraavasti.

- Ensin etsitään muistipaikoista vastinkohdat kohdassa 2. esittämälläni tavalla (D=4).
- Kunkin kommentin ja hännän katsotaan liittyvän siihen käskyyn, joka esiintyy ko. rivillä viimeisenä.
- Suoritetaan luurankomuodossa olevien muistipaikkojen käskyrivien luokittelu neljään luokkaan yllä kuvatulla tavalla.
- Vertaillaan vastinkäskypareja toisiinsa ja tilastoidaan kukin pari taulukon 2 mukaisesti johonkin 16 mahdollisesta luokasta.

Vertailussa ovat siis mukana vain ne käskyt, joille vertailtavissa tiedostoissa löytyy vastinosat (kun D=4). Vertailun ulkopuolelle jäävät käskyt, joille vastinosaa ei löydy sekä rivit, joilla on pelkkä kommentti eikä ainoatakaan käskyä. Karkeasti arvioiden vertailuun saa tällä menetelmällä mukaan noin puolet ohjelman riveistä. Vertailussa ei myöskään kiinnitetä mitään huomiota häntien/kommenttien pituuteen, sisennyksiin eikä asiasisältöön.

Suoritin ensimmäisenä kontrollikokeen, jossa jaoin tiedoston VFPARTSO kahteen yhtäsuureen osaan ja vertasin niitä keskenään. Tulokset ovat nähtävissä taulukossa 3. Kuhunkin taulukon ruutuun on merkitty tapausten lukumäärä ja niiden prosenttiosuus sarekesummasta. Taulukon voi tulkita siten, että pystysarakkeet kuvaavat, miten vastinrivit muuttuvat siirryttäessä tiedostosta OSA2 tiedostoon OSA1.

		VFPARTSO, Osa2				
		E	H	K	KH	
VFPARTSO, Osa1	E: ei kommenttia, ei häntää	620 (80%) 0	41 (48%) ++	69 (23%) +	2 (22%) +	<b>732 (63%)</b>
	H: häntä, ei kommenttia	50 (6%) +	27 (31%) --	16 (5%) --	1 (11%) --	<b>94 (8%)</b>
	K: kommentti, ei häntää	99 (13%) +	14 (16%) +	208 (69%) -	2 (22%) 0	<b>323 (28%)</b>
	KH: kommentti ja häntä	7 (1%) +	4 (5%) -	7 (2%) -	4 (44%) --	<b>22 (2%)</b>
		<b>776 (66%)</b>	<b>86 (7%)</b>	<b>300 (26%)</b>	<b>9 (1%)</b>	<b>1171 (100%)</b>

**Taulukko 3:** Tiedoston VFPARTSO loppuosa verrattuna sen alkuosaan (prosenttiluvut eivät pyöristysvirheistä johtuen summaudu aina sataan).

Esimerkiksi tiedoston OSA1 300 komenttirivistä (pystysarake K) 69 (23%) muuttuu luokan E riveiksi, 16 (5%) muuttuu hänniksi, 208 (69%) pysyy tavallisina kommentteina ja 7 (2%) muuttuu hännällisiksi kommentteiksi.

Taulukosta voi havaita mm. sen, että kommentteja tuntuu olevan yllättävänkin systemaattisesti samoissa kohdissa, vaikka kyse on nyt varmasti kahdesta eri ohjelmasta: Tiedoston OSA2 riveistä 300 sisältää kommentin. Jos häntää pitää aikeena kirjoittaa kommentti, säilyy 77% kommenttiriveistä joko kommenttina tai aikeena kirjoittaa kommentti.

Taulukkoa 3 voi pitää vertailukohtana verrattaessa tiedostoja keskenään. Vertailussa täytyy tosin pitää mielessä, että tässä vertailtavat ohjelmat tekevät eri asioita, kun taas myöhemmissä vertailuissa mahdollisesti saman ohjelmoijan kirjoittama ohjelma tekee samoja asioita. Lisäksi on pidettävä mielessä, että ohjelmien täysin erilaiset osat, tyypillisesti noin puolet, jää vertailun ulkopuolelle. **Nyt siis vertaillaan ohjelmia vain niiltä osin, joilta ne muistuttavat toisiaan eniten.** Absoluuttisia lukumääriä tarkasteltaessa on muistettava ottaa huomioon tiedostojen koko — pienemmät tiedostot johtavat pienempiin lukumääriin.

Vertailin edellä kuvatulla tavalla tiedostoja SYSTEKSO ja VFPARTSO. Tulos on taulukossa 4. Taulukossa huomio kiintyy lähinnä K-sarakkeen K&E-ruudun poikkeamiseen referenssimatriisin (taulukko 3) käyttäytymisestä. Näyttäisi, että epäilyttävän suuri osa kommentteista muuttuu hänniksi (163 kpl, 18%, kun prosenttiluku referenssimatriisissa on vain 5%). Toisaalta pystysaraketta H tarkastelemalla voi todeta, että myös vahvasti kopiointia vastaan puhuvia rivejä löytyy runsaasti: Systekin 41 hännästä 15 säilyy häntänä (vahvasti kopioinnin puolesta), 16 häntää katoaa kokonaan (vahvasti kopiointia vastaan) ja loput 10 muuttuvat joko kommentteiksi tai hännällisiksi kommentteiksi.

Tein vastaavan vertailun tiedostojen TL ja SYSTEKSO välillä. Tiedosto VFPARTSO sisältää TL:n muistipaikat lähes sellaisinaan, ja samankaltaisuus sen osalta osoittautui jo aikaisemmissakin tarkasteluissa suuremmaksi kuin tiedoston VFPARTSO osalta. Tulos on taulukossa 5. Huomiota kiinnittää, että kaikki vertailun kohteeksi joutuneet Systekin hännät ovat mukana myös TL:ssä.

		SYSTEKSO				
		E	H	K	KH	
<b>VFPARTSO</b>	<b>E:</b> ei kommenttia, ei häntää	1628 (81%) o	16 (39%) ++	139 (16%) +	2 (7%) +	1785 (60%)
	<b>H:</b> häntä, ei kommenttia	74 (4%) +	15 (37%) --	163 (18%) --	8 (28%) --	260 (9%)
	<b>K:</b> kommentti, ei häntää	298 (15%) +	9 (22%) +	535 (60%) -	11 (38%) o	853 (29%)
	<b>KH:</b> kommentti ja häntä	16 (1%) +	1 (2%) -	60 (7%) -	8 (28%) --	85 (3%)
		2016 (68%)	41 (1%)	887 (30%)	29 (1%)	2983 (100%)

**Taulukko 4:** VFPARTSO vs. SYSTEKSO.

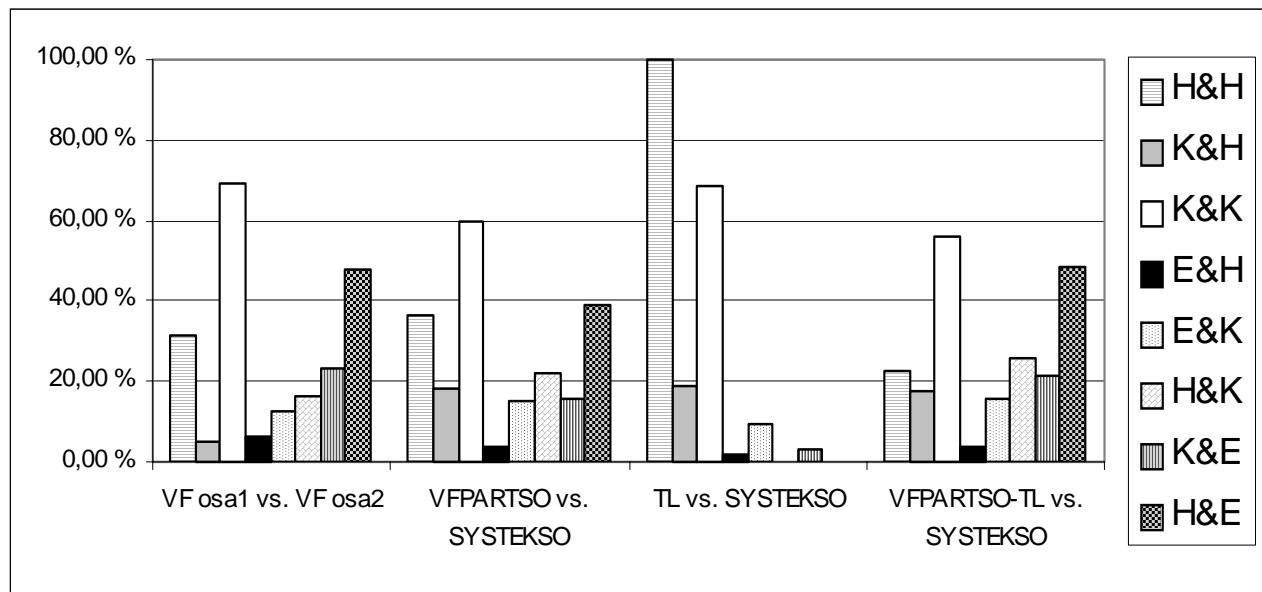
		SYSTEKSO				
		E	H	K	KH	
<b>TL</b>	<b>E:</b> ei kommenttia, ei häntää	285 (88%) o	0 (0%) ++	9 (3%) +	0 (0%) +	294 (46%)
	<b>H:</b> häntä, ei kommenttia	6 (2%) +	8 (100%) --	58 (19%) --	4 (58%) --	76 (11%)
	<b>K:</b> kommentti, ei häntää	31 (10%) +	0 (0%) +	210 (69%) -	1 (12%) o	242 (38%)
	<b>KH:</b> kommentti ja häntä	1 (0%) +	0 (0%) -	28 (9%) -	2 (29%) --	31 (5%)
		323 (50%)	8 (1%)	305 (47%)	7 (1%)	643 (100%)

**Taulukko 5:** TL vs. SYSTEKSO.

Viimeisessä kokeessa poistin vielä tiedoston VFPARTSO aineistosta tietoliikenneosuuden muistipaikat. Tulos on taulukossa 6. Nyt taulukon alkioden jakauma muistuttaa jo melkoisesti referenssitaulukkoa (taulukko 3).

		SYSTEKSO				
		E	H	K	KH	
VF PAR TSO TL pois- tettu	E: ei kommenttia, ei häntää	1339 (79%) o	15 (48%) ++	125 (21%) +	2 (10%) +	1481 (64%)
	H: häntä, ei kommenttia	68 (4%) +	7 (23%) --	105 (18%) --	4 (19%) --	184 (8%)
	K: kommentti, ei häntää	262 (16%) +	8 (26%) +	327 (56%) -	9 (43%) o	606 (22%)
	KH: kommentti ja häntä	16 (1%) +	1 (3%) -	30 (5%) -	6 (29%) --	53 (2%)
		1685 (73%)	31 (1%)	587 (25%)	21 (1%)	643 (100%)

**Taulukko 6:** VF PARTSO verrattuna tiedostoon SYSTEKSO kun TL-muistipaikat on poistettu.



**Kuva 14:** Yhteenvedo häntä- ja kommenttiverailusta.

Yleiskuvan saaminen taulukoiden lukuarvoja vertailemalla on vaikeaa. Tästä syystä piirsin pylväsdiagrammit taulukoista. Jätin diagrammeista hännälliset kommentit pois, koska niiden määrä taulukoissa on pieni. Jo yhden uuden tapauksen lisääminen aineistoon saattaa jossain tapauksessa lähes kaksinkertaistaa prosenttiluvun. Tulos kuvassa 14. Kuvassa on esitetty “häntä- ja kommenttiprofili” ensin referenssitaulukosta (vasemmalla) ja sitten muista tekemistäni vertailuista.

Tarkasteltaessa tulosta, kannattaa pitää mielessä, että nyt tarkastelussa ovat mukana vain ne osat ohjelmista, joista vertailuohjelma löysi samankaltaisia kohtia. Vertailussa TL vs. SYSTEKSO mukaan

---

tuli n. 69% tiedoston TL käskyistä. Vertailussa VFPARTSO vs. SYSTEKSO mukaan saatiin mukaan n. 51% tiedoston VFPARTSO käskyistä. Kun VFPARTSO-tiedostosta poistetaan tietoliikenneosuus, saadaan mukaan n. 48% VFPARTSO:n käskyistä. Referenssivertailussa tiedoston VFPARTSO alku- ja loppuosien välillä vertailuun saatiin mukaan n. 37% käskyistä. Pienempi prosenttiluku on tässä ymmärrettävä, koska vertailtava tiedosto on noin puolet pienempi kuin muissa vertailuissa. Tällöin todennäköisyys, että samanlaista koodia löytyy sattumalta on myös pienempi. (Tämän osoittamiseksi vertasin VFPARTSO-tiedoston alkupuoliskoa myös sen viimeiseen neljännekseen. Nyt vertailuun saa mukaan enää 31% käskyriveistä.)

Vertailemalla kuvan 14 kohtaa *VF osa1* vs. *VF osa2* muista taulukoista tehtyihin profiileihin, vahvistuu jo VTT:n ensimmäisessä lausunnossa esitetty käsitys, jonka mukaan samankaltaisuus ohjelmissa keskittyy pääasiassa tietoliikennettä käsitteleviin muistipaikkoihin.

## 8. VTT:n uusi raportti

Kommentoin tässä VTT:n uutta lausuntoa (Ari Heikkilä, Teemu Tommila 9.4.1997) vain sen tiedostoja SYSTEKSO, VFPARTSO ja TL koskevilta osilta.

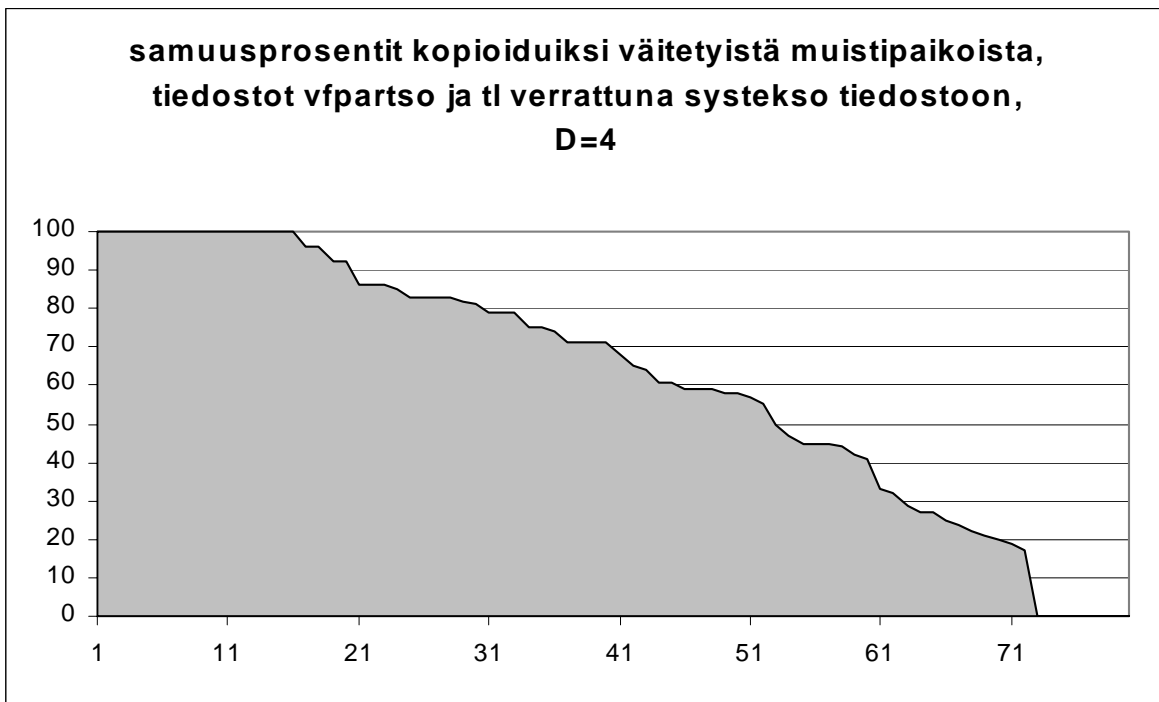
Haltuuni saama materiaali koostuu tutkintapyyntöstä, lausunnosta ja sen liitteistä. Liitteistä keskeiset ovat Liite 4 (tiedoston VFPARTSO muistipaikkoja) ja Liite 5 (tiedoston TL muistipaikkoja). Muihin liitteisiin en ehtinyt paneutua.

Lausunnon argumentointi perustuu pääasiassa em. liitteissä rinnakkain asetettujen muistipaikkojen vertailuun. Kopioituksi väitettyjä muistipaikkoja on yhteensä laskujeni mukaan 86 (yhteensä tiedostossa VFPARTSO on laskutavasta riippuen joko 333 tai 295 muistipaikkaa, taulukko 1). Näistä noin 40 muistipaikkaa olen kommentoinut jo aikaisemmassa lausunnossani.

Muistipaikat on pääsääntöisesti järjestetty siten, että tapaukset, joissa kopiointi näyttää selvältä on sijoitettu ensimmäisiksi, jolloin lausunnon pintapuolisessa tarkastelussa saattaa helposti syntyä vaikutelma, että koko kopioituksi väitetty materiaali on samanlaista. Tarkempi tarkastelu osoittaa kuitenkin, että luettelossa on runsaasti muistipaikkoja, joiden samankaltaisuus ei ole mitenkään itsestäänselvää. Lisäksi samankaltaisuus on monin paikoin selitettävissä pankkistandardien vaatimuksilla. Esimerkiksi koottaessa vakiomuotoista tietoliikennesanomaa, siihen on koottava tietyt merkit tietyssä järjestyksessä. TCL-kieli ohjaa jopa kaksi toisistaan riippumatonta ohjelmoijaa tekemään tämän samalla tavalla.

Kävin liitteiden muistipaikkavertailut pintapuolisesti läpi. Valitettavasti VTT:n lausunto ei sisällä VTT:n tutkijoiden perusteluita muistipaikkakohtaisesti. Joissain tapauksissa on vaikea keksiä, miksi ko. muistipaikat on vertailussa asetettu rinnakkain. Tein liitteistä seuraavia havaintoja

- Liitteissä on muistipaikkoja vertailtu yhteensä 84 sivulla. Yleensä siis yksi muistipaikkapari/sivu.
- Neljältä sivulta Systekin muistipaikkaa ei löydy tiedostosta SYSTEKSO (VTTI5s31, s33, s39 ja s43).
- VTT:n raportissa mainitaan (VTTs4), että muistipaikkapari s621&v931 esiintyy kahteen kertaan. Tämä pitää paikkansa (VTTI5s36 ja s50). VTT:n raportissa ei kuitenkaan mainita, että myös pari s829&v542 esiintyy kahdesti (VTTI5s29 ja s47). Lisäksi Systekin muistipaikat s364 ja s935 esiintyvät vertailussa kahdesti. Sivulla VTTI5s22 mp:n v599 väitetään olevan kopio muistipaikasta s364 ja sivulla VTTI5s40 sen väitetään olevan kopio muistipaikasta s282.



**Kuva 15:** Samuus kopioiduksi väitettyjen muistipaikkojen osalta.

Vastaavasti sivulla VTTI5s11 muistipaikkaa v597 väitetään muistipaikan s935 kopioksi ja sivulla VTTI5s37 sen väitetään olevan kopio muistipaikasta s928.

- Vertailussa on mukana muutamia todella lyhyitä muistipaikkoja. Esimerkiksi muistipaikoissa s869 ja s871 on vain kaksi käskyä (VTTI5s17 ja s25).
- Näiden lisäksi vertailussa on mukana joukko muistipaikkapareja, joista ei mitenkään tahdo löytää samankaltaisuutta. Esimerkeinä muistipaikkaparit
  - s364&v282, VTTI5s40
  - s621&v931, VTTI5s36
  - s319&v559, VTTI5s42
  - s935&v928, VTTI5s37
  - s265&v551, VTTI5s48
  - s295&v864, VTTI5s46
  - s872&v388, VTTI5s54
  - s923&t498, VTTI4s27
  - s415&v239, VTTI5s52
  - s893&v240, VTTI5s53

Jotkut muistipaikkaparit näyttävät toteuttavan täysin eri asioita (esimerkiksi s319 ja v559).

Yhteenvetona voi siis todeta, että samankaltaisuutta on korkeintaan noin 60 muistipaikassa. Osa tästä on selitettävissä standardien vaatimuksilla, ja sillä, että samat ohjelmoijat ovat toteuttaneet samoja asioita tekeviä ohjelmapätkiä.

Kun puuttuvat muistipaikat ja useaan kertaan esiintyvät muistipaikat poistetaan, vertailtavaksi jää siis lopulta 76 liitesivua. Laskin näille muistipaikoille läheisyysmitat. Lopputulos on kuvassa 15. Yli 80% samuus n. 30 muistipaikassa, sekä häntävertailu näyttävät osoittavan, että osassa VF Partnerin koodissa on käytetty lähtökohtana Systekin koodia ja muutamissa kohdissa on todennäköisesti tapahtunut tiedostokopiointi. Tämä vastaa jo Urho Pulkisen VTT:n ensimmäisessä lausunnossa esittämää arviota.

VTT:n lausunnon johtopäätös on kiteytetty sen loppuun “Edellä todetun ja erityisesti kohdan 6 perusteella voidaan pitää erittäin todennäköisenä, että VF Partner Oy:n lähdekieliseen ohjelmiston lähtökohtana on käytetty Systek Oy:ltä kopioitua lähdekielistä ohjelmistoa.” (VTTs9). Tässä huomio kiintyy erityisesti sanoihin “lähtökohtana” ja “kopioitua”. Jos tällä tarkoitetaan sitä, että lausunnossa on pystytty osoittamaan, että VF Partnerin ohjelma on tehty kopioimalla Systekin ohjelmätiedosto, ja sitten muokkaamalla sen muistipaikkoja on hämäysmielessä yritetty häivyttää jäljet, olen eri mieltä. Aikaisemman VTT:n lausunnon (Pulkkinen) lisäksi on mielestäni vain osoitettu, että todennäköisesti joidenkin muistipaikkojen osalta on tapahtunut jossain vaiheessa suora tiedostosta tiedostoon kopiointi (häntävertailu).

Miten on sitten mahdollista, että kaksi (tai oikeastaan kolme) tutkimusta päättyy näin erilaiseen lopputulokseen? Uskoisin, että VTT:n uuden lausunnon johtopäätöksen syynä on se, etteivät lausunnon antajat tunne VF Partnerin ohjelmiston taustaa. He olettavat ohjelman tekijöiden olevan eri henkilöitä. Lausunnossa todetaan mm. sivulla 4 hieman puolen välin alapuolella: “...on erittäin epätodennäköistä, että kaksi riippumatonta ohjelmoijaa päätyy lähellekään samanlaiseen ratkaisuun edes sovelluksen osatoimintojen toteutuksessa.”. Kohdassa *Yhteen veto, kohta 2*, todetaan sama asia johtopäätösten perusteluna uudestaan. Jos kyseessä todella olisivat “riippumattomat ohjelmoijat”, olisin samaa mieltä edellä esitetystä näkemyksistä. Saamani tiedon mukaan VF Partnerin ohjelman tehneet ohjelmoijat (Juselius ja Salojää) työskentelivät Systekin ohjelmakoodin parissa yli viisi vuotta käytännössä päätoimisesti ollessaan Systekin palveluksessa. Systekin ohjelmakoodi on osittain heidän tekemäänsä ja he ovat joka tapauksessa ylläpitotyössä joutuneet perehtymään sen kaikkiin yksityiskohtiin. Kun tämä otetaan huomioon, voi VTT:n raportin perusteella päätyä myös toisenlaiseen johtopäätökseen. Todellista ristiriitaa lausuntojemme välillä ei siis juurikaan ole, kyse on eri lähtökohtiin perustuvista päätelmistä.

## 9. Ohjelmien kääntämisestä

VeriFonen maksupäätteen yhteydessä käytetty terminologia poikkeaa alalla yleisesti käytetyistä termeistä. Esimerkiksi termi **muistipaikka** tarkoittaa yleensä vain yhden tai korkeintaan pari käskyä sisältävää muistin osaa, eikä kokonaista usean kymmenen käskyn aliohjelmaa. Erityisen hämäävää on termien **kääntäminen** ja **konekielinen muoto** käyttäminen. Koska tämä liittyy käsillä olevaan oikeustapaukseen mahdollisen yrityssalaisuuden loukkauksen osalta, täsmennän käsitteiden tulkintaa seuraavassa. Esitys on ymmärrettävästi melkoisesti yksinkertaistettu, mutta yritän kuitenkin tuoda oleelliset asiat esille.

Ohjelmat kirjoitetaan nykyisin lähes aina jollain ns. korkean tason ohjelmointikielillä (esimerkiksi C, C++, Cobol, Pascal). Ohjelmointikielten kääntämisen yhteydessä näitä ohjelmoijien käyttämiä kieliä sanotaan yleisesti **lähdekieliksi** (eli kansanomaisesti “sorsakoodiksi”, source language).

Koska on otaksuttavaa, että tämän raportin lukija ei ole perillä ohjelmoinnin erikoisuuksista, tarkastellaan ohjelmointia seuraavaksi pienen äärimmäisen yksinkertaisen esimerkin avulla. Oletetaan, että ohjelmassa täytyy laskea loppusumma summaamalla veroton hinta ja ALV. Ensinnäkin kaikille näille tiedoille on varattava tilaa tietokoneen muistista. Tämän voi tehdä kirjoittamalla sopivaan kohtaan lähdekielistä ohjelmaa rivin:

integer loppusumma, verotonHinta, ALV; (esimerkki 1)

Tässä oletetaan yksinkertaisuuden vuoksi, että kaikki luvut ovat kokonaislukuja. Ohjelman kohdassa, jossa loppusumma lasketaan, ohjelmoija voi nyt kirjoittaa lähdekielisen käskyn:

loppusumma = verotonHinta + ALV; (esimerkki 2)

Tämä ohjelman ns. lähdekielinen muoto kertoo kenelle tahansa ko. ohjelmointikieltä taitavalle, että loppusumma saadaan kasvattamalla verotonta hintaa arvonlisäverolla. Tietokoneohjelmat koostuvat

suuresta joukosta tämäntapaisia käskyrivejä. Esimerkiksi toisen tätä raporttia varten laatimani ohjelman koko on noin 500 käskyriiviä. Taulukkolaskentaohjelman koko saattaa olla joitakin miljoonia rivejä. Puhelinkeskuksesta rivien määrä voi olla vaikkapa 15 miljoonaa riviä.

Tietokoneen suoritin (prosessori, CPU) puolestaan osaa suorittaa vain hyvin yksinkertaisia binäärimuodossa olevia käskyjä, ns. **konekieltä**. Konekielellä käskyt esitetään jonoina nollia ja ykkösiä. Tyypillisiä käskyjä ovat esimerkiksi kahden luvun yhteenlasku ja tiedon siirtäminen muistissa muistiosoitteesta toiseen. Muistiosoitteet ovat koneen muistipaikkojen numeroita, siis kokonaislukuja. Lähdekielinen ohjelma muutetaan suoritusta varten konekieliseen muotoon kääntäjäohjelman avulla. Kääntäjäohjelma lukee lähdekielisen ohjelman, varaa tilaa muistista ohjelmoijan määrittelemille muuttujille ja muuttaa ohjelmointikielillä kirjoitetut lähdekieliset käskyt konekieliseen muotoon. Tämä käänös on yksisuuntainen: konekielisessä muodossa olevaa ohjelmaa ei pysty kääntämään takaisin lähdekieliseen muotoon. Esimerkiksi edellä esitetty loppusumman laskeva käsky voisi konekielisessä muodossa näyttää vaikkapa seuraavalta (Pentium-prosessorin konekieltä):

```
1000101101010101111101000000011010101011110000100010010101010110001011 (esimerkki 3)
```

Asiantuntija pystyy halutessaan selvittämään, että bittijono sisältää kolme konekäskyä: MOV (tiedon siirto rekisteriin), ADD (yhteenlasku) ja MOV (tiedonsiirto rekisteristä muistiin). Tieto siitä, minkä nimisistä muuttujista on kysymys on hävinnyt. Kääntäjä on muuttanut ne muistiosoitteiksi (kokonaislukuja). Aivan samanlainen bittijono voisi syntyä myös aivan toisenlaisesta lähdekoodin pätkästä, ja toisaalta kääntäjä voi toisessa tilanteessa kääntää ko. lähdekoodin pätkän aivan toiseksi konekäskyiksi. Vähänkin monimutkaisemmassa esimerkissä ohjelmalogiikan ymmärtäminen konekielisen esitysmuodon perusteella käy siis käytännössä mahdottomaksi.

Ohjelman ostaja saa tavallisesti ohjelman vain konekielisessä muodossa, eli käytännössä siis vain tietokoneen ymmärtämän, esimerkiksi 10 miljoonan bitin pituisen jonon nollia ja ykkösiä. Näin myyjä voi olla varma, ettei ohjelman *osia* kopioida muualle, eikä ostaja myöskään voi muokata saamastaan ohjelmasta uutta kilpailevaa versiota. Koko ohjelmanhan ostaja voi tietysti laittomasti kopioida, mikä onkin tämän päivän suurin ongelma laittomassa ohjelmien kopioinnissa. Muutosten tekeminen kopioituun ohjelmaan ei kuitenkaan ole käytännössä mahdollista ilman lähdekielistä koodia.

TCL-kieli poikkeaa edellä esitetystä kahdella merkittävällä tavalla: ensinnäkään se ei ole korkean tason kieli ja toiseksi sitä ei käännetä konekieliseen muotoon (sanan edellä esitettyssä merkityksessä).

TCL ei ole korkean tason kieli, koska siinä ohjelmoijan on itse huolehdittava muistin varauksesta eri tarkoituksiin, kielestä puuttuvat alkeellisimmatkin tavat ilmaista asioita havainnollisesti jne. Esimerkiksi tehdäkseen edellä kuvatun loppusumman laskemisen, ohjelmoijan täytyy ensin mielessään päättää, että hän käyttää loppusummaan esimerkiksi muuttujaa #0, verottomaan hintaan muuttujaan #5 ja arvonlisäveroon muuttujaa #6. Tämän jälkeen hän voi kirjoittaa käskysarjan

```
*n#0.#5 (esimerkki 4)  
*o#0.#6
```

Jotta ohjelmasta saisi myöhemmin helpommin selvää, hän saattaa lisätä käskysarjaan kommentin, jolloin ohjelmanpätkä on ymmärrettävämpi:

```
*n#0.#5 ; summataan #0:aan loppusumma. Veroton hinta on (esimerkki 5)  
*o#0.#6 ; #5:ssa ja ALV#6:ssa
```

(Esimerkki on sikäli huono, että muuttujien kapasiteetti on TCL:ssä vain 16 bittiä, joten tässä se ei välttämättä riittäisi.)

TCL-kieltä ei varsinaisesti käännetä, vaan maksupäätteessä toimii ns. tulkkiohjelma, joka suorittaa TCL-kielen käskyt yksi kerrallaan. Niinpä lähdekielinen ohjelma on lähes sellaisenaan talletettuna maksupäätteen muistiin. Kääntämisellä tarkoitetaan yksinkertaisesti tarpeettoman tiedon poistamista lähdekielisestä ohjelmasta: siitä poistetaan turhat rivinvaihdot ja kommentit. Maksupäätteen muistiin talletettua ohjelmaa pystyy tutkimaan maksupäätteen näppäimistön avulla ja sen pystyy myös lataamaan

---

maksupäätteestä PC-tietokoneeseen tutkittavaksi. Edellä oleva esimerkin 4 lähdekielinen kahden käskyn sekvenssi olisi siis luettavissa maksupäätteeseen talletetusta TCL-kielen “konekielisestä” muodosta — toisin kuin käännettävässä ohjelmointikielissä. Käännösprosessissa hävinneet rivinvaihdot olisi helppo lisätä siihen tekstieditorilla tai yksinkertaisella konversio-ohjelmalla. Esimerkissä 5 lisätyt kommentit sensijaan häviävät “käännöksessä”, eikä niitä saa esille maksupäätteen muistiin talletetusta ohjelmasta.

Hallussani olevat maksupäätteohjelmat sisältävät paikoitellen paljonkin kommentteja. Muutamien kymmenien muistipaikkojen koodin läpikälyksen antamalla kokemuksella voin kuitenkin todeta, että kaikkien hallussani olevien ohjelmien kommentointi on erittäin puutteellista ja sekavaa, eikä kommenttien pois jättäminen paljoakaan lisää ohjelman ymmärtämisen aiheuttamaa vaivaa — varsinkin, jos koodin lukija tuntee sovelluksen ennestään ja tietää mitä on etsimässä.

## 10. Loppuyhteenveto

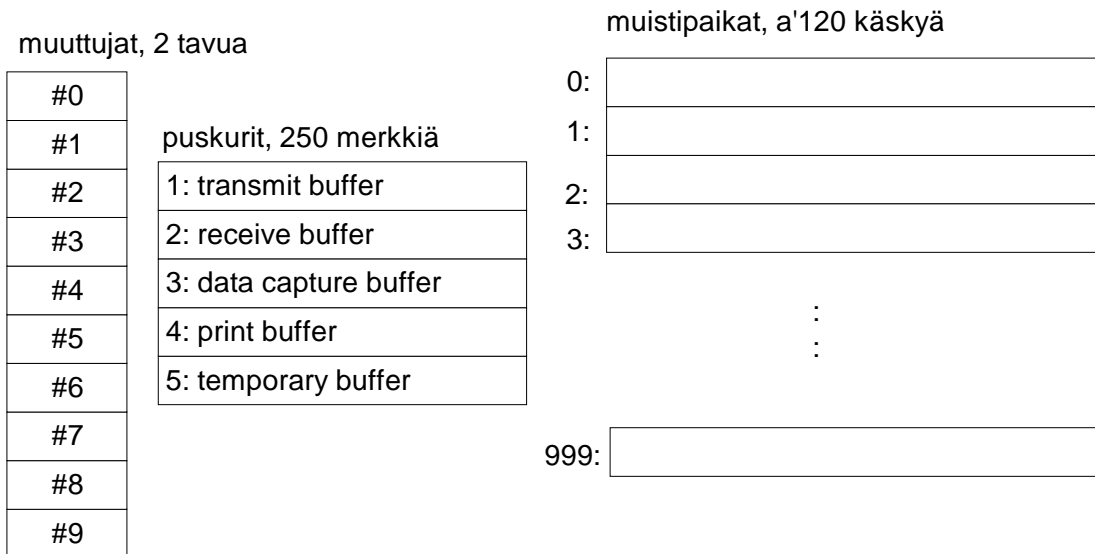
Aikaisemmassa lausunnossani 8.12.1996 päädyin johtopäätökseen: VF Partnerin ohjelma on sen omaa tuotantoa eikä kopio Systekin ohjelmasta. Lausunto perustui oletukseen, jonka mukaan VTT:n ensimmäinen Unto Pulkkinen tekemä lausunto oli paikkansa pitävä. Tästä syystä perehdyin vain ohjelman ko. raportissa viitattuihin osiin. Tässä raportissa kuvaamani analyysiohjelmistoihin perustuva kartoitus on huomattavasti perusteellisempi ja kattaa koko ohjelmiston. Yrityksistäni huolimatta en onnistunut löytämään ohjelmistosta merkittäviä uusia todisteita kopioinnista. Ainoa merkittävä lisä aikaisempiin havaintoihin on muutamien muistipaikkojen osoittautuminen identtisiksi häntävertailun avulla. Tämä ei mielestäni anna aihetta muuttaa aikaisempaa johtopäätöstäni. Ohjelmissa havaitut samankaltaisuudet voidaan mielestäni selittää suurelta osin seuraavilla seikoilla.

- Ohjelman ovat laatineet samat ohjelmoijat.
- Ohjelmat toteuttavat saman toiminnallisuuden.
- Käyttöliittymän on oltava sama.
- Tietoliikennekommunikaation pankin suuntaan on oltava sama.
- Kommunikaation maksupäätteen oheislaitteiden kanssa on oltava sama.
- TCL-kieli tarjoaa ohjelmoijalle vain vähän variointivaihtoehtoja ohjelman kirjoittamiseen.
- VeriFonen maksupäätteen rakenne pakottaa toteuttamaan tiettyjä asioita tietyillä tavoilla.

Ohjelman osien uudelleenkäyttö, joka tapahtuu useimmiten kopioimalla, on ohjelmoinnissa yleinen käytäntö. Ohjelmoinnissa koodia kopioidaan paikasta toiseen ja tarvittaessa modifioidaan editorilla uuteen käyttötarkoitukseen sopivaksi. Se, että muutamien muistipaikkojen sisältö voidaan osoittaa identtiseksi, ei mielestäni ole osoitus siitä, että koko ohjelma olisi kopio Systekin ohjelmasta. Samuusmittaan perustuvan vertailun perusteella näyttää päinvastoin siltä, että ohjelmat ovat pääasiassa erilaisia. Samankaltaiset ohjelmankohdat eivät myöskään sisällä mitään erityisiä oivalluksia tai hienouksia vaan ovat aivan tavanomaista rutiinikoodia.

Yrityssalaisuuden osalta totean, että mielestäni salatusta ohjelmasta ei voi puhua silloin, kun ohjelma (tosin ilman kommentteja) on lähdekielisessä muodossa luettavissa suoraan maksupäätelaitteelta.

# Liite A: VeriFonen maksupäätteen ohjelmoinnista



**Kuva 1:** Ohjelmointiympäristö.

Kyseessä oleva maksupäätējärjestelmä on nykypäivän mittapuun mukaan melko yksinkertainen järjestelmä (kuva 1). Sen tärkein toiminto on maksutapahtuman kirjaaminen näppäimistöä ja maksukortinlukijaa käyttäen. Tapahtumasta tulostetaan kuittikirjoittimella kuitti ja tapahtuma voidaan varmentaa ottamalla yhteys pankin järjestelmään. Tapahtumat talletetaan tapahtumientallennusmuistiin, josta ne esimerkiksi päivän päätteeksi lähetetään pankin järjestelmään jatkokäsittelyä varten. Käyttäjä saa toiminnoista palautetta pienen näytön kautta. Kellon avulla järjestelmä pystyy kirjaamaan tapahtuma-ajat. Laitteeseen voidaan liittää myös muita oheislaitteita, kuten esimerkiksi näppäimistö, jolla asiakas voi syöttää PIN-tunnuksensa.

Maksupäätteen ohjelmiston perustana on käyttöjärjestelmä, jossa laitteen sovellusohjelmaa suoritetaan. Käyttöjärjestelmän tarjoama ohjelmointiympäristö on ohjelmoijan kannalta tarkasteltuna erittäin primitiivinen (kuva 1). Toisin kuin ohjelmoinnissa yleensä, ohjelmoija ei voi vapaasti valita käyttämiään muuttujia, vaan järjestelmä määrittelee ne valmiiksi: käytettävissä on muuttujat nimeltä #0, #1...#9. Sanomien käsittelyä varten on samassa hengessä määritelty viisi 250-merkin sanomapuskuria. Puskureita voi käyttää muihinkin tarkoituksiin kuin mihin niiden nimet viittaavat. Ohjelmat kirjoitetaan TCL-kielillä ja talletetaan koneen muistipaikkoihin, joita on tuhat kappaletta. Ohjelmoija paloittelee ohjelmansa muistipaikkoihin sopiviin osiin, yhteen muistipaikkaan mahtuu 120 merkkiä.

TCL-kieli on erittäin primitiivistä. Sillä ohjelmointi muistuttaa lähinnä 1950- ja 1960-luvuilla verraten tavallista konekielistä ohjelmointia, jossa ohjelmoijan on itse huolehdittava ohjelman sijoittamisesta muistiin. Toisaalta ympäristö on niin yksinkertainen, että ohjelmoija oppii kaiken tarvittavan muutamassa päivässä. Maksupäättesovelluksesta tarkastelemani ohjelmanosat olivat yksikertaisia ja verraten helposti toteutettavissa näinkin alkeellisilla työkaluilla.

Nykypäivän mittapuun mukaan ohjelmointi TCL-kielellä on yksinkertaista mutta hankalaa. Hankaluus aiheutuu ohjelmointiympäristön primitiivisyydestä: ohjelmoijan aika kuluu ohjelmaa laadittaessa toisarvoisten kysymyksien miettimiseen, kuten muistipaikkojen varaaminen ja muuttujien käyttö eri tarkoituksiin. Lopputuloksena syntyvät ohjelmat ovat vaikeita ylläpitää useastakin syystä. Ainakin seuraavien neljän seikan uskoisin aiheuttavan harmaita hiuksia.

Ensinnäkin, ohjelmat ovat erittäin vaikeaselkoisia, ellei niitä ole kommentoitu kunnolla. Esimerkiksi tavanomaisella ohjelmointikielellä kirjoitettu käsky

```
tarkistussumma = tarkistussumma + merkki
```

kertoo jokaiselle ohjelmoijalle, että tarkoitus on kasvattaa tarkistussummaa sanoman seuraavalla merkillä. TCL-ohjelmassa voisi vastaavassa kohdassa lukea esimerkiksi

```
*O#.#1
```

TCL-kieltä opiskeleva oppii nopeasti muistamaan, että tämä on lyhennysmerkintä käskystä

```
*O#0.0.#1
```

\*O kertoo, että kyseessä on aritmeettinen operaatio, jonka operandeina ovat muuttujat #0 ja #1. Operaatiokoodi 0 tarkoittaa yhteenlaskua, eli siis ideana on laskea yhteen muuttujat #0 ja #1 ja tallettaa lopputulos muuttujaan #0. Vaikka ohjelmoija oppiikin nopeasti muistamaan ulkoa tämäntapaisen koodauksen, tieto siitä, että muuttujaan #0 kerätään tarkistussummaa, ei ilmene mitenkään käskystä (kuten tavanomaisessa ohjelmointikielessä). Eli jos ohjelmoija joutuu myöhemmin muuttamaan ohjelmaa, aikaa kuluu runsaasti vanhan ohjelman logiikan selvittämiseen. Tämä ongelma on ratkaistavissa ohjelman kurinalaisella ja selkeällä kommentoinnilla; ts. ohjelmoija kirjoittaa käskyjen lisäksi ohjelmakommentteina selväkielisen selityksen siitä, mitä käskyn on tarkoitus tehdä; esimerkiksi

```
*O#.#1 ; lisätään seuraava merkki tarkistussummaan
```

(TCL-kielessä puolipisteen jälkeen tuleva osa riviä tulkitaan kommentiksi). Vielä tätäkin tärkeämpää olisi ryhmittää käskyt muutamien käskyjen kokonaisuuksiksi, joille lisätään yhteinen kommentti; esimerkiksi *“sanoman pituus on nyt muuttujassa #3. Seuraavat käskyt käyvät läpi vastaanottopuskurin (puskuri 2) ja summaavat muuttujaan #1 tarkistussumman. Muuttuja #3 nollautuu samalla.”* Kummassakaan ohjelmistoversiossa”. (Systek, VF Partner) ei ole viljelty johdomukaisesti näitä käytäntöjä.

Toiseksi, järjestelmässä ei ole kunnollista aliohjelmankutsumekanismia. Käytännössä suuri ohjelma jaetaan aina aliohjelmiin, jolloin ohjelman hallittavuus ja ymmärrettävyys paranevat. Samalla toteutustyötä voidaan jakaa eri henkilöiden kesken, koska tietyn henkilön tehtäväksi voidaan luontevasti sopia joukko yhteenkuuluvia aliohjelmiä (“moduuli”). Esimerkiksi maksutapahtuman käsittelyssä voisi olla aliohjelmat “lue maksukortti”, “tarkasta kortti”, “syötä tapahtumatiedot näppäimistöltä”, “varmenna pankista”, “kirjoita maksukuitti” ja “talleta tapahtuma”. Maksutapahtuman käsittely koostuu näiden aliohjelmien kutsuista. Monimutkaisemmat aliohjelmat jakautuvat edelleen aliohjelmiin. Esimerkiksi “varmenna pankista” -aliohjelman toteutuksessa kutsutaan tietoliikenteen hoitavia aliohjelmiä. TCL-kielessä on mahdollista tehdä aliohjelma sijoittamalla sen käskyt yhteen tai useampaan muistipaikkaan. Aliohjelmakutsuun, siis aliohjelman suoritukseen siirtymiseen, ei kuitenkaan ole minkäänlaista parametrienvälitysmekanismia. Parametrit on välitettävä tekemällä sopimuksia, jotka liitetään kommentteina ohjelmalistaan; esimerkiksi: *“Aliohjelmaan tultaessa puskuri 1 sisältää sanoman, jonka tarkistussumma lasketaan. Aliohjelman päättyessä tarkistussumma on muuttujassa #0. Muuttujan #3 arvo nollautuu.”*. Hallussani olevista ohjelmista tämäntapaiset kommentit usein puuttuvat.

---

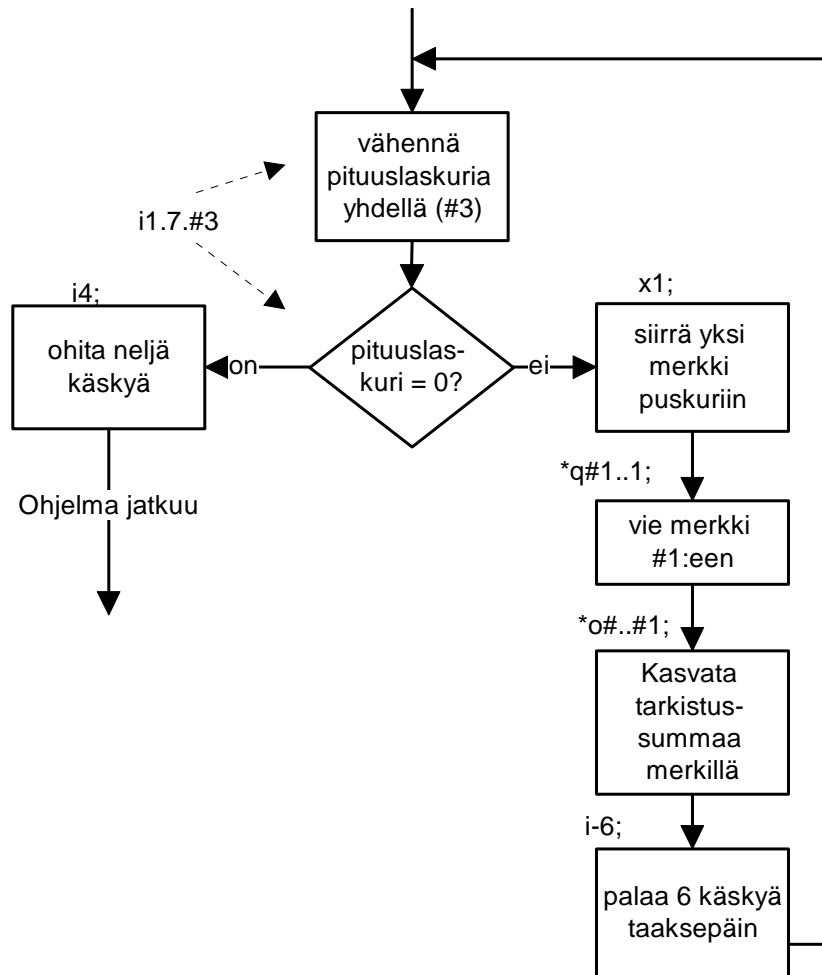
Kolmanneksi, aliohjelmilla ei ole paikallisia muuttujia, vaan ainoat käytettävissä olevat muuttujat ovat #0...#9. Jos nyt esimerkiksi #3 sisältää sanoman pituuden ja kutsutaan aliohjelmaa "laske tarkistussumma", voi aliohjelma nollata muuttujan #3, vaikka aliohjelmaa kutsuva ohjelma voisi tarvita pituutta vielä kutsun jälkeenkin. Yleisesti ottaen aliohjelman kutsuja ei tiedä, mitä muutoksia aliohjelma aiheuttaa muuttujissa ja puskureissa ja toisaalta aliohjelman tekijä ei välttämättä tiedä, mitä muuttujia uskaltaa käyttää. Tätäkin ongelmaa voi lievittää huolellisella suunnittelulla ja kommentoinnilla. Molempien ohjelmaversioiden osalta kommentoinnissa on myös tässä kohden huomattavasti toivomisen varaa.

Neljänneksi, TCL-kieli ei sisällä edes alkeellisia tapoja osoittaa muistipaikkoja ja käskyjä symbolisilla nimillä. On siis vain tiedettävä, että "muistipaikka 875 sisältää tarkistussumman laskevan koodin". Aivan erityisen hankala piirre on se, että ainoa keino osoittaa haaraantumiskohta ohjelmakoodissa on kertoa, kuinka monen käskyn päässä se on. Otetaan seuraava esimerkki

```
i1.7.#3
i4
x1
*q#1..1
*o#..#1
i-6
```

Koodin toiminta selviää kuvasta 2, jossa jokaisen laatikon yhteyteen on liitetty siihen liittyvä käsky. Ohjelmassa lasketaan sanoman tarkistussummaa, ts. lasketaan yhteen sanoman kaikki merkit. Aluksi muuttujassa #3 on sanoman pituus ja lähdepuskurissa itse sanoma. Lähde- ja kohdepuskurit on ennen tätä osaa ohjelmasta asetettu esimerkiksi käskyllä B2.4, joka määrittelee lähdepuskuriksi puskurin 2 ja kohdepuskurin puskuriksi 4. Alussa oleva käsky "i1.7.#3" vähentää aina muuttujaa #3 yhdellä, ja ohittaa seuraavan käskyn (i4) jos tulos on positiivinen. Sanoma siirretään merkki kerrallaan kohdepuskuriin (käsky x1). Kohdepuskurista merkki siirretään laskentaa varten muuttujaan #1 (käsky \*q#1..1), ja tarkistussummaa kerätään yhteenlaskulla muuttujaan #0 (käsky \*o#..#1). Tämän jälkeen palataan taas silmukan alkuun vähentämään muuttujan #3 arvoa yhdellä (käsky i-6). Kun #3 lopulta nollautuu, suoritetaan käsky i4, joka aiheuttaa neljän käskyn ohituksen, eli ohjelman suorituksen jatkumisen käskyä i-6 seuraavasta käskystä.

Viimeisenä kuvaamani piirre on ylläpidon kannalta erittäin hankala. Jos ohjelmakoodiin jostain syystä joudutaan lisäämään yksi käsky esimerkiksi neljännen ja viidennen käskyn väliin, on lisäksi muistettava muuttaa käsky i4 muotoon i5 ja käsky i-6 muotoon i-7. Hallussani olevien ohjelmien koodissa on runsaasti tätä esimerkkiä monimutkaisempia siirtymiskäskykombinaatioita. Pienellä haeskelulla löysin esimerkiksi kohdan, jossa ylitetään 30 käskyä. Jos lisäksi ohjelmakoodin kasvun seurauksena koodi joudutaan jakamaan kahteen muistipaikkaan, voi ylläpito muodostua mahdottomaksi ilman suuria muutoksia, koska haaraantumiskäskyillä ei voida haarautua toisen muistipaikan sisälle. Tätä ongelmaa voi TCL-kielessä helpottaa kommentailla ja sisentämällä ohjelmarivejä ohitettavien käskyjen kohdalla. Hallussani olevissa ohjelmissa on monin paikoin käytetty sisennystä — ei kuitenkaan johdonmukaisesti kaikkialla.



**Kuva 2:** Esimerkiohjelman toiminta

Yhteenvetona TCL-ohjelmoinnista voi todeta, ettei se vaadi erityistä koulutusta. Kuka tahansa jonkin verran ohjelmointia harrastanut (esimerkiksi Basic-kielellä) voi kielen käsikirjan luettuaan ja esimerkkiohjelmiä tutkittuaan ryhtyä laatimaan ohjelmia. Työ edellyttää huolellista suunnittelua ja tarkkuutta ohjelmien kirjoittamisessa sekä edellä kuvaamani ongelmakohtien tunnistamista ja ratkomista jollain tavalla. Pitkä ylläpitovaihe saattaa verraten helposti johtaa tarpeeseen kirjoittaa ohjelmisto kokonaan uudelleen. Koodin vaikeaselkoisuus rajoittaa sen uudelleenkäyttöä uudessa sovelluksessa. Toisin sanoen, jos ohjelmaa joudutaan muuttamaan, saattaa olla järkevämpää kirjoittaa koodi kokonaan uudelleen kuin yrittää muuttaa vanhaa ohjelmaa. Toisaalta kokemus saman ohjelmiston aikaisemmasta versiosta helpottaa ratkaisevasti uuden version tekemistä.

---

# Liite B: Raporttia varten tehdyt ohjelmat ja niiden testaus

Tiedostojen tutkimista varten toteutin kaksi ohjelmaa. Aluksi toteutin ohjelman blanks, jonka avulla yritin muodostaa kuvaa tiedostoissa olevista kommentteista ja hännistä. Yrittäessäni ensin vertailla muistipaikkoja käsin ja samalla tuskastuneena siihen, että uusia kopioituiksi väitettyjä muistipaikkoja näytti putkahtavan esiin asian käsittelyn joka käänteessä, päätin lopulta toteuttaa ohjelman, joka vertailee ohjelmatiedoston muistipaikkoja kaikkiin toisen ohjelmatiedoston muistipaikkoihin. Näin syntyi ohjelma cmp. Molemmat ohjelmat on toteutettu siten, että tulosten jatkokäsittely Excel-taulukkolaskentaohjelmalla on helppoa. Kuvaan seuraavassa lyhyesti molemmat ohjelmat ja myös tulosten varmistamiseksi lopuksi suorittamiani testejä.

## 1. Ohjelma blanks

### 1.1. Toiminta

Ohjelma lukee tiedoston rivi kerrallaan ja tulostaa siitä seuraavat tiedot:

```
Tiedosto: systekso
Rivien kokonaismaara: 6125
Kommentoitujen rivien kokonaismaara: 2826
Muistipaikkojen kokonaismaara: 285
Kommentoimattomien hantien maara: 224
Kommenttien hantien maara: 153
```

Kommenttiriveiksi tulkitaan rivit, joilla on vähintään yksi puolipiste. Näin kommenttiriviksi voidaan vahingossa tulkita myös käskyrivi, jolla on vakiomerkkijonoon sijoitettu puolipiste. Tällaista riviä en ylimalkaisessa manuaalisessa tarkastelussa huomannut rivien joukossa. Tästä kommenttirivien määrään mahdollisesti aiheutuva virhe on vain muutamien rivien luokkaa.

Ohjelma tulostaa lisäksi jokaisen löytämänsä häntärivin varustettuna muistipaikan numerolla.

### 1.2. Testaus

- Tehty tiedostolla SYSTEKSO, paitsi viimeinen vertailu.
- Rivien kokonaismäärä tarkastettu Unixin wc-ohjelmalla.
- Puolipisterivien määrä tarkistettu Unixin komennolla grep “;” SYSTEKSO | wc.
- Muistipaikkojen kokonaismäärä tarkistettu komennolla grep “\$ “ SYSTEKSO | wc. Tulos hieman liian suuri, koska muitakin dollareita sisältäviä rivejä on. Määrä tarkistettu more-komennon avulla.
- Tarkastettu, että ohjelman tulostamien häntärivien määrä on sama kuin sen antamien kommentoimattomien häntien ja kommenttien häntien summa wc syhan
- Häntien määrät tarkistettu pistokokein editorilla seuraavilta riveiltä (s=space, t=tab). Kaikki hännät löytyivät, pituudet OK.
  - alku ;s; 6;t; 1;&128=492029..10915111150200111;; SYP-VISA
  - alku ;s; 0;t; 1;&144=492076..12550111150700091;; STS-VISA-PREMIER
  - alku ;s; 1;t; 0;@\syp\syv0001\PERUS.SYP ; TCLOAD:ssa tarpeellinen
  - alku ;s; 1;t; 0;@\syp\syv0001\LAITE.syp ; TCLOAD:ssa tarpeellinen
  - 371;s; 0;t; 2;
  - 371;s; 1;t; 0;\*L836 ;\* haetaan korttitaulukon koodia vastaava pituus
  - 820;s; 4;t; 0; I5
  - 828;s; 6;t; 2;
  - 336;s; 0;t; 1;
  - 843;s; 0;t; 2;

- 864;s:: 1;t:: 0;I2.2.45 ; etumiinus mukaan
- 864;s:: 0;t:: 1;
- 642;s::37;t:: 0;642\$
- 642;s:: 1;t:: 0; gn5w\*m
- 342;s:: 7;t:: 0;
- 875;s:: 5;t:: 0;i-6
- 930;s:: 7;t:: 0;
- 387;s:: 1;t:: 0;387\$
- 375;s:: 1;t:: 0;a214.1 ; viesti,, ei ole ml.214 ssa,joten hyppy
- 932;s:: 4;t:: 0;932\$
- 882;s:: 5;t:: 0; i-6
- 363;s:: 1;t:: 0; \*o#.5.63 ; AND 63
- 807;s:: 1;t:: 0; GR27R82R5 ; esc+r+SWEDEN
- 807;s:: 1;t:: 0;I2.1.48 ; if ei-kannett.
- 807;s:: 2;t:: 0;U994

- Tarkastettu tiedostosta VFPARTSO muistipaikat v251 (VTTI5s19) ja v825 (VTTI5s17): kaikki hännät löytyneet, ei ylimääräisiä häntiä.
- HUOM: jotkut rivit päättyvät pisteisiin “...” VTT:n raporttia lukiessa tämä voi näyttää hännältä, vaikka ei olekaan.

## 2. Ohjelma cmp

Pääosa esitetyistä tuloksista perustuu tähän ohjelmaan. Ohjelma kehittyi pikkuhiljaa tutkimuksen edistyessä. Jokaisen muutoksen ja lisäyksen jälkeen testasin ohjelmaa muutamilla esimerkkitapauksilla ja tarkastin tulokset käsin. Samalla tuli usein testattua lisää jo aikaisemmin toteutettuja ominaisuuksia. Häntämatriisin laskennan lisäys tapahtui viimeiseksi, joten sitä on testattu kaikkein vähiten. Häntien piteuden laskevat osat kopiaoin blanks-ohjelmasta, joten ne oli jo valmiiksi testattu. Raportin viimeistelyn yhteydessä vietin vielä jonkin verran aikaa tulosten tarkastamiseen käsin. Seuraavassa on kuvattu lyhyesti ohjelman toimintaidea ja lopuksi suoritettut testit.

### 2.3. Toiminta

Ohjelma lukee aluksi tietorakenteisiinsa vertailtavat tiedostot: tdsto1 ja tdsto2. Ne muutetaan luurankomuotoon ja samalla kirjataan kustakin rivistä kommentin pituus (0= ei kommenttia) ja hännän pituus (välilyönnit+tabulointimerkit).

Ohjelmassa on komentorivikäyttöliittymä, jolla voi tehdä seuraavat toiminnot:

- Läheisyysparametrin D asettaminen, oletusarvo on 4
- Läheisyysmitan arvon laskeminen tdsto1 muistipaikan i ja tdsto2 muistipaikan j välillä. Myös häntämatriisi tulostuu.
- Eniten annettua tdsto1 muistipaikkaa muistuttavan tdsto2 muistipaikan etsiminen. Myös häntämatriisi tulostuu.
- Eniten muistuttavan tdsto2 muistipaikan hakeminen kaikille tdsto1 muistipaikoille. Tulostus tapahtuu tiedostoon, joka voidaan siirtää Excelliin. Samalla tulostuu häntämatriisit.
- Kuten yllä, mutta toiseksi eniten muistuttavien paikkojen tulostus, läheisyysmatriisi ei tulostu.
- Tietyn tdsto1 muistipaikan läheisyysmitan laskeminen kaikkiin tdsto2 muistipaikkoihin.
- Muistipaikan luurankomuodon tulostaminen tiedostoon.
- Vertailtavien tiedostojen muistipaikkojen ja käskyjen lukumäärän tulostus.
- Tdsto1:n muistipaikan poistaminen vertailusta.
- Seuraavien kommentojen lukeminen tiedostosta.

#### 2.4. Testi: systekso vs systekso

Testi, vaihe1: verrattu SYSTEKSO itseensä, tulostettu tiedostoon muistipaikkojen parhaat kaverit.

Tulos:

- Lähes kaikille muistipaikoille löytyi parhaaksi kaveriksi muistipaikka itse.
- Kolme muistipaikkaa liian lyhyitä, niille kaveriksi 0 (380, 869, 871)
- Kolmelle muistipaikalle löytyi väärä kaveri (843-369, 880-879, 877-876): varmistettu, että luurankomuodot ovat sattumalta samat.

Vaihe2: poistettu em. ongelmapaikat aineistosta (6 riviä), tarkasteltu summamatriisia. Nollasta poikkeavia arvoja vain diagonaalilla kuten pitääkin

#### 2.5. Käsin tehdyt muistipaikkavertailut

Vaikka olin varmentanut tuloksia useaan otteeseen tutkimuksen edetessä käsin, tein varmuuden vuoksi vielä lopuksi muutamia muistipaikkavertailuja. Tarkastelun kohteena oli sekä läheisyysmitta että häntämatriisi.

- MP v457 vs. s379 läheisyys 79%
- MP v446 vs, s924 läheisyys 59%
- MP v446 vs s379 läheisyys 0%
- MP v56 vs s659 läheisyys 86%
- MP v436 vs s717 läheisyys 100%

Häntävertailun summatietoja tarkastelemalla voi varmistua, että matsiisin kaikkia tapauksia näyttää esiintyvän aineistossa. Asian varmistamiseksi etsin tuloslistaa tarkastelemalla myös harvinaisemmille tapauksille yhden esimerkkitapauksen jokaiselle matriisin alkiolle. Ne on kirjattu seuraavaan matriisiin.

	E	H	K	KH
<b>E:</b> ei kommenttia, ei häntää	lähes jokaisessa vertailussa	v513 & s 849, 1	v551 & s265, 1	v259 & s639, 1
<b>H:</b> häntä, ei kommenttia	t483 & s876, 1	v490 & s875, 1	v35 & s35, 1	v430 & s925, 1
<b>K:</b> kommentti, ei häntää	v430 & s925, 1	v697& s628, 1	v697& s628, 3	v436 & s717, 1

---

**KH:** kommentti  
ja häntä

v573 & s668, 1

v571 & s767, 1

v489 & s343, 1

v489 & s343, 1