

Lausunto: Ohjelmistojen samankaltaisuus, VF Partner vs. Systemk

8.12.1996
Prof. Ilkka Haikala,
TTKK/Ohjelmistotekniikka
PL 553
33101 Tampere

(03)3652911
ijh@cs.tut.fi

1. Lausunnon antajasta

Olen Tampereen teknillisen korkeakoulun professori, viran alana ohjelmistotuotanto (Software Engineering). Minulla on ohjelmistotekniikasta n. 25 vuoden kokemus. Oma käytännön kokeneisuuteni ohjelmistotekniikasta on mielestäni poikkeuksellisen laaja-alainen: olen ollut toteuttamassa laajahkoja ohjelmistoja kymmenkunnalla eri ohjelmointikielellä useisiin eri järjestelmiin. Joukkoon mahtuu sekä tutkimushankkeissa kehitettyjä ohjelmistoja että käytännön asiakasprojekteja. Nykyisessä tehtävässäni yksi päätehtävistäni on diplomitöiden ohjaus. Laitoksemme diplomityöt tehdään lähes poikkeuksetta yritysten tuotekehitysprojekteissa. Tyypillisessä työssä kehitetään ohjelmisto tai sen osa, joten diplomityöt tarjoavat poikkeuksellisen näköalapaikan teollisuudessa tehtävään ohjelmistonkehitystyöhön. Viimeisen noin 10 vuoden aikana olen ohjannut noin 170 diplomityötä. Kuriositeettina voi tuoda esille, että yhdessä diplomityöprojektissa itse asiassa kehitettiin ohjelmisto maksupäätteeseen. Olen kirjoittanut ohjelmistotekniikkaan liittyviä oppikirjoja, joista tuorein on Jukka Märijärven kanssa yhdessä kirjoittamani *Ohjelmistotuotanto* (Suomen ATK-kustannus oy 1995).

2. Lausunnon pohjana olevat taustatiedot

Lausuntoni pohjana olevan materiaalin olen saanut Asianajotoimisto Pello&Tiivolalta. Lisäksi olen materiaalin luovutuksen yhteydessä keskustellut VF Partnerin edustajien (Salojää & Juselius) kanssa arvioinnin kohteena olevan järjestelmän teknisistä ominaisuuksista ja arvioitavina olevien ohjelmistojen rakenteesta. Ohjelmien lähdekieliset versiot toimitettiin minulle myöhemmin levykkeellä. Asian yhteydessä esille tulleet henkilöt ovat kaikki minulle ennestään tuntemattomia. Samaten tarkasteltavana oleva ohjelmisto oli minulle ennestään tuntematon. Hallussani oleva materiaali on seuraava.

- 1) TRANZ 420 Programmers Manual: Arvioinnin kohteena olevan maksupäätejärjestelmän ohjelmointimanuaali. Manuaali sisältää kaiken

tarvittavan tiedon maksupäätteen ohjelmien laatimiseksi (ja lukemiseksi).

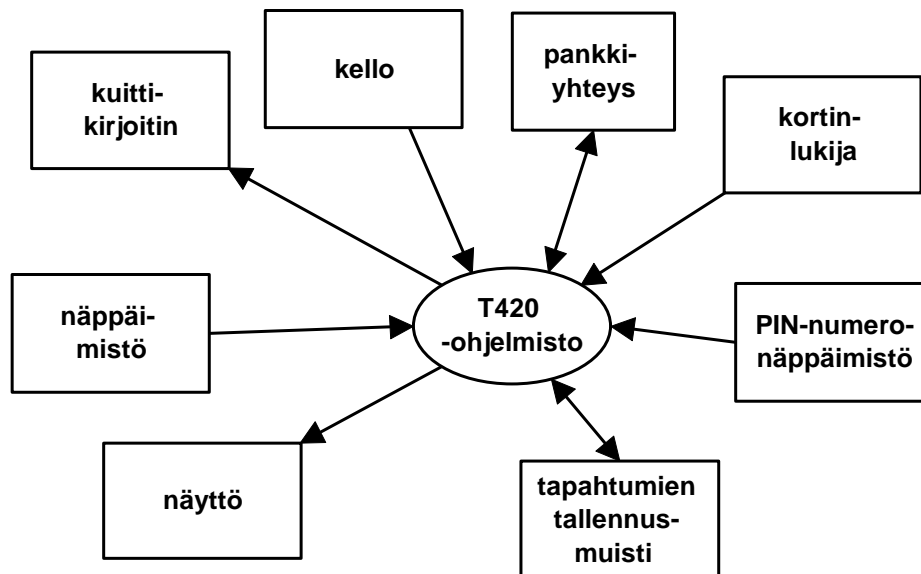
- 2) Levyke, jolla on kiistan kohteena olevien ohjelmistojen koodi. Koodi on sekä ohjelmoijan kirjoittamassa muodossa että vaikeampiselkoisessa "konekielisessä" muodossa, jossa se ladataan järjestelmään.
- 3) Kopio VTT-teollisuusautomaation tuottamasta vertailuraportista, jonka on laatinut Unto Pulkkinen (päivätty 22.12.1994).
- 4) Kopiot Teemu Kallialan VTT:lle lähettämästä em. raportin tarkennuspyynnöstä ja Unto Pulkkinen siihen laatimasta vastauskirjeestä (päivätty 23.3.1995).
- 5) Kopio Rauno Pyyhtiän (Systek) 1.11.94 ilmeisesti VTT:lle lähettämästä kirjeestä, jossa on osoitettu kohtia, joihin vertailussa kannattaa kiinnittää huomiota.
- 6) Kopio Rauno Pyyhtiän 18.8.95 päivästä lausunnosta, jossa pyritään osoittamaan ohjelmien samankaltaisuus.
- 7) Kopiot Juseliuksen ja Salojään antamista lausunnoista
 - Juselius ja Salojää: Verifone T420 ohjelmiston toteutus VF Partner Oy:ssä, 24.10.1994.
 - Salojää: Kommentit muistipaikkoihin, 29.12.1994.
 - Salojää: Perustelu luvun 245 käytölle, 27.11.1996.

Lisäksi arviointi perustuu Salojään ja Juseliuksen ilmoitukseen, jonka mukaan he ovat kirjoittaneet (ainakin merkittävilta osin) Systekin ohjelmaversioita ja lisäksi ylläpitäneet sitä useita vuosia Systekillä ennen siirtymistään VF Partnerin palvelukseen.

3. Arvioinnin suorittaminen

Käytimme asiaan perehtymiseen ja tämän raportin kirjoittamiseen noin kolme henkilötyöpäivää. Minun lisäksi asiaan perehtyi laitoksellamme tutkijana työskentelevä Matti Rintala. Ajasta kului noin yksi työpäivä asian taustojen selvittämiseen, yksi työpäivä ohjelmakoodin analysointiin ja yksi työpäivä tämän lausunnon kirjoittamiseen. Käytettävissä olleen ajan puitteissa ei perusteellinen perehtyminen ohjelmistoon ollut luonnollisestikaan mahdollista. Tästä syystä keskityimme asiasta aikaisemmin annetuissa lausunnoissa osoitettuihin kohtiin ja lausunnoissa esitettyjen väittämien arviointiin. Arviointi kohdistuu edellisen kohdan materiaaliluettelon viitemateriaaleihin 2-4 ja 6. Erityisen tarkastelun kohteena olivat Pyyhtiän lausunnossa (viitemateriaali 6) osoitetut kohdat, koska oletimme hänen ohjelmiston tuntevana osaavan osoittaa keskeisimmät kohdat. Salojään ja Juseliuksen kommenttien (viitemateriaali 7) arviointiin käytettävissä oleva aika ei juurikaan riittänyt.

Matti Rintala teki ohjelmistolle muutamia tietokoneanalyyskejä, joilla yritimme etsiä ohjelmistosta lisää samankaltaisia kohtia. Vertailussa "konekielisistä" versioista poistettiin tietoa, jota olisi ohjelmistoa laittomasti kopioitaessa helppo "naamiointitarkoituksessa" muuttaa (mm. rivinumeroita ja numeeriset vakiot). Tämä vertailu ei tuonut esille uusia samankaltaisuuksia aikaisemmin osoitettujen lisäksi.



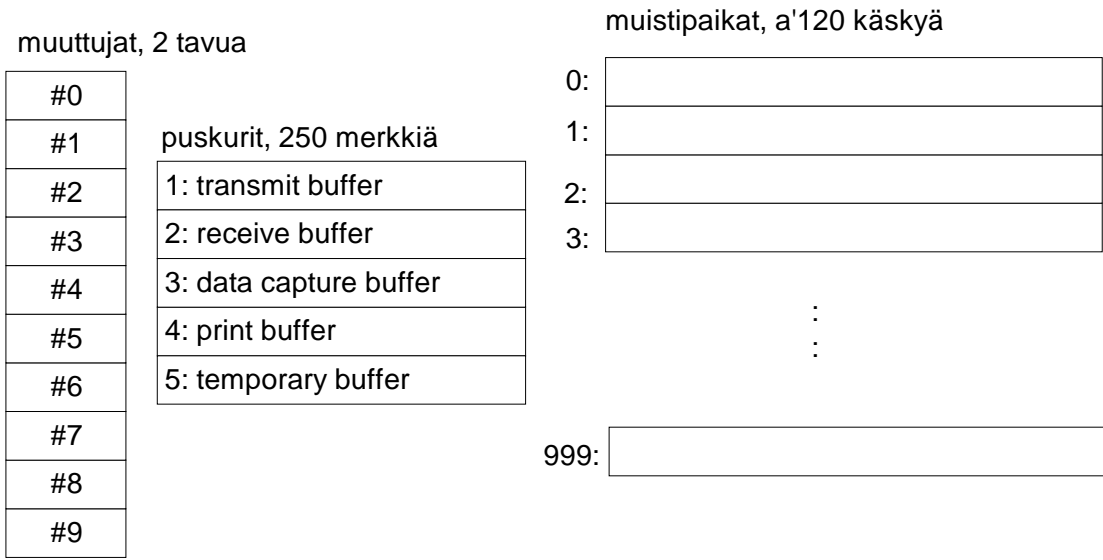
Kuva 1: T420-maksupääteohjelmisto

Seuraavassa selostuksessa joudutaan pakostakin puuttumaan ohjelmistojen tuottamisen tekniikkaan ja ohjelmointiin. Yritän kuitenkin tehdä sen niin kansantajuisesti, että maallikkokin pystyy seuraamaan esitystä ainakin yleisellä tasolla. Esityksen suppeudesta johtuen tekstiin jää kuitenkin väkisinkin kohtia, joiden perusteellinen ymmärtäminen vaatii ainakin jonkinasteista ohjelmointitaitoa. Yritän myös selvittää ohjelmistojen tuottamiseen liittyviä taustatietoja tarvittavassa määrin.

4. Arvioinnin kohteena oleva järjestelmä

Kyseessä oleva maksupäätejärjestelmä 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 2). 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. Ohjel-



Kuva 2: Ohjelmointiympäristö.

mat kirjoitetaan TCL-kielellä 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. Maksupäätössovelluksesta 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 ohjelmakommenteina selväkielisen selityksen siitä, mitä käskyn on tarkoitus tehdä; esimerkiksi

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

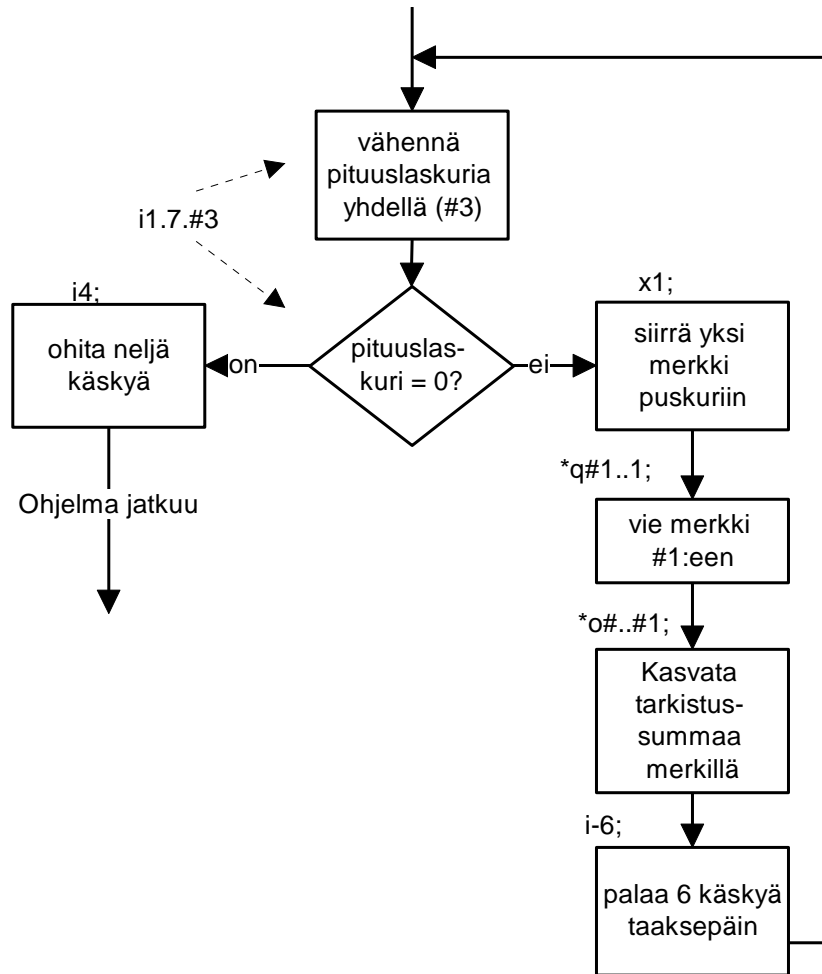
(TCL-kielessä puolipisteen jälkeen tuleva osa riviä tulkitaan kommentiksi). Vielä tätäkin tärkeämpää olisi ryhmittää käskyt muutamien käskyjen kokonaisuuksiksi, joille lisätään yhteinen kommentti; esimerkiksi *“sanoman pituus on nyt muuttujassa #3. Seuraavat käskyt käyvät läpi vastaanottopuskurin (puskuri 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, aliohjelmillä 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
```



Kuva 3: Esimerkkiohjelman toiminta

Koodin toiminta selviää kuvasta 3, 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 nolautuu, 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 kommentteilla ja sisentämällä ohjelmarivejä ohitettavien käskyjen kohdalla. Hallussani olevissa ohjelmissa on monin paikoin käytetty sisennystä — ei kuitenkaan johdonmukaisesti kaikkialla.

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 kuvaamieni 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.

5. Ohjelmistojen tuottaminen

Asiaa tuntemattomat samaistavat usein ohjelmistojen tuottamisen ja ohjelmoinnin. Käytännössä ohjelmointi, eli siis itse koodin kirjoittaminen, on vain pieni osa koko prosessia. Ohjelmiston tuottaminen voidaan jakaa vaiheisiin useilla eri tavoilla. Käytännössä voidaan yleensä erottaa ainakin seuraavat asiakokonaisuudet: määrittely, suunnittelu, ohjelmointi ja testaus. Määrittely kuvaa ohjelman toiminnallisuuden käyttäjän kannalta. Maksupääteohjelmiston tapauksessa määritellään mitä toimintoja päätteessä on ja miten niitä käytetään. Esimerkiksi kuvataan, miten maksutapahtuman käsittely etenee: kortti vedetään läpi kortinlukijasta, järjestelmä tarkastaa, onko kortti sulkulistalla, jos se ei ole, näyttöön tulostetaan kortti OK jne. Määrittelyssä ei siis käsitellä järjestelmän teknisiä ominaisuuksia muuta kuin niiltä osin, kun ne asettavat rajoituksia toteutukselle. Suunnitteluvaiheessa määritellään ohjelman jako osajärjestelmiin ja osajärjestelmien sisällä edelleen pienempiin osiin (aliohjelmista koostuviin moduuleihin). Ohjelmointivaiheessa kirjoitetaan ohjelman käskyt ja testausvaiheessa kokeillaan systemaattisesti ohjelman toimivuutta.

Ajallisesti nämä asiakokonaisuudet eivät useinkaan muodosta selkeitä peräkkäisiä vaiheita, vaan varsinkin kokemattomien suunnittelijoiden tapauksessa ja/tai jouduttaessa tekemisiin täysin uudentyyppisen sovelluksen kanssa, ne sekoittuvat vaikeasti toisistaan erotettavissa olevaksi kokonaisuudeksi, jossa määritellään, suunnitellaan, ohjelmoidaan ja testataan ilman mitään järjestelmällisyyttä.

Tärkein em. vaiheista on yleensä määrittely: sen suorittaminen vaatii ko. liiketoiminnan, tässä tapauksessa maksukorttijärjestelmän tuntemusta. Toteutuksen tärkein vaihe on suunnittelu — itse ohjelman kirjoittaminen on hyvin suunnitellussa järjestelmässä yleensä yksinkertaisena pidetty tehtävä. Työmäärän jakautuminen vaiheittain on luonnollisesti täysin tapauskohtaista. Maksupääteohjelmiston tapauksessa yksi lähtökohta voisi olla esimerkiksi: n. 1/3 määrittelyä ja alustavaa suunnittelua, 1/3 yksityiskohtaista suunnittelua ja ohjelmointia ja 1/3 testausta. Toteutettaessa sama ohjelmisto uudelleen määrittelyvaihe jää kokonaan pois, ja suunnittelukin on huomattavasti ensimmäistä versiota helpompaa.

Maksupääteohjelmiston tapauksessa pidän helppona ja suoraviivaisena projektia, jossa samanlaisen ohjelmiston jo kerran toteuttaneet suunnittelijat toteuttavat vastaavan toiminnallisuuden omaavan ohjelmiston uudelleen. Näin siinäkin tapauksessa, että alkuperäistä ohjelmaa ei ole käytettävänä toteutuksen pohjaksi. Jos ohjelmisto on aiottu joka tapauksessa suunnitella uudelleen, ei aikaisemmasta ohjelmakoodista ole välttämättä erityistä hyötyä, varsinkin jos vanha koodi on sellaista, että sen muuttaminen uuteen järjestelmään on vaikeaa ja virhealtista.

6. Ohjelmien oikeudellisesta suojasta

Ohjelmistojen oikeudelliseen suojaan liittyy lukuisia ongelmia, joita yritetään ratkoa tuomioistuimissa eri puolilla maailmaa. En voi väittää olevani minkäänlainen asiantuntija tällä oikeudellisella alueella. En kuitenkaan malta — tilaisuuden saatuani — olla puuttumatta tähän mielenkiintoiseen ongelmaan. Yritän seuraavassa valottaa ongelmaa ohjelmistotekniikan kannalta.

Tavallisesti suoja perustuu tekijänoikeuslainsäädäntöön. Tämä on ehkä perinteisistä suojamuodoista ohjelmistoille sopivin (vrt. mallisuoja, patentti), mutta siihen liittyy ohjelmistojen erityispiirteistä johtuvia ongelmia. Jos joku kopioi orjallisesti toisen yrityksen tuottaman ohjelman, on kopiointi normaalitapauksessa yleensä verraten helppo osoittaa. Jos uuden ohjelmiston tekevät samat ohjelmistosuunnittelijat, voivat ohjelmat kuitenkin olla hyvin samankaltaisia ilman, että suoranaista kopiointia on tapahtunut. Edellä kohdassa 5 esittämäni perusteella on selvää, että ohjelmisto on muutakin kuin ohjelman ohjelmarivit. Konkreettisesti ohjelmasta on sen omistajayrityksen hallussa ohjelmakoodi, mutta sen lisäksi ohjelmistoon liittyy myös määrittelyssä ja suunnittelussa hankittu ja luotu uusi tieto, joka ei yleensä ole dokumentoituna minnekään, ja vain vaivoin kaivettavissa esiin itse ohjelmasta.

Suuri osa ohjelmaan sisältyvästä ”pääomasta” ei siis ole itse ohjelmassa, vaan sen suunnitelleissa ja sitä ylläpitävissä ihmisissä. Tällaista pääomaa on erittäin hankala suojata. Se, ettei tämä ongelma ole tullut vielä kärjistetympin esille ohjelmistojen yhteydessä johtuu luultavasti alan teknologian nopeasta kehityksestä. Vain harvoin sattuu tilanteita, joissa kannattaa kopioida kilpailijan tuote. Yleensä kopioitu tuote on valmistuessaan auttamattomasti vanhentunut, ja kilpailijallakin on jo markkinoilla tuotteen seuraava sukupolvi.

7. Ohjelmien samuuden tutkiminen

Tietokoneohjelmien samankaltaisuutta vertailtaessa voi edellä esittämäni (kohta 5) perusteella kiinnittää huomiota seuraaviin seikkoihin

- 1) ohjelmilla saattaa olla samankaltainen käyttöliittymä ja/tai toiminnallisuus (määrittely kopioitu),
- 2) ohjelmat saattavat olla rakenteeltaan samanlaisia (suunnittelu kopioitu) tai
- 3) ohjelmat saattavat sisältää samoja algoritmeja ja identtistä ohjelmakoodia.¹

VeriFone-maksupäätteen tapauksessa ei voi mitenkään ajatella, että täsmälleen tai lähes toiminnallisesti samanlaisen maksupäätteen tekeminen loukkaisi tekijänoikeutta. Tekijänoikeushan ei suojaa ohjelman ideaa ja toiminnallisuutta. Lisäksi käyttöliittymä on yksinkertainen ja alaan liittyvät standardit sekä käytettävä laitteisto ja sen käyttöjärjestelmä pakottavat samankaltaisen toiminnallisuuden toteuttamiseen.

Ohjelman rakenteella tarkoitetaan tapaa, jolla ohjelmisto on sisäisesti jaettu osiin (moduuleihin ja aliohjelmiin), ja tapoja, joilla nämä osat välittävät tietoa toisilleen. Ohjelmien rakenteen samankaltaisuus voisi mahdollisesti osoittaa, että myöhemmin laaditun ohjelman toteutuksessa on käytetty mallina aikaisemmin laadittua ohjelmaa. Nyt käsillä olevassa vertailussa molempien ohjelmistojen suunnittelijoina on kuitenkin samat henkilöt, joten rakenteen vertailuun ei kannata ryhtyä, koska rakenteen mahdollinen samankaltaisuus olisi helposti selitettävissä.

Ohjelmakoodin ja algoritmien vertailussa tulee esille monia pulmallisia kysymyksiä. Jopa eri ohjelmoijien tekemissä ohjelmissa saattaa esiintyä hämmästyttävässä määrin täysin samanlaisia kohtia, puhumattakaan saman ohjelmoijan uudelleen kirjoittamasta ohjelmakoodista. Syitä on monia; esimerkiksi:

- 1) Ohjelmoijat koulutetaan ja kouliintuvat työssään ratkaisemaan ongelmia samalla tavalla. Tämä aiheuttaa samantapaisuutta algoritmitasolla. Esimerkiksi tarkistussumman laskemiseen ei ole kovin montaa vaihtoehtoa. Kerran toimiviksi havaitut algoritmit toistuvat yhä uudelleen. Esimerkiksi VF Partnerin tapauksessa ainakin kolmessa samaksi osoittautuneessa koodinpätkässä kyseessä oli itse asiassa sama rutiininomainen algoritmi (tarkistussumman laskeminen).
- 2) Ohjelmointia opetellaan lukemalla esimerkkiohjelmiä manuaaleista ja koulutusmateriaaleista sekä lukemalla ja ylläpitämällä toisten ohjelmoijien tekemiä ohjelmia. Näin samat (hyvät tai huonot) käytännöt kopioituvat ohjelmasta toiseen.
- 3) Ohjelmoijat koulutetaan ja kouliintuvat työssään käyttämään ohjelmointikieltä samalla tavalla. Kun jokin ratkaisu on havaittu hyväksi ja toimivaksi,

1. Algoritmilla tarkoitetaan tässä, kuten ohjelmistotekniikassa yleensäkin, ohjelman tai sen osan logiikkaa. Kun algoritmi ilmaistaan jollain ohjelmointikielillä, saadaan ohjelmariveistä koostuva ohjelma tai ohjelman osa. Toisin päin sanottuna ohjelma on ohjelmointikielillä kuvattu algoritmi. Esimerkiksi kuva 3 esittää tarkistussumman laskevan algoritmin ja sen yhteydessä on sekä kuvassa että tekstissä esitetty ohjelmaan liittyvät koodirivit, siis ohjelma eli koodi eli ohjelmakoodi. Algoritmin ei siis tarvitse olla, eikä se yleensä olekaan, mikään hieno "keksintö" tai erikoinen oivallus, vaan sillä tarkoitetaan yksinkertaisesti tapaa, jolla ohjelman logiikka etenee.

ohjelmoija soveltaa sitä yhä uudelleen. Tällaisista perusratkaisuista saattaa tulla tietynlaista kansanperinnettä, jonka käyttäjä ei edes välttämättä tiedä, miksi ratkaisu toimii. Esimerkiksi tietoliikenneprotokollan alustussanomassa saattaa olla kenttiä, joiden merkitystä kukaan ei tiedä, mutta on “yleisesti tiedossa”, että “näin ne pitää olla, muuten ei homma pelaa”. Kenttien arvot on esimerkiksi aikoinaan voitu kopioida ohjelmointimanuaalissa olevasta esimerkistä.

- 4) Ohjelmointiympäristö saattaa ohjata tietyn tyyppisiin ratkaisuihin. Tässä suhteessa VeriFonen maksupäätteen on mielestäni äärimmäinen esimerkki: ohjelmarakennetta ohjaa muistin jaottelu muistipaikkoihin, muuttujat on määritelty ennalta (#0...#9) ja tietojen käsittely tapahtuu aina puskureissa 1...5. Käskykannan rakenne pakottaa tietyntyyppeihin ohjelmistoratkaisuihin. Esimerkiksi erittäin tyyppilinen käskysarja on “b3.4 g”, joka asettaa lähde- ja kohdepuskurit sekä tyhjentää kohdepuskurin (tässä puskurit 3 ja 4). Lisäksi käyttöjärjestelmän palvelut ohjaavat tietyntyyppeihin ratkaisuihin. Toinen esimerkki on ohjelmoinnissa yleisesti käytetyn laskuriin perustuvan silmukkarakenteen käyttö kuvassa 3. Vastaava silmukkarakenne toistuu tarkastelemissani ohjelmankohdissa usein.
- 5) Rajapinnat ulkomaailmaan ohjaavat ohjelmiston rakennetta. Maksupäätteen tapauksessa ensinnäkin käyttöliittymä on ilmeisesti sama tai lähes sama kuin vanhassa ohjelmistossa. Toiseksi pankkiyhteyteen liittyy Kermit-protokolla ja muut pankkien kanssa sovitut käytännöt, jotka edellyttävät samanlaisten sanomien samanlaista käsittelyä molemmissa ohjelmistoissa.

Olen perehtynyt molempiin ohjelmistoihin vain keskeisimpien samankaltaisiksi väitettyjen ohjelmanosien osalta. Mielestäni on mahdotonta todeta — ainakin ilman perusteellisempia selvityksiä — etteikö sama ohjelmoija voisi kirjoittaa saman ohjelmapätkän uudelleen täsmälleen samalla tavalla, varsinkin kun ohjelmoijalla on voinut olla käytössään esimerkiksi omia muistiinpanoja, käsikirjoja ja koulutusmateriaalia, joiden perusteella ohjelmistoa on alunperin suunniteltu.

Joka tapauksessa nyt osoitetut ohjelmarivit ovat hyvin pieni osa ohjelmasta ja ne sisältävät osittain samankaltaisia ohjelmaosia. Ne ovat rakenteeltaan tavanomaista rutiinikoodia, jonka kuka tahansa ohjelmointia taitava pystyy tarvittaessa tuottamaan.

8. Arvio VTT:n lausunnoista

Unto Pulkkisen laatima raportti on asiallisen ja asiantuntevan tuntuinen. Siitä ei kuitenkaan selviä, miten vertailu on tehty. Olisi mielenkiintoista tietää, mitä menetelmiä on sovellettu ja kuinka paljon työaikaa on käytetty. Raportista saa sen kuvan, että vertailu on koko ohjelmiston kattava. Perusteellisen vertailun tekeminen vaatii koko ohjelmiston läpikäyntiä käsky käskyltä. Lisäksi molempien ohjelmien suunnitteluratkaisut tulee käydä läpi jne. Koska koodi on verraten vaikeaselkoista, eikä ohjelmia ole kommentoitu kunnolla, tämä on erittäin työlästä ja työkustannukset voivat nousta hyvin korkeiksi. Työtä voisi todennäköisesti helpottaa laatimalla joitain yksinkertaisia työkaluja, joilla informaatio saataisiin helpommin ymmärrettävään muotoon.

Tutustumieni ohjelmanosien perusteella voin pääosin yhtyä Pulkkisen raportin johtopäätöksiin. Viimeisessä kappaleessa Pulkkinen tosin toteaa, että ohjelmassa on kohtia, joita ei suurella todennäköisyydellä voi kirjoittaa ulkomuistista. Tämä saattaa pitää paikkansa, mutta ehdottoman varmaa se ei missään tapauksessa ole, eikä Pulkkinenkaan ota kantaa siihen, millaisen lähteen kopioinnista olisi kyse. Voihan kyseessä olla esimerkiksi ohjelmoijan omat muistiinpanot, manuaalissa tai koulutusmateriaalissa oleva esimerkki, tai kyseessä voi myös olla hyvämuistinen ohjelmoija, joka ei mitenkään epätavallinen ilmiö. Lisäksi kyse voi aivan hyvin olla myös edellisessä kohdassa esille tuomistani ohjelmien samankaltaisuutta aiheuttavista seikoista. Esimerkiksi kuvassa 3 esitetty koodisekvenssi toistuu tarkastelemissani ohjelman osissa suunnilleen samanlaisena kolme eri kertaa. Se näyttää olevan TCL-ohjelmoijien hyväksi havaitsema tapa tehdä silmukoita, joissa puskurissa oleva tieto käydään tavu tavulta läpi esimerkiksi tarkistussumman laskemiseksi. Mielestäni ei olisi hämmästyttävää, vaikka toinen ohjelmoija kirjoittaisi tämän osan ohjelmaa täsmälleen samoilla käskyillä, puhumattakaan tilanteesta, jossa sama ohjelmoija kirjoittaa koodin uudelleen.

Raportin kirjoittamisen jälkeen Pulkkiselta on pyydetty tarkennuksia raportin sisältöön kysymyksiä muodossa (viitemateriaali 4). Seuraavassa lyhyesti Pulkkisen esittämät tarkennukset ja omat kommenttini niihin. Numerointi on sama kuin ko. viitemateriaalissa.

- 1) Pulkkinen toteaa, että kyseessä ei ole rutiinikoodi. Mistään ei selviä mitä tällä termillä tarkoitetaan. Jos se tarkoittaa, että koodin kirjoittaminen on jotenkin tavallisuudesta poikkeavan vaikeaa, hän on väärässä, ellei sitten vaikeutena pidetä itse kehitysympäristön aiheuttamia ongelmia, joita kuvasin kohdassa 4.
- 2) Pulkkinen toteaa Systekin ohjelmassa olevan 13165 käskyriiviä ja VF Partnerin ohjelmassa 8370 käskyriiviä. Tätä en tarkastanut, mutta luvut vastannevat todellisuutta.
- 3) Pulkkinen toteaa, että ohjelmistotuotannossa tuotettujen käskyrivien määrä on kaikki työvaiheet huomioon ottaen 5-50 riviä / työpäivä. Tämä vastaa alalla vallitsevaa yleistä käsitystä. Tässä kannattaa kuitenkin lisäksi mainita, että 1) tuottavuuserot ohjelmoijien välillä voivat olla huimia (esimerkiksi jopa 10-20 -kertaisia) ja 2) toteutettaessa samaa ohjelmaa uudelleen tuottavuus voi olla ensimmäiseen toteutuskertaan verrattuna moninkertainen. Tämä johtuu mm. siitä, että määrittely- ja suunnitteluvaiheet jäävät lähes olemattomiksi, ohjelman kirjoittelu sujuu aieman kokemuksen perusteella nopeammin ja lisäksi testaus helpottuu, koska kokemus auttaa välttämään tavallisimmat sudenkuopat. VF Partnerin 8320 käskyrivin ohjelman vaatimaa työmäärää on käytettävissä olevien tietojen perusteella mahdoton arvioida, mutta kaikki edellytykset poikkeuksellisen korkealle tuottavuudelle ovat olemassa.
- 4) Pulkkinen toteaa, että 1000 riviä ohjelmakoodia voidaan tehdä käytännössä äärettömän monella eri tavalla. Näin onkin, jos kyseessä olisi yksi yhtenäinen ohjelman osa. Nyt näin pitkiä yhdenmukaisia osia ei nyt tarkasteltavassa ohjelmassa ole, vaan yhdenmukaisuudet jakautuvat suhteellisen lyhyisiin ohjelman osiin.

- 5) Pulkkinen toteaa, että kaksi ohjelmoijaa voi pienessä määrin tuottaa identtisiä ohjelmarivejä. Viittaan kohdassa 7 esittämiini seikkoihin.
- 6) Pulkkinen toteaa, ettei sama ohjelmoija eri päivinä voi tuottaa identtisiä käskyriivejä: jos kyse on esimerkiksi 50 käskystä, poikkeavat ne ainakin detaljien osalta hieman — poikkeavuuden todennäköisyys on Pulkkinen mukaan yli 90%. Tässä olen Pulkkinen kanssa eri mieltä, ellei poikkeamina pidetä eri muistipaikkojen, muuttujien ja puskureiden käyttöä (ja nämä poikkeamathan VF Partnerin ohjelmistosta löytyvätkin). Jos kyseessä on esimerkiksi standardimuotoisen sanoman muodostaminen puskuriin, tekee sama ohjelmoija sen suurella todennäköisyydellä aina samalla tavalla. Joka tapauksessa Pulkkinen arvio yli 90% varmuudesta ei voi perustua mihinkään arvausta parempaan.
- 7) Pulkkinen toteaa, että käytössä on ollut pieni osa Systekin lähdekoodista. Arviota Systekin koodin orjallisesta kopioinnista edes vähäisessä määrin raportissa kuitenkin mielestäni kyetä osoittamaan oikeaksi. Joka tapauksessa on mahdoton osoittaa, missä määrin tieto on siirtynyt ohjelmoijan päässä, missä määrin ohjelmoijan omissa muistiinpanoissa tai manuaaleissa ja koulutusmateriaalissa, onko käytössä ollut ohjelmalistaus, vai onko kenties suoraan siirretty ohjelman osia tiedostoina, ja editorilla muokattu siinä muistipaikkojen, muuttujien ja puskurien numerot sekä ohjelmakomentit.
- 8) Lopuksi Pulkkinen toteaa, että ainakin lähtökohtana VF Partnerin ohjelmille on ollut Systekin ohjelma. Näinhän tietenkin on pakkokin olla, koska ohjelman ovat toteuttaneet samat ohjelmoijat. Se, sisältyykö samankaltaisuuteen tekijänoikeusrikkomus, on kokonaan toinen juttu. Tässä kohdassa Pulkkinen myös viittaa kahteen ilmeisesti varmana pitämänsä todisteeseen. Toinen on Pulkkinen mukaan tähdillä ympäröity kommentti (Systek: **** OK **** ja VF Partner: **** Valmis ****). Tässä Pulkkinen kuitenkin erehtyy. Kyseessä ei ole kommentti, vaan maksupäätteen näytölle tulostuva teksti. On oikeastaan hämmästyttävää, ettei teksti ole molemmissa tapauksissa täsmälleen sama, onhan ohjelmassa varmasti pyritty käyttäjän kannalta täsmälleen samanlaiseen toimintaan. Toinen kohta on kommentti, jossa VF Partnerin ohjelmassa esiintyy samalla rivillä kaksi komenttia. Jälkimmäinen kommentti on sama kuin Systekin ohjelmassa samalla rivillä oleva kommentti.

Mielestäni Pulkkinen oleellisin kannanotto yllä on kohdassa 7, jossa hän päätyy arvioimaan, että käytössä on ollut pieni osa Systekin koodia — ottamatta kantaa siihen miten tieto olisi siirtynyt Systekistä VF Partneriin.

9. Komentit Rauno Pyyhtiän lausunnosta 18.8.1996

Pyyhtiä toteaa lausunnossaan olevansa vakuuttunut siitä, että Systekin ohjelmaa on kopioitu VF Partnerin ohjelmaan. Seuraavaksi Pyyhtiä epäilee, että VF Partnerin ohjelmaa on tahallaan sotkettu lisäämällä hyppyjä taaksepäin. Tätä on kuitenkin vaikea uskoa, koska todennäköisesti olisi helpompaa kirjoittaa ohjelmakoodi uudelleen kuin alkaa sekoittaa jo valmiiksi sekavan ohjelmakoodin käskyjä.

Tämän jälkeen Pyyhtiä luettelee joukon kohtia, joissa ohjelmat ovat samanlaisia. Ensi silmäyksellä näyttää, että tekstissä viitataan kymmeneen eri kohtaan. Tarkempi tarkastelu osoittaa, että ensinnäkin kaksi viimeistä kohtaa on aikaisemmin esitettyjen kohtien toistoa. Ne on otettu esille uudelleen muutaman yksityiskohdan analysointia varten. Toiseksi ensimmäinen ja kahdeksas kohta ovat itse asiassa sama muistipaikka, joka toistuu listassa ilman mitään erityistä syytä. Kolmessa kohdassa kyse on käytännössä hyvin samantapaisesta ohjelmakoodin osasta, eli tarkastussumman laskemisesta (vrt. kuva 3). Viitattuja eri muistipaikkoja on siis 7 ja näistäkin kolmen sisällöt ovat hyvin samantapaiset (eli siis em. tarkastussumman laskeminen). Samankaltaisiksi väitettyjä kohtia on siis laskentatavasta riippuen 5 tai 7. Kaikissa tapauksissa kyse on tavanomaisesta rutiinikoodista.

Lopuksi hän tarkastelee kahta kohtaa koodissa, joiden hän arvelee todistavan koodin kopioiduksi. Molemmat kohdat liittyvät tietoliikennesanomien käsittelyyn. Ensimmäisessä kohdassa hän tarkastelee lukua 245, jonka hän väittää olevan täysin satumanvaraisesti valittu. Salojään selvityksen perusteella (viitemateriaali 7) luvun valinta ei ole sattuma, vaan luvuksi kelpaisi ilmeisesti jokin luku välillä 92-255 (tätä en pystynyt ao. manuaalien ja ajan puutteen vuoksi varmistamaan). Se, että luvuksi on valittu tältä väliltä juuri luku 245 saattaa olla esimerkki ns. "hyväksi havaitusta" parametrin arvosta. Toisessa kohdassa pohdiskellaan vaihtoehtoisia tapoja viedä kaksi välilyöntimerkkiä puskuriiin. Pyyhtiä väittää kyseessä olevan virheen, joka on vahingossa jäänyt koodiin. Salojään selvityksessä merkintätavan käytölle annetaan uskottavan tuntuinen selitys, jota minun ei kuitenkaan ollut mahdollista varmentaa, koska käytettävissä ei ollut Kermit-protokollan sanomakuvauksia (eikä myöskään aikaa). Tässä Pyyhtiän ja Salojään lausunnot ovat tässä selkeästi ristiriidassa keskenään. Mielestäni kumpikaan kohta riittää osoittamaan että kopiointia olisi tapahtunut.

10. Yhteenveto

Omat johtopäätökseni perustuvat seuraaviin oletuksiin:

- 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 kopioiduksi väitetyt osat tai ovat ainakin ylläpitäneet niitä ollessaan työsuhteessa Systekiin, joten he ovat tunteneet Systekin ohjelmassa käytetyt ratkaisut hyvin toteuttaessaan VF Partnerin ohjelmistoa.

Näistä oletuksista lähtien päädyn johtopäätökseen, jonka mukaan VF Partnerin ohjelma on sen omaa tuotantoa, eikä kopio Systekin ohjelmasta. Siinä olevat samankaltaisuudet Systekin ohjelmaan voi suurelta osin selittää kohdassa 7 esittämilläni seikoilla. Lisäksi samankaltaisiksi havaitut rivit eivät sisällä mitään erityisiä keksintöjä tai oivalluksia, vaan ovat aivan tavallista sovelluskoodia, jota kuka tahansa ohjelmoija pystyy helposti tuottamaan perehdyttyään ensin VeriFonen laitteistoon ja käyttöjärjestelmään sekä ohjelmistossa sovellettaviin standardeihin. Tämä ei tietenkään sulje pois mahdollisuutta, että tekijöillä olisi ollut käytettävissään ohjelmaa kirjoittaessaan Systekin

ohjelmaan liittyviä muistiinpanoja, ohjelmalistaus tai ainakin osia siitä. Tämän osoittaminen kiistattomasti ei ole ainakaan tähän mennessä esitettyjen argumenttien perusteella mahdollista.

Tampereella 8.12.1996

Ilkka Haikala