Anssi Takku

Ohjelmiston laadun ja luotettavuuden estimointi luotettavuusmallien avulla

Tietojärjestelmätieteen
pro gradu -tutkielma
27.06.2004

Jyväskylän yliopisto
Tietojenkäsittelytieteiden laitos
Jyväskylä

TIIVISTELMÄ

Takku, Anssi Juhani
Ohjelmiston laadun ja luotettavuuden estimointi luotettavuusmallien avulla / Anssi Takku
Jyväskylä: Jyväskylän yliopisto, 2004.
97 s.
Pro gradu -tutkielma

Matkapuhelinsovellusten lyhentyneet kehittämisprosessit ja nopea teknologian muuttuminen asettavat paineita tuotteen laadukkuudelle ja luotettavuudelle. Tässä tutkimuksessa pohditaan soveltuvatko ohjelmiston luotettavuusmallit matkapuhelinsovellusten kehittämiseen. Hyvä luotettavuusmalli auttaa päättämään, onko ohjelmisto jo sopiva asiakkaan käyttöön vai pitääkö kehitystä jatkaa ja kuinka kauan.

Tutkimukseen valittiin kolme erityyppistä luotettavuusmallia. Valintaan vaikutti menetelmien referenssit, alan kirjallisuuden kritiikki ja tutkimuksessa suoritetun kohdeyrityksen kartoituksen tulokset. Mallien toimintaperiaatteet esitetään ja niitä arvioidaan tutkimuksessa kehitetyn viitekehyksen avulla. Lisäksi tutkimukseen kartoitettiin globaalissa kohdeyrityksessä kehitettyjä laadun ja luotettavuuden parantamiseen tähtääviä menetelmiä. Viitekehys ei soveltunut niiden analysoitiin, joten niiden perusolettamuksia tyydyttiin tarkastelemaan.

Tämän hetken luotettavuusmallien heikkous on etteivät ne tarjoa tarpeeksi hyötyjä verrattuna niiden vaatimiin resursseihin. Luotettavuusmallit osoittautuivat vaikeiksi prosesseiksi, ja voi olla, ettei tulevaisuudessakaan tulla kehittämään yhtä yleisesti tarkkaa mallia. Luotettavuusmalleja parempana vaihtoehtona voi olla tehostaa laatumenetelmien käyttöä kehittämisprosessin aikana. Näin estetään ohjelmistovirheiden syntymistä kehitettävään tuotteeseen.

AVAINSANAT: matkapuhelinsovellus, laatujohtaminen, ohjelmistovirhe, ohjelmiston luotettavuusmalli

SISÄLLYS

1 JOHDANTO 6

2 MATKAPUHELINTEN OHJELMISTOKEHITYS 8

2.1 NYKYTILA JA TULEVAISUUS 8
2.2 OHJELMISTOARKKITEHTUURIT 10
2.2.1 Ensimmäinen arkkitehtuuri 10
2.2.2 Kehittyneempi arkkitehtuuri 11
2.2.3 Älypuhelinarkkitehtuuri 12

3 LAATUJOHTAMINEN JA SEN SOVELTAMINEN OHJELMISTOKEHITYKSEEN 15

3.1 MITÄ LAATU ON? 15
3.2 LAATUJOHTAMISEN PERIAATTEET 16
3.3 LAATUJOHTAMISEN SOVELTAMINEN OHJELMISTOKEHITYKSEEN 18
3.3.1 Jatkuva kehittyminen 19
3.3.2 ISO 9000 -standardi 21
3.3.3 Muodolliset katselmoinnit 22
3.3.4 Metriikat 24
3.4 LAATUJOHTAMISEN ONGELMIA KÄYTÄNNÖSSÄ 26

4 OHJELMISTOVIRHEET 29

4.1 OHJELMISTO VS. LAITTEISTO 30
4.2 KUSTANNUKSET 33
4.3 PARETO-ANALYYSI 35
4.4 VIRHEIDEN ENNUSTAMINEN 37

5 TUTKIMUSYMPÄRISTÖ JA -MENETELMÄ 40

5.1 TUTKIMUSYMPÄRISTÖ 40
5.2 TUTKIMUSMENETELMÄ 43
5.2.1 Tiedonkeruumenetelmät 44
5.2.2 Viitekehyksen toteuttaminen 46

6 OHJELMISTON LUOTETTAVUUSMALLIT 52

6.1 HAASTATTELUIDEN JA KYSELYIDEN TULOKSET 53
6.2 KIRJALLISUUDESSA ESITETTYJÄ LUOTETTAVUUSMALLEJA 56
6.2.1 Tilastolliset vikamallit - SRE 58
6.2.2 Kausaaliset analyysit - Bayes-verkot 64
6.2.3 Ääripäiden välimaasto - ODC 69
6.3 KOHDEYRITYKSESSÄ KÄYTÖSSÄ OLEVIA MENETELMIÄ 73
6.3.1 Virhetietokannat 74
6.3.2 Tositestaaminen 74
6.3.3 Nopea markkinapalaute 75
6.3.4 Komponenttitehtaiden käytännöt 75

7 MENETELMIEN ARVIOINTI JA JATKOTOIMENPITEET 77
7.1 MENETELMIEN ARVIOINTI 77
7.1.1 Kirjallisuuden luotettavuusmallit 77
7.1.2 Kohdeyrityksen käytössä olevat menetelmät 80
7.2 SUOSITUS KOHDEYKSIKÖN JATKOTOIMENPITEIKSI 82
7.3 JOHTOPÄÄTÖKSET 85
LÄHTEET 88

LIITE 1. SÄHKÖPOSTIKYSELYN RUNKO 95

KUVIOT

KUVIO 1. NOKIAN KÄYTTÄMIEN OHJELMISTOARKKITEHTUURIEN MYYNTIVOLYYMIT 2001-2003 11
KUVIO 2. LAATUJOHTAMISEN KOLME PERUSPERIAATETTA JA NIIDEN VÄLISET YHTEYDET. 19
KUVIO 3. PDCA-SYKLI JA SEN VAIHEITA TUKEVIA TEKNIIKOITA SEKÄ TAITOJA 20
KUVIO 4. OHJELMISTOVIRHEEN ELINKAARI OHJELMISTOKEHITYKSESSÄ 29
KUVIO 5. LAITTEISTON VIKATAAJUUS SUHTEESSA SEN ELINKAAREEN 31
KUVIO 6. OHJELMISTON VIKATAAJUUS SUHTEESSA SEN ELINKAAREEN 32
KUVIO 7. VIRHEEN KORJAAMISESTA AIHEUTUVAT SUHTEELLISET KUSTANNUKSET OHJELMISTON ELINKAAREN ERI VAIHEISSA 34
KUVIO 8. PARETO-ANALYYSI - 80 PROSENTTIA VIOISTA AIHEUTUU 20 PROSENTISTA KOMPONENTTEJA 36
KUVIO 9. MOBIILI IP:N VERKKOARKKITEHTUURI 41
KUVIO 10. TESTAUKSEN V-MALLI 42
KUVIO 11. TUTKIMUKSEEN VALITTUJEN LUOTETTAVUUSMALLIEN SIJOITTUMINEN MALLIEN KIRJOON 57
KUVIO 12. SRE-MALLIN YDINVAIHEET AJALLISESTI SIJOITTUNEENA OHJELMISTON KEHITTÄMISEN ELINKAARELLE 59
KUVIO 13. LUOTETTAVUUDEN HAVAINNOINTIKAAVIO 62
KUVIO 15. YKSINKERTAINEN BAYES-VERKKOESIMERKKI TODENNÄKÖISYYSARVOINEEN 66
KUVIO 16. BAYES-VERKON TOPOLOGIA VIRHEIDEN ESTIMOINTIIN 67
KUVIO 17. VIKATYYPPIEN JAKAUTUMISEN MUUTOS KEHITTÄMISVAIHEITTAIN 72

TAULUKOT

TAULUKKO 1. LYUN JA NIKORAN VIITEKEHYKSEN KOMPONENTIT 48
TAULUKKO 2. KEHITETTY VIITEKEHYS OHJELMISTON LUOTETTAVUUSMALLIEN VERTAILUUN 50
TAULUKKO 3. SRE:N ANALYSOINTI VIITEKEHYKSEN AVULLA 63
TAULUKKO 4. BAYES-VERKKOJEN ANALYSOINTI VIITEKEHYKSEN AVULLA 68
TAULUKKO 5. TUNNISTETTAVAT OMINAISUUDET, KUN VIKA ON LÖYDETTY 70
TAULUKKO 6. TUNNISTETTAVAT OMINAISUUDET, KUN VIKA ON KORJATTU 71
TAULUKKO 7. VIKATYYPPIEN YHTEYDET OHJELMISTON KEHITTÄMISVAIHEISIIN 71
TAULUKKO 8. ODC:N ANALYSOINTI VIITEKEHYKSEN AVULLA 73
TAULUKKO 9. VIITEKEHYKSEN AVULLA ANALYSOIDUT LUOTETTAVUUSMALLIT 78

1 JOHDANTO

Matkapuhelinmarkkinat ovat kasvaneet kiihtyvällä vauhdilla jo vuosikymmenen ajan. Vuonna 1992 oli maailmassa kymmenen miljoonaa matkapuhelimen käyttäjää. Kesäkuussa 2002 matkapuhelimien käyttäjien määrä ylitti ensimmäisen kerran miljardin (Nokia 2003a). Samalla ohitettiin kiinteiden puhelinliittymien tilaajamäärä. Markkinoiden kasvamisen johdosta joutuvat matkapuhelinvalmistajat julkaisemaan yhä useammanlaisia tuotteita markkinoiden tyydyttämiseksi. Esimerkiksi Nokia (Nokia 2003a) julkaisi ennätykselliset 33 uutta tuotetta vuonna 2002. Nopea julkaisutahti johtaa puhelinten kehittämisprosessien optimointiin eli tuote pyritään kehittämään yhä tehokkaammin ja nopeammin. Lyhentyneiden kehittämisprosessien ja nopean teknologian muuttumisen seurauksena, tuotteen laadun ja luotettavuuden takaaminen ovat kehittämisprosessin aikana yhä vaikeampia.

Ohjelmiston luotettavuudella (*Reliability*) tarkoitetaan mm. O´Connorin (1995, 3) mukaan tuotteen todennäköisyyttä suorittaa tietty toiminto ilman esiintyviä virheitä tietyissä olosuhteissa tietyn aikajakson verran. Ohjelmiston luotettavuuden mallintamista on tutkittu Fentonin ja Neilin (1999) mukaan jo yli 30 vuotta. Hyvä luotettavuusmalli auttaa päättämään, onko ohjelmisto jo sopiva asiakkaan käyttöön vai pitääkö kehitystä jatkaa ja kuinka kauan. Erilaisia malleja (*Reliability Model*) on kehitetty runsaasti, mutta niistä yhtäkään ei Brocklehurstin ja Littlewoodin (1992) mielestä voida yleisesti suositella. Heidän mukaan tarkkuus vaihtelee mallien välillä liian dramaattisesti. Eri luotettavuusmallien välistä vertailua on alan kirjallisuudessa suoritettu vähän ja yleisesti mallien taustalla olevaa teoriaa ei ole yhdistetty. Tämän huomion perusteella tässä tutkimuksessa tutkitaan seuraavia ongelmia:

- Soveltuvatko ohjelmiston luotettavuusmallit matkapuhelinsovelluksen kehittämiseen?
- Mitkä ovat tämän hetken keskeisimpiä ohjelmiston luotettavuusmalleja?
- Kuinka luotettavuusmallit toimivat?

- Mikä tekee luotettavuuden mallintamisen niin vaikeaksi?

Tutkimuksen tarkoituksena on luoda matkapuhelinsovelluksia kehittävälle tutkimuksen kohdeyksikölle käytänteet, joilla voidaan verifioida kehitettyjen tuotteiden laatu ja luotettavuus ennen niiden julkaisua.

Tutkimus hyödyntää konstruktiivisen tutkimuksen periaatteita. Tutkimukseen valittiin kolme erityyppistä luotettavuusmallia tarkempaan analyysiin. Valintaan vaikutti mallien referenssit, alan kirjallisuuden kritiikki ja tutkimuksessa suoritetun kohdeyrityksen kartoituksen tulokset. Mallien perusteet käydään läpi ja ne arvioidaan tutkimuksessa kehitettävän viitekehyksen avulla. Arviointi suoritetaan teoreettiselta kannalta. Lisäksi tutkimuksessa kartoitetaan kohdeyrityksen eri yksiköiden käytänteitä aihealueelta. Tarkemmin läpikäytävien luotettavuusmallien ja toisten yksiköiden käytäntöjen arvioinnin pohjalta annetaan suositukset kohdeyksikön jatkotoimenpiteiksi.

Tutkimus aloitetaan laajalla teoriaosuudella. Toisessa luvussa käydään matkapuhelinten ohjelmistokehitys tiivistetysti läpi. Luvun tarkoituksena on kuvata minkälaiseen ohjelmistoympäristöön tutkimus sijoittuu. Kolmannessa luvussa käydään laatujohtamisen periaatteet läpi ja kuinka sen menetelmiä voidaan hyödyntää ohjelmistokehityksessä. Luvussa esitetään neljä laatujohtamisen menetelmää, joiden avulla estetään tehokkaasti virheiden syntyminen kehitettävään ohjelmistoon. Neljännessä luvussa käydään ohjelmistovirheiden teoria läpi. Painotuksena luvussa on, kuinka ohjelmistovirheet ovat kompleksisia, vaikeita estimoida ja suuria kustannuksia aiheuttavia. Teoriaosan jälkeen esitetään viidennessä luvussa tutkimusympäristö ja -menetelmä. Luvussa luodaan viitekehys, jonka avulla valittuja ohjelmiston luotettavuusmalleja tullaan analysoimaan. Kuudennessa luvussa käydään valittujen luotettavuusmallien ja kohdeyrityksen kartoituksessa esiintyneiden käytänteiden perusteet läpi. Seitsemännessä luvussa edellisen luvun luotettavuusmallit arvioidaan viitekehyksen avulla ja käytänteiden perusteita tarkastellaan. Arvioinnin jälkeen annetaan kohdeyksikölle suositeltavat jatkotoimenpiteet.

2 MATKAPUHELINTEN OHJELMISTOKEHITYS

Luvussa keskitytään matkapuhelinten kehittämiseen etenkin kohdeyrityksen ohjelmistoarkkitehtuurien kannalta. Ohjelmistoarkkitehtuureista esitellään historiajärjestyksessä ensimmäinen varteenotettava arkkitehtuuri CUI (*Common User Interface*), siitä kehittyneempi ISA (*Intelligent Software Architecture*) ja älypuhelimia varten suunnattu Symbian. Painopisteenä on etenkin Symbian-arkkitehtuuri, koska siinä nähdään alan tulevaisuus ja sen rakenne on kaikille kehittäjille avoin. Luvun tarkoituksena on antaa kuvaa minkälaiseen ohjelmistoympäristöön tutkimus keskittyy.

2.1 Nykytila ja tulevaisuus

Matkapuhelinkäyttäjien määrä on ylittänyt 2002 ensimmäisen kerran miljardin (Nokia 2003a). Tämän takia mm. Päivärinnan ja Luukan (2001, 271) mukaan puhelinvalmistajien tulee kasvattaa tuotevalikoimaansa markkinoiden tyydyttämiseksi. Heidän mukaansa ihmiset haluavat pieniä ja halpoja matkapuhelimia, jotka sisältävät yhä suuremman määrän ominaisuuksia. Tässä piilee ristiriita, minkä Päivärinta ja Luukka myöntävät. Yleisesti ominaisuudet maksavat ja aivan kaikkea ei helposti kehitetä, sillä rajana ovat matkapuhelimen fyysiset ominaisuudet. Muistia on rajattu määrä, näytön koko on rajallinen sekä käytettävän prosessorin nopeus asettaa myös omat rajoituksensa ohjelmistolle.

Matkapuhelin koostuu noin 90-prosenttisesti ohjelmistosta, ja siksi ohjelmiston kehittäminen on puhelimen kehittämisen vaativin vaihe. Matkapuhelinohjelmisto kehitetään kuin normaali ohjelmistokin eli käytetään tilanteeseen parhaiten soveltuvia metodeja ja tekniikoita. Yksi suosittu kehittämismalli matkapuhelimen kehittämisessä on v-malli, jonka suuri testausmäärä soveltuu vaativan ohjelmiston kehittämiseen. Matkapuhelimen ohjelmiston kehittämisen tekevät vaativaksi mm. muistin, prosessorin nopeuden ja näytön koon rajat. Lisäksi matkapuhelin saattaa olla yhtäjaksoisesti kuukausia käynnissä. Tämän takia kehittämisessä on erityisen tärkeää välttää muistivuotoja.

Matkapuhelinten kehittämisajat ovat lyhentyneet koko ajan. Esimerkiksi Nokia julkaisi vuonna 2002 ennätykselliset 33 uutta matkapuhelinmallia ja korostaa vuosikatsauksessaan (Nokia 2003a), että parempia ja virheettömiä tuotteita pitäisi tuottaa yhä lyhyemmässä ajassa markkinoille. Vastauksena haasteeseen on modulaarinen suunnittelutyyli, joka perustuu ajatukseen, että samoja ohjelmistokomponentteja voidaan käyttää eri tuotteissa. Musa (1998, 182) käyttää kontekstista nimeä mobiilikoodi (*mobile code*), joka viittaa laajalle levinnyttä ja erilaisiin sovelluksiin sisällytettyä uudelleenkäytettyä koodia. Hänen mukaansa tämän suuren luokan muutoksen takana ovat oliokeskeisen kehittämisen, agenttien ja Internetin kautta laajalle levinneiden ohjelmistotyökalujen yleistyminen. Kehittämisprosessit ovat Musan mielestä lyhentyneet, mutta hän korostaa etteivät perimmäiset syyt ohjelmistojen epäluotettavuuteen ole hävinneet. Virheitä päätyy yhä tuotteisiin. Kehittämisprosessin muuttuminen on johtanut organisaatioiden rakenteellisiin muutoksiin, jossa tietyt osastot kehittävät ohjelmistokomponentteja, joita eri tuoteprojektit voivat käyttää. Esimerkiksi tutkimuksen kohdeyrityksessä tämänkaltaisia osastoja kutsutaan komponenttitehtaiksi.

Karin ja Kilpeläisen (2001, 287) mukaan tulevaisuuden mobiilimaailmassa tulemme todistamaan kahden viime vuosikymmenen menestyksekkäimmän ilmiön, mobiilitekniikan ja Internetin sulautumisen. Viimeisinä vuosina Internet on levinnyt räjähdysmäisesti useiden miljoonien käyttäjien keskuuteen. Sen suosio perustuu Karin ja Kilpeläisen mielestä käyttäjien mahdollisuuteen päästä käsiksi kaikenlaisiin palveluihin, tietoon ja viihteeseen yhden ainoan rajapinnan kautta. Näitä palveluja tuottavat yritykset sekä loppukäyttäjät itse. Karin ja Kilpeläisen mukaan kaiken tekevät mahdolliseksi avoimet Internet-protokollat, standardit palveluiden kehittämiskielet ja verkon modulaarisuus. Samaan aikaan mobiililaitteiden, kuten matkapuhelimien ja PDA-laitteiden, käyttö on kasvanut voimakkaasti. Matkapuhelimien ansiosta olemme tottuneet saamaan nopean yhteyden muihin ihmisiin tai palveluihin missä tahansa ja milloin vain. Yhdistämällä Internetin avoimen palvelukehityksen ja sen suuren tiedon sekä palveluiden määrän mobiiliverkon tuomaan tavoitettavuuteen saamme mobiilin Internetin. Nyt Internet TCP/IP protokollapino alkaa löytää tiensä myös mobiililaitteisiin – IP konvergoituu mobiilimarkkinoille (Darby 1999, 564; Sengodan

ym. 2000, 504). Konvergoituminen tulee vaatimaan enemmän ohjelmistoarkkitehtuureilta, joiden päälle matkapuhelinsovelluksia kehitetään.

2.2 Ohjelmistoarkkitehtuurit

Matkapuhelinsovellukset ovat monimutkaisia ja ne tuotetaan hektisellä tahdilla. Lisäksi vaadittavien resurssien määrä on myös kasvanut. Näitä sovelluksiin kohdistuvia vaatimuksia hallitaan ohjelmistoarkkitehtuurilla (Garlan 2000). Garlanin mielestä hyvä ohjelmistoarkkitehtuuri auttaa takaamaan järjestelmän tehokkuus-, luotettavuus-, siirrettävyys- ja yhteentoimivuusvaatimukset. Huono voi olla hänen mukaansa tuhoisa. Arkkitehtuurin avulla hallitaan järjestelmän eri yksiköiden välistä yhteistoimintaa. Se määrittelee, millä tavoin ja millaisella protokollalla yksiköt kommunikoivat keskenään. Sen avulla voidaan rakentaa pienistä yksiköistä isompia kokonaisuuksia. Garlan näkee ohjelmistoarkkitehtuurin toimivan yleensä siltana vaatimusmäärittelyn ja toteutuksen välillä. Uusia mobiiliohjelmistoja kehitettäessä on hyvä arkkitehtuurisuunnittelu kaiken perusta (Paakkunainen 2000, 7).

Seuraavaksi käydään lyhyesti läpi kohdeyrityksen pääohjelmistoarkkitehtuurit historiajärjestyksessä: CUI, ISA ja Symbian. CUI ja ISA -luvut pohjautuvat kohdeyrityksen aiempiin tutkimuksiin.

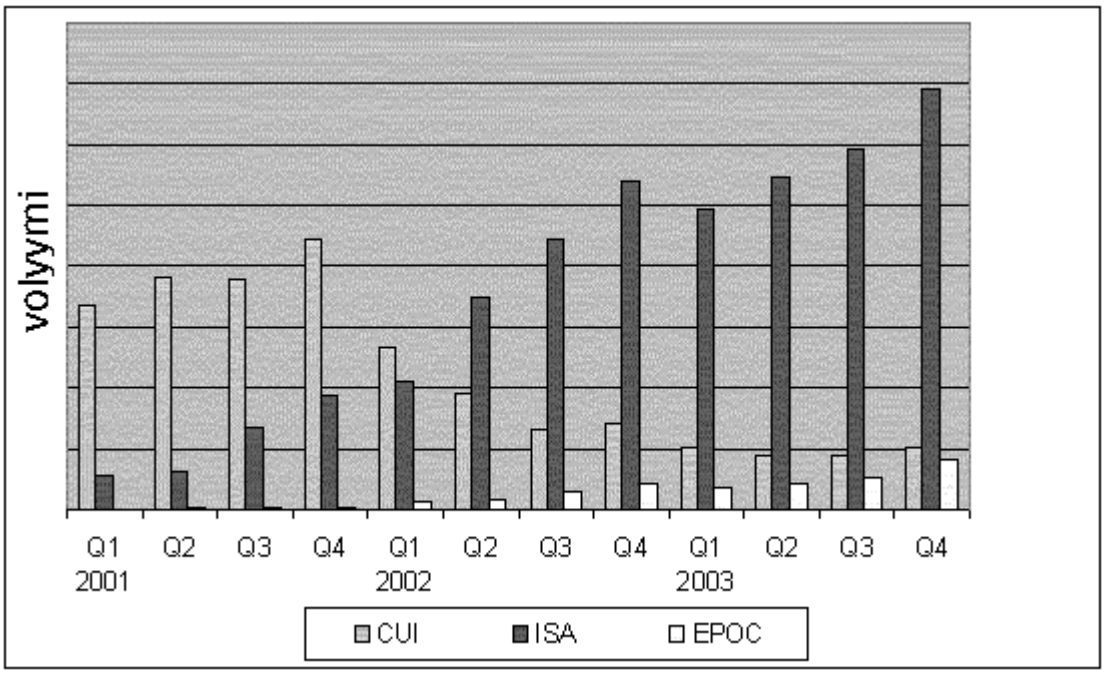
2.2.1 Ensimmäinen arkkitehtuuri

CUI (*Common User Interface*) -arkkitehtuurin kehittäminen alkoi 1990-luvun alussa. Arkkitehtuuri perustuu puhelimen resursseja jakaviin sekä hallitseviin aliohjelmiin. Jokainen näistä on itsenäinen osa koko ohjelmistoa. Karvosenojan (2000, 19) mielestä CUI:n periaate, enemmän funktiokutsuja ja vähemmän viestejä, antaa koodille lisää nopeutta ja tehokkuutta. Kuitenkin hänen mukaansa ohjelman modulaarisuus on epäselvää ja konfiguraatio vaikeaa, minkä vuoksi jo tehtyjä ominaisuuksia on vaikea hyödyntää uusissa tuotteissa. Arkkitehtuurin ongelmana on sen vaikeaselkoisuus. Lisäksi ongelmaksi muodostui resurssien hallinta. Paakkunaisen (2000, 7) mukaan alunperin UI-osajärjestelmä (käyttöliittymä) hallitsi kaikkia puhelimen resursseja. Hän

jatkaa, että ongelma ratkaistiin antamalla myös joillekin muille osajärjestelmille hallintaoikeudet, mutta tämä tapahtui ainoastaan UI:n kautta ja siksi UI muodostuikin koko arkkitehtuurin pullonkaulaksi. Tarvittiin uusi helppokäyttöisempi ohjelmistoarkkitehtuuri.

2.2.2 Kehittyneempi arkkitehtuuri

Vuonna 2002 Nokian CUI-pohjaisten matkapuhelimien myynnin ohittivat ISA (*Intelligent Software Architecture*) -arkkitehtuuriin pohjautuvat mallit (KUVIO 1). ISA kehitettiin nimenomaan sen vuoksi, että päästäisiin CUI:n ongelmista eroon. Paakkunaisen (2000, 8) mukaan arkkitehtuuri on hyvin standardoitu ja yleisesti hyväksytty.



KUVIO 1. Nokian käyttämien ohjelmistoarkkitehtuurien myyntivolyymit 2001-2003 (EPOC = Symbian) (Nokia 2003b)

Karvosenojan (2000, 20) mielestä suurin ero ISA-arkkitehtuurissa aikaisempaan CUI-arkkitehtuuriin on ohjelmiston jakaminen sovelluksiin ja palvelimiin. Hänen mukaansa arkkitehtuurin pääidea on resurssien hallinnan jakaminen palvelimille. Näin vältytään CUI-arkkitehtuurin ongelmalta, missä yksi osajärjestelmä vastaa kaikista resursseista.

Karvosenoja toteaa palvelimien tarjoavan palveluita selkeän rajapinnan kautta, minkä takia niitä voidaan muokata ilman että muutokset vaikuttavat sovelluksiin. Hänen mukaansa palvelimien ei myöskään tarvitse tietää mitään sovelluksista, jotka käyttävät niiden palveluita. Tästä seuraa, että sovelluksia voidaankin lisätä ilman vaikutusta palvelimiin. ISA-arkkitehtuurin ansiosta voivat kolmannen osapuolen ohjelmistosuunnittelijat luoda sovelluksia matkapuhelimeen (Paakkunainen 2000, 8).

ISA konseptissa on myös joitain huolia ja haittoja. Esimerkiksi Paakkunaisen (2000, 10) mukaan parannetun joustavuuden takia tarvitaan enemmän huomiota turvallisuusvaatimusten täyttämiseksi. Myös parannettu modulaarisuus ja säädettävyys aiheuttavat hänen mielestään ongelmia ohjelmiston koodimäärän kasvaessa.

ISA-arkkitehtuuriin pohjautuvat tuotteet hallitsevat tällä hetkellä kohdeyrityksen myyntivolyymia. Kumminkin tulevaisuudessa yhä useampi puhelin tulee olemaan älypuhelin (Evans-Pughe 2003). ISA-arkkitehtuuri ei sovellu tehottomuutensa ja vähäisten resurssiensa takia älypuhelimiin eli niitä varten tarvitaan oma arkkitehtuuri.

2.2.3 Älypuhelinarkkitehtuuri

Evans-Pughen (2003) mukaan älypuhelin on PDA-laitteen ja matkapuhelimen risteytys, jolla voidaan olla yhteydessä verkkoon, tarkistaa sähköpostit, järjestellä tapaamisia, linkittyä PC-tietokoneeseen, muokata taulukkolaskentaohjelman taulukoita ja tekstinkäsittelyohjelman tiedostoja sekä ladata uusia sovelluksia ja palveluita. Yhdeksi johtavaksi älypuhelinarkkitehtuuriksi on noussut Symbian-käyttöjärjestelmä.

Symbian-käyttöjärjestelmä sai alkunsa EPOC:sta eli Psion-merkkisten kannettavien tietokoneiden käyttöjärjestelmästä. Vuonna 1998 Psion, Ericsson, Motorola ja Nokia muodostivat yhteishanke Symbianin, jonka käyttöjärjestelmäksi valittiin EPOC (Symbian 2003a). EPOC:n nimi vaihdettiin väärinkäsitysten välttämiseksi Symbian OS:ksi. Symbian omistaa Symbian-käyttöjärjestelmän ja lisensioi sitä laitevalmistajille, mukaan lukien Symbianin omistajat. Tällä hetkellä Symbian-yhteisyrityksen omistajat valmistavat 80 prosenttia kaikista maailmalla myydyistä matkapuhelimista (Evans-

Pughe 2003; Symbian 2003b). Käyttöjärjestelmän tulevaisuus näyttää siis erittäin hyvältä.

Ydin Symbianissa on 32-bittinen oliosuuntautuneesti toteutettu käyttöjärjestelmä, joka sisältää ytimen (*kernel*), tiedostonhallinnan ja käyttöliittymätuen. Lisäksi käyttöjärjestelmä sisältää väliohjelmiston ja sitä tukevia sovelluksia, tehden siitä hyvän järjestelmäalustan käsilaitteiden ohjelmistosuunnittelulle. (Nokia 2003c) Käyttöjärjestelmä onkin keskittynyt pieniin, mobiili- ja datakeskeisiin laitteisiin. Oliosuuntautuneisuus on Symbian-käyttöjärjestelmän tärkein suunnitteluperiaate. Koko käyttöjärjestelmä sekä kaikki ohjelmointirajapinnat on oliosuuntautuneesti toteutettu ja suurin osa järjestelmäalustasta on toteutettu C++ -ohjelmointikielellä. Symbian-käyttöjärjestelmä ei sisällä käyttöliittymää älypuhelimille. Esimerkiksi Nokia on kehittänyt Symbianin päälle oman Nokia Series 60 -järjestelmäalustan, jota yritys lisensioi muillekin puhelinvalmistajille (Nokia 2003c). Tosin tämä herättää kysymyksiä Symbian-leirin yhtenäisyydestä. Symbian-käyttöjärjestelmä sisältää monia ainutlaatuisia suunnittelupäätöksiä, jotka tähtäävät tehokkaampaan resurssien hallintaan ja vankempaan koodiin. Ne tekevät kuitenkin Symbian-ohjelmoinnin vaikeammaksi oppia. (Viitala 2003, 16) Ohjelmoinnin vaikeudesta huolimatta käyttöjärjestelmä tarjoaa etuja sekä operaattoreille että ohjelmistokehittäjille. Operaattorit hyötyvät samaan standardiin pohjautuvista päätelaitteista, jolloin sovellusten kehittäminen tulee kustannustehokkaammaksi. Kehittäjät puolestaan pääsevät ensimmäistä kertaa tekemään omia mobiilisovelluksiaan Symbianin tarjoaman avoimen järjestelmäalustan päälle.

Jorma Ollila ilmoitti 2001 (esim. Nokia 2003c), että puolet Nokian kolmannen sukupolven puhelimista pohjautuu Symbian-käyttöjärjestelmiin vuoden 2004 loppuun mennessä. Symbian on suunniteltu riippumattomaksi laitteistosta, jotta sitä voitaisiin käyttää useissa eri puhelintyypeissä. Evans-Pughe (2003) listaa Symbian-käyttöjärjestelmän kilpailijoiksi Microsoftin Smartphone-, PDA-laitteiden PalmSource-, ja Linux-käyttöjärjestelmän.

Luvussa kuvattiin tutkimukseen liittyvää ohjelmistoympäristöä. Matkapuhelimet ovat kehittymässä yhä monikäyttöisemmiksi laitteiksi. Kehittymisen myötä

matkapuhelinsovellukset ovat yhä monimutkaisempia. Näitä sovelluksiin kohdistuvia vaatimuksia hallitaan ohjelmistoarkkitehtuurilla. Tulevaisuuden ohjelmistoarkkitehtuuriksi on nousemassa Symbian, jota käytetään älypuhelimissa. Tämän tutkimuksen ohjelmistoympäristönä toimii juuri Symbian-arkkitehtuuri. Tutkimuksen kohdeyksikkö kehittää Symbian-arkkitehtuurin päälle tosiaikasovelluksia.

3 LAATUJOHTAMINEN JA SEN SOVELTAMINEN OHJELMISTOKEHITYKSEEN

Jonesin (1996, 375) mukaan laatu ja luotettavuus ovat loogisesti yhteenkuuluvia käsitteitä. Hänen mukaansa hyvällä laadulla tehdään luotettavia tuotteita. Tässä luvussa tiivistetään laatujohtamisen perusperiaatteet ja kuinka sitä voidaan hyödyntää ohjelmistokehityksessä. Ensiksi selvitetään mitä laatu-käsitteellä tarkoitetaan, tämän jälkeen käydään läpi laatujohtamisen periaatteet sekä kuinka laatujohtamisen menetelmiä voidaan soveltaa ohjelmistokehitykseen. Lopuksi pohditaan laatujohtamisen ongelmia.

Laadusta puhuttaessa on hyvä muistaa Oaklandin (1993, 9) tiivistämä periaate: laatua tulee johtaa – se ei vain tapahdu.

3.1 Mitä laatu on?

Silénin (1998, 13) mukaan laadun käsite on muuttunut alkuperäisestä tuotteen virheettömyydestä kokonaisvaltaiseksi liikkeenjohdon käsitteeksi. Nykyisin laatu käsitetään hänen mielestään yhä useammin yrityksen laaja-alaiseksi kehittämiseksi, jonka tavoitteena on asiakkaiden tyytyväisyys, kannattava liiketoiminta ja pitkällä aikavälillä myös kilpailukyvyn säilyttäminen ja kasvattaminen. Oakland (1993, 5) on kerännyt teoksessaan alan asiantuntijoiden näkemyksiä, mitä laatu oikein tarkoittaa:

- soveltuvuutta käyttötarkoitukseen - Juran
- tuotteen tai palvelun piirteiden ja ominaisuuksien kokonaisuus, jotka täyttävät määrätyt ja odotettavat tarpeet - BS 4778, 1987 (ISO 8402, 1986) Quality Vocabulary: Part 1, International Terms
- asiakkaan nykyisten ja tulevien tarpeiden täyttämistä laadun avulla - Deming
- tuotteen tai palvelun markkinoinnin, insinööriosaamisen, tuotannon ja huollon kautta määrittyviä piirteitä, joiden avulla pystytään täyttämään asiakkaan tarpeet - Feigenbaum
- vastaavuutta vaatimuksiin - Crosby.

Yleisesti ottaen voidaan tiivistää, että laadulla tarkoitetaan asiakkaan vaatimusten täyttämistä. Lecklin (1997, 23) kuitenkin muistuttaa ettei asiakastyytyväisyys ole itsetarkoitus, johon pitää pyrkiä hinnalla millä hyvänsä. Esimerkiksi pankin asiakkaat

ovat varmasti tyytyväisiä, jos he saavat lainan nollakorolla. Lecklin muistuttaa ettei tämä kuitenkaan tarkoita, että pankin toiminta olisi laadukasta, pikemminkin päinvastoin, koska pankin oma kannattavuus kärsii saamatta jäävien tuottojen seurauksena. Käsite sisältää nykyään monta tärkeää merkitystä ja siitä on kehitetty oma johtamistyyli, laatujohtaminen, jota käsitellään tarkemmin kohdassa 3.2.

Silénin (1998, 14) mukaan laadun voi jakaa tuotteen ja toiminnan laatuun. Tuotteen laadulla tarkoitetaan kuinka asiakas kokee organisaation tuotteen. Tämä on kilpailun takia tärkeää organisaatioille. Organisaatioiden tavoitteena on tehdä laadukkaita tuotteita ja näin voittaa asiakkaiden tuoteuskollisuus. Toiminnan laadulla puolestaan organisaatiot saavuttavat edellä mainitun laadun. Toiminnan laatu on siis organisaation sisäisen toiminnan ja prosessien tehokkuutta ja virheettömyyttä. Laadukkaita tuotteita ei voi tehdä kuin laadukkaalla työnohjauksella.

3.2 Laatujohtamisen periaatteet

Silénin (1998, 38) mukaan laatujohtamisen kehitystyö on tapahtunut käytännössä pitkälti toisen maailmansodan jälkeen Japanissa. Hänen mukaansa keskeisenä lähtökohtana oli 1950-luvulla amerikkalaisen Edwards Demingin ahkera luennointi japanilaisille johtajille ja insinööreille tilastollisesta laadunvalvonnasta. Japanissa laatujohtamisella onkin pitkät perinteet. Jonesin (1996, 347) mukaan on juuri laatukontrollin eikä kustannustehokkuuden tai oikea-aikaisten markkinoiden ansiota, että japanilaiset yritykset ovat pärjänneet ennätyksellisen hyvin modernin teknologian kilpailukyvyssä. Silénin (1998, 37) mielestä yksi keskeisemmistä laatujohtamisen esiinnousua edesauttavista tekijöistä länsimaissa oli havainto Yhdysvaltojen autoteollisuuden laatukilpailukyvyn jäämisestä jälkeen Japanin autoteollisuudesta 1970-luvun lopulla. USA:n suuret autovalmistajat alkoivat tuolloin soveltaa laajasti laatujohtamisen oppeja turvatakseen kilpailukykynsä ja ne levittivät laatujohtamisen oppeja myös omiin sopimustoimittaja- ja alihankkijaverkostoihinsa. Silénin mukaan keskeiset eurooppalaiset yritykset seurasivat perässä 1980-luvun alkupuolella laatujohtamisen oppeja soveltaen. Hän jatkaa, että vähitellen 1980-luvun puolivälin jälkeen laatujohtamisen käyttö on yleistynyt keskeisissä länsimaisissa organisaatioissa.

Laatujohtaminen käsitetään monella eri tavalla organisaatioissa sekä alan kirjallisuudessa. Tutkimuksessa käytiin läpi Oaklandin (1993), Bellefeuillen (1993), Jonesin (1996), Vondarembsen sekä Whiten (1996), Silénin (1998), Lecklinin (1999), Pressmanin (2001) ja Kanin (2003) näkemyksiä laatujohtamisesta. Jokaisella oli omakohtaiset näkemykset siitä mistä laatujohtaminen koostuu, mutta seuraavat kolme perusperiaatetta oli havaittavissa tulkinnoissa: asiakkaisiin keskittyminen, jatkuva kehittyminen ja kaikkien osallistuminen. Seuraavaksi käydään periaatteet lyhyesti läpi.

Kuten kohdassa 3.1 todettiin, laatu-käsite tarkoittaa yleisesti asiakkaan vaatimusten täyttämistä. Tämä on aivan luonnollinen perussääntö. Jos organisaatiossa on suuret määrät tietotaitoa, mutta jos se on fokusoitu vääriin tuotteisiin, niin tulosta ei synny. Organisaatio, joka nopeimmin reagoi asiakkaiden vaatimuksiin, dominoi markkinoita. Siksi markkinatutkimuksista on tullut yrityksille tärkeä osa-alue. Kanin (2003, 375) mukaan laatujohtamisen tuloksena yhä useampi yritys suorittaa tutkimuksia mitatakseen asiakkaidensa tyytyväisyyttä. Bellefeuille (1993) kertoo esimerkiksi General Motorsin Cadillac-divisioonan testaavan istuinmallit asiakkailla ennen lopullisia suunnitelmia. Nykyään markkinatutkimusten tekeminen näkyy kuluttajille esimerkiksi yhä enemmissä määrin postissa tulevilla tuotekyselyillä ja aina väärään aikaan soitettavilla markkinatutkimushaastatteluilla. Kanin (2003, 375) mielestä tuotteen laatu ja asiakastyytyväisyys yhdessä muodostavat todellisen laadun merkityksen.

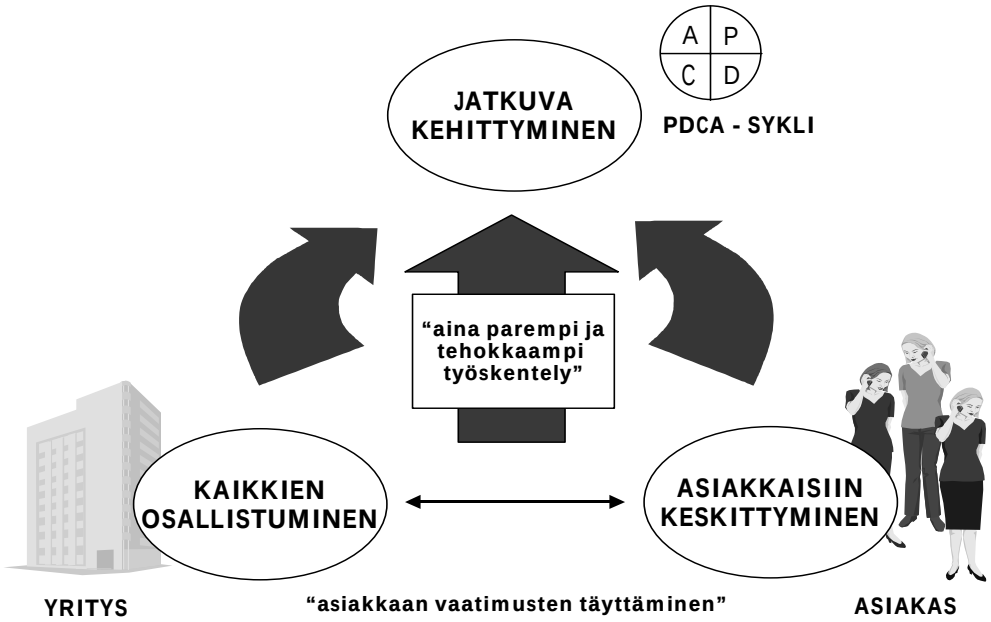
Jatkuva kehittyminen on laatujohtamisen johtava periaate kaikissa tutkimusta varten läpikäydyissä johtamistyylin tulkinnoissa. Bellefeuillen (1993) mukaan jokaisella työllä on kaksi päämäärää: ensinnäkin toimittaa tuote tai palvelu ja toiseksi parantaa tuotteiden tai palveluiden toimittamista. Parannukset eivät synny sattumalta, vaan niiden takana on suunnitelmallisuus. Jatkuvan kehittymisen päämetodina on Bellefeuillen mielestä tieteellinen lähestymistapa ongelmien ratkaisuun. Hänen mukaansa japanilaiset kutsuvat tätä Demingin sykliksi, koska Edwards Deming kehitti syklin. Deming käytti syklin pohjana Walter Shewhartin opetuksia ja kutsui itse sitä Shewhartin sykliksi. Syklillä on monta nimeä, mutta länsimaissa se tunnetaan Bellefeuillen mukaan yleisesti nimellä PDCA-sykli (Plan, Do, Check, Act). Hänen mielestään jatkuva kehittyminen on systeemi, jonka avulla yksilöt ja organisaatio etsivät keinoja tehdä asioita paremmin.

Systeemi perustuu yleensä muutosten hallintaan. Jatkuvasta kehittymisestä ja PDCA-syklistä enemmän alakohdassa 3.3.1.

Kaikkien mukaansaanti on erityisen tärkeää laatujohtamisen aikaansaamiseksi. Sama periaate pätee muillekin johtamistyyleille. Jos tietty johtamistyyli halutaan ajaa organisaatioon sisään, vaatii se pitkäkestoisen strategian. Ensinnäkin ylempien johtajien täytyy ymmärtää johtamistyylin sisällöllinen perusta ja oikeansuuntainen tulkinta. Tämän jälkeen ajatusmalli tulee iskostaa organisaation muiden työntekijöiden ajatusmaailmaan. Oaklandin (1993, 385) mukaan useimmissa organisaatioissa on neljä henkilökuntatasoa: ylemmät johtajat, keskijohtajat, työnjohtajat ja työntekijät. Jokaisella on erilaiset näkemykset laatujohtamisesta. Oaklandin mielestä ylempien johtajien tulee taata, että kaikki näkevät laatujohtamisen suotuisaksi. Ohjelmiston kehittämisessä varsinkin ongelmaksi on havaittu, että kehittämisprojektin jäsenet odottavat, että ohjelmiston laadusta vastaa koko projektin laatuvastaava. Pressman (2001, 193) muistuttaa kumminkin tosiasiasta, että jokainen ohjelmiston kehittämisprosessin osallinen on vastuussa laadusta. Laatujohtamisen kolme perusperiaatetta ja niiden väliset yhteydet on esitetty oheisessa kuviossa (KUVIO 2). Siinä laatujohtamisen pohjan luo yritys, jossa kaikki työntekijät osallistuvat laatujohtamiseen ja jonka liiketoiminnan pohjana on asiakkaisiin keskittyminen. Kaikki yrityksen prosessit pyritään tekemään jatkuvan kehittymisen mukaan paremmin ja tehokkaammin. Apuna tehokkaamman työskentelyn saavuttamiseksi käytetään PDCA-sykliä.

3.3 Laatujohtamisen soveltaminen ohjelmistokehitykseen

Laatujohtamisesta on kirjallisuudessa monia eri tulkintoja ja tämä näkyy kehiteltyjen laatujohtamista tukevien metodien määrässä. Metodeja on aivan yksinkertaisista taulukoista laajoihin koko organisaatiota koskeviin järjestelmiin. Alan kirjallisuudesta on vaikea hahmottaa mitkä tekniikat lopulta sijoittuvat laatujohtamisen alle. Tähän tutkimukseen on kerätty kirjoittajan mielestä soveltuvimmat laatujohtamisen metodit, joita voidaan hyödyntää ohjelmistokehityksessä. Menetelmien valinnassa painotettiin niiden laajuutta, alan kirjallisuuden kritiikkiä ja helppokäyttöisyyttä.



KUVIO 2. Laatujohtamisen kolme perusperiaatetta ja niiden väliset yhteydet.

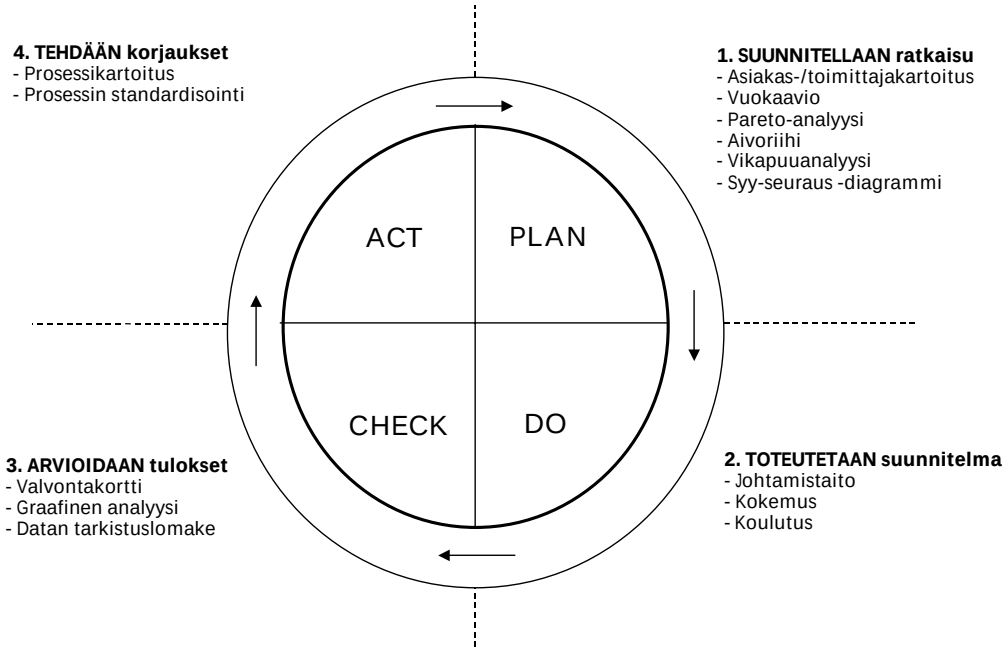
3.3.1 Jatkuva kehittyminen

Jatkuva kehittyminen on Edwards Demingin (1987) kehittämä menetelmä. O´Connorin (1995, 327-328) mukaan jatkuva kehittyminen johtaa melkein aina vähentyneisiin kustannuksiin, korkeampaan tuottavuuteen ja luotettavuuteen. Hänen mielestään menetelmä on varsinkin Japanissa, jossa siitä käytetään nimeä *kaizen*, otettu innostuneesti käyttöön. Menetelmän periaatteena on työskennellä aina paremmin ja tehokkaammin. Pressmanin (2001, 199) mukaan laatujohtaminen on sovellettavissa ohjelmistokehitykseen, koska se perustuu jatkuvalle kehittymiselle. Hänen mielestään menetelmän idea on lähellä nykyään ohjelmistojen toteutuksessa käytettävää iterointia, mikä helpottaa suunnittelutyylien yhteensovittamista. Ohjelmistoa kehitetään jokaisella iterointikierroksella vaatimuksia vastaavaksi ja tähän kehittämiseen voidaan käyttää hyväksi PDCA-sykliä. Seuraavat osatekijät ovat välttämättömiä jatkuvan kehittymisen -menetelmälle (Vonderembse & White 1996, 93-94):

- standardoidut ja dokumentoidut proseduurit,
- ryhmät, jotka etsivät parannettavia alueita,
- metodi- ja ongelmanratkaisutyökalujen analysointi,

- Plan - Do - Check - Act (PDCA) sykli,
- dokumentoidut parannetut proseduurit.

Varsinkin PDCA-sykli on tärkeä metodi jatkuvassa kehittymisessä. Sen avulla voidaan ratkoa pienempiä ongelmia tai laatia pitkäkestoisia strategioita. Metodissa suunnitellaan ensin ratkaisu ongelmaan (PLAN), sen jälkeen tehdään suunnitelmien mukaan (DO), tämän jälkeen arvioidaan toiminnan tulokset ja laatu (CHECK) ja lopuksi tehdään tarvittavat korjaukset (ACT). Tämän jälkeen sykli umpeutuu ja aloitetaan uusi kierros jälleen suunnittelulla. PDCA-syklistä tarkempi kuvaus ohessa (KUVIO 3). Kuviossa jokaiseen vaiheeseen on yhdistetty mahdollisia helpottavia tekniikoita sekä taitoja. Kuvion lähteinä on käytetty Lecklinin (1999, 54-55) teosta ja australialaisen laatukonsultointitoimiston (HCi 2003) www-materiaalia.



KUVIO 3. PDCA-sykli ja sen vaiheita tukevia tekniikoita sekä taitoja.

Aikaisemmin taylormaisessa johtamisessa ei otettu kantaa lainkaan työntekijöiden työnteon parantamiseen. Nykyisin, kuten Breyfogle (1999, 552) toteaa, *kaizen* tarjoaa työntekijöille sekä mahdollisuuden että keinot parantaa omaa työtään.

3.3.2 ISO 9000 -standardi

Asiakkaalla on yleisesti tarve varmistua toimittajan tuotteen laadusta. Pressmanin (2001, 200) mielestä tämä korostuu etenkin tietyillä aloilla (esim. telelaitteisto, lääketieteelliset laitteet ja metalliteollisuus). Hänen mukaansa standardit ohjelmiston laadunvarmistukselle esiteltiin sotilaallisten ohjelmistojen parissa 1970-luvulla ja siitä ne ovat levinneet nopeasti kaupalliseen ohjelmistomaailmaan. Näiden laatujärjestelmiin liittyvien standardien pohjalta syntyi ISO 9000 -standardisarja. Se hyväksyttiin kansainvälisen standardointijärjestön toimesta vuonna 1987 (Lecklin 1999, 315). Monilla toimialoilla on omat laatustandardinsa, mutta Euroopassa kehitetty ISO 9000 on noussut merkittävimmäksi yleisstandardiksi. Esimerkiksi vuonna 1998 oli Silénin (1998, 17) mukaan yli tuhannella suomalaisella yrityksellä ISO 9000 -sertifikaatti ja koko maailmassa sertifikaatteja oli yli 170 000.

ISO 9000 -standardin mukaisen laatujärjestelmän voi myöntää akkreditoitu kolmas osapuoli eli sertifiointilaitos. Lecklin (1999, 321-322) kuvaa sertifiointimenettelyn seuraavanlaiseksi. Sertifiointimenettelyyn kuuluvat pakollisina seuraavat vaiheet: hakemus, suunnittelukokous ja laatujärjestelmän arviointi. Hakemuksessa kuvataan yrityksen laatujärjestelmä ja vastataan haettavaan standardiin liittyvään kysymyssarjaan. Luokittelulaitos tarkastaa toimitetun hakemuksen ja nimeää hakemusta käsittelevän arviointiryhmän. Arviointiryhmässä tulee olla vähintään yksi henkilö, joka omaa hyvän kokemuksen toimialasta. Suunnittelukokouksessa annetaan palaute yrityksen laatujärjestelmästä ja sovitaan varsinaisen arvioinnin aikataulusta ja arvioitavista kohteista. Itse arvioinnissa perehdytään mahdollisimman hyvin sekä monipuolisesti yrityksen toimintaan. Käytännön toimintaa seuraamalla huomataan vastaako yrityksen kuvaama laatujärjestelmä todellisuutta. Arviointi kestää yhdestä päivästä moneen viikkoon riippuen yrityksen koosta. Kaikki havaitut poikkeamat kirjataan ylös ja esitetään yhteenvetokokouksessa. Kun esitetyt poikkeamat on korjattu ja tarkastettu uudelleen, myönnetään yritykselle laatusertifikaatti.

ISO 9000 -standardisarjasta löytyy standardeja eri toimialoille. Esimerkiksi Lecklinin (1999, 276) mukaan ohjelmiston rakentamistyöhön on pyritty lisäämään ryhdikkyyttä

ISO 9000-3 -standardilla. Standardi antaa perustan yrityksen laatukulttuurille. ISO 9000 kertoo mitä sertifikaatin saamiseksi tulee tehdä, mutta se ei kuvaile kuinka tavoitteet saavutetaan (Pressman 2001, 217). Yrityksen tulee näin kehittää omat metodit ja tekniikat tavoitteiden saavuttamiseksi. Näin sertifikaatin ansaitseva yritys on luonut perustan omalle laatujohtamiselleen. Jones (1996, 337) kritisoi ISO-standardien tuovan kohtalaisen vähän parannusta ohjelmiston laatuun. Ne nostavat hänen mielestään hieman ironisesti pikemminkin tuotteiden hintaa, mihin syynä on ehkä huomattava paperin kulutuksen kasvu. ISO 9000-3 -standardin avulla yritys voi kehittää ohjelmistotuotantonsa käytänteet oikeille raiteille ja jatkaa sertifikaatin hankkimisen jälkeen käytänteiden parantamista yrityksen liiketoimintaa tukeviksi. Silénin (1998, 19) mukaan valitettavasti suurin osa yrityksistä ei jatka käytänteiden parantamista, vaan tyytyy jäämään sertifikaatin edellyttämälle tasolle. Hän korostaakin standardien edustavan aina keskinkertaisuutta, eikä keskinkertaisuudella voi differentioitua markkinoilla.

3.3.3 Muodolliset katselmoinnit

Katselmoinnit ovat olleet kauan käytössä ohjelmistojen kehittämisessä. Jonesin (1996, 356) mukaan katselmointeja on pidetty ohjelmistojen alkuajoista lähtien. Hänen mielestään katselmointien suosio perustuu siihen, että testauksen huomattiin sijaitsevan liian myöhäisessä kehittämisvaiheessa. Tämän vuoksi testauksella voidaan vaikuttaa liian vähän tuotteen laatuun. Katselmoinnit ovat jo varsin yleisiä monissa yrityksissä. Niitä voi olla muodollisia ja virallisia tai vähemmän virallisia, esimerkiksi kahvihuonekeskustelut, kokouksen esitelmät, jne. Pääasia niissä on löytää virheet tuotteesta, ennen kuin ne päätyvät seuraavaan ohjelmistotuotantovaiheeseen tai asiakkaalle tehtävään julkistukseen. Esimerkiksi pelkän testauksen käytöllä saatetaan vaatimusmäärittelyn virheet havaita vasta kehittämisprojektin loppupuolella ja vahingot ovat tällöin jo suuret. Katselmoinnit sekoitetaan helposti tarkastuksiin. Erona näillä kahdella on katselmointien painottuminen suunnitelmien ja tarkastusten koodin oikeellisuuteen. Pressmanin (2001, 206) mukaan muodollisen katselmoinnin (*Formal Technical Review*) tavoitteet ovat:

1. Löytää virheet ohjelmiston toiminnallisuudesta, logiikasta tai toteutuksesta.

2. Verifioida, että ohjelmisto vastaa sille asetettuja vaatimuksia.
3. Varmistaa, että ohjelmisto on tuotettu ennalta määriteltyjen standardien mukaan.
4. Saavuttaa ohjelmisto, joka on tehty yhtenäisten käytäntöjen mukaan.
5. Tehdä projekteista helpommin johdettavia.

Pressmanin mielestä katselmoinnit tarjoavat nuoremmille kehittäjille hyvän mahdollisuuden tarkkailla erilaisia ohjelmistoanalyysejä, -suunnitteluita ja -toteutuksia. Muodolliset katselmoinnit edistävät näin ohjelmistojen varmistamista ja jatkuvuutta, sillä enemmän ihmisiä tulee tutuksi ohjelmiston osien kanssa, joita he eivät ole koskaan nähneet.

Muodollinen katselmointi tulee järjestää hallitusti tarkan kontrollin alla, nimittäin huonosti kontrolloitu katselmointi voi vain pahentaa tilannetta. Pressman (2001, 206) on kerännyt hyvän ohjelistan muodollisen katselmoinnin pääsäännöistä:

- Katselmointiin osallistuvien ihmisten määrän tulee (tyypillisesti) olla kolmen ja viiden henkilön välillä.
- Ennakkotyötä tulisi tapahtua, mutta kumminkin maksimissaan kaksi tuntia per henkilö.
- Katselmoinnin kestoaika tulee olla vähemmän kuin kaksi tuntia.

Jones (1996, 357) lisää tärkeiksi säännöiksi myös, että katselmointiin osallistuville on roolit selvillä (kuka on puheenjohtaja, sihteeri ja jne.), työjärjestys tehtynä ja tärkeimpänä muistaa arvioida tuotetta eikä sen tekijää. Katselmoinnin aikarajan takia tulee katselmoinnin keskittyä melko pieneen osaan tuotetta, jotta toiminta olisi tehokkainta. Chillaregen, Bhandarin, Chaarin, Hallidayn, Moebusin, Rayn ja Wongin (1992) mielestä katselmoinnin kriittisin osa on arvioida onko dokumentit katselmoitu riittävän monen ja ammattitaitoisen ihmisen toimesta. He painottavat erityisesti, että dokumentit katselmoitaisiin eri ammattitaidon hallitsevien työntekijöiden toimesta. Näin tarkastettaisiin eri osa-alueiden oikeellisuus katselmoitavissa dokumenteissa. Oikein suoritettu muodollinen katselmointi on yksi tehokkaimmista laadunvarmistajista.

Katselmointien tapaan toimivat koodin tarkastukset ovat myös tehokas tapa löytää ohjelmistovirheet ja korjata ne. Tosiasiassa suurin osa ohjelmistotuotteen elinkaaren

virheistä esiintyy suorittamatta koodia. Esimerkiksi koodin tarkastaminen voi paljastaa 40 prosentista 60 prosenttiin kaikista vioista. (Chillarege 1994; Jones 1996, 356) Koodin tarkastaminen voi Musan, Ianninon ja Okumoton (1987, 13) mukaan koostua kääntäjän diagnooseista, suunnittelu- ja koodikatselmoinneista tai koodin lukemisesta.

Jonesin (1996, 357) mielestä katsastukset eivät ole kalliita, mutta kuitenkin todella tehokkaita. Kumminkin Oakland (1993, 289) kehottaa laatuosastoa käyttämään enemmän aikaa virheiden estämiseen kuin tarkastamaan tai korjaamaan niitä.

3.3.4 Metriikat

Metriikoiden käyttö on katselmointien ohella tärkein laatujohtamisen metodi ohjelmistokehityksessä. Metriikoiden avulla pystytään arvioimaan ohjelmiston kypsyyttä. Fentonin ja Neilin (2000) mielestä ohjelmistometriikat on kollektiivinen termi, jota käytetään kuvaamaan ohjelmistokehittämisen mittaamiseen käytettäviä erilaisia toimintoja. Nämä toiminnot vaihtelevat perinteisistä ohjelmistokoodin ominaisuuksia tarkkailevista mittareista aina malleihin, jotka auttavat ennustamaan ohjelmiston laatua ja tarvittavien resurssien määrää. Lecklin (1999, 165) kiteyttää metriikoiden pääidean hyvin seuraavaan lauseeseen: ”Jos et voi mitata prosessia, et voi ohjata sitä, ja jos et voi ohjata, niin et voi johtaa ja hallita sitä.” Metriikoiden avulla voidaan siis mitata ja hallita prosessia.

Fentonin ja Neilin (2000) mukaan ohjelmistometriikat on aiheena yli 30 vuotta vanha, mutta ne ovat silti vain hädin tuskin saavuttaneet suosiota ohjelmistokehityksessä. Syynä tähän on ollut se, ettei metriikoiden tarjoamaa informaatiota ole osattu muokata tukemaan johdon päätöksentekoa. Heidän mielestään metriikoiden käyttämisen kaksi päämotivoijaa ovat halu ennustaa kehittämisprosessin hyödyt/kustannukset ja tulevien ohjelmistotuotteiden laatu. Jones (1996, 3) näkeekin, että suunnittelu ja estimointi ovat mittaamisen peilikuvia. Hänen mukaansa yhä enemmän metriikoiden avulla saatuja mittaustietoja käytetään tulevaisuuden ennustamiseen. Jonesin mielestä on olemassa täydellinen korrelaatio hyvän mittaustarkkuuden ja estimointitarkkuuden välillä – yhtiöt, jotka käyttävät metriikoita hyvin, estimoivat myös hyvin. Kanin (2003, 85)

mukaan ohjelmistometriikat voidaan luokitella kolmeen kategoriaan: tuotemetriikkoihin, prosessimetriikkoihin ja projektimetriikkoihin. Hänen mukaansa tuotemetriikat kuvaavat tuotteen ominaisuuksia kuten kokoa, kompleksisuutta, suunnittelun ominaispiirteitä, tehokkuutta ja laadun tasoa. Prosessimetriikoita voidaan käyttää parantamaan ohjelmiston kehittämistä ja ylläpitoa. Esimerkiksi virheiden poistamisen tehokkuutta voidaan parantaa. Projektimetriikoilla kuvataan projektin luonnetta ja toteuttamista. Tällaisia ovat Kanin mukaan mm. ohjelmistokehittäjien lukumäärä, kustannukset, aikataulu ja tuottavuus. Organisaation aloittaessa datan keräämisen metriikoiden avulla voi ongelmaksi muodostua ylikerääminen ja puutteellinen analysointi. Fentonin ja Neilin (2000) mielestä analysointi on metriikoiden käyttämisen tärkein vaihe. Jopa pienellä määrällä dataa saadaan hyvän analysoinnin ansiosta arvokasta informaatiota kehittämisprosessista. Siksi on tärkeää, että metriikkaprojektit ovat analysointipainotteisia datapainotteisuuden sijaan.

Ilman mittaamista on mahdotonta vertailla onko prosessi parantunut vai ei. Vertailupohjana mitatulle materiaalille voidaan käyttää laajaa edustavaa otantaa tai edellisiä ohjelmistoprojekteja. Jos aiempien projektien ohjelmistokehittäjät eivät ole mittausmetriikoita käyttäneet, jää tehokkain hyöty saavuttamatta. Kan (2003, 474) suosittelee, että organisaatioissa, joissa on yli 100 työntekijää, tulisi olla vähintään yksi kokoaikainen metriikkahenkilö. Hänen mukaansa metriikkahenkilö päättäisi organisaation laatutavoitteisiin sopivat metriikat, suunnittelisi niiden keräämisen, vastaisi datan laadusta, analysoisi datan ja tarjoaisi lopuksi palautetta johdolle sekä kehittämisprosessille. Näin taattaisiin työn tekeminen kunnolla, koska Pressmanin (2001, 98) mukaan suurin osa ohjelmiston kehittäjistä ei vieläkään harrasta mittausta ja valitettavasti monilla heistä ei ole halua aloittaakaan. He ovat kuitenkin avainroolissa datan hankkimisessa. Ohjelmistokehittäjien asennoitumista voi parantaa selvittämällä heille metriikoiden käytön myötä saavutettavat hyödyt.

Metriikoiden käyttö on yritykselle olennainen tekniikka, kun ohjelmistoprosessien kehittäminen aloitetaan. Niiden avulla voidaan todistaa, että parannusta on syntynyt. Mittaamisessa on hyvä muistaa Albert Einsteinin lausahdus (Pressman 2001, 81): ”Not

everything that can be counted counts, and not everything that counts can be counted.” Tärkeintä on päättää mitä mitata ja miten mittauksia arvioida.

3.4 Laatujohtamisen ongelmia käytännössä

Laatujohtaminen on leviämässä vähitellen suomalaisten yritysten johtamistyyliin. Joissakin yrityksissä se on viety jo menestyksekkäästi pitkälle ja joissain se on jäänyt polkemaan paikallaan. Ongelmana on juuri laatujohtamisen jääminen ISO 9000 -sertifioinnin hankkimisen jälkeen tietylle tasolle, eikä johtamisfilosofiaa enää paranneta enemmän yrityksen toimintaa tukevaksi. Silénin (1998, 144) mielestä yrityksissä on keskitytty usein toiminnasta irrallisten ISO 9000 -laatujärjestelmien rakentamiseen ja laatupalkintokriteereiden mekaaniseen soveltamiseen sen sijaan, että olisi rakennettu toimivia laatukulttuureja organisaatioihin. Hänen mukaansa johtajien tulee ymmärtää, että ISO 9000 -standardi antaa vain perustan laatujohtamiselle. Silénin (1998, 18) ja Jonesin (1996, 338) mielestä ISO 9000 -standardi synnytti viime vuosikymmenellä Suomessa ja muuallakin maailmassa keinotekoiset sertifiointimarkkinat, jotka ovat hyödyttäneet liiketaloudellisesti enemmän sertifiointilaitoksia kuin yrityksiä. Ongelmana koko järjestelmässä on, että kolmas osapuoli eli auditointilaitoksen tarkastaja määrittelee, mikä on laatua ja mikä ei. Tällöin organisaatio saattaa keskittyä pitämään asiakkaan sijasta tarkastajan tyytyväisenä. ISO 9000 -standardi voi erota huomattavasti yrityksessä jo käytössä olevasta laatukulttuurista. Riskiksi voi muodostua, että yritykseen rakennetaan toinen laatujärjestelmä sertifikaatin takia jo olemassa olevan rinnalle. Tämä pysäyttää helposti organisaation toimintaprosessit ja laaduntuottokyvyn. Silén (1998, 18) korostaakin ettei EU-komission mukaan ole selkeää näyttöä siitä, että sertifiointi olisi tuonut kilpailuetua Eurooppaan ja että tuotteiden laatu olisi sertifioiduissa yrityksissä parempi kuin sertifioimattomissa.

Laatujohtamisen yhtenä perusperiaatteena on kaikkien sitoutuminen johtamisfilosofiaan. Oaklandin (1993, 316) mukaan yrityksissä koetaan liian usein laatujohtamisen keskittyvän vain erityisten laatuihmisten keskuuteen. Hänen mielestään periaatteiden omaksumisessa syntyy syvä kuilu ylempien johtajien ja tuotannon välille. Moni kokee, että laatujohtaminen saa tekemään valtavan ja turhan määrän

dokumentaatiota ja hidastaa alati tuotantoa. Todellisuudessa laatujohtaminen ei ole pelkästään dokumenttien luomista. Se on laadun luomista. Parempi laatu johtaa vähentyneeseen muokkaustyöhön, nopeampaan toimitusaikaan ja laadukkaampaan tuotteeseen. Jonesin (1996, 345) mukaan valitettavasti vain noin puolet laatujohtamismenetelmien kokeiluista Yhdysvalloissa ovat onnistuneita ja toinen puoli epäonnistuneita. Hänen mukaansa menestyminen korreloi vahvasti siihen, kuinka vakavasti päälliköt ja johtavat sitoutuvat laatujohtamiseen sekä kuinka syvällisesti he sen ymmärtävät. Jonesin tutkimukset paljastavat, että laatujohtamista menestyksekkäästi ohjelmiston parissa käyttävät yritykset omaavat yleisesti kolme ominaisuutta: (1) ne käyttävät tehokkaita mittausohjelmia, mitkä selvittävät vikojen alkuperän, vakavuuden ja niiden poistamisen tehokkuuden, (2) ne hyödyntävät formaaleja katselmointeja ja tarkastuksia ennen testauksen alkamista ja (3) niiden ohjelmiston laatu oli hyvä tai erinomaista ennen kuin laatujohtaminen aloitettiin. Tämän perusteella Jonesin (1996, 345) mielestä laatujohtaminen toimii, kun sitä käytetään oikein.

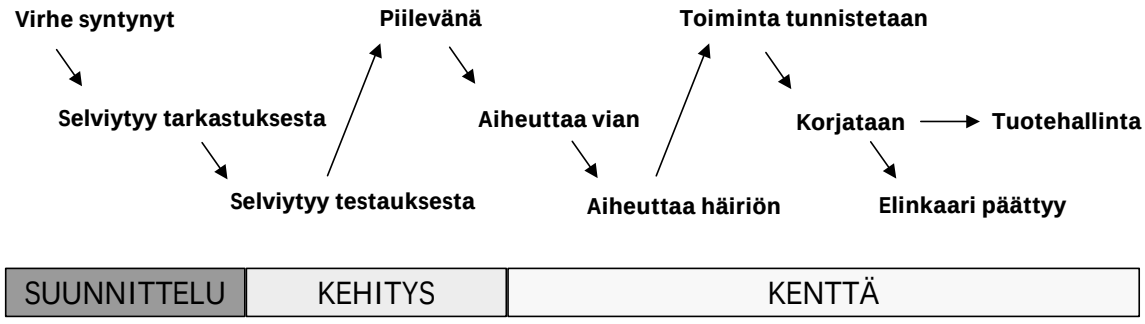
Laatujohtamista tukemaan on kehitetty lukuisa määrä erilaisia metodeja, ja osa niistä on liian hankalia. Esimerkiksi kirjallisuudessa paljon kehuttu laadun talo -menetelmä (*Quality Function Deployment*, QFD) tarjoaa mielestäni vähän tuloksia verrattuna siihen, kuinka paljon resursseja menetelmän käyttö vaatii. Lecklinin (1999, 202) mukaan menetelmän avulla taataan asiakkaan asettamat laatuvaatimukset suunnittelussa, tuotannossa ja jakelussa. Siinä rakennetaan asiakkaiden tarpeista laaja taloa muistuttava matriisi, jota luetaan jopa kuudesta eri näkökulmasta (Lecklin 1999, 203). Kun vielä matriisiin sijoitettavien tarpeiden määrä kasvaa yli kymmenen, on vaikea uskoa, että syntyvällä sekasotkulla voisi jotenkin laatua parantaa. Yritysten tuleekin tarkkaan tutkia, mitkä menetelmät soveltuvat parhaiten niiden käyttöön.

Vaikka hyvä laaduntuottokyky onkin yrityksen keskeinen kilpailutekijä, yrityksen ei pidä keskittyä pelkästään laatuun, vaan laatu tulee ottaa huomioon osana yrityksen toiminnan kokonaisuutta. Silén (1998, 6) muistuttaakin, että laatua tulee kehittää yhdessä muiden kilpailukyvyn osatekijöiden kanssa.

Luvussa tiivistettiin laatujohtamisen perusteet ja esiteltiin kuinka sitä voidaan soveltaa ohjelmistokehityksessä. Valittujen menetelmien periaatteet käytiin läpi ja lopulta pohdittiin laatujohtamisen ongelmia. Laatujohtamisen avulla kehittämisprosessin laadukkuutta voidaan parantaa ja tämä näkyy lopputuotteen luotettavuudessa. Seuraavassa luvussa tarkastellaan ohjelmistovirheitä.

4 OHJELMISTOVIRHEET

Virheet ovat poikkeamia asiakkaiden odotuksista, vaatimuksista ja tarpeista. Esimerkiksi Iresonin, Coombs Jr:n ja Mossin (1995, 22.4) mukaan, kun tekstinkäsittelyohjelma lukkiutuu eikä vastaa käyttäjän toimiin, niin silloin on asiakkaan kannalta kyseessä virhe. Oheisessa kuviossa (KUVIO 4) käydään läpi ohjelmistovirheen muodostuminen. Ohjelmistohäiriöiden päälähteenä ovat suunnitteluvirheet (Musa, Iannino & Okumoto 1987, 7). Virhe syntyy suunnittelussa ja selviytyy tarkastuksista sekä katselmoinneista itse julkaistavaan tuotteeseen. Siellä se pysyy piilevänä, kunnes käyttöolosuhteet laukaisevat vian, joka aiheuttaa ohjelmistohäiriön. Häiriön esiintyessä sen käyttäytyminen rajataan ja toiminta tunnistetaan. Kun tarpeellinen tietämys häiriöstä on saavutettu, se korjataan. Lopuksi esiintynyt virhe dokumentoidaan ja tapaus käsitellään loppuun. On kumminkin harhaluuloa ajatella, että kaikki ohjelmistovirheet syntyvät suunnitteluvaiheessa ja ne korjataan ohjelmistokehityksen lopussa. Virheitä ilmenee koko prosessin ajan. Niiden korjaaminen voi tapahtua aikaisemmin tai myöhemmin riippuen virheiden tietämyksestä ja kyvystä tehdä muutos tuotteeseen.



KUVIO 4. Ohjelmistovirheen elinkaari ohjelmistokehityksessä (Chillarege 1994)

Ohjelmiston luotettavuudella (*Reliability*) tarkoitetaan mm. O´Connorin (1995, 3) mukaan tuotteen todennäköisyyttä suorittaa tietty toiminto ilman esiintyviä häiriöitä tietyissä olosuhteissa tietyn aikajakson verran.

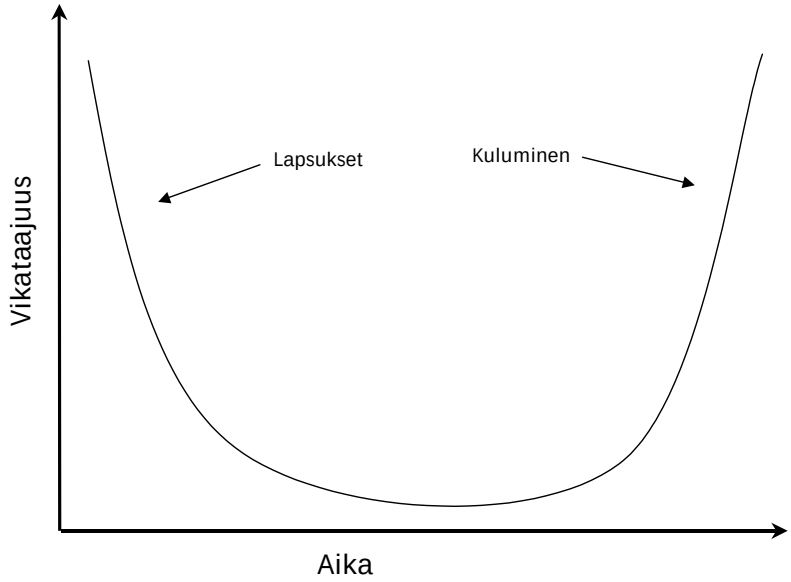
Luvussa käydään ensin läpi, millaisia ohjelmistovirheet ovat, kuinka ne eroavat laitteistovirheistä ja miten ne esiintyvät verrattuna ohjelmiston elinkaareen. Tämän

jälkeen pohditaan niistä aiheutuvia kustannuksia. Seuraavaksi esitetään luotettavuuden parantamisen yksinkertainen apuväline Pareto-analyysi. Lopuksi pohditaan mikä tekee ohjelmistovirheiden estimoinnin niin vaikeaksi.

4.1 Ohjelmisto vs. laitteisto

Ymmärtääksemme ohjelmistoa meidän pitää tutkia sen tunnusmerkkejä, mitkä tekevät sen erilaiseksi kuin muut ihmisten rakentamat tuotteet. Esimerkiksi laitteistoon verrattuna ohjelmisto on enemmänkin looginen kuin fyysinen systeemin osa. Pressmanin (2001, 6) mukaan laitteistoa rakennettaessa ihmisten luovat prosessit (analyysi, suunnittelu, rakentaminen ja testaus) lopulta muuntuvat fyysiseen muotoon. Jos rakennamme uuden tietokoneen, alkuperäiset hahmotelmamme, formaalit suunnittelupiirustuksemme ja koekytketty prototyyppi päätyvät fyysisten tuotteiden sisään (siruihin, piirilevyihin, virtalähteisiin, jne.). Vaikka joitain samoja ominaisuuksia on sekä ohjelmistokehityksessä että laitteiston valmistuksessa, ovat ne fundamentaalisesti täysin erilaisia. Pressmanin (2001) mielestä molemmissa toiminnoissa korkea laatu saavutetaan hyvän suunnittelun kautta, mutta laitteiston valmistusvaihe voi tuoda uusia (tai helposti korjattavia) laatuongelmia ohjelmistolle. Esimerkiksi Iresonin ym. (1995, 22.4) mukaan Hubble-teleskoopin alun värähtelyongelmia vaimennettiin ohjelmistomuutoksilla.

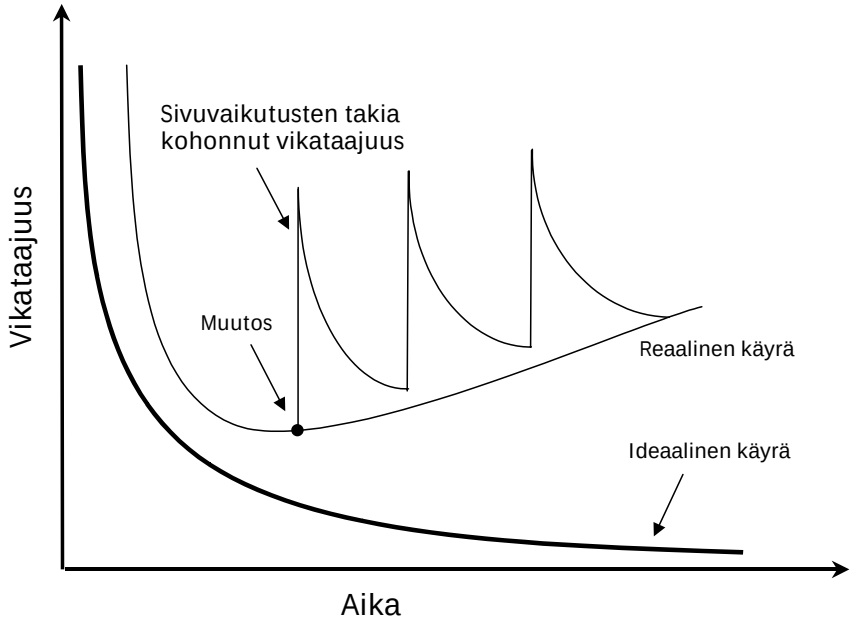
Ohessa (KUVIO 5) on esitetty laitteiston vikataajuusfunktio ajan suhteen. Pressmanin (2001, 7) mukaan syysuhde, josta käytetään usein nimeä ”kylpyammekurvi”, osoittaa, että laitteistolla ilmenee kohtalaisen korkea vikataajuus sen elinkaaren alussa. Nämä viat johtuvat hänen mielestään usein suunnittelu- ja valmistusvioista. Kun viat korjataan, laskee vikataajuus tasaiselle tasolle joksikin aikaa. Ihannetapauksessa tämä taso on mahdollisimman alhainen. Kuitenkin, kun aika kuluu, vikataajuus nousee uudestaan, kun laitteistokomponentit alkavat Pressmanin mukaan kärsiä ympäristön kumulatiivisista vaikutuksista, kuten liasta, värähtelystä, väärinkäytöstä, ääripään lämpötiloista ja muista vastaavanlaisista. Toisin sanottuna laitteisto alkaa kulua. Vikataajuusfunktion käyrän kurvien jyrkkyydet vaihtelevat eri laitteistojen suhteen, mutta lähes kaikilla tämänkaltainen ilmiö esiintyy.



KUVIO 5. Laitteiston vikataajuus suhteessa sen elinkaareen (Pressman 2001, 7)

Ohjelmistolla ei vastaavaa ympäristöstä johtuvaa kulumista esiinny. Siksi ohjelmiston vikataajuusfunktiossa (KUVIO 6) onkin esitetty ideaalinen käyrä, jota vikojen jakaantuminen tulisi elinkaaren aikana noudattaa. Pressman (2001, 7-8) toteaa, että todellisuudessa ohjelmisto ei kulu vaan se heikentyy. Tämän takia ohjelmistoa joudutaan tietyin väliajoin päivittämään käyttäjien uusia vaatimuksia vastaamaan. Pressman kuvailee ohjelmiston elinkaaren ja vikataajuuden suhdetta seuraavasti. Löytämättömät viat aiheuttavat korkean vikataajuuden ohjelmiston elinkaaren alussa. Kuitenkin nämä saadaan korjattua (ihanteellisesti, aiheuttamatta uusia vikoja) ja käyrä tasoittuu kuvion mukaiseksi. Ohjelmiston elinkaaren aikana sitä joudutaan päivittämään. Kun päivityksiä tehdään, syntyy todennäköisesti uusia vikoja, jotka aiheuttavat vikataajuuden piikkimäisen kasvun. Ennen kuin vikataajuus palaa entiselle tasolle, joudutaan ohjelmisto uudelleen päivittämään, aiheuttaen jälleen piikin. Vähitellen minimivikataajuustaso alkaa nousta - ohjelmisto heikentyy päivitysten takia. Heikentymistä kuvaa kuviossa reaalinen vikataajuuskäyrä.

Kuvioita tukee hyvin Musan, Ianninon ja Okumoton (1987, 7) toteamus, että vikojen päälähde ohjelmistoissa on suunnitteluvirheissä, kun laitteistovikojen pääsyynä on ollut fyysinen heikentyminen.



KUVIO 6. Ohjelmiston vikataajuus suhteessa sen elinkaareen (Pressman 2001, 8)

Ohjelmisto pääsee yleensä viallisempana markkinoille kuin laitteisto (Ireson 1995, 22.8). Syynä tähän on Musan, Ianninon ja Okumoton (1987, 7) mukaan ohjelmiston luotettavuuden ominaisuus pyrkiä muuttumaan jatkuvasti testauksen aikana. Tämä tapahtuu heidän mielestään joko uusia ongelmia löydettäessä, uutta koodia kirjoitettaessa tai kun korjaustoimenpide ongelmille tehdään. Musan ym. mukaan laitteiston luotettavuus voi muuttua tietyissä aikajaksoissa, kuten alkuvaiheen vanhentamisessa (*burn-in*) tai elinkaaren loppupuolella. O´Connor (1995, 7) tarkentaa, että vanhentamisessa laitetta käytetään jo tehtaalla äärimmäisissä olosuhteissa tietty aikajakso. Näin saavutetaan pitkäaikainen alhaisen vikataajuuden aikajakso, joka tulee laitteen tilaajan käyttöön. Laitteistolla on paljon suurempi pyrkimys kuin ohjelmistolla pysyvää tasoa kohti (Musa, Iannino & Okumoto 1987, 7). Iresonin (1995, 22.7) mukaan ohjelmisto on 40 - 50 kertaa alttiimpaa virheille kuin laitteisto. Tosin kaikkia ohjelmiston virheitä ei pidetä fataaleina, koska ne eivät häiritse prosessia ajamalla koko järjestelmää alas.

Kun ohjelmistovika esiintyy, se esiintyy kaikissa ohjelmiston kopioissa, ja jos se aiheuttaa vian tietyssä olosuhteissa, niin vika aiheutuu aina näissä olosuhteissa. Siksi

ohjelmistovirheet voivat olla äärimmäisen vakavia. (O´Connor 1995, 249) Laitteiston puolella vika koskee aina vain yhtä laitetta, eikä vika esiinny pakosti samoissa olosuhteissa muissa saman sarjan laitteissa. Iresonin (1995, 22.9) mukaan laitteiston luotettavuus voi olla teoriassa ennustettavissa, jos tunnetaan suunnittelu, käyttö ja ympäristön tuomat rasitustekijät. Vastaavasti hän näkee, ettei ohjelmiston luotettavuutta voida ennustaa minkään fyysisten tekijöiden pohjalta, koska se riippuu täysin inhimillisistä tekijöistä suunnittelussa. Iresonin mielestä on kuitenkin joitain apriorisia lähestymistapoja olemassa, jotka pohjautuvat käytettyyn kehittämisprosessiin ja koodin laajuuteen.

4.2 Kustannukset

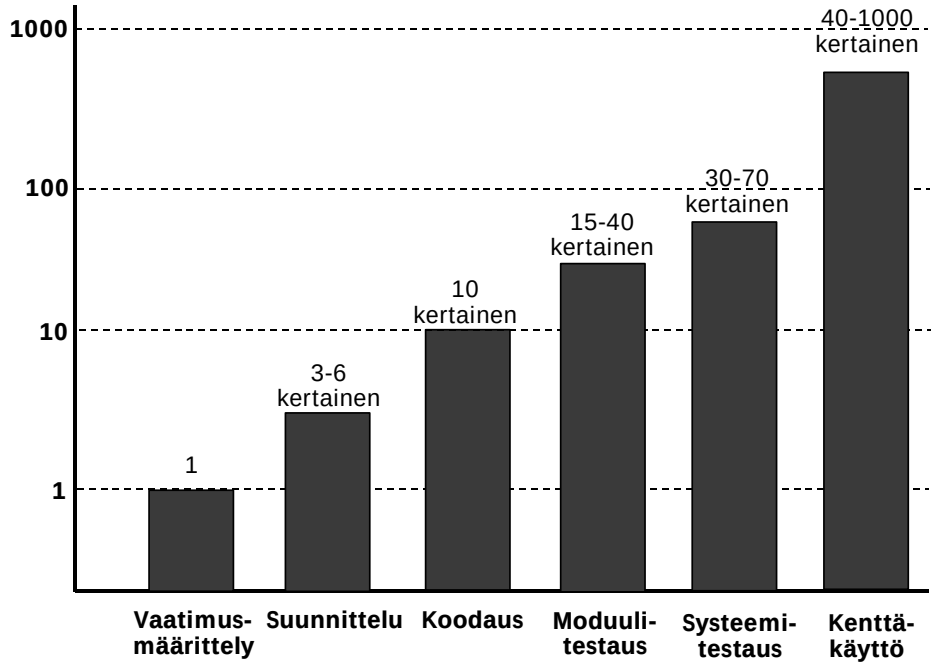
Ohjelmistovian ilmaantuessa kentällä ovat korjauskustannukset suuret. Musan ym. (1987, 191) mukaan vian sattuessa korjausprosessi sisältää seuraavat toimenpiteet:

1. Palvelupuhelun soittaminen.
2. Vioittuneen ohjelman lopettaminen oikealla tavalla.
3. Relevantin tiedon rekonstruointi.
4. Ohjelman uudelleen alustaminen ja käynnistäminen.
5. Olosuhteiden, joissa virhe esiintyi, dokumentointi.

Näihin toimenpiteisiin asiakas ei yksin kykene. Siksi hän joutuu aina ottamaan yhteyttä huoltoon. Kuten aiemmin todettiin, jos virhe esiintyy ohjelmistossa, se esiintyy kaikissa sen kopiossa. Virheen löytymisen myötä kaikkiin ohjelmiston kopioihin joudutaan tekemään siksi päivitys. Tämä on kallis toimenpide yritykselle. Musan ym. (1987, 4) mukaan virheistä syntyneet vahingot sisältävät välittömien kustannusten lisäksi tuotteen vahingonkorvausvastuun tuomat riskit ja mahdollisen vaurion yrityksen maineelle. Viimeisimmällä voi olla heidän mielestä dramaattinen vaikutus yrityksen markkinaosuuteen ja tuottavuuteen. Laitteiston suhteen tilanne on helpompi. Kun kyseessä on fyysinen tuote, voi asiakas vian ilmaantuessa korjata sen esimerkiksi varaosien avulla. Tässä tulee ilmi edellisessä kappaleessa mainittu laitteiston ja ohjelmiston fyysinen ero – ohjelmisto on yleensä ladattu laitteiston sisään. Siksi

ohjelmiston huoltaminen on asiakkaalle hyvin vaikeaa ja pienenkin virheen sattuessa joudutaan turvautumaan huoltoon. Tämä nostaa ohjelmiston virhekustannuksia.

Oheisesta kuviosta (KUVIO 7) käy ilmi kuinka suhteelliset kustannukset löytää ja korjata ohjelmistovirhe nousevat dramaattisesti sen mukaa, mitä myöhemmässä ohjelmiston elinkaaren ajankohdassa se esiintyy. Esimerkiksi jos virhe huomataan jo vaatimusmäärittelyn aikana, tulee sen korjaaminen maksamaan yritykselle 1000€, mutta jos samainen virhe pääsee kentälle asti, voi sen korjaaminen maksaa jopa 1 000 000€.



KUVIO 7. Virheen korjaamisesta aiheutuvat suhteelliset kustannukset ohjelmiston elinkaaren eri vaiheissa (Pressman 2001, 198)

Kuten kuviosta näkyy, virheiden löytämiseen kannattaa panostaa ohjelmiston elinkaaren alussa. Pressmanin (2001, 198) mukaan laatukustannukset voivat syödä pahasti ohjelmistoprojektin budjettia, mutta näin varaudutaan pahimmalta. Testaus on hänen mielestään välttämätöntä, vaikka se on myös todella kallis tapa löytää virheet. Pressman kehottaa kuluttamaan aikaa kehittämisprosessin alussa virheiden löytämiseen. Näin voidaan vähentää testaus- ja virheenpoistokustannuksia.

Laatukustannukset on yleisesti jaettu kolmeen eri kategoriaan: ennaltaehkäisy-, arviointi- ja virhekustannuksiin (O´Connor 1995, 353; Jones 1996, 379; Pressman 2001, 197). Ennaltaehkäisykustannukset liittyvät toimiin, jotka pyrkivät ehkäisemään virheiden esiintymistä. Jonesin (1996, 379) mukaan ohjelmiston parissa ennaltaehkäisy käsittää metodit, joilla yksinkertaistetaan kompleksisuutta ja taipumusta tehdä inhimillisiä virheitä. Esimerkkejä ennaltaehkäisykustannuksista ovat henkilökunnan koulutus uusimpiin suunnittelu- ja koodaustekniikoihin. Arviointikustannukset sisältävät Jonesin (1996, 380) mukaan henkilöstön suorittaman tuotteen verifioinnin ja validoinnin. Tämä tapahtuu hänen mielestään tarkastusten, katselmointien, mittausten, laadun tarkkailun ja erinäisten testausten avulla. Virhekustannukset syntyvät julkaisun jälkeen esiintyvissä virhekorjauksissa. Tällaisia ovat Jonesin mukaan esimerkiksi asiakaspalvelu, ylläpito, takuuhuolto ja joissain tapauksissa maksuvelvollisuus vahingoissa sekä oikeudenkäyntikustannuksissa.

4.3 Pareto-analyysi

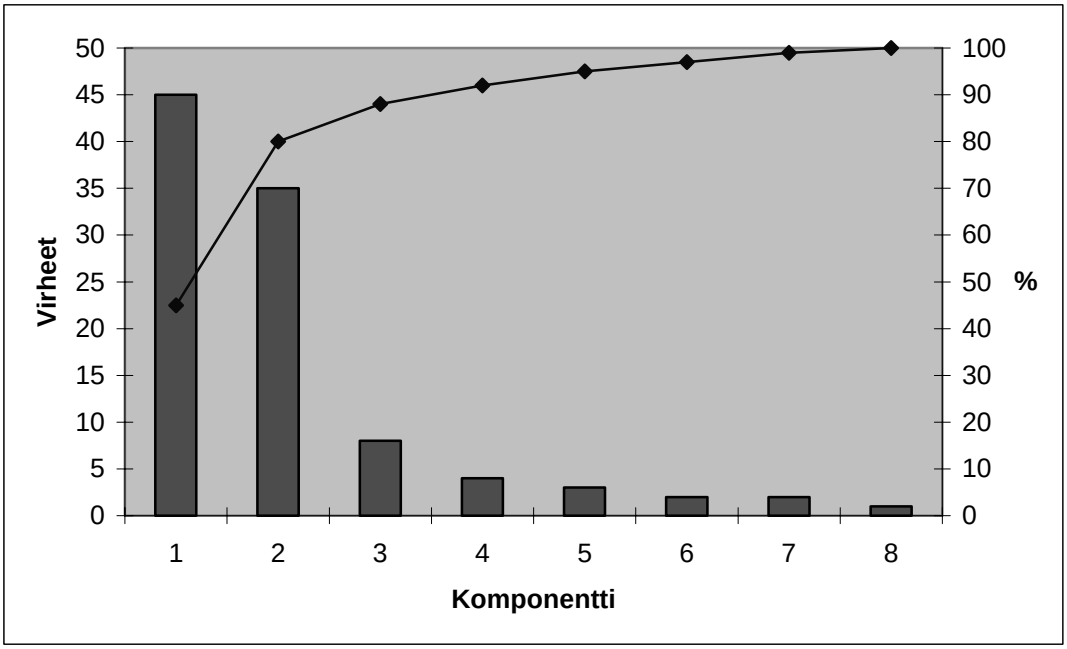
Luotettavuuden parantamisen yksi yksinkertaisimmista apuvälineistä on Pareto-analyysi. Se on graafinen työkalu, jota käytetään ongelmien ratkaisuun. Pareto-analyysi pohjautuu oletukseen, että suuri osa tuotteen virheistä aiheutuu pienestä määrästä syitä (ainakin Juran 1988, 22.19; O´Connor 1995, 286; Vonderembse & White 1996, 437; Lecklin 1999, 193; Breyfogle 1999, 37). Missä tahansa ympäristössä on rajaton määrä ongelmia ratkaistavaksi. Ongelmana on, ettei ole varaa ratkaista kaikkia. Koska ei ole järkevää korjata ongelmaa itseään, vaan ongelman aiheuttaja, täytyy ongelmaa tarkentaa aiheuttajan löytämiseksi. Ohjelmistossa virheiden aiheuttajaksi tarkentuvat moduulit tai komponentit. Pareto-periaatteen mukaan oletetaan, että pieni määrä ohjelmistomoduuleita todennäköisesti aiheuttaa suurimman osan ohjelmistovirheistä. Tätä kutsutaan usein ”20-80 säännöksi” merkiten, että 20 prosenttia ohjelmistomoduuleista aiheuttaa 80 prosenttia kaikista vioista (Fenton & Ohlsson 2000, 800).

Breyfogle (1999, 37) on jakanut Pareto-kaavion rakentamisen seuraaviin vaiheisiin:

1. Valitse ongelma ja ongelman osa-alueet diagrammille.

2. Määritä ajanjakso (esim. viikko, päivä, vuoro).
3. Laske jokaisen osa-alueen esiintymismäärä.
4. Aseta osa-alueet esiintymismäärän mukaiseen järjestykseen.
5. Piirrä osa-alueet esiintymismäärän mukaisesti laskevaan järjestykseen vasemmalta oikealle.

Seuraavassa on esimerkki Pareto-kaavion muodostamisesta Breyfoglen ohjeiden mukaisesti. Ongelmaksi on valittu ohjelmiston testaus ja testauksessa esiintyvät ohjelmistovirheet. Virhemäärää tarkastellaan komponenttikohtaisesti. Ohjelmisto koostuu kahdeksasta eri komponentista, ja näissä esiintyviä virheitä tarkkaillaan neljän viikon, testausvaiheen, verran. Kahden viikon kuluttua virheet lasketaan komponenttikohtaisesti yhteen ja piirretään Pareto-kaavio tulosten perusteella (KUVIO 8). Kaaviosta pystymme päättelemään, että ohjelmiston virheherkimmät komponentit ovat 1 ja 2. Ne aiheuttavat yhteensä 80 prosenttia kaikista ohjelmistovirheistä. Jos ohjelmistoprojekti panostaa näiden komponenttien virheiden poistamiseen, laskee koko ohjelmiston virhetaajuus radikaalisti.



KUVIO 8. Pareto-analyysi - 80 prosenttia vioista aiheutuu 20 prosentista komponentteja

Fentonin ja Ohlssonin (2000) mukaan pareto-säännön virheolettamus on johtanut monet ammatinharjoittajat etsimään menetelmiä virhealttiiden moduuleiden ennustamiseen

mahdollisimman aikaisin kehitys- ja testausvaiheessa. Nämä menetelmät ovat Fentonin ja Ohlssonin mielestä jakaantuneet kahteen eri kategoriaan:

1. Aikaisen virhedatan käyttö ennustamaan jälkivirheiden määrää.
2. Tuotemetriikoiden käyttö ennustamaan vikoja ja virheitä.

Aikaisemman virhedatan käyttö perustuu olettamukseen, että testauksessa paljon virheitä sisältäneet moduulit tulevat olemaan myös testauksen jälkeen virhealttiita. Fenton ja Ohlsson kumminkin kritisoivat teoriaa. Heidän tutkimuksen mukaan kaikkein kompleksisimmat moduulit tuntuvat olevan kaikkein virhealttiimpia ennen testausta, mutta testauksen jälkeen niissä ei ole juuri yhtään virhettä. Kaikkein virhealttiimmat moduulit testauksen jälkeen näyttävät olevan vähemmän kompleksisia moduuleita. Fentonin ja Ohlssonin mielestä syynä tähän voi olla testauksen jakautuminen moduuleittain – moduulit, jotka vaikuttavat olevan kompleksisia, testaan paljon tarkemmin kuin yksinkertaisemmat moduulit.

Laajasta uskomuksesta huolimatta Pareto-analyysista löytyy vain vähän julkaistuja todisteita sen paikkansa pitävyydestä, ja se mitä löytyy, tukee toisenlaisia lukemia (alkaen heikosta 20-50:stä vahvaan 10-80 tyyppiseen sääntöön) (Fenton & Ohlsson 2000, 800). Kuitenkin analyysin avulla pystytään poistamaan tehokkaasti ohjelmistovirheitä, mutta kuten kohdassa 4.1 todettiin, yksikin ohjelmistovirhe voi olla fataali ja kaataa koko ohjelman.

4.4 Virheiden ennustaminen

Edellä olevissa kohdissa todettiin ohjelmiston luotettavuuden ennustamisen olevan todella vaikeaa. Kun ohjelmisto on julkaistu, saavat ohjelmistotoimittajat palautetta käyttäjiltä ohjelmiston luotettavuudesta. Kuitenkin tällöin se on liian myöhäistä. Ohjelmistotoimittajien tulee tietää, onko ohjelmisto tarpeeksi luotettava, ennen kuin se toimitetaan asiakkaille. (Wood 1996) Fentonin ja Neilin (1999) mukaan organisaatiot kyselevät vieläkin, kuinka ne voivat ennustaa kehittämiensä ohjelmistojen luotettavuuden ennen kuin ohjelmistoa on käytetty, vaikka vastausta tähän kysymykseen on tutkittu jo yli 30 vuotta. Tämä on heidän mielestään aihealue, joka saattaa nousta kaikkein vaikeimmaksi ongelmaksi ohjelmistoteollisuudessa. Kanin

(2003, 211) mukaan aihealue on ollut jo viimeaikoina yksi ohjelmistoteollisuuden aktiivisimmista. Hänen havaintojen perusteella yli sata mallia erilaisine olettamuksineen, kykyineen ja rajoitteineen on esitelty tieteellisissä julkaisuissa ja ohjelmistokonferensseissa. Valitettavasti Kanin mukaan monia malleja ei ole edes testattu oikeanlaisessa ympäristössä oikealla datalla ja vielä vähemmän malleja on tällä hetkellä käytössä. Monia erilaisia luotettavuusmalleja on siis kehitetty, mutta kuten Brocklehurst ja Littlewood (1992) toteavat, ei yhtäkään voida yleisesti suositella. Heidän mielestään luotettavuusmallien tarkkuus vaihtelee mallien välillä dramaattisesti. Jotkut mallit antavat joskus hyviä tuloksia, kun taas jotkut ovat yleisesti kammottavia ja minkään mallin ei voida olettaa olevan aina tarkka. Tämä huono tilanne on varmasti ollut pääsyynä siihen, etteivät luotettavuusmallit ole yleistyneet. Jos käyttäjät ovat kokeilleet jotain mallia huonoin lopputuloksin, eivät he tämän jälkeen ole halukkaita kokeilemaan uusimpia tekniikoitakaan. Mitä me sitten hyötyisimme tarkoista ohjelmiston luotettavuusmalleista?

Woodin (1996) mielestä tieto jäännösvirheiden määrästä auttaa päättämään, onko koodi jo sopivaa asiakkaan käyttöön vai pitääkö testausta vielä jatkaa ja kuinka kauan. Lisäksi tieto tarjoaa hänen mukaansa estimaatin häiriömäärästä, jonka asiakas tulee kohtaamaan kun ohjelmistoa käyttää. Tämä estimaatti auttaa määrittelemään oikeantasoisen ylläpidon, mitä tarvitaan vikojen korjaamisen, kun ohjelmisto on toimitettu. Kanin (1991) mukaan hyvä luotettavuusmalli on objektiivinen toteamus ohjelmistosysteemin koodin laadusta. Kun laatu saadaan korkeaksi, voi organisaatio laskea kustannuksia, koska säästöjä syntyy pieleen menneiden tuotteiden, uudelleen tehtävän työn ja asiakkaiden takuuvaatimusten vähenemisellä.

Kirjallisuudessa on paljon tutkimuksia, artikkeleita ja raportteja, jotka puoltavat joitain malleja, metriikoita ja ratkaisuita. Yleisesti ongelmaa on pyritty ratkaisemaan kolmesta eri perspektiivistä (Neil & Fenton 1996):

1. Ennustamalla järjestelmässä olevien virheiden määrää ohjelmiston koko- ja kompleksisuusmetriikoiden avulla.
2. Päättelemällä virheiden määrä testausinformaation avulla.
3. Arvioimalla suunnittelun tai prosessin kypsyyden vaikutusta virheiden lukumäärään.

Koko- ja kompleksisuusmetriikat perustuvat olettamukseen, että kehittämisprosessin aikana löydetyt virheet muodostavat lineaarisen mallin, josta pystytään ennustamaan jäännösvirheiden määrä. Neil ja Fenton (1996, 2) kritisoivat voimakkaasti varsinkin kokometriikoihin perustuvia menetelmiä, sillä heidän mielestään jäännösvirheiden määrän ennustaminen pelkän koodirivimäärän perusteella on yhtä viisasta kuin ennustaa jonkun älykkyysosamäärää kengän numeron perusteella.

Testausinformaation käyttäminen jäännösvirheiden päättelyyn on lupaava näkökulma. Yhdeksi ongelmaksi paljastui kumminkin Fentonin ja Ohlssonin (2000) tutkimuksessa oletetun virhekäyttäytymisen paikkansapitämättömyys. Tutkimus paljasti, että testauksessa vähiten virhealttiit moduulit olivat itse käytössä kaikkein virhealttiimpia. Tutkijat arvioivat syyksi ilmiöön testauksessa ennalta kaikkein kompleksisimmiksi arvioitujen moduuleiden tarkemman testaamisen muihin verrattuna.

Neilin ja Fenton (1996) mielestä moni asiantuntija väittää prosessien laadun olevan paras ennustaja koko ohjelmiston laadulle. Tältä kannalta on kehitetty ohjelmiston luotettavuusmenetelmiä, jotka päättelevät luotettavuuden ohjelmiston suunnittelun tai itse kehittämisprosessin kypsyyden pohjalta. Neilin ja Fentonin mukaan yksinkertaisin prosessin laadun metriikka on 5-tasoinen järjestysasteikko CMM (*Capability Maturity Model*). Sen suuresta levinneisyydestä huolimatta ei heidän mielestään ole olemassa yhtään vakuuttavaa todistetta, että korkeamman kypsyyden omaavat organisaatiot valmistaisivat vähemmän jäännösvirheitä sisältäviä tuotteita kuin matalamman kypsyyden omaavat organisaatiot.

Luvussa esiteltiin ohjelmistovirheiden teoriaa. Ohjelmistovirheet voivat olla äärimmäisen vakavia, koska ne esiintyvät kaikissa ohjelmiston kopioissa. Virheistä aiheutuvat kustannukset voivat tämän vuoksi nousta erittäin suuriksi, varsinkin jos virhe havaitaan ohjelmiston elinkaaren myöhäisessä vaiheessa. Pareto-analyysin mukaan pieni määrä ohjelmistokomponentteja todennäköisesti aiheuttaa suurimman osan ohjelmistovirheistä. Analyysin periaatteeseen perustuu osa luotettavuusmalleista. Erilaisia luotettavuusmalleja on kehitetty pitkään, mutta yhtään yleisesti tarkkaa ei ole vielä kehitetty.

5 TUTKIMUSYMPÄRISTÖ JA -MENETELMÄ

Tässä luvussa käydään läpi tutkimusympäristö ja -menetelmä. Tutkimuksen kohdeyrityksenä toimii Nokia ja tarkempana kohteen yksikkö, joka keskittyy Internet Protocol -konvergenssin kehittämiseen. Yksikkö toimii Nokia Mobile Phones (*NMP*) toimialaryhmän alla. Kohdeyritystä ja -yksikköä kuvaillaan hieman luvun alkuun, että saataisiin kuva, minkälaiseen toimintaympäristöön ohjelmiston luotettavuusmallia tullaan integroimaan.

Tutkimusympäristön tarkastelun jälkeen kuvataan tutkimusmenetelmä. Ensin käsitellään tutkimuksen tiedonkeruumenetelmät ja tämän jälkeen luodaan viitekehys luotettavuusmallien vertailuun. Viitekehyksen luonnin pohjalla ovat aikaisemmat vastaavanlaiset viitekehykset sekä alan kirjallisuuden painotukset. Lisäksi kohdeyksikkö tuo omia rajoitteitaan viitekehykseen.

5.1 Tutkimusympäristö

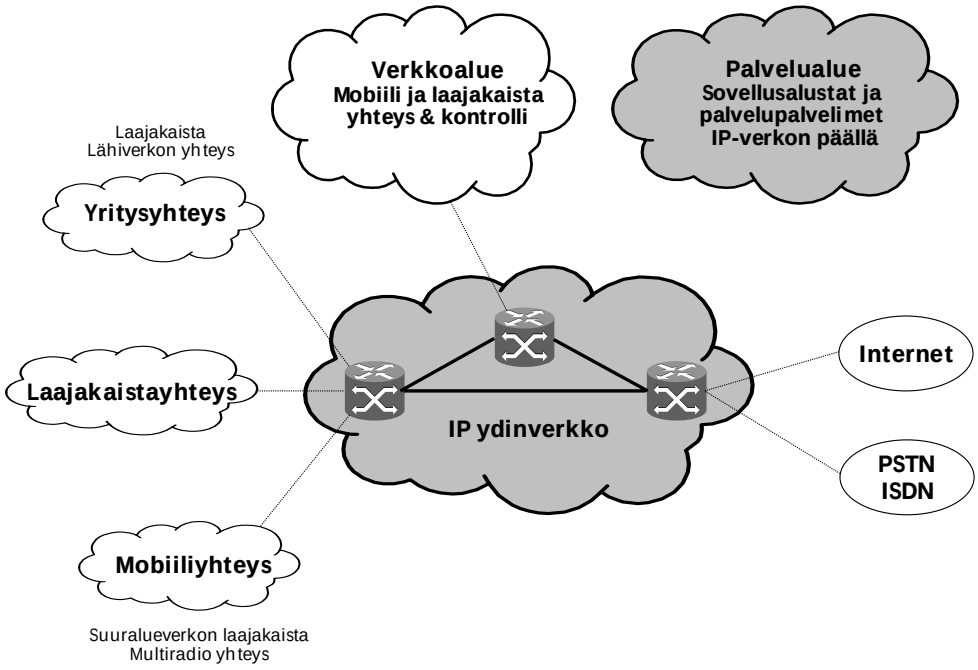
Nokia on globaali yritys, jonka päämaja sijaitsee Suomessa, Espoossa. Nykyään yritys keskittyy ydinosaamiseensa telealaan, toimittaen matkapuhelimia ja matkapuhelinverkkoja. Jorma Ollila, nykyinen Nokian pääjohtaja, kiteyttää Nokian nopean kansainvälisen kasvun Steinbockin (2001, 1) teoksessa seuraavasti: ”1980-luvulle saakka Nokia oli suomalainen yritys. 1980-luvulla Nokia oli pohjoismaalainen ja 1990-luvun alussa eurooppalainen yritys. Nyt me olemme globaalinen yritys”.

Nokiaan kuuluu kaksi toimialaryhmää: Nokia Mobile Phones (*NMP*) ja Nokia Networks sekä erillinen Nokia Ventures Organization ja yhtymän sisäinen tutkimusyksikkö Nokia Research Center (Nokia 2003d). NMP on maailman suurin matkapuhelinvalmistaja, jonka tuotevalikoima kattaa kaikki asiakasryhmät ja matkapuhelinstandardit. Henkilöstön määrä koko konsernissa oli vuonna 2002 noin 52 000, josta noin 26 000 eli puolet työskenteli Nokia Mobile Phones:issa (NMP) (Nokia 2003d). Nokian liikevaihto vuonna 2002 oli n. 30 miljardia euroa, josta n. 77 prosenttia oli NMP:n ansiota. NMP:n

suuren tuottavuuden taustalla oli yrityksen onnistuminen 38 prosentin markkinaosuuden saavuttamisessa koko maailman matkapuhelinmyynnistä. (Nokia 2003a) Nykyään useampi kuin joka kolmas maailmassa myyty matkapuhelin on siis Nokian valmistama.

Nokian liiketoimintastrategian yhtenä kohtana on olla aktiivinen Internet Protokolla (*IP*) konvergenssissa (Nokia 2003a). Tähän osa-alueeseen yhtiössä panostaa IP konvergenssi -yksikkö, joka toimii tämän tutkimuksen kohdeyksikkönä. Seuraavassa yksikön toimintaympäristön lyhyt esittely.

IP-konvergenssi tarkoittaa yksinkertaisesti Internetin ja mobiliteetin konvergenssia. IP-konvergenssin verkkoarkkitehtuuri on esitetty ohessa (KUVIO 9). Arkkitehtuurin pohjana on ydin IP-verkko, jota käyttävät Internet, yleinen puhelinverkko (PSTN), digitaalinen monipalveluverkko (ISDN) sekä mobiili-, yritys- ja laajakaistayhteydet. IP-ydinverkon päälle voidaan rakentaa suojattuja verkko- ja palvelualueita (*Domain*).

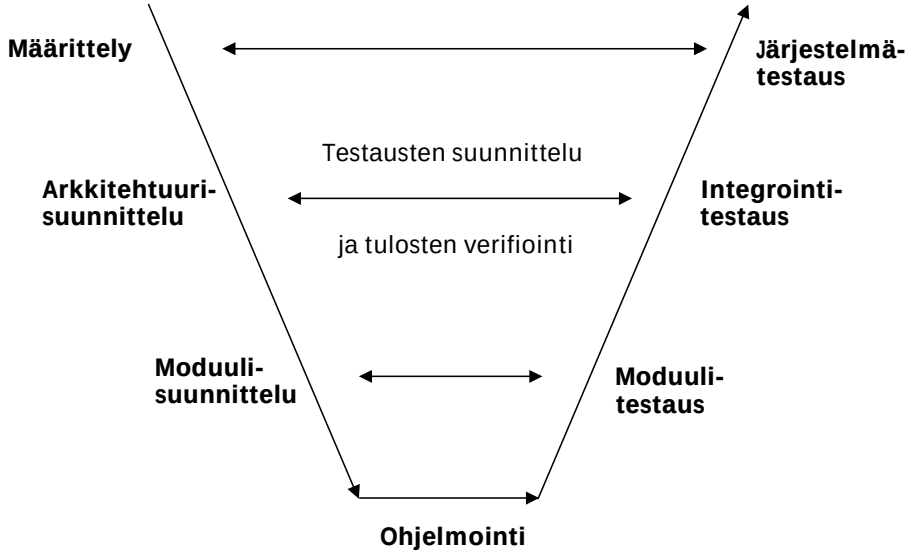


KUVIO 9. Mobiili IP:n verkkoarkkitehtuuri (Kari & Kilpeläinen 2001, 289)

IP-konvergenssi luo uuden markkina-alueen, jonka parissa kohdeyksikkö toimii. Yksikkö vilkastuttaa markkinakasvua ja sovellusten kehittämistä, takaa NMP:n paikan

alan arvoketjussa ja valmistelee sekä testaa tulevia markkinoita. Karin ja Kilpeläisen (2001, 291) mukaan Nokian kannalta IP-konvergenssi luo vakiintuneiden palveluparadigmojen päälle, esim. SMS ja MMS, uusia toiminnallisuuksia, sisältöä ja palvelutyyppejä. Kohdeyksikkö tuottaa Symbian-arkkitehtuurin ohjelmistoja. Tuotettavat ohjelmistot ovat tosiaikasovelluksia, jotka pohjautuvat SIP (*Session Initiation Protocol*) -protokollaan.

Ohjelmistokehitys perustuu avoimen lähdekoodin periaatteeseen, ohjelmiston osat toteutetaan joko yrityksen sisäisillä komponenttitehtailla tai alihankkijoilla. Kohdeyksikkö integroi muualla toteutetut ohjelmiston osat ja suorittaa pääosan testauksesta. Testaus noudattaa pitkälti V-mallia. V-malli on kuvattu ohessa (KUVIO 10).



KUVIO 10. Testauksen V-malli (Haikala & Märijärvi 1996, 234)

Kohdeyksikkö toteuttaa ohjelmistokehityksessään V-mallin vasemman puolen eli suunnittelun. Ohjelmointi suoritetaan joko komponenttitehtailla tai alihankkijoilla. Moduulitestaus suoritetaan toteuttajan toimesta eli yksikön ulkopuolella. Integrointi- ja järjestelmätestaus jäävät yksikön vastuulle. Lisäksi yksikkö suorittaa suuren määrän muita testauksia ohjelmistolle. Haikala ja Märijärvi (1996, 235) muistuttavat, että mitä korkeammalla V-mallin testaustasolla ollaan, sitä kalliimmaksi virheiden korjaus tulee.

Tämänlainen ohjelmistokehitys vaatii yksikön ja alihankkijoiden välillä hyvää kommunikointia ja etenkin luottamusta, että moduulitestaus hoidetaan riittävällä tarkkuudella. Fentonin ja Ohlssonin (2000, 811) mukaan tuntuu väistämättömältä, että pieni määrä järjestelmän moduuleita aiheuttaa suuren määrän julkistusta edeltävistä virheistä ja että pieni määrä moduuleita tulee sisältämään suuren osan julkistuksen jälkeisistä vioista. Tämän väittämän mukaisesti ohjelmistokehityksen kaikkein tärkein testausvaihe suoritetaan yksikön ulkopuolella.

Tämä tutkimus suoritetaan IP-konvergenssiyksikön toimesta ja sen tarkoituksena on kehittää yksikölle soveltuva malli, jolla voidaan verifioida yksikön kehittämien matkapuhelinsovellusten luotettavuus. Tämänkaltaisella mallilla yksikkö voi todentaa kehittämiensä sovellusten laadukkuuden ja luotettavuuden ennen sovellusten luovuttamista asiakkaille. Näin varmistetaan ettei asiakastuotteille päädy fataaleja vikoja, jotka nostaisivat huoltokustannuksia ja mahdollisesti vahingoittaisivat yrityksen mainetta. Yksikkö on vielä nuori ja sen alaisuudessa ei ole vielä yhtään ohjelmistoprojektia valmistunut. Näin historiatietoja yksikölle tyypillisistä ohjelmistoprojekteista ei ole saatavana. Historiatietojen avulla voitaisiin potentiaalisten luotettavuusmallien toimivuutta tarkastella, vertaamalla niiden antamia tuloksia oikeisiin toteutuneisiin.

5.2 Tutkimusmenetelmä

Tutkimusmenetelmä on konstruktiivinen, mutta siinä on myös käsitteellis-teoreettisia piirteitä. Käsitteellis-teoreettisia piirteitä tuo tutkimuksessa suoritettava kirjallisuustutkimus ohjelmiston luotettavuusmalleista. Tarkoituksena on selvittää aihealueen tämän hetkinen tila ja se mitkä mallit ovat suosittuja maailmalla sekä kohdeyrityksen sisällä. Tiedonkeruulähteinä ovat alan tieteellinen kirjallisuus ja kohdeyrityksen kartoitus. Tutkimuksen konstruktiivisessa osassa kehitetään kohdeyksikölle sopiva luotettavuusmalli käyttäen hyväksi eri menetelmien teorioita. Tutkimukseen on valittu alan kirjallisuudesta kolme mallia lähempään tarkasteluun ja niitä analysoidaan kohdassa 5.2.2 kehitetyn viitekehyksen avulla. Analyysien avulla

saadaan selville jokaisen mallin heikkoudet ja vahvuudet, mitkä toimivat pohjana kohdeyksikön jatkotoimenpiteille.

Tässä luvussa käydään ensin läpi tutkimuksen tiedonkeruumenetelmät ja sitten tutkimusta varten kehitetyn viitekehyksen tieteellinen pohja.

5.2.1 Tiedonkeruumenetelmät

Hirsjärven, Remeksen ja Sajavaaran (1997, 181) mukaan tutkimus usein alkaa siitä, että tutkija yrittää kartoittaa kentän, jossa hän toimii. Tämä tutkimus käynnistyi myös tutkimusalueen kartoittamisella. Tärkeänä osana on ollut tutkijan osallistuminen kohdeyrityksen ja täten tutkimusympäristön jokapäiväiseen työntekoon. Kokouksissa ja epävirallisissa keskusteluissa saatu informaatio on ohjannut tutkimuksen kulkua paljon. Muuten tiedonkeruu tätä tutkimusta varten on tapahtunut pääosin alan kirjallisuuteen tutustumalla. Lisäksi tietoa on kerätty epävirallisissa keskusteluissa, haastatteluissa, puhelinneuvotteluissa ja sähköpostin välityksellä tapahtuvissa kyselyissä.

Pohjatietona tutkimukselle käytettiin laatujohtamisen, IP-konvergenssimaailman, ohjelmistovirheiden ja -luotettavuuden teoriaa. Tutkimuksessa primaariaineiston keruu tapahtui alan kirjallisuuteen tutustumalla ja kohdeyrityksen kartoittamisella. Oikeanlaisen primaariaineiston löytämiseksi käytettiin tutkimuksessa sekundaariaineistona Internetistä löytyviä tiivistelmäartikkeleita ja kohdeyrityksen intranetiä. Kohdeyrityksen kartoittaminen tapahtui pääsääntöisesti sähköpostikyselyiden avulla. Sähköposti oli ainoa looginen vaihtoehto organisaation globaalisuuden takia (ks. 5.1). Myös puhelinneuvotteluita ja haastatteluita käytettiin avuksi. Kohdeyrityksen kartoituksella haluttiin selvittää, minkälaisia laatua ja luotettavuutta parantavia menetelmiä yrityksessä on käytössä ja minkälaisia menetelmiä on tutkittu. Tiedonkeruumenetelmänä on käytetty kvalitatiivista aineiston keruuta ja etenkin kyllääntymistä (*saturation*). Tällä tarkoitetaan sitä, että tutkija alkaa kerätä aineistoa päättämättä etukäteen, miten monta tapausta hän tutkii. Hän voi aloittaa haastattelut ja jatkaa niitä niin kauan, kuin haastattelut tuovat tutkimusongelman kannalta uutta tietoa. Aineisto on riittävä, kun samat asiat alkavat kertautua haastatteluissa. (Hirsjärvi, Remes

& Sajavaara 1997, 181) Tämän tutkimuksen osalta haastattelut tai kyselyt liittyvät kohdeyrityksen kartoittamiseen, millaisia tapauksia eli laatua ja luotettavuutta parantavia menetelmiä on käytössä tai tutkittuna.

Tutkimuksen alussa tiedonkeruumenetelmänä käytettiin haastatteluita. Haastateltaviksi pyrittiin löytämään henkilöitä, jotka omaavat hyvät tiedot tutkimusalueesta. Lopulliseksi haastattelumääräksi kertyi neljä haastattelua, joissa haastateltavina oli yhteensä kuusi henkilöä. Puolet haastatteluista jouduttiin suorittamaan puhelinhaastatteluina pitkien välimatkojen takia. Haastatteluissa pyrittiin Järvisen ja Järvisen (2000, 153) ohjeiden mukaan selvittämään ilmiöalueen kattavuus ja hankkimaan aihealueesta tietoa. Ennen kaikkea oltiin kiinnostuneita haastateltavien kokemuksista ohjelmiston luotettavuutta ja laatua parantavista menetelmistä. Minkälaisia menetelmiä he ovat käyttäneet tai käyttävät sekä kuinka ne ovat toimineet? Sisältökysymysten lisäksi selvitettiin keitä muita tulisi haastatella. Haastattelut suoritettiin avoimina eli tutkimusteemojen ohjaamina (Järvinen & Järvinen 2000, 153).

Haastatteluiden pohjalta luotiin sähköpostikysely, joka lähetettiin kohdeyrityksen henkilöille, joilla saattoi olla tietämystä ohjelmiston laatua ja luotettavuutta parantavista menetelmistä. Valittuina henkilöinä oli haastatteluissa saadut uudet kontaktit. Kysely lähetettiin yhteensä neljälletoista henkilölle. Järvisen ja Järvisen (2000, 155-156) ohjeiden mukaan käytettiin avoimia kysymyksiä, joita tulee käyttää silloin, kun kysymysten kohteena oleva aihepiiri ei vielä ole jäsentynyt. Tällöin tutkija odottaa, että tutkittavat vastauksillaan ilmaisisivat käytössä olevia hahmottamistapoja. Sähköpostikyselyn runko löytyy liitteestä 1.

Vastauksia sähköpostikyselyyn saatiin seitsemältä henkilöltä, mutta niistä vain neljä oli valideja. Muut ohjasivat ottamaan yhteyttä uusiin henkilöihin, jotka lopulta eivät omanneet tietämystä tutkimusalueesta. Vastausprosentti, eli kuinka moni kyselylomakkeen saaneista todella palauttaa lomakkeen täytettynä, muodostaa kyselytutkimuksissa merkittävän luotettavuuskriteerin (Järvinen & Järvinen 2000, 162). Tässä tutkimuksessa luotettavuuskriteeri jää alhaiseksi, mutta silti kyselyn vastaukset

auttoivat tutkijaa hahmottamaan tutkimusaluetta ja tätä myötä ohjaamaan tutkimuksen kulkua.

Lopullinen aihealueesta tietäneiden haastateltujen ja kyselyyn osallistuneiden henkilöiden määrä tässä tutkimuksessa oli kymmenen. Heidän lisäksi tutkija sai paljon hyödyllistä tietoa aihealueesta epävirallisissa kokouksissa ja käytäväkeskusteluissa eli tutkimusympäristön jokapäiväisessä työnteossa. Haastatteluiden ja kyselyiden tuottamat vastaukset käsitellään kohdassa 6.1.

5.2.2 Viitekehyksen toteuttaminen

Lyun ja Nikoran (1992) mukaan ensimmäinen luotettavuusmalli kehitettiin vuonna 1972. Tähän päivään mennessä Kanin (2003, 211) havaintojen perusteella on kehitetty yli sata erilaista mallia. Kumminkaan yksikään näistä menetelmistä ei ole saavuttanut de facto -standardin asemaa, koska niiden tarkkuutta ei ole voitu yleisesti verifioida. Mallien välistä vertailua on kirjallisuudessa suoritettu vain vähän. Lyu ja Nikora (1992) kehittävät artikkelissaan viitekehyksen, jonka avulla voidaan vertailla kehitettyjä malleja. Tutkimus on suoritettu 1990-luvun alussa, jolloin matemaattiset tilastolliset vikamallit olivat suosiossa. Siksi siinä esitelty viitekehys keskittyy vahvasti matemaattisten suureiden vertailemiseen. Nykyään ovat kumminkin kausaaliset analyysit nostaneet suosiotaan ja ne poikkeavat huomattavasti tilastollisista vikamalleista (Fenton & Neil 1999). Mallien muuttuneiden ominaisuuksien takia Lyun ja Nikoran viitekehystä ei voida vain parantaa ja soveltaa, vaan täytyy kehittää kokonaan uusi viitekehys mallien vertailuun.

Tutkimuksen aihealuetta vaivaa yleinen nuoren tieteenalan dilemma. Luotettavuusmalleja on kehitetty runsaasti, mutta yhtäkään ei ole yleisesti hyväksytty. Lisäksi eri mallien vertailu on jäänyt vähäiseksi, koska tutkijat panostavat uusien ja olemassa olevien mallien kehittämiseen tavoitteenaan julkaista ensimmäinen de facto -standardi. Ainoa löytämäni aihealueen viitekehys, joka on kehitetty vertailemaan eri luotettavuusmalleja, on Lyun ja Nikoran (1992) käsialaa.

Lyu ja Nikora (1992) kuvaavat 1990-luvun alun luotettavuusmalleja sarjaksi tekniikoita, jotka pohjautuvat todennäköisyysteorioihin ja tilastollisiin analyyseihin. Siksi aikakauden luotettavuusmallit olivat erittäin matemaattisia. Lyun ja Nikoran viitekehys onkin osittain hyvin matemaattinen ja sen käyttämisen edellytyksenä on, että vertailtavia malleja on käytetty saman datan avulla ja tuloksia vertaillaan. Oheisessa taulukossa (TAULUKKO 1) on esitetty ja kuvattu viitekehyksen komponentit. Jokainen luotettavuusmalli pohjautuu johonkin perusolettamukseen. *Olettamuksen paikkansapitävyydellä* tarkastellaan mallin antamien tulosten yhteneväisyyttä olettamusten kanssa. Komponenteista tärkein on *menetelmän validiteetti* -komponentti. Sen avulla mitataan matemaattisesti mallin tarkkuutta, poikkeamia, tendenssiä ja häiriöitä ajetulla testidatalla. *Mittauksen helppoudella* tarkkaillaan, kuinka paljon parametreja malli tarvitsee ja kuinka helposti ne ovat mitattavissa. Lyun ja Nikoran mukaan helposti mitattavat parametrit eivät ainoastaan laske mittauskustannuksia, vaan auttavat helpommin tulkitsemaan luotettavuusmallia, jonka ansiosta voidaan tarjota palautetta ohjelmiston kehittäjille. *Kykenevyydellä* viitataan Lyun ja Nikoran mukaan mallin kykyyn estimoida ohjelmiston luotettavuuteen liittyviä suureita, kuten esimerkiksi tämänhetkinen luotettavuus, odotettu päivä jolloin asetetut luotettavuustavoitteet saavutetaan ja vaadittavat kustannukset tavoitteiden saavuttamiseksi. *Soveltuvuus* tarkastelee mallin toimivuutta eri toimintaympäristöissä ja erityyppisissä kehittämisprosesseissa. *Yksinkertaisuus* kuvaa luotettavuusmallin käytön vaikeutta. Varsinkin matemaattisille luotettavuusmalleille käytön helppous on tärkeä. Mallia pitää ymmärtää ja osata tulkita ilman suurta perehtymistä siihen. Mallin syötteenä oleva data voi sisältää virheitä, jotka johtuvat esimerkiksi inhimillisestä virheestä. *Epäherkkyys häiriötekijöille* -komponentti ilmaisee, kuinka luotettavuusmallin tarkkuus kärsii, jos syöttödata on virheellistä.

Viitekehyksen käytössä panostetaan varsinkin mallin validiteetin mittaamiseen. Tämä tapahtuu matemaattisesti, mikä soveltuu hyvin tilastollisiin vikamalleihin. Uusimpiin luotettavuusmalleihin matemaattisia kaavoja on vaikea soveltaa ja tämä laskee viitekehyksen käyttöarvoa. 1990-luvun alun mallien ongelmana on liika painotus luotettavuuden ja koodin koon korrelointiin, jota Neil ja Fenton (1996) kritisoivat voimakkaasti.

TAULUKKO 1. Lyun ja Nikoran viitekehyksen komponentit (Lyu & Nikora 1992)

Komponentti	*Kuvaus*
Olettamusten paikkansa pitävyys	**Olettamukset, joihin malli perustuu tulee olla lähellä saatuja arvoja**
Mallin validiteetti	**Mallin toimivuus tarkkuuden, poikkeamien, tendenssin ja häiriöiden kannalta**
Mittauksen helppous	**Tarvittavien parametrien määrä ja niiden arviointi**
Kykenevyys	**Mitä kaikkea mallin avulla voidaan mitata**
Soveltuvuus	**Toimivuus eri kehittämis- ja toimintaympäristöissä ja elinkaaren vaiheissa**
Yksinkertaisuus	**Suotavaa kaikkein matemaattisimmille malleille**
Epäherkkyys häiriötekijöille	**Mallin tulee toimia pienistä häiriöistä huolimatta**

Musa, Iannino ja Okumoto (1987, 19-20) listaavat hyvän ohjelmiston luotettavuusmallin tärkeitä ominaisuuksia:

1. Antaa hyvän ennusteen tulevaisuuden virheiden käyttäytymisestä.
2. Laskee hyödyllisiä suureita.
3. Malli on yksinkertainen.
4. Se on laajasti soveltuva.
5. Malli perustuu järkeviin olettamuksiin.

Lista on pääpiirteittäin samankaltainen kuin aiemmin esitelty Lyun ja Nikoran viitekehys. Viitekehys tarkentaa listausta vielä kahdella komponentilla. Ainoana merkittävänä erona on listan kohta (1): antaa hyvän ennusteen tulevaisuuden virheiden käyttäytymisestä. Viitekehyksestä puuttuu kokonaan tulevaisuuden estimointi, mikä on tärkeä ominaisuus ohjelmiston kehittämisprosessin hallinnointiin. Estimoinnin avulla voidaan reagoida nopeasti, jos prosessi on ajautumassa väärään suuntaan.

Fentonin ja Neilin (200, 357) mukaan ohjelmistometriikoiden tulevaisuus lepää suhteellisen yksinkertaisten olemassa olevien metriikoiden varassa. Nämä tukevat johdon päätöksentekotyökaluja, jotka yhdistävät eri puolia ohjelmiston kehittämis- ja testausprosessista, luoden johdolle erilaisia ennusteita, olettamuksia ja vaihtoehtoja ohjelmiston elinkaaren aikana. Myös ohjelmiston luotettavuusmallilla voidaan saada vastauksia johdon kysymyksiin kehittämisprosessin tilasta ja aikataulusta. Kolme tämänlaista kysymystä ovat (Musa, Iannino & Okumoto 1987, 23):

1. Onko tämä ohjelmisto valmis julkaisuun?
2. Milloin se tulee olemaan valmis julkaisuun?
3. Pitäisikö meidän ottaa käyttöön edellinen versio nykyisen sijaan?

Vastaus ensimmäiseen kysymykseen saadaan, jos malli laskee ohjelmiston tämänhetkisen luotettavuuden. Valmis julkaisuun -kysymykseen vastaus voidaan päätellä, jos malli ennustaa tulevaisuuden virhekäyttäytymistä. Voimme päättää, tehdäänkö regressio edelliseen versioon, jos nykyisen version häiriötiheys on liian suuri. Jos uusi versio ei vastaa asetettuja häiriöintensiteettitavoitteita ja vanhempi vastaa, häiriöintensiteettieron ollessa versioiden välillä huomattava, on regressio vaivan arvoista (Musa, Iannino & Okumoto 1987, 26).

Tutkimuksen kohdeyksikkö tuo kehitettävään viitekehykseen myös omat rajoitteensa. Tärkeimmät rajoitteet ovat:

1. Malli käyttää jo kerättäviä metriikoita.
2. Se ei vaadi suuria resursseja.
3. Malli ei saa olla liian teoreettinen.
4. Mallin avulla voidaan reagoida nopeasti kehitettävään ohjelmistoon.

Kohdeyrityksen yksikkö, jolle ohjelmiston luotettavuusmallia kartoitetaan tai mahdollisesti kehitetään, on vastikään perustettu ja sen alaisuudessa käynnistyneiden projektien metriikoita on vajaan vuoden verran kerätty. Toivottavaa olisi, että mahdollinen luotettavuusmalli hyödyntäisi kerättyä informaatiota. Malli ei saisi vaatia liian suuria resursseja käyttöönsä. Kustannusten tulisi vastata mallin tarjoamia hyötyjä. Teoreettisuus ei saisi nousta liian korkeaksi. Muuten luotettavuusmallin käyttöarvo johdon apuvälineenä laskisi. Fenton ja Neil (2000) esimerkiksi painottavat artikkelissaan, että johdolle tarjottavien metriikoiden tulee olla yksinkertaisia ja helposti opittavia. Mallin avulla tulee myös voida reagoida nopeasti, jos kehitettävä ohjelmisto on luotettavuuden suhteen ajautumassa väärään suuntaan.

Lyun ja Nikoran viitekehyksessä, alan kirjallisuudessa ja kohdeyksikön rajoitteissa toistuvat osittain samat ominaisuudet, joiden perusteella on helppo rakentaa uuden viitekehyksen perusta. Toistuvat ja viitekehykseen valittavat komponentit ovat:

ohjelmistovirheiden estimointi, kykenevyys, mallin helppokäyttöisyys, soveltuvuus ja yksinkertaisuus. Lyun ja Nikoran viitekehyksestä jätetään pois kaksi komponenttia, mallin validiteetti ja epäherkkyys häiriötekijöille, mitkä vaativat analysoitavan luotettavuusmallin käyttöä tietyllä testidatalla. Komponentteja ei voida käyttää, koska tässä tutkimuksessa rakennetaan viitekehystä, joka vertailee luotettavuusmalleja teoreettisesti. Lyun ja Nikoran viitekehyksestä otetaan ensimmäiseksi ja kategorisoivaksi komponentiksi *perusolettamus*, jonka tarkoituksena on luokitella vertailtava luotettavuusmalli sen perusolettamuksen perusteella. Viitekehykseen lisätään myös yhdeksi komponentiksi Musan, Ianninon ja Okumoton painottama *johdon päätöksenteon tukeminen*. TAULUKKO 2 kuvaa kaikki kehitettyyn viitekehykseen valitut komponentit.

TAULUKKO 2. Kehitetty viitekehys ohjelmiston luotettavuusmallien vertailuun

Komponentti	***Kuvaus***
Perusolettamus	**Mihin olettamukseen malli perustuu?**
Ohjelmistovirheiden estimointi	**Antaako hyvän ennusteen tulevaisuuden virheiden käyttäytymisestä?**
Kykenevyys	**Kykeneekö estimoimaan ohjelmiston luotettavuuteen liittyviä suureita?**
Mallin helppokäyttöisyys	**Onko parametrejä vähän ja onko niitä helppo arvioida?**
Soveltuvuus	**Toimiiko erilaisissa toimintaympäristöissä ja kehittämisprosesseissa?**
Yksinkertaisuus	**Voiko mallin tuloksia ymmärtää ja tulkita helposti?**
Johdon päätöksenteon tukeminen	**Onko ohjelmisto valmis julkaisuun? Milloin se tulee olemaan valmis?**
Kerättävien metriikoiden käyttö	**Hyödyntääkö malli kohdeyksikön keräämiä metriikoita?**
Vaaditut resurssit	**Vaatiiko malli suuria resursseja?**

Ohjelmistovirheiden estimointi käsittelee mallin kykyä ennustaa tulevaisuuden virhekäyttäytymistä. Hyvällä estimoinnilla voidaan reagoida nopeasti ohjelmiston kehittämisen väärään suuntaan ajautumiseen. *Kykenevyys*-kriteeri viittaa mallin kykyyn estimoida ohjelmiston luotettavuuteen liittyviä suureita. *Mallin helppokäyttöisyys* käsittelee parametrien määrää sekä niiden arvioinnin helppoutta. Helposti mitattavat parametrit eivät pelkästään laske mittauskustannuksia vaan myös auttavat laatuihmisiä tulkitsemaan mallin avulla ohjelmiston fyysistä tilaa ja antamaan siitä palautetta ohjelmistokehittäjille. *Soveltuvuus* tarkastelee mallin kykyä tarjota tarkkoja estimaatteja eri toimintaympäristöissä ja erityyppisissä kehittämisprosesseissa. Mitä

yksinkertaisempi malli on, sitä helpommin voidaan kerätä tarvittava data, käyttää perusparametrejä tietokannasta ja ymmärtää sekä tulkita mallin tuloksia.

Taulukossa kaksoisviivalla erotetaan kohdeyksikön rajoitteet muista komponenteista. Jos viitekehystä käytetään jatkossa tieteellisessä tarkastelussa, riittävät kaksoisviivan yläpuolella olevat komponentit tarkasteluun. Ne muodostavat viitekehyksen perustan, jota voidaan käyttökontekstin suhteen laajentaa tarvittavilla lisäkomponenteilla, joita tässä tutkimuksessa edustavat kohdeyksikön rajoitteet. Rajoitteina on, että malli käyttäisi jo kerättäviä metriikoita ja sen käyttäminen ei tulisi vaatimaan suuria resursseja.

6 OHJELMISTON LUOTETTAVUUSMALLIT

Hirsjärven, Remeksen ja Sajavaaran (1997, 183) mukaan metodin eli menetelmän käsite on moniselitteinen. Heidän mielestään metodi on sääntöjen ohjaama menettelytapa, jonka avulla tieteessä tavoitellaan ja etsitään tietoa tai pyritään ratkaisemaan käytännön ongelma. Näin myös ohjelmistovirheiden ja -luotettavuuden mallintamisen pohjalta on kehittynyt menetelmiä. Pienet ohjelmistoyritykset luottavat Jonesin (1996, 3) mielestä omien asiantuntijoidensa arvioihin ohjelmiston luotettavuudesta, kun isommat yhtiöt käyttävät mieluummin oikeanlaisia työkaluja ja menetelmiä luotettavuuden mallintamiseen. Chillaregen ym. (1992) mukaan luotettavuusmallit jakautuvat fundamentaalisilta ominaisuuksiltaan kahteen ääripäähän: tilastollisiin vikamalleihin ja kausaalisiin analyyseihin. Kan (2003, 187-188) luokittelee vastaavanlaisesti mallit staattisiin ja dynaamisiin malleihin. Tässä tutkimuksessa käytetään Chillaregen ym. käyttämää luokittelua. He kuvaavat tilastollisten vikamallien päämäärän olevan ennustaa ohjelmistotuotteen luotettavuus. Tyypillisesti luotettavuus on mitattavissa jäännösvirheiden määrässä ja tuotteen häiriötaajuudella. Chillaregen ym. mielestä tilastolliset vikamallit takaavat hyvän tilanneraportoinnin, mutta ne eivät tarjoa itse ohjelmistokehittäjille palautetta. Tutkijoiden mukaan kausaaliset analyysit puolestaan analysoivat ja tunnistavat vikojen perimmäiset syyt ja käynnistävät toimia vikalähteiden korjaamiseksi. Toimien ansiosta ohjelmistokehittäjät saavat palautetta työstään. Chillarege ja muut listaavat kausaalisten analyysien ongelmaksi niiden vaatimat suuret taloudelliset ja ajalliset resurssit. Näiden kahden kirjon ääripään, kvantitatiivisten tilastollisten vikamallien ja kvalitatiivisten kausaalianalyysien, välillä vallitsee suuri kuilu. Tämä kuilu aiheutuu Chillaregen ja muiden mielestä ohjelmistokehittäjille mielekkäiden ja hyviä kehittämismalleja hyödyntävien mittausmenetelmien puutteesta.

Tässä luvussa käydään ensin läpi kohdeyrityksessä suoritetun kartoituksen tulokset. Tuloksien jälkeen valitaan alan kirjallisuudesta kolme luotettavuusmallia, joiden perusteet käydään läpi ja kerätään niistä luvussa 5.2.2 kehitetyn viitekehyksen vaatimat ominaisuudet. Mallien valintaan vaikuttivat niiden referenssit, alan kirjallisuuden kritiikki ja tutkimuksessa suoritetun kohdeyrityksen kartoituksen tulokset. Luvun lopussa käydään lävitse kartoituksen avulla löytyneitä kohdeyrityksen sisäisiä laatua ja

luotettavuutta parantavia menetelmiä. Kohdeyrityksen käytössä olevat menetelmät ovat vielä suhteellisen nuoria ja osa on vasta kehitteillä. Lisäksi menetelmien toimivuuden tarkkuudesta ei ollut vielä olemassa valideja tietoja. Siksi niiden perusteet käydään vain tiivistetysti läpi.

6.1 Haastatteluiden ja kyselyiden tulokset

Haastatteluiden ja kyselyiden avulla kartoitettiin kohdeyrityksen laatu- ja luotettavuusmallien tilanne. Tarkoituksena oli löytää minkälaisia malleja on käytössä ja tutkittuina. Aluksi haastateltiin kuutta yrityksen laatuasiantuntijaa. Haasteluiden tarkoituksena oli selvittää ilmiöalueen kattavuus ja hankkia lisää tietoa aihealueesta.

Ensimmäisenä haastateltava oli terminaalisovelluksia tuottavan yksikön virhehallintaprosessin kehittäjä. Hänen kanssaan käytiin läpi, kuinka ks. yksikkö käyttää apunaan nopeaa markkinapalautetta tuottamiensa sovellusten luotettavuuden estimoimiseen. Menetelmä on selitetty tarkemmin kohdassa 6.3.3. Menetelmä oli juuri otettu käyttöön, eikä sen toimivuudesta ollut vielä tarkempia tuloksia, mutta ensimmäiset tulokset olivat olleet kannustavia. Haastateltava toi myös esille menetelmän tämänhetkiset puutteet, jotka on sisällytettynä menetelmän arviointiin kohdassa 7.1.2. Haastattelussa pohdittiin myös olisiko nopeaan markkinapalautteeseen pohjautuva menetelmä integroitavissa tutkimuksen kohdeyksikön käyttöön. Haastateltavan mielestä menetelmä soveltuisi kohdeyksikön käyttöön pienillä muutoksilla ja jatkokehittelyllä.

Toisessa haastattelussa oli mukana kaksi laatuasiantuntijaa ISA-puhelimia (ks. 2.2.2) kehittävästä yksiköstä. Toinen asiantuntijoista oli erikoistunut luotettavuuden määrittelyyn ja toinen virheiden hallintaan. Yksikkö erosi kooltaan tutkimuksen kohdeyksiköstä huomattavasti. Henkilöstöä siellä oli kymmenkertaisesti, minkä takia yksiköiden väliset käytänteet erosivat huomattavasti. Yksiköllä on käytössä laaja tietokanta, jonne palautettujen puhelimien virheet kirjataan. Kantaan merkitään tällä hetkellä viallisen tuotteen oire. Aikaisemmin merkittiin oireen arvioitu syy ja oireen korjaava toimenpide, joka yleisesti oli uuden ohjelmiston lataaminen tuotteeseen. Näin

virheiden perimmäiset syyt eivät päätyneet ohjelmistokehittäjille asti. Varsinkin kun oireen aiheuttavan syyn kirjasi ylös usein puutteellisen koulutuksen omaava huoltoliikkeen työntekijä. Tietokannasta saatujen tietojen perusteella voitiin laskea tuotteiden häiriötaajuudet ja käyttää niitä apuna kehitettävien tuotteiden luotettavuuden estimointiin. Yksikössä on alettu panostaa ohjelmiston luotettavuusestimaattien tekemiseen tarkistuspisteissä (*milestone*), koska ohjelmiston osuus tuotteissa on suuri. Yksikössä on myös alettu tutkia, kuinka ohjelmiston kypsyyden (*maturity*) määrittämisen avulla taattaisiin tuotteen laadukkuus ja luotettavuus.

Kolmannessa haastattelussa oli mukana kaksi tutkimusyksikön työntekijää. Toinen heistä oli tutkinut ohjelmiston luotettavuuden määrittämistä ja toinen ohjelmistoprosessien kehittämistä. Suurin ongelma heidän mielestään kohdeyrityksessä on tällä hetkellä ettei ohjelmistovirheiden alkuperää voida tarkasti jäljittää. Esimerkiksi komponenttitehdas ei koskaan tiedä kehittämänsä ohjelmiston todellista häiriötiheyttä, koska historiatietoja julkaistavista tuotteista ei ole saatavilla. Syynä tähän on, että kun tuote palautetaan takuuhuoltoon vian takia, todellista ongelmaa ei useinkaan pystytyä selvittämään, vaan ohjelmisto ladataan tuotteeseen uudestaan ja toivotaan, että vika häviää. Tarkkoja tilastoja ei näin saada ja siksi historiatietoihin pohjautuvia estimointimalleja ei voida kohdeyrityksessä tehokkaasti käyttää. Toisen haastateltavan mukaan testaus kannattaisi ottaa enemmän huomioon tutkimuksessa. Tällä hetkellä käytössä olevan tositestaamisen (ks. 6.3.2) avulla saadaan helposti dataa, jonka avulla lopullisen tuotteen luotettavuutta voitaisiin estimoida, mutta se suoritetaan liian myöhäisessä kehitysvaiheessa. Toisen haastateltavan mielestä vaihe on aivan liian myöhäinen tärkeiden päätösten kannalta, joista esimerkkinä hän mainitsi päätöksen onko tuotteen laatu saavuttanut riittävän tason. Ohjelmiston luotettavuutta tutkinut haastateltava suositteli perehtymistä Bayes-verkkoihin ja ohjelmistoprosessien kehittämistä tutkinut ODC (*Orthogonal Defect Classification*) -menetelmään. Heidän mielestään nämä soveltuisivat hyvin tutkimukseen.

Viimeinen haastattelussa oli mukana ohjelmiston laatuspesialisti, joka on vahvasti ollut kehittämässä systeemitestausta. Tällä hetkellä haastateltava tutkii SRE (*Software Reliability Engineering*) -mallin soveltuvuutta tuoteprojekteihin. Hänen tiiminsä oli

juuri koekäyttämässä mallia erään tuotteen kehityksessä. SRE:n avulla he pyrkivät parantamaan tuotteen luotettavuutta ja estimoimaan luotettavuuden kehittymistä. Haastattelussa käytiin läpi mallin perusidea ja miten haastateltavan tiimi integroi sen tuotekehitykseen. Mallin toimimista käytännössä kuvataan enemmän kappaleessa 6.2.1. Haastateltava kertoi oppineensa kehitysprojektin aikana, että monen projektin jäsenen mielestä tämänkaltaisissa malleissa on liikaa epävarmuutta ja siksi he epäilevät lopputuloksia. Hänen mielestä tutkimusalue on erittäin hankala, mutta ehkä juuri sen takia haastava. Täydellisti toimivaa mallia tullaan tuskin koskaan kehittämään. Haastateltavan mielestä on kumminkin pakko olla olemassa jokin matemaattinen malli, joka yhdistää käyttäjien, laitteiden, verkkojen ja ajan tuoman luotettavuuden.

Haastatteluiden tavoitteena oli hankkia lisätietämystä tutkimuksen aihealueesta, varsinkin minkälaisten mallien kohdalla kohdeyritys on ollut aktiivinen, ja selvittää keitä muita tulisi aihealueen tiimoilta haastatella. Uusia kontakteja kertyi haastatteluiden pohjalta neljätoista kappaletta. Näille henkilöille päätettiin lähettää sähköpostikysely aihealueesta (ks. LIITE 1). Kyselyn ideana oli saada selville minkälaisia malleja henkilöt käyttävät tai ovat käyttäneet luotettavuuden saavuttamiseksi tai sen estimoimiseksi. Kyselyyn saatiin viisi validia vastausta, joiden tärkein anti esitetään seuraavassa tiivistetysti.

Erään komponenttitehtaan virhepäällikkö kertoi, että heillä vikamäärä estimoidaan implementointitiimin tekemän tehokkuusestimaatin perusteella. Tällä hetkellä tehokkuusestimaatin arvioivat kokeneemmat ohjelmistokehittäjät eli se ei perustu mihinkään matemaattiseen kaavaan. Menetelmän käytön ensimmäinen kierros on meneillään, joten kokemusta sen toimivuudesta ei vielä ole. Seuraaville kierroksille tehokkuusestimaatin tarkkuutta yritetään parantaa ensimmäisen kierroksen toteutumien mukaan.

ISA-puhelimia kehittävän yksikön eräs laatujohtaja tiivisti pitkän kokemuksensa perusteella kohdeyrityksen tilanteen luotettavuusmalleista. Hänen mukaansa mallit ovat yleisesti olleet ad hoc -tyyppisiä tietylle tuotteelle suunnattuja. Mallien perustana ovat olleet aikaisemmat tuotteet, joiden vikamäärien tilastollisia keskiarvoja tai vastaavia

menetelmiä on käytetty pohjana uusien tuotteiden estimaateissa. Tärkeintä laatujohtajan mielestä on, että kun tietynlaista luotettavuusmallia käytetään, tulee varmistaa, että kaikki käyttäjät ymmärtävät tarkasti mallin pääidean ja eri osatekijöiden toiminnallisuuden. Hän suositteli perehtymistä IBM:n kehittämään ODC-malliin.

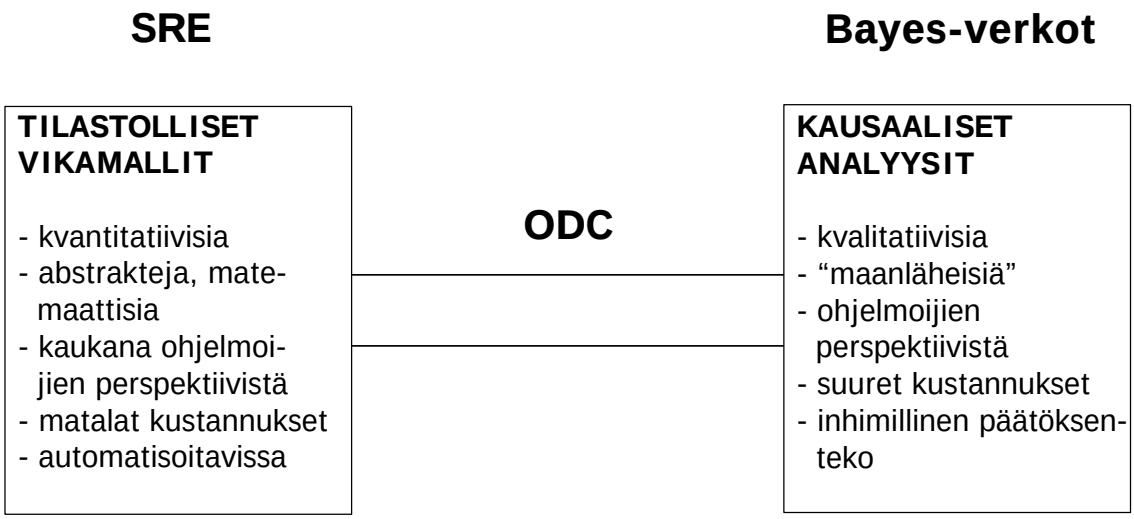
Asiakaspään huoltoa kehittävän projektin projektipäällikkö muistutti, että suuri prosentuaalinen ero ohjelmisto- ja laitteistovioissa johtuu yleisesti siitä, että laitteistovikoja pyritään monesti korjaamaan ohjelmiston avulla. Hänen mielestään on ongelmallista, että huollossa puhelimet ohjelmoidaan uudelleen selvittämättä tarkemmin, mikä on todellinen vika. Näin ei saada jäljitettyä vian alkuperäistä aiheuttajaa ja ehkäistyä vastaavanlaisten vikojen uudelleen syntyminen. Projektipäällikkö kehotti perehtymään tositestaamiseen ja ottamaan yhteyttä palauteanalyysejä tekeviin tiimeihin.

Eräs ohjelmistoprojektin projektipäällikkö, joka omaa pitkän laatutaustan, totesi historiatietoihin pohjautuvien virhe-estimointien olevan todella vaikeita. Hän esitti esimerkkinä kaksi tuoteprojektia, joissa toisessa estimaatti oli ollut aivan liian korkea ja toisessa aivan liian matala. Projektipäällikön mielestä historiatietoihin pohjautuva virhe-estimointi saataisiin tarkemmaksi jos se aloitettaisiin jo komponenttitehtaan tasolta.

6.2 Kirjallisuudessa esitettyjä luotettavuusmalleja

Tähän päivään mennessä on Kanin (2003, 211) mukaan kehitetty yli sata erilaista menetelmää luotettavuuden mallintamiseksi. Hän jatkaa, että valitettavasti kovin montaa mallia näistä ei ole testattu käytännössä oikealla datalla ja vielä vähemmän malleja on oikeasti käytössä. Luotettavuusmallien ongelmiksi ohjelmistokehityksen kannalta Kan luottelee niiden tarvitseman datan keräämisen kalleuden, vaikean ymmärrettävyyden ja yksinkertaisesti niiden toimimattomuuden koekäytössä. Kan käy kattavassa teoksessaan (2003) joitain malleja tarkemmin läpi. Hän toteaa tarkasteltavien luotettavuusmallien valinnan perustuneen pelkästään kokemukseen, eivätkä ne pakosti ole parhaimpia mitä kirjallisuudesta tällä hetkellä löytyy. Yleisesti ohjelmiston luotettavuusmallien luokittelua tai keskinäistä vertailua en alan kirjallisuudesta löytänyt.

Tähän tutkimukseen oli tarkoituksena valita kirjallisuudesta kolme hyväksi tunnustettua luotettavuusmallia lähempään tarkasteluun. Tarkastelun pohjana toimii kohdassa 5.2.2 kehitetty viitekehys. Tarkasteluun valittavien mallien tulisi edustaa luvun kuusi alussa kuvatun Chillaregen ym. (1992) mukaisen jaottelun ääripäitä. Näin saataisiin tarkempi kuvaus erityyppisistä luotettavuusmalleista. Kohdeyrityksen kartoituksessa esiintyneet mallit SRE, Bayes-verkot ja ODC sijoittuvat jaotteluun hyvin. Musan (1997) mukaan SRE edustaa tilastollisia vikamalleja, Fentonin ja Neilin (1996) mielestä Bayes-verkot kausaalisia analyyseja ja Chillaregen ym. (1992) mukaan sijoittuu ODC näiden kahden ääripään välimaastoon. Kuviossa 11 mallit on sijoitettu Chillaregen ym. (1992) kuvaamaan jaotteluun. Kuvion pohjana toimii Chillaregen (1994) hahmotelma ODC:n asettumisesta mallien kirjoon. Kaikki kolme luotettavuusmallia omaavat myös hyvät referenssit ja ne ovat alan kirjallisuudessa tunnettuja. Näiden perusteluiden ansiosta nämä luotettavuusmallit valittiin tutkimukseen tarkempaan tarkasteluun.



KUVIO 11. Tutkimukseen valittujen luotettavuusmallien sijoittuminen mallien kirjoon (Chillarege 1994)

Seuraavaksi käydään valittujen luotettavuusmallien perusteet tiivistetysti läpi. Jokainen malleista tarjoaa myös muita tutkimuksessa mainitsemattomia muuttujia ja mittareita, mutta niiden tarkastelu on jätetty pois, koska tutkimus keskittyy selventämään vain mallien pääidean.

6.2.1 Tilastolliset vikamallit - SRE

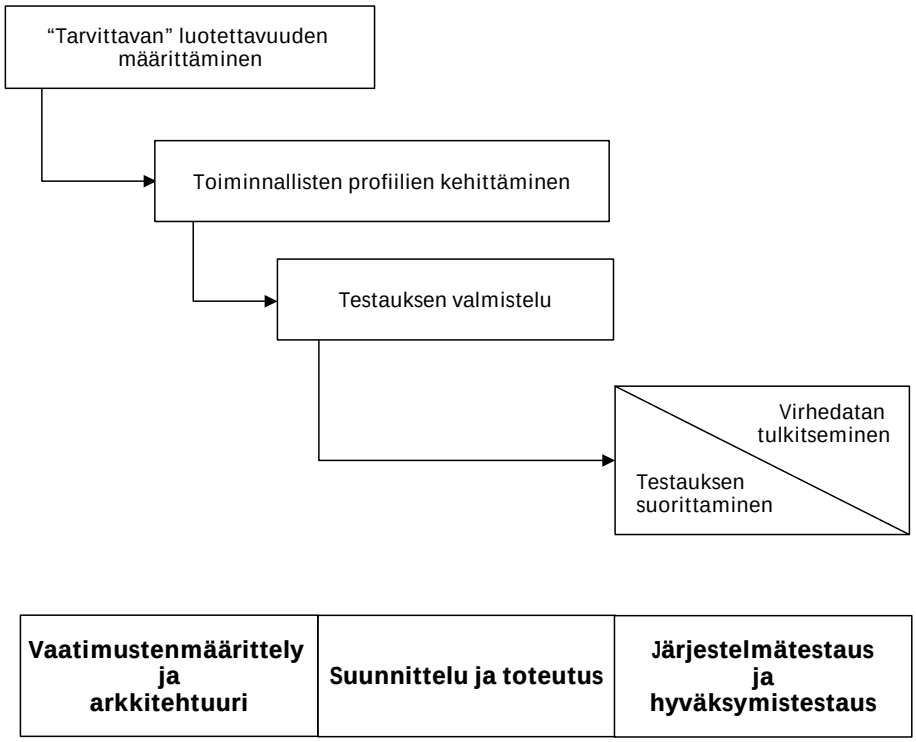
Tilastollisia vikamalleja tutkimuksessa edustaa SRE (*Software Reliability Engineering*). SRE on yhdysvaltalaisessa teleyhtiö AT&T:ssä kehitetty ohjelmiston luotettavuuden parantamiseen ja ennakoimiseen tähtäävä menetelmä. Sen pääkehittäjä on John D. Musa, ja menetelmä virallistettiin ensimmäisen kerran vuonna 1991 (Musa 1997, 3). Mallin tarkastelu perustuu lähinnä Musan (1997) artikkeliin ja hänen ylläpitämäänsä mallin viralliseen Internet-sivustoon.

Musan (1998) mukaan ohjelmistokehittämisen pääongelma on tasapainoilla luotettavuuden/käytettävyyden, kehittämisajan ja kustannusten välillä. SRE on kehitetty ratkaisemaan tämä ongelma. Se on käytäntö kvantitatiiviselle suunnittelulle ja ohjaukselle ohjelmiston kehittämiseen sekä testaukseen korostaen etenkin luotettavuutta ja käytettävyyttä (Musa 2003). SRE eroaakin muista vastaavista menetelmistä ollen pääsääntöisesti kvantitatiivinen. Kvantitatiivista tietoa menetelmässä käytetään apuna kun päätetään, mikä on kohderyhmän kannalta kannattavin ohjelmiston luotettavuusstrategia. Menetelmässä tarkkaillaan tuotteen kahta ominaisuutta: sen toimintojen suhteellista käyttöä ja vaadittuja laatuominaisuuksia. Suhteellisella käytöllä saadaan selville tuotteen kannalta kaikkein kriittisimmät toiminnot, joihin voidaan käyttää enemmän resursseja ja testata niitä realistisemmassa ympäristössä. Musan (2003) mukaan tuotteen vaaditut laatuominaisuudet ovat luotettavuus, käytettävyys, toimituspäivä ja elinkaarikustannukset. Musan (1997) mukaan testaajat ovat SRE:n käytön kannalta avainasemassa, mutta myös ohjelmiston kehittäjät, arkkitehdit ja käyttäjät ovat sitoutuneita menetelmään. SRE:n vahvuutena on, että malli mahdollistaa ohjelmiston kehittäjien ja testaajien tehokkaan yhteistyöskentelyn, jonka avulla taataan tuotteen luotettavuuden vastaavuus asiakkaan vaatimuksiin, nopeutetaan tuotteen markkinoille pääsyä, alennetaan tuotekustannuksia, parannetaan asiakastyytyväisyyttä ja lisätään sekä kehittäjien että testaajien tehokkuutta.

SRE:n referenssit ovat mairittelevat. IEEE:n *Computer Society's Technical Committee on Software Reliability Engineering* on kasvanut kuudessa vuodessa sen perustamisesta

40 ihmisestä yli tuhanteen, vuosittaisen kasvun ollessa 70 prosenttia (Musa 1997). Järjestö julkaisee jäsenlehteä, sponsoroi vuosittaista SRE-symposiota ja osallistuu moniin muihin toimintoihin, kuten standardointiin. Musan (2003) mukaan SRE:tä käyttävät mm. seuraavat yritykset: Alcatel, AT&T, Ericsson Telecom, Hewlett Packard, Hitachi, IBM, NASA's Jet Propulsion Laboratory, Microsoft, Saab Military Aircraft ja U.S. Air Force.

SRE koostuu seitsemästä päävaiheesta. Kaksi ensimmäistä koostuvat päätöksenteosta ja ne muodostavat perustan mallin viidelle ydinvaiheelle. Päätöksentekovaiheet kestävät yleisesti vain muutamia tunteja. Niissä päätetään mitkä tuotteeseen liittyvät järjestelmät tarvitsevat erillisen testauksen ja mitä testityyppejä millekin järjestelmälle tarvitaan. Erillisellä testauksella varmistetaan järjestelmän toimivuus erilaisissa laitteistokonfiguraatioissa ja viestintäprotokollissa eri maissa. Ydinvaiheet puolestaan levittäytyvät julkaisun koko elinkaarelle. (Musa 1997) KUVIO 12 esittää mallin ydinvaiheet kehittämisen elinkaareen ajallisesti sijoitettuna.



KUVIO 12. SRE-mallin ydinvaiheet ajallisesti sijoittuneena ohjelmiston kehittämisen elinkaarelle (Musa 1997, 5)

”Tarvittavan” luotettavuuden määrittäminen. Vaiheessa määritellään häiriöiden vakavuusasteet, päätetään häiriöiden intensiteettitavoite ja valitaan käytettävät luotettavuusstrategiat (Musa 1997). Häiriön vakavuusaste koostuu häiriöistä, jotka vaikuttavat samankaltaisesti käyttäjään. Vakavuusaste ilmaisee, kuinka vakava seuraamus häiriöstä voi syntyä. Seuraamus voi koskea ihmishenkiä, kustannuksia tai toisia toimintoja. Esimerkiksi matkapuhelinsovelluksessa 1. luokan häiriö voi olla, että puhelua ei voida suorittaa eli tällöin häiriö on käyttäjän kannalta vakava häiriö. 4. luokan häiriö voisi olla ettei osoitekirjaan voida lisätä henkilölle postinumeroa. Tämänkaltainen häiriö voi olla käyttäjälle kiusallinen muttei häiritse itse tuotteen käyttöä. Vaiheen oleellisimpia osia on häiriöiden intensiteettitavoitteen FIO:n (*Failure Intensity Objective*) määrittäminen. Määrittämistä varten tarvitsee määrittää tuotteen luonnollinen mittayksikkö. Mittayksikkönä voi olla esimerkiksi käyttöaika, tulostetut sivut, soitetut puhelut, lähetetyt tekstiviestit, jne. Esimerkiksi tulostimen FIO voi olla 1 häiriö 10000 tulostettua sivua kohti, matkapuhelimessa 1 häiriö 1000 puhelua tai tekstiviestiä kohti ja pankkiautomaattijärjestelmässä 1 häiriö kolmen viikon aikana. FIO:n määrittämiseen vaikuttavat tulevien käyttäjien tarpeet, olemassaolevan järjestelmän luotettavuus, järjestelmän laitteisto- sekä ohjelmistokomponentit ja kilpailevat järjestelmät (Musa 1997). Musan (1997) mukaan on olemassa kolme pääsääntöistä luotettavuusstrategiaa: vian ehkäiseminen, vian poistaminen ja vikatoleranssi. Vikojen ehkäisemisellä pyritään vähentämään vikojen syntyä kehitettävään ohjelmistoon. Apuna käytetään hyviä vaatimusmäärittely-, suunnittelu- ja ohjelmointitekniikoita sekä prosesseja. Niiden lisäksi panostetaan vaatimusmäärittely- ja suunnittelukatselmointeihin. Vikojen poistaminen ohjelmistosta suoritetaan koodin tarkastuksien ja testauksen avulla. Vikatoleranssi tarkoittaa, että keskitytään löytämään ohjelmistossa esiintyviä poikkeamia. Poikkeamat korjataan ennen kuin ne saattavat kehittyä häiriöiksi asti.

Toiminnallisten profiilien kehittäminen. Toiminto on tehtävä, jonka järjestelmä suorittaa. Toiminnallinen profiili koostuu joukosta toimintoja ja toimintojen esiintymistodennäköisyydestä. (Musa 2003) Esimerkiksi pankkiautomaattijärjestelmässä saldon tarkistaminen voi olla toiminnallisen profiilin yksi jäsen ja sen esiintymistodennäköisyys on 0.1, mikä tarkoittaa että

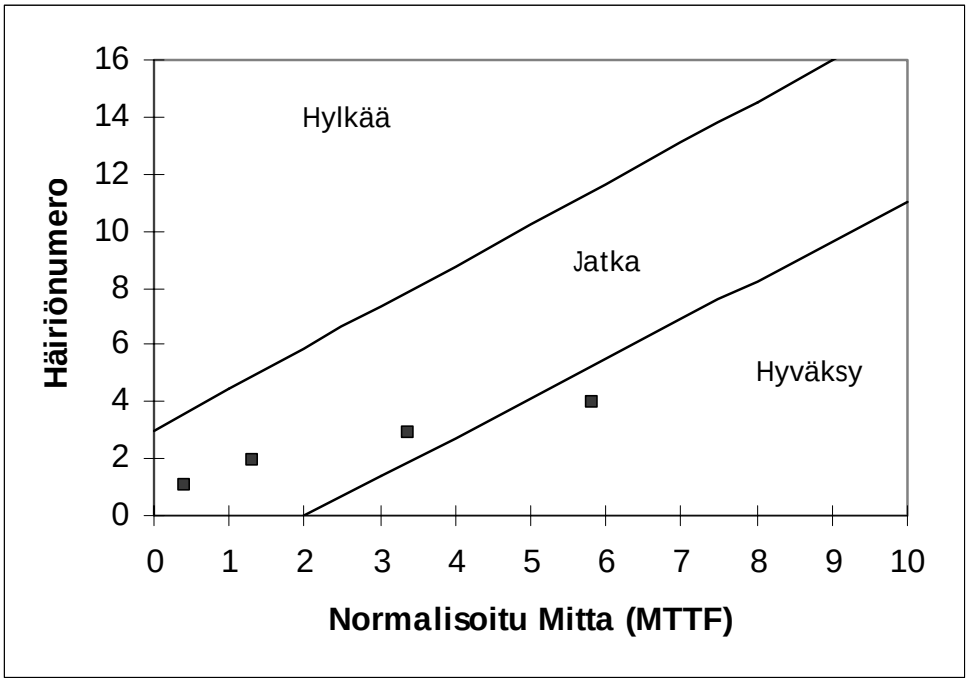
pankkiautomaatin käyttäjien koko käytöstä 10 prosenttia kuluu saldon tarkistamiseen. Musan (2003) mukaan ensimmäistä kertaa SRE-mallia toteuttavat kokevat esiintymistodennäköisyyksien päättämisen vaikeaksi. Apuna voidaan tällöin käyttää edeltäviä vastaavanlaisia järjestelmiä, tai jos järjestelmä on kokonaan uusi, tulee esiintymistodennäköisyydet määrittää yhdessä markkinoinnin kanssa.

Testauksen valmistelu. Tässä vaiheessa valmistellaan testitapaukset, menettelytavat ja kaikki automaattiset työkalut, joita on päätetty käyttää (Musa 1997). Testauksen valmistelussa käytetään hyväksi aikaisemmin luotuja toiminnallisia profiileja. Niiden perusteella jaetaan testiresurssit. Esimerkiksi jos raha-automaattijärjestelmää varten suoritetaan 500 testitapausta ja saldon tarkistaminen -profiilin esiintymistodennäköisyys on 0.1, käytetään 50 testitapausta saldon tarkistamiseen. Musan (1997) mielestä SRE voidaan suorittaa ilman testauksen automatisointia, mutta testauksen hallinta ja häiriöntunnistustyökalut tavallisesti nopeuttavat ja tehostavat prosessia.

Testauksen suorittaminen. Tässä vaiheessa suoritetaan testaus ja tunnistetaan häiriöt – milloin ne esiintyvät ja kuinka vakavia ne ovat (Musa 1997). Esiintyvistä häiriöistä saatu informaatio toimii seuraavan vaiheen syötteenä.

Virhedatan tulkitseminen. Vaihe auttaa tuotteen julkaisupäätöksiä estimoimalla järjestelmän häiriöintensiteettiä ja vertaamalla sitä asetettuihin tavoitteisiin (Musa 1997). Julkaisupäätökseen vaikuttavat testauksessa ilmenneiden häiriöiden vakavuusasteet ja esiintymistiheys. Apuna vaiheessa on luotettavuuden havainnointikaavio, joka perustuu määriteltyyn FIO-arvoon. Oheisessa kuviossa (KUVIO 13) on esitetty Musan teoksiin (1997; 2003) pohjautuva havainnointikaavio. Testauksessa esiintyneiden häiriöiden esiintymisajat muutetaan normalisoituun mittaan MTTF:en (*Mean Time To Failure*) ja asetetaan kaavioon. Normalisointi tapahtuu kertomalla virheen esiintymisaika FIO-arvolla. Esimerkiksi jos matkapuhelinsovelluksen FIO-arvo on 1/200 (1 häiriö 200 tunnissa) ja testauksessa 1. häiriö esiintyy 80 tunnin kuluttua, saadaan normalisoiduksi mitaksi 0.4. 2. virhe esiintyy 240 tunnin kuluttua ja sen normalisoitu mitta on 1.2. Häiriöiden esiintymisajat ovat kumulatiivisia. Kun normalisoitu mitta häiriölle on laskettu, asetetaan se kaavioon ja

tarkkaillaan, koska häiriötiheys on asetetussa tavoitteessa. Jos häiriö asettuu kaaviossa hylkää-alueelle, ei kehitetty järjestelmä vastaa annettuja tavoitteita ja se pitää joko hylätä tai sen kehittämistä tulee jatkaa. Jatka-alueelle sijoittunut häiriö merkitsee testaamisen jatkamista, ja kun häiriö sijoittuu hyväksy-alueelle, on järjestelmä yltänyt asetettuihin tavoitteisiin ja se voidaan julkaista. KUVIO 13:ssa häiriöt 1-3 sijoittuvat testaamisen jatkaminen alueelle ja neljäs häiriö sijoittuu hyväksyttävälle alueelle, jolloin testaaminen voidaan lopettaa. Havainnointikaavion alueiden leveyteen ja jyrkkyyteen vaikuttaa määritelty FIO-arvo.



KUVIO 13. Luotettavuuden havainnointikaavio (Musa 1997, 11; 2003)

Musa (2003) muistuttaa ettei SRE-prosessi pääty kun tuote on julkaistu. Julkaisun jälkeen kerätään kenttätietoja seuraavia julkaisuja tai toisia tuotteita varten.

Musa ja hänen työtoverinsa AT&T:llä päättivät aluksi, että ainoastaan ohjelmiston kehittäjät määräävät FIO-arvon ja toiminnalliset profiilit, mutta tämä ei toiminutkaan käytännössä. Testaajat ovat riippuvaisia näistä arvoista ja ovat siksi enemmän motivoituneita niiden saavuttamiseksi. Musa ja hänen kollegansa ratkaisivat ongelman sijoittamalla testaajia ohjelmiston kehittämis- ja arkkitehtitiimeihin. Testaajilla on myös

enemmän kosketuspintaa tulevien käyttäjien kanssa ja näin he kykenevät tuomaan hyvin käyttäjien toiveita ohjelmiston suunnitteluun. Ratkaisun myötä myös ohjelmiston kehittäjien ja arkkitehtien ymmärrys testausta kohtaan kasvoi. Oheisessa taulukossa (TAULUKKO 3) on malli analysoitu viitekehyksen avulla.

TAULUKKO 3. SRE:n analysointi viitekehyksen avulla

Komponentti	***SRE***
Perusolettamus	**Tilastollinen vikamalli, mikä painottaa testaajien ja ohjelmistokehittäjien vahvaa yhteistyötä. Käyttää hyväksi kvantitatiivista tietoa**
Ohjelmistovirheiden estimointi	**Estimoi tulevaisuuden häiriöiden aikavälin MTTF (Mean Time To Failure)**
Kykenevyys	**Virheiden aikaväli (MTTF), häiriöiden vakavuusaste, häiriöiden intensiteettitavoite (FIO)**
Mallin helppokäyttöisyys	**Parametreja ei paljoa ja niiden arviointi loogista**
Soveltuvuus	**Vaatii hyvän testauksen asiantuntemuksen**
Yksinkertaisuus	**Tuloksia helppo ymmärtää ja tulkita**
Johdon päätöksenteon tukeminen	**Kertoo milloin ohjelmisto on saavuttanut kehittämisprosessin alussa sille asetetun luotettavuuden**
Kerättävien metriikoiden käyttö	**Ei käytä**
Vaaditut resurssit	**Mahdollisten testauksen automaatiotyökalujen hankkiminen**

Tutkimukseen haastateltavista henkilöistä yksi oli juuri ollut testaamassa SRE:n toimivuutta eräässä tuoteprojektissa. Projekti pyrki SRE:n avulla parantamaan ja estimoimaan kehitettävän ISA-tuotteen luotettavuutta. SRE:tä käytettiin eniten käytettyjen sekä kriittisempien toiminnallisten profiilien testaamiseen. Oleellisinta SRE:n käytössä oli sitä varten luodun testaustyökalun soveltaminen ja kehittäminen. Testaustyökalu oli automaattinen ja sen avulla voitiin luoda virtuaalitestauspäiviä ja näin nopeuttaa testausprosessia. Automaattinen testaus oli tarpeen, sillä tuoteprojektissa erilaisia käyttötapauksia syntyi paljon. Tällä hetkellä testaustyökalut toimivat vain ISA-tuotteiden (ks. 2.2.2) testauksessa, mutta ne olisi tarkoitus tulevaisuudessa kehittää myös Symbian-tuotteiden (ks. 2.2.3) testaamiseen sopivaksi. Laajoissa tuotteissa SRE:n

ongelmaksi muodostuu juuri käyttötapauksien liiallisuus. Haastateltavan mukaan projektissa opittiin, että SRE:n mukaisessa testauksessa ei pyritä kattamaan koko ohjelmistoa, vaan vain käyttäjien näkökulmasta olevat pääosat siitä. FIO-arvoksi matkapuhelimelle projektissa määriteltiin 1/200h eli yksi virhe saa tapahtua kahdessasadassa käyttötunnissa.

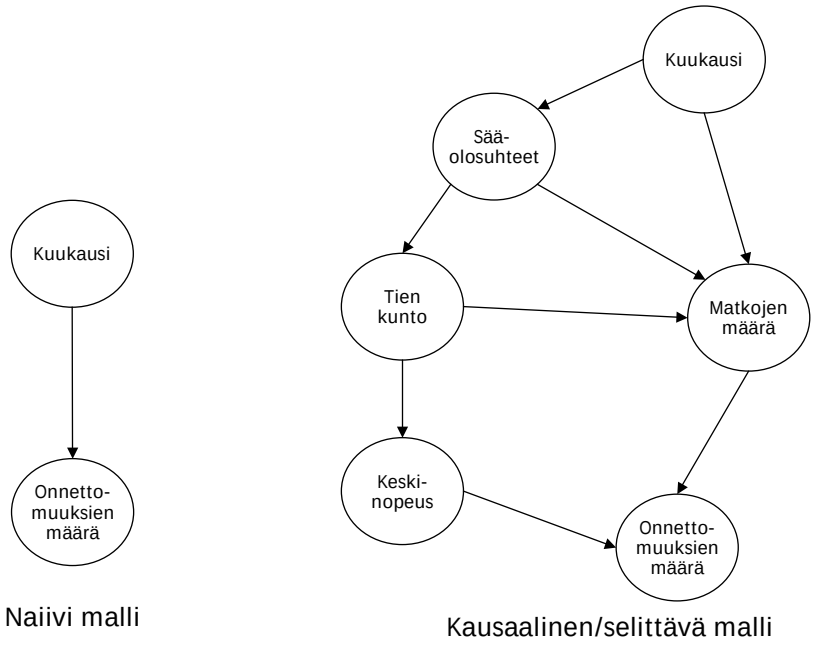
Tutkimusta tehdessä tarkempia tietoja menetelmän toimivuudesta kehittämisprojektissa ei ole vielä saatu. Tällä hetkellä tiedossa on, että SRE:n käyttö kehittämisprojektissa vaatii paljon resursseja. Siksi menetelmän tuoma hyöty tulee olla suuri, jos sitä aiotaan tulevaisuudessa myös käyttää. Eniten resursseja kului automaattisen testaustyökalun konfigurointiin. Testaustyökalun kehittyessä voi resurssien tarve pienentyä.

6.2.2 Kausaaliset analyysit - Bayes-verkot

Bayes-verkot edustavat tutkimuksessa tilastollisista vikamalleista vahvasti poikkeavia kausaalisia analyyseja. Teoria johon Bayes-verkot pohjautuvat on kehitetty kauan aikaa sitten, mutta vasta viime aikoina realististen verkkojen rakentaminen ja toteuttaminen ovat tulleet mahdollisiksi tehokkaiden algoritmien ja niitä toteuttavien ohjelmistotyökalujen ansiosta (Fenton & Neil 2000). Menetelmän soveltuvuutta ohjelmistovirheiden estimointiin ovat paljolti tutkineet mm. Norman Fenton ja Martin Neil. Menetelmään pohjautuvaa markkinoiden johtavaa ohjelmistotyökalua kehittävä Hugin Expert (2003) -yhtiön asiakkaina ovat mm. Hewlett-Packard, Intel Corporation, Siemens, Phillips Research Laboratories, Motorola, Cisco ja NASA/Johnson Space Center. Bayes-verkot ovat tähän päivään mennessä osoittaneet hyödyllisen käyttönsä käytännöllisissä sovelluksissa kuten lääketieteellisissä diagnooseissa ja mekaanisten vikojen diagnooseissa. Viimeksi Bayes-verkkoja on Fentonin ja Neilin (1999) mukaan ansiokkaasti käyttänyt Microsoft, jonka Microsoft Officen apuvelhot (*help wizards*) pohjautuvat Bayes-verkkoihin. Menetelmän perusteiden läpikäynti perustuu lähinnä Fentonin ja Neilin (1996, 1999, 2000) teoksiin.

Tilastolliset mallit, kuten regressiomallit, ovat hallinneet ohjelmistometriikoita, vaikka todellisuudessa tarvitaan kausaalisia malleja (Fenton & Neil 2000). Regressiomallit

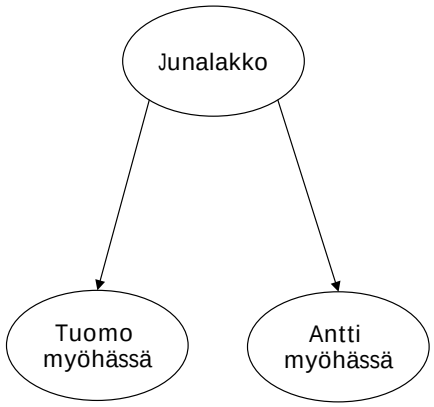
johtavat usein syyn ja seurauksen väärinymmärtämiseen. Kuitenkin tämä on tilanne, missä moni ohjelmiston kehittäjä on – hyväksytään yksinkertaiset mallit, joista tiedetään puuttuvan todella tärkeitä muuttujia. Esimerkiksi kuukauden ja liikenneonnettomuuksien korrelaatio ei tarjoa todistetta kausaalista suhteesta. KUVIO 14 havainnollistaa naiivin regressiomallin ja kausaalisen mallin eroavaisuuden kuukauden ja liikenneonnettomuuksien suhteen kuvaamisesta. Naiivi regressiomalli kertoo eri kuukausina tapahtuvien liikenneonnettomuuksien määrän muttei selitä määrässä tapahtuvia poikkeamia. Kausaalista mallista voimme päätellä esimerkiksi, että talvikuukausina huonojen sääolosuhteiden takia teiden kunto on heikko. Tämän takia ihmiset ajavat vähemmän ja kun he ajavat on keskinopeus alhaisempi, siksi liikenneonnettomuuksien määrä on ajankohtana alhaisempi.



KUVIO 14. Liikenneonnettomuuksien kuvaaminen (Fenton & Neil 2000, 360)

KUVIO 15 esittää Fentonin (2003) kuvaileman yksinkertaisen esimerkin Bayes-verkon käytöstä. Kuviossa solmukohdat esittävät muuttujia, jotka voivat olla diskreettejä tai jatkuvia. Esimerkiksi ’Junalakko’ voi olla diskreetti muuttuja kahdella mahdollisella arvolla ’Tosi’ ja ’Epätosi’. Nuolet esittävät kausaalisia suhteita muuttujien välillä. Toinen muuttuja voi olla ’Tuomo myöhässä’, millä on myös diskreetit arvot ’Tosi’ ja

’Epätosi’. Koska junalakko voi aiheuttaa Tuomon myöhästymisen, piirretään muuttujien välille kausaalisuhde. Bayes-verkot ovat verkkoja, jotka ilmentävät todennäköisiä suhteita muuttujien välillä (Neil & Fenton 1996).



Tuomo myöhässä	Junalakko	
	Tosi	Epätosi
Tosi	0.8	0.1
Epätosi	0.2	0.9

Antti myöhässä	Junalakko	
	Tosi	Epätosi
Tosi	0.6	0.5
Epätosi	0.4	0.5

KUVIO 15. Yksinkertainen Bayes-verkkoesimerkki todennäköisyysarvoineen

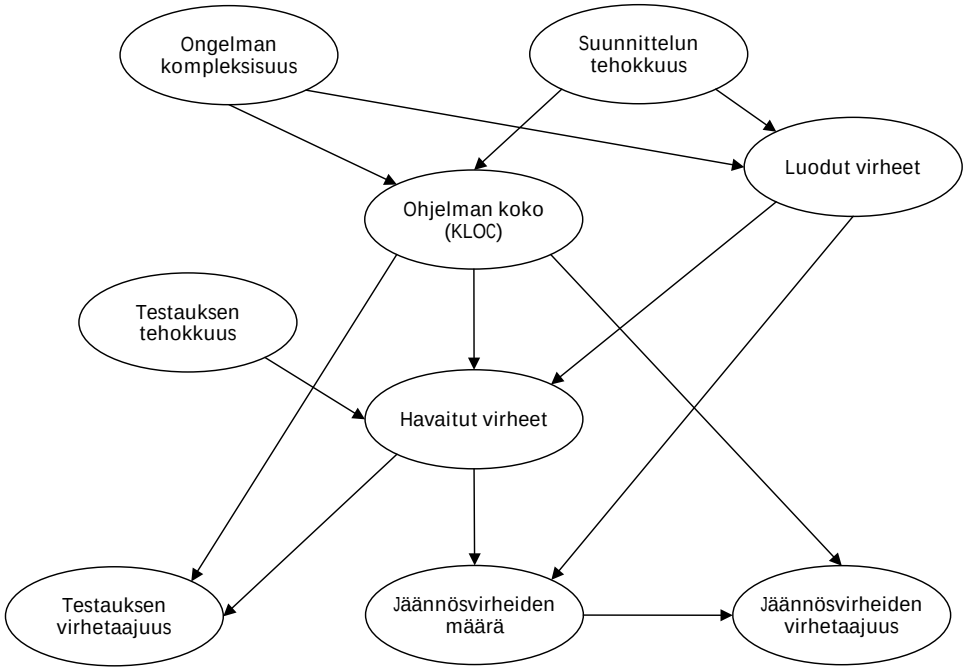
Kuvion 15 ylemmästä todennäköisyystaulukosta näemme, että Tuomon myöhästyminen on normaalisti hyvin epätodennäköistä (todennäköisyys Tuomon myöhästymiseen, kun kyseessä ei ole junalakko, on 0.1), mutta junalakon sattuessa myöhästyminen on erittäin todennäköistä (todennäköisyys 0.8). Oletetaan, että Tuomolla on työtoveri Antti, joka yleensä ajaa autolla töihin. Junalakko voi myös myöhästyttää Anttia, koska silloin muukin liikenne ruuhkautuu. Siksi piirrämme kausaalisuhteen myös ’Junalakko’ ja ’Antti myöhässä’ muuttujien välille. Kuvion alemmasta todennäköisyystaulukosta näemme, että Antti on usein myöhässä ja junalakon sattuessa myöhästymistodennäköisyys kasvaa vain vähän. Kuitenkin junalakon sattuessa Antti on epätodennäköisemmin myöhässä kuin Tuomo.

Jos pystymme esimerkiksi tilastollisten tietojen avulla määrittelemään junalakon todennäköisyydet, voimme laskea todennäköisyydet Tuomon ja Antin myöhästymiselle. Oletetaan, että junalakko on erittäin epätodennäköinen (todennäköisyys 0.1). Näillä arvoilla laskemme Tuomon myöhästymistodennäköisyyden Fentonin (2003) käyttämän kaavan avulla:

p(Tuomo myöhässä) = p(Tuomo myöhässä | junalakko) * p(junalakko) + p(Tuomo myöhässä | ei junalakkoa) * p(ei junalakkoa)
= (0.8 * 0.1) + (0.1 * 0.9) = 0.17

Samalla lailla voimme laskea, että Antti myöhästyy todennäköisyydellä 0.51. Ohjelmiston luotettavuutta arvioitaessa kasvaa muuttujien määrä radikaalisti. Tämän takia estimoitavien arvojen laskeminen ei ole enää inhimillistä työtä. Bayes-verkkojen avulla suoritettavaan ohjelmiston luotettavuuden estimointiin onkin kehitetty tietokonepohjaisia työkaluja. Fentonin (2003) mukaan alan johtavin työkalu on Hugin Expert.

Avainominaisuutena Bayes-verkoissa on kyky mallintaa ja perustella epävarmuutta (Fenton 2003). Oheisella kuviolla (KUVIO 16) Fenton ja Neil (1999) mallintavat kuinka Bayes-verkon avulla voidaan ennustaa ohjelmiston virheiden määrää.



KUVIO 16. Bayes-verkon topologia virheiden estimointiin (Fenton & Neil 1999)

Suurin osa mallin muuttujista omaa tilat: ”erittäin korkea”, ”korkea”, ”keskitasoinen”, ”matala” tai ”erittäin matala”. Poikkeuksina ovat ohjelman koko ja

virhemäärämuuttujat, joissa käytetään kokonaislukuja, sekä virhetaajuudet, joissa käytetään reaaliarvoja. Tiloihin kiinnitetyt todennäköisyydet ovat fiktiivisiä, mutta ne pohjautuvat alan kirjallisuuden analysointiin tai terveeseen järkeen perustuviin olettamuksiin muuttujien välisten suhteiden suunnista ja vahvuuksista (Fenton & Neil 1999). Malli perustuu olettamukseen, että muuttujat vaikuttavat vahvasti toisiin muuttujiin. Esimerkiksi muuttujien ”ongelman kompleksisuus” ja ”suunnittelun tehokkuus” väillä oleva epätasapaino vaikuttaa muuttujiin ”ohjelman koko” ja ”luodut virheet”.

Malli on erittäin matemaattinen ja sen käyttäminen edellyttää ohjelmistotyökalun apua. Työkaluun ohjelmoidaan verkon topologia ja muuttujien ehdolliset jakaumat. Syötettävinä arvoina ovat pääteltävissä olevat ongelman kompleksisuus, suunnittelun tehokkuus, ohjelman koko ja testauksen tehokkuus. Muuttujille valitaan sopivin vaihtoehto viidestä eri tilasta, paitsi ohjelman koko syötetään kokonaislukuna. Näiden neljän syötetyn muuttujan perusteella työkalu laskee muiden muuttujien jakaumat. Arvojen perusteella voidaan päätellä ohjelmiston luotettavuus ja sen avulla kypsyys julkaistavaksi. Oheisessa taulukossa (TAULUKKO 4) on esitetty mallin ominaisuudet tiivistetysti.

TAULUKKO 4. Bayes-verkkojen analysointi viitekehyksen avulla

Komponentti	***Bayes-verkot***
Perusolettamus	**Kausaalinen analyysi, jonka pohjana on vanha todennäköisyysteoria. Luotettavuuden määrittelevät muuttujat, jotka vaikuttavat toisiinsa**
Ohjelmistovirheiden estimointi	**Estimoi pohjatietojen pohjalta ohjelmiston jäännösvirheiden määrän**
Kykenevyys	**Jäännösvirheiden määrä, jäännösvirheiden virhetaajuus**
Mallin helppokäyttöisyys	**Bayes-verkon muuttujien todennäköisyydet vaikea määritellä. Verkon toimiessa parametrien määrittäminen ja arviointi todella helppoa**
Soveltuvuus	**Laajoissa ohjelmiston kehittämisprosesseissa muuttujien suuren määrän takia toiminta kankeaa**
Yksinkertaisuus	**Tuloksia helppo ymmärtää ja tulkita**
Johdon päätöksenteon tukeminen	**Määrittelee ohjelmiston tietyn hetken jäännösvirheiden määrän. Ei estimoi tulevaisuuden jäännösvirheitä, jos kehittämisprosessia jatketaan**
Kerättävien metriikoiden käyttö	**Voidaan hyödyntää Bayes-verkon rakentamisvaihessa, kun muuttujien välisiä todennäköisyyksiä määritellään**
Vaaditut resurssit	**Ohjelmistotyökalun hankinta ja toimivan verkkotopologian rakentamiseen kuluva aika**

6.2.3 Ääripäiden välimaasto - ODC

ODC (*Orthogonal Defect Classification*) on 1990-luvun alussa IBM:n tutkimusyksikössä, Watson Research Centerissä, kehitetty malli ohjelmistoprosessin mittaamiseksi ja analysoimiseksi (esim. Chillarege 1994). Mittaamisen ja analysoinnin pohjana toimivat ohjelmiston kehittämisessä ilmentyneet viat. Mallia on käytetty ja kehitetty jo kymmenen vuotta IBM:llä. Sitä on käytetty yli 60 projektissa ja yli 4000 ohjelmistoammattilaisen parissa. Myös muut yritykset, kuten Motorola, Tandem ja NorTel ovat omaksuneet mallin. (IBM 2003; Chillarege 2003) Mallin kehittämiseen osallistui monia IBM:n työntekijöitä, mutta pääkehittäjän roolissa ja etenkin mallin jatkokehittäjänä on toiminut Ram Chillarege. Mallin perusteiden kuvaaminen perustuu lähinnä Chillaregen ym. (1992) artikkeliin.

Perinteisten tilastollisten vikamallien ja kausaalisten analyysien väliin on syntynyt syvä kuilu, mutta kuten Chillarege (1994) toteaa, rakentaa ODC sillan tämän kuilun yli. Malli rakentuu ääripäiden parhaiden ominaisuuksien päälle. Tämä nostaa tuntuvasti ODC:n teoreettisuutta. Mallissa esiintyneet viat kirjataan tilastollisten vikamallien tapaan, mutta niitä tarkastellaan tarkemmin syy-seuraus -suhteessa niin kuin kausaalisissa analyyseissa.

ODC:n periaate on, että luokittelemme viat tiettyihin luokkiin, jotka osoittavat prosessin osan, joka tarvitsee erityistä huomiota (Chillarege ym. 1992). Näin voimme tarjota ohjelmistonkehittäjille prosessinaikaista palautetta. Ohjelmistovikoja jäljitettäessä eri informaation palaset tulevat voimaan eri aikoina (Bassin, Kratschmer & Santhanam 1998). Siksi vikoja koskeva informaatio kirjataan, kun vika on löydetty tai kun se on korjattu. Bassin ym. (1998) mukaan ODC:n kahdeksasta vika-attribuutista kolme (aktiviteetti, vian laukaisija ja vaikutus) saadaan kun vika ilmaantuu ja loput viisi kun vika on korjattu. Vikainformaatiota analysoimalla voidaan tulkita kehitettävän ohjelmiston kypsyys.

TAULUKKO 5 esittää kirjattavia ominaisuuksia kun vika on löydetty. Löydettäessä ollaan kiinnostuneita vian esiintymisolosuhteista ja vaikutuksesta asiakkaaseen.

Aktiviteetti on suoritettava vaihe, jonka aikana vika esiintyy. Esimerkiksi vika voi esiintyä yksikkötestauksessa. Aktiviteetti voi sisältää monta eri laukaisijaa. *Laukaisija* on ympäristö tai tila, jossa vika ilmaantuu. Kun tuote julkaistaan oletetaan, että kaikki sen funktiot ja operaatiot on testattu. Kentällä tietyissä olosuhteissa vika saattaa kuitenkin ilmaantua toisin kuin testausympäristössä. Täten, vaikka vikatyyppi on sama, sen ilmaantumiseksi tarvitaan tietyt olosuhteet. Siksi laukaisijat, toisin kuin vikatyypit, tunnistetaan vian elinkaaren alussa. (Chillarege ym. 1992) Kehitettävässä ohjelmistossa löytyneen vian *vaikutus asiakkaaseen* tulee arvioida jos sitä ei olisi löydetty kehityksen yhteydessä. Vaikutus asiakkaaseen -informaatiota käytetään hyväksi, kun lasketaan tuotteen asiakastyytyväisyyttä. TAULUKKO 5 esittää kirjattavat attribuutit kun vika havaitaan – missä aktiviteetissa eli suoritettavassa vaiheessa vika havaitaan, minkälaisessa ympäristössä tarkemmin ja mikä on sen vaikutus asiakkaaseen.

TAULUKKO 5. Tunnistettavat ominaisuudet, kun vika on löydetty (Bassin, Kratschmer & Santhanam 1998)

Aktiviteetti				**Vaikutus**
Tarkastus	**Yksikkötestaus**	**Integrointitestaus**	**Järjestelmätestaus**	**asiakkaaseen**
Suunnittelun yhdenmukaisuus	Yksinkertainen polku	Testikattavuus	Työkuormitus/rasitus	Asennettavuus
Logiikka/datavirrat	Kompleksi polku	Testipoikkeamat	Käynnistys/uudelleenkäynnistys	Huollettavuus
Komponenttien yhteensopivuus		Testisekvenssit	Elpyminen/poikkeukset	Standardi
Taaksepäin yhteensopivuus		Testivuorovaikutukset	Laitteistokonfiguraatio	Eheys/turvallisuus
Kieliriippuvuus			Ohjelmistokonfiguraatio	Siirrettävyys
Samanaikaisuus			Suojatut testit	Luotettavuus
Sivuvaikutukset				Suorituskyky
Harvinaiset tapaukset				Dokumentaatio

TAULUKKO 6 esittää kirjattavia ominaisuuksia kun vika on korjattu. Kirjauksen suorittaa vian korjannut ohjelmoija. *Kohteella* tarkoitetaan kehittämisvaihetta, jossa virhe on syntynyt. *Vikatyyppi* kuvaa esiintyneen vian luonnetta. ODC:ssä viat jaotellaan seitsemään eri tyyppiin. Ortogonaalisuudella tarkoitetaan kohtisuoruutta ja ODC:ssä tällä tarkoitetaan erityisesti vikojen luokittelun kohtisuoruutta. Toisin sanoen vikatyypit on suunniteltu niin etteivät ne voi olla päällekkäisiä. Vian *määrite* osoittaa puuttuuko joku tietty ominaisuus, onko ominaisuus toteutettu virheellisesti vai onko vika ulkopuolinen. *Lähde* tarkentaa missä virhe on alun perin syntynyt ja *ikä* virheen ajallisen historian.

TAULUKKO 6. Tunnistettavat ominaisuudet, kun vika on korjattu (Bassin, Kratschmer & Santhanam 1998)

Kohde	Vikatyyppi	Määrite	Lähde	Ikä
Suunnittelu Ohjelmointi	Sijoitus/alustus Tarkistus Algoritmi/metodi Funktio/luokka/olio Ajoitus/sarjallistuvuus Rajapinta/OO-viestit Yhteys	Puuttuu Virheellinen Ylimääräinen	Talon sisällä kehitetty Kirjastosta uudelleenkäytetty Alihankkijalla teetetty Koodin siirto toisesta laiteympäristöstä	Vanha Uusi Muokattu Korjattu

Viat ovat yhteydessä tiettyihin ohjelmiston kehittämisvaiheisiin. Yhteydet saattavat vaihdella organisaatioittain. TAULUKKO 7 kuvaa miten vikatyypit yleisesti sijoittuvat neljän ohjelmiston pääkehittämisvaiheen kanssa. Vaikka funktiovika on löytynyt järjestelmä- tai yksikkötestauksessa, se osoittaa kumminkin vaatimusmäärittelyssä tai arkkitehtuurisuunnittelussa tapahtuneeseen virheeseen (Chillarege ym. 1992). Samalla lailla esimerkiksi ajoitusvirheet yhdistetään joko arkkitehtuuri- tai moduulisuunnittelussa tapahtuneeseen virheeseen.

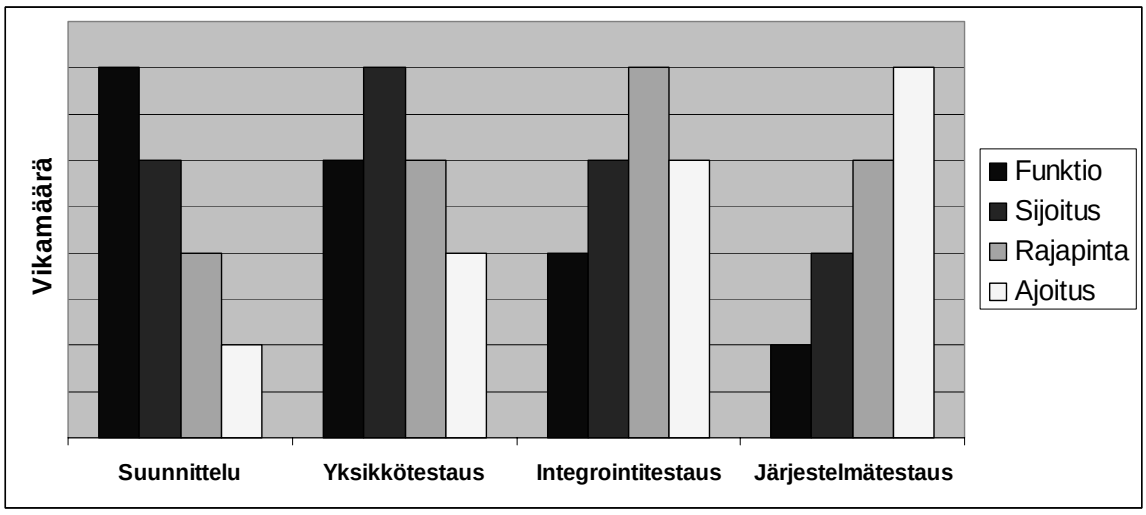
TAULUKKO 7. Vikatyyppien yhteydet ohjelmiston kehittämisvaiheisiin (Chillarege 2000)

Vaihe / vikatyyppi	Sijoitus/ alustus	Tarkistus	Algoritmi/ metodi	Funktio/luokka/ olio	Ajoitus/ sarjallistuvuus	Rajapinta/ OO-viestit	Yhteys
Vaatimustenmäärittely				X			
Arkkitehtuurisuunnittelu				X	X	X	X
Moduulisuunnittelu			X		X	X	X
Ohjelmointi	X	X	X				

Näiden yhteyksien takia tietty vikatyyppi voi aiheuttaa ohjelmiston verifiointivaiheissa vikapiikin. KUVIO 17 näyttää esimerkin, kuinka voimme havainnollisesti hyödyntää ortogonaalisia vikatyyppejä. Kuviossa esitetään kehittämisprosessin verifiointivaiheiden neljän vikatyypin jakautuminen. Kehittämisprosessin odotettu käyttäytyminen voidaan kuvailla. Esimerkiksi suurimmassa osassa kehittämisprosesseja, missä suunnittelu suoritetaan ohjelmointia ja testausta ennen, tulisi funktioviat löytää prosessin alkupuolella ja ihanteellisesti vain muutamat järjestelmätestauksessa. Täten funktiovikoja vastaavan pylvään tulisi pienentyä prosessin kuluessa. (Chillarege ym.

1992) Jos funktiotyyppisten vikojen määrä kasvaa myöhemmissä vaiheissa, on se merkki tuotteeseen jääneistä puutteista, ja silloin tuotteen julkistaminen sekä myyntiin päästäminen on suuri riski. Samoin voidaan olettaa, että ajoitus- ja sarjallistuvuusviat esiintyvät kehittämisprosessin loppupuolella, etenkin järjestelmätestauksen yhteydessä. Periaatteena on, että vikatyyppien jakautuma muuttuu ajan mittaan ja jakautuma osoittaa missä vaiheessa ohjelmistokehitys on.

Kehittämisprosessin vikatyyppien jakauman muutosta havainnollistavasta histogrammista voidaan päätellä ohjelmiston kypsyys. Onko kehittäminen loogisesti samassa paikassa kuin fyysisesti? Esimerkiksi jos järjestelmätestauksessa vikojen jakautumisprofiili näyttää siltä, kuin sen pitäisi sopia paremmin yksikkö- tai integrointitestaukseen, silloin vikajakautuminen osoittaa tuotteen olevan ennenaikainen järjestelmätestaukseen (Chillarege ym. 1992). TAULUKKO 8 esittää ODC:n analysoinnin viitekehyksen avulla.



KUVIO 17. Vikatyyppien jakautumisen muutos kehittämisvaiheittain (Chillarege ym. 1992)

Tutkimuksessa suoritetussa kyselyssä ilmeni, että eräässä kohdeyrityksen komponenttitehtaassa on omaksuttu vahvat vaikutteet ODC:sta. Virheiden kirjauksessa käytetään paljolti samoja attribuutteja kuin ODC:ssa. Joitain attribuutteja on jätetty pois tai hieman muunneltu alkuperäiseen menetelmään verrattuna.

TAULUKKO 8. ODC:n analysointi viitekehyksen avulla

Komponentti	*ODC*
Perusolettamus	**Käyttää sekä tilastollisten vikamallien että kausaalisten analyysien ominaisuuksia. Perustana virheiden tarkka kirjaaminen ja analysoiminen**
Ohjelmistovirheiden estimointi	**Analysoi kirjattujen virheiden perusteella ohjelmiston kypsyyden**
Kykenevyys	**Ohjelmiston kypsyysaste, pohjainformaatio asiakastyytyväisyydelle, nopea palaute ohjelmistokehittäjille**
Mallin helppokäyttöisyys	**Parametreja on paljon ja ongelmana on niiden kirjaamisen yhdenmukaisuus**
Soveltuvuus	**Toimii parhaiten laajoissa ohjelmiston kehittämisprosesseissa**
Yksinkertaisuus	**Tulosten ymmärtäminen ja tulkinta vaikeaa. Vaatimuksena hyvä perehtyminen menetelmään**
Johdon päätöksenteon tukeminen	**Määrittelee ohjelmiston tietyn hetken kypsyysasteen. Voidaan estimoida milloin ohjelmisto on laadullisesti valmis julkaistavaksi**
Kerättävien metriikoiden käyttö	**Vaatisi nyt kerättävien metriikoiden tarkemman jaottelun ja lisäksi keräämistä tulisi täydentää muilla metriikoilla**
Vaaditut resurssit	**Kun malli on sisäistetty (vie todennäköisesti kauan), kuluttaa se vähän resursseja**

Komponenttitehdas suorittaa tällä hetkellä tutkimusta vuoden vanhan menetelmän parantamiseen. Tutkimuksen tarkoituksena on selvittää kuinka menetelmää pystyttäisiin keventämään, koska tällä hetkellä sen käyttö sitoo liikaa resursseja. ODC soveltuu hyvin juuri komponenttitehtaiden käyttöön, koska menetelmä tarjoaa nopeaa palautetta ohjelmiston kehittäjille.

6.3 Kohdeyrityksessä käytössä olevia menetelmiä

Tutkimuksessa kartoitettiin kohdeyrityksen käytössä olevia laatua ja luotettavuutta parantavia menetelmiä. Kartoittaminen suoritettiin haastatteluiden ja sähköpostikyselyiden avulla (ks. LIITE 1). Kartoittamisella haluttiin selvittää, minkälaisia luotettavuusmalleja yrityksessä on käytössä ja minkälaisia menetelmiä on tutkittu. Kartoittaminen paljasti, että yhtään validia luotettavuusmallia ei ole käytössä, vaan yleisesti on kehitetty monia ad hoc -tyyppisiä luotettavuuden tai kehittämisprojektin virhemäärän estimointimenetelmiä. Suurin osa kehitetyistä menetelmistä on ollut käytössä vain vähän aikaa, eikä niiden toimivuuden tarkkuudesta

ole tarkkaa tietoa. Komponenttitehtaiden käytäntöjen pohjalla toimii Kytöahon (2003) diplomityö, jossa hän tutki, kuinka ohjelmistoprojektit arvioivat projektin alussa niissä esiintyviä virheitä ja niiden ajoittumista. Tutkimusta varten hän kartoitti yrityksen komponenttitehtaiden käytäntöjä. Menetelmiä voidaan pitää luotettavuutta arvioivina, koska niiden avulla voidaan verrata, kuinka kehittämisprojektista löytyneet virheet käyttäytyvät samantyyppisiin projekteihin verrattuna. Tähän lukuun on kerätty sekä tutkijan että Kytöahon kartoittamien menetelmien perusolettamuksia ja ne käydään lyhyesti läpi.

6.3.1 Virhetietokannat

Kohdeyrityksessä on yleisesti käytössä virhetietokannat, joihin ohjelmistoprojektissa esiintyneitä virheitä kirjataan. Virheet kirjataan kantaan jokaisessa kehittämisprojektissa erikseen sovittujen avainsanojen mukaisesti. Projektin virheiden jakautuminen avainsanojen suhteen saadaan tietokannasta tarkempaan analysointiin. Tietoja analysoimalla voidaan hallita projektia, mutta estimaatteja niiden avulla on vaikea tehdä. Suurin osa kohdeyrityksen käytössä olevista luotettavuusmenetelmistä perustuu juuri tietokantoihin tallennettuihin virheisiin. Nimittäin osa yksiköistä käyttää apunaan aikaisempia projektejaan ja niiden virhedataa. Esimerkiksi kyselyn mukaan eräs yksikkö selvittää uutta ohjelmistoprojektia käynnistettäessä projektin luonteen. Kehitetäänkö siinä paljon uusia ominaisuuksia vai käytetäänkö vanhoja komponentteja hyväksi? Tämän jälkeen lasketaan aiemmista vastaavantyyppisistä projekteista virheiden esiintymisen keskiarvo eri etappien välillä. Keskiarvojen avulla saadaan arvioitua käynnistyvän ohjelmistoprojektin virheiden määrä ja niiden ajoittuminen projektin suhteen.

6.3.2 Tositestaaminen

Kohdeyrityksessä käytetään paljon, erityisesti tuoteprojektien yhteydessä, ns. tositestaamista (*TRUE testing*). Idea on sama kuin yleisesti ohjelmistomaailmassa tunnetussa beetatestauksessa. Eli kehityksessä olevia tuotteita annetaan yrityksen työntekijöiden käyttöön ja he raportoivat käytössä havaituista virheistä sekä tekevät

parannusehdotuksia. Tarkoituksena on näin saada ennusteita todellisten käyttäjien reaktioista ja tuotteen laadusta verrattuna muihin yrityksen tuotteisiin. Tositestaamisessa testataan myös käyttöoppaan toimivuutta ja ongelma-alueita, mitä siihen pitäisi käyttäjää helpottamaan lisätä.

6.3.3 Nopea markkinapalaute

Haastattelun mukaan eräässä liiketoimintayksikössä käytetään nopeaa markkinapalautetta (*Fast field feedback*) apuna luotettavuuden estimointiin. Yksikkö kehittää älypuhelinsovelluksia. Ongelmana on, että kun matkapuhelin palautetaan huoltoon ohjelmistovian takia, ei vikoja huoltoliikkeissä erotella tarkemmin, vaan vika merkitään yleisesti ohjelmistoviaksi. Tämän takia yksikkö ei saa palautetta, onko vika ilmaantunut sen kehittämässä sovelluksessa vai jossain muussa. Nopeassa markkinapalautteessa tarkkaillaan ensimmäisiä käyttäjien takuuhuoltoon palauttamia puhelimia. Huoltoliikkeet lähettävät sovitun määrän palautetuista puhelimista kehittämisprojektille tarkempaan analyysiin. Analyysissa selvitetään kuinka suuri osuus prosentuaalisesti ohjelmistovikojen takia palautetuista puhelimista johtuu yksikön kehittämistä sovelluksista. Prosentuaalisen osuuden avulla voidaan estimoida kuinka moni matkapuhelin koko tuotantovolyymista tullaan ainakin kerran palauttamaan huoltoon yksikön kehittämän sovelluksen takia. Lisäksi vikojen syyt selvitetään ja korjataan seuraavaan ohjelmistojulkaisuun. Näin saadaan pahimmat viat korjattua ennen suurimpien tuote-erien valmistusta.

6.3.4 Komponenttitehtaiden käytännöt

Osassa komponenttitehtaista on Kytöahon (2003, 39) mukaan käytössä menetelmiä, jotka pohjautuvat koko- ja kompleksisuusmittoihin. Hänen mukaansa eräässä komponenttitehtaassa virhemäärien estimointi pohjautuu täysin koodin kokoon. Menetelmä pohjautuu laskelmiin, että tuhatta koodiriviä kohti pääsee komponenttitehtaan tuottamaan koodiin keskimäärin kolme virhettä. Laskennan pohjana on käytetty komponenttitehtaan aiempien projektien keskiarvoa. Toisessa komponenttitehtaassa virhemäärien arviointiin käytetään Kytöahon mukaan

konkreettisia kaavoja, jotka pohjautuvat koodin kokoon ja kehitysaikaan. Kytöahon tietojen mukaan menetelmää on testattu vain yhdessä ohjelmistoprojektissa, jossa se antoi kohtuullisen tarkan arvion. Kolmannessa Kytöahon kartoittamassa komponenttitehtaassa käytettiin hyväksi toimintopisteiden määrää koodissa. Toimintopiste koostuu käytettävästä ohjelmointikielestä riippuen tietystä rivimäärästä koodia ja yhdessä toimintopisteessä on 5 virhettä. Menetelmä on siis käytännössä samankaltainen kuin pelkkään koodin kokoon perustava menetelmä. Oikeaoppisessa toimintopisteanalyysissä toimintopisteiden määrään vaikuttaa Jonesin (1996, 50) mukaan ohjelmiston syötteet, tulosteet, loogiset tiedostot, tehdyt kyselyt ja liittymät.

Tässä luvussa kuvattiin alan kirjallisuudessa ja monien suurien yritysten käytössä olevien luotettavuusmallien sekä kohdeyrityksen sisäisten käytäntöjen perusteita. Seuraavassa luvussa arvioidaan menetelmien vahvuuksia ja heikkouksia.

7 MENETELMIEN ARVIOINTI JA JATKOTOIMENPITEET

Tässä luvussa arvioidaan edellisessä luvussa esitettyjä luotettavuusmalleja ja kartoituksessa ilmenneiden kohdeyrityksen käytäntöjen perusteita. Tämän jälkeen annetaan suositus kohdeyksikön jatkotoimenpiteiksi. Lopuksi käydään tutkimuksen johtopäätökset läpi.

7.1 Menetelmien arviointi

Arviointi painottuu tieteellisestä kirjallisuudesta valittujen luotettavuusmallien analysointiin viitekehyksen avulla. Kohdeyrityksen sisäisessä käytössä olevat menetelmät olivat vielä niin kehittymättömiä, että niiden perusteellinen arviointi on vaikeaa. Niiden perusolettamuksia kuitenkin tarkastellaan lyhyesti.

7.1.1 Kirjallisuuden luotettavuusmallit

Kirjallisuudesta valittujen luotettavuusmallien arviointi perustuu kappaleessa 5.2.2 kehitetyn viitekehyksen käyttöön. Oheisessa taulukossa (TAULUKKO 9) on koottu mallien tarkasteltavat ominaisuudet samaan taulukkoon vertailun helpottamiseksi. Viitekehyksen avulla saadut tulokset käydään seuraavaksi mallikohtaisesti lävitse.

SRE

SRE on tilastollinen vikamalli, joka käyttää hyväkseen kvantitatiivista informaatiota. Mallin pääideana on testata eniten käytetyt toiminnot. Niiden toimiessa moitteettomasti pysyy käyttäjä tyytyväisenä. Tosin tärkeimmät toiminnot, kuten esimerkiksi matkapuhelimessa hälytysnumeroon soitto, täytyy toimia ehdottomalla varmuudella silloin kun niitä käytetään. SRE tukee testaajien ja ohjelmistokehittäjien vahvaa yhteistyötä. Mallin avulla tiedetään milloin ohjelmisto on saavuttanut sille asetetun laatutavoitteen. Laatutavoitteen mittana pidetään virheiden esiintymisen aikaväliä. Lisäksi mallin avulla voidaan määritellä häiriöiden vakavuusasteet ja intensiteettitavoitteet. Parametreja ei paljoa vaadita ja ne on helppo arvioida. Malli

omaa laajan referenssilistan ja se soveltuu hyvin kaikkiin ohjelmiston kehittämisprosesseihin, kunhan testaus on tarpeeksi asiantuntevalla tasolla. SRE:n vahvana puolena on juuri testaajien ja ohjelmistokehittäjien tehokas yhteistyö. Mallin tulokset ovat helposti ymmärrettävissä ilman perehdytystä sen perusteisiin. Tämän takia SRE soveltuu hyvin johdon päätöksenteon tueksi. Malli ei käytä hyväkseen kohdeyksikön jo keräämiä metriikoita. Käyttöönottaminen vaatii normaalien asennuksen ja kouluttamisen ohella automaattisten testaustyökalujen hankintaa. Testaustyökalujen konfigurointi vaatii aikaa ja niiden käyttämiseen tulee kiinnittää henkilöstöä.

TAULUKKO 9. Viitekehyksen avulla analysoidut luotettavuusmallit

Komponentti / Malli	**SRE**	**Bayes-verkot**	**ODC**
Perusolettamus	Tilastollinen vikamalli, mikä painottaa testaajien ja ohjelmistokehittäjien vahvaa yhteistyötä. Käyttää hyväksi kvantitatiivista tietoa	Kausaalinen analyysi, jonka pohjana on vanha todennäköisyysteoria. Luotettavuuden määrittelevät muuttujat, jotka vaikuttavat toisiinsa	Käyttää sekä tilastollisten vikamallien että kausaalisten analyysien ominaisuuksia. Perustana virheiden tarkka kirjaaminen ja analysoiminen
Ohjelmistovirheiden estimointi	Estimoi tulevaisuuden häiriöiden aikavälin MTTF (Mean Time To Failure)	Estimoi pohjatietojen pohjalta ohjelmiston jäännösvirheiden määrän	Analysoi kirjattujen virheiden perusteella ohjelmiston kypsyyden
Kykenevyys	Virheiden aikaväli (MTTF), häiriöiden vakavuusaste, häiriöiden intensiteettitavoite (FIO)	Jäännösvirheiden määrä, jäännösvirheiden virhetaajuus	Ohjelmiston kypsyysaste, pohjainformaatio asiakastyytyväisyydelle, nopea palaute ohjelmistokehittäjille
Mallin helppokäyttöisyys	Parametreja ei paljoa ja niiden arviointi loogista	Bayes-verkon muuttujien todennäköisyydet vaikea määritellä. Verkon toimiessa parametrien määrittäminen ja arviointi todella helppoa	Parametreja on paljon ja ongelmana on niiden kirjaamisen yhdenmukaisuus
Soveltuvuus	Vaatii hyvän testauksen asiantuntemuksen	Laajoissa ohjelmiston kehittämisprosesseissa muuttujien suuren määrän takia toiminta kankeaa	Toimii parhaiten laajoissa ohjelmiston kehittämisprosesseissa
Yksinkertaisuus	Tuloksia helppo ymmärtää ja tulkita	Tuloksia helppo ymmärtää ja tulkita	Tulosten ymmärtäminen ja tulkinta vaikeaa. Vaatimuksena hyvä perehtyminen menetelmään
Johdon päätöksenteon tukeminen	Kertoo milloin ohjelmisto on saavuttanut kehittämisprosessin alussa sille asetetun luotettavuuden	Määrittelee ohjelmiston tietyn hetken jäännösvirheiden määrän. Ei estimoi tulevaisuuden jäännösvirheitä, jos kehittämisprosessia jatketaan	Määrittelee ohjelmiston tietyn hetken kypsyysasteen. Voidaan estimoida milloin ohjelmisto on laadullisesti valmis julkaistavaksi
Kerättävien metriikoiden käyttö	Ei käytä	Voidaan hyödyntää Bayes-verkon rakentamisvaiheessa, kun muuttujien välisiä todennäköisyyksiä määritellään	Vaatisi nyt kerättävien metriikoiden tarkemman jaottelun ja lisäksi keräämistä tulisi täydentää muilla metriikoilla
Vaaditut resurssit	Mahdollisten testauksen automaatiotyökalujen hankkiminen	Ohjelmistotyökalun hankinta ja toimivan verkkotopologian rakentamiseen kuluva aika	Kun menetelmä on sisäistetty (vie todennäköisesti kauan), kuluttaa se vähän resursseja

Bayes-verkot

Bayes-verkot pohjautuvat matemaattisiin todennäköisyysteorioihin ja tarvitsevat siksi avukseen ohjelmistotyökalun hankkimisen. Malli käyttää apunaan metriikoiden tuottamaa informaatiota, kun muuttujien todennäköisyyksiä määritellään. Täten kohdeyrityksen jo kerättyjä metriikoita hyödynnettäisiin. Fentonin ja Neilin (2000, 368) mielestä parhaana tulevaisuuden vaihtoehtona on juuri rakentaa ja käyttää yksinkertaisiin olemassaoleviin metriikoihin perustuvia kausaalisia menetelmiä. Toimivan Bayes-verkon rakentaminen on mallin tarkkuuden kannalta oleellisinta. Rakentaminen vaatii aikaa ja muuttujien välisten todennäköisyyksien määrittäminen sekä kokemusta että taitoa analysoida kerättyjä metriikoita. Kun Bayes-verkko saadaan toimintaan, estimoi se kehitettävän ohjelmiston jäännösvirheiden määrän sekä jäännösvirheiden taajuuden. Sen käyttäminen ja tulosten tulkitseminen on helppoa. Johto voi menetelmän avulla tarkkailla ohjelmiston estimoitujen jäännösvirheiden määrää ja päättää sen perusteella milloin tietty kypsyysaste julkaisua varten on saavutettu. Bayes-verkkojen käyttäminen soveltuu hyvin muihin kuin laajoihin ohjelmiston kehittämisprosesseihin. Niissä muuttujien suuren määrän takia todennäköisyyksien määrittäminen on hankalaa ja samalla inhimillisten virheiden todennäköisyys kasvaa. Bayes-verkot tulevat kehittymään ja ne voivat olla tulevaisuudessa tarkkoja mallintamisvälineitä.

ODC

ODC on hyvin teoreettinen malli. Sen valjastaminen tehokkaaseen käyttöön vaatii suuria ajallisia resursseja. Ensiksi vikojen kirjaaminen tulee kaikkien työntekijöiden sisäistää ja pyrkiä suorittamaan samalla tavalla. Vikainformaation perusteella tehtävä ohjelmiston luotettavuuden analysointi vaatii kokemusta ja aikaa. Malli voi tarjota oikein käytettynä paljon erilaista kehitettävää ohjelmistoa tukevaa informaatiota, kuten pohjatietoa asiakastyytyväisyydelle ja nopeaa palautetta ohjelmistokehittäjille. Rahalliset kustannukset mallissa ovat pienet. Tosin Chillaregen ym. (1992) mukaan ODC:n käynnistämiseksi joudutaan varautumaan alkukustannuksiin, jotka sisältävät koulutuksen, työkalujen ja prosessien vaihtamisen. He toteavat myös, että kun malli on saatu toimintaan ovat ohjelmistokehittäjäkohtaiset kustannukset minimaaliset. Yksi mallin heikkous on vikojen luokittelun inhimillinen prosessi. Ihmiset tulkitsevat viat eri

tavalla ja näin saattaa syntyä näkemyseroja. Tosin jos vikaluokkien jaottelu on ortogonaalinen, tulisi vikojen jaottelun olla helppoa. Suuriin ohjelmistojen kehittämisprosesseihin malli sopii hyvin, koska niissä on yleensä nimetty erikseen pelkästään luotettavuusmallin parissa työskentelevät ihmiset. Etuna tästä on vikojen yhdenmukaisempi kirjaaminen. ODC:n tarkoituksena on tarkasti kirjattujen virheiden perusteella arvioida kehitettävän ohjelmiston kypsyys. Kypsyysasteen perusteella voi johto päätellä, onko ohjelmisto valmis julkaistavaksi, tai estimoida milloin se olisi. Kohdeyksikön tulisi ODC:ta varten jaotella tarkemmin kerättäviä vikametriikoita sekä lisättävä huomattavasti kerättävien metriikoiden määrää.

Kanin (2003, 268) mielestä ODC:n merkittävin panos näyttää olevan testauksen tehokkuuden lisääminen suuren tietomäärän avulla. Hän suositteleekin yritysten tekevän syvällisiä syy-seuraus- ja vikatyyppi-analyyseja integroituna osana prosessimetriikoita. Kan muistuttaa, että edellytyksenä tähän on suuret resurssit.

7.1.2 Kohdeyrityksen käytössä olevat menetelmät

Kohdeyrityksen sisäiset menetelmät ovat vielä sen verran nuoria ja kehittymättömiä, että niitä ei voi viitekehyksen avulla analysoida. Tässä arvioinnin pohjana toimivat menetelmien perusolettamukset.

Virhetietokannat

Suurin heikkous virhemäärien ennustamisessa on, ettei virheiden lukumäärä kerro välttämättä, kuinka luotettava ja vakaa tuote on. Tuote voi toimia käytännössä luotettavasti, vaikka siinä olisi jäännösvirheitä paljon. Tässä toimii kohdassa 4.3 esitellyn Pareto-analyysin periaate: pieni osuus kaikista virheistä saa aikaan lähes kaikki havaitut toimintavirheet. Lisäksi ongelmana on virheiden syöttäminen. Monen eri henkilön syöttäessä virheitä kantaan ei voida saavuttaa yhdenmukaisuutta. Virheet syötetään kantaan avainsanojen perusteella ja eri henkilöt kokevat avainsanat erilailla. Virhetietokantojen avainsanoiksi kannattaisi valita ODC:n ortogonaaliset vikatyypit, koska ne ovat kuvaavia ja ei-päällekkäisiä. Hyvänä puolena virhetietokantojen käytössä on, että niiden avulla saadaan seuraavia vastaavanlaisia projekteja varten aikaisempien

historiatiedot avuksi. Kanin (2003, 279) mukaan kannattaa aina käyttää aikaisempien vastaavankaltaisten projektien historiatietoja apuna, kun uuden projektin estimaatteja rakennetaan.

Tositestaaminen

Mielestäni tositestaamisen idea on hyvä, sillä sen avulla varmistetaan tuotteen toimiminen moitteettomasti käyttäjien parissa. Tämä on sen ansioita, että testaamisen suorittavat koneiden sijaan ihmiset. Tästä aiheutuu kumminkin kaksi ongelmaa. Ensinnäkin testauksen suorittavat ihmiset eivät ole todellisia käyttäjiä, vaan teknisen koulutuksen omaavia henkilöitä. Siksi he osaavat käyttää vaikeitakin teknisiä tuotteita, vaikka ne eivät olisi lähelläkään helppokäyttöisyyttä. Toiseksi testaamiseen kuluva aika on todellista aikaa, eikä automaattisen testauksen virtuaaliaikaa. Tositestaaminen onkin todella hidasta ja kuluttaa työntekijöiden resursseja, mikä tekee siitä kalliin. Menetelmä pohjautuu tositestaajien hyvin laadittujen raporttien tarkkaan analysoimiseen. Tositestaamisen suurin ongelma on sen myöhäinen vaihe ohjelmiston kehittämisessä. Testaamisessa voi ilmetä vikoja joihin olisi pitänyt reagoida paljon aikaisemmin kehittämisprosessissa ja joihin ei enää voida paljoa vaikuttaa ellei kehittämisprosessin aikatauluja muokata.

Nopea markkinapalaute

Nopeaan markkinapalautteeseen pohjautuvan menetelmän kehittäjä tiedosti haastattelussa menetelmän tämän hetken huonot puolet, jotka ovat menetelmän sijoittuminen kehittämisprojektin myöhäiseen vaiheeseen ja, että pienen määrän perusteella estimoidaan suurta tuotantovolyymia. Menetelmän myöhäisen ajankohdan takia ohjelmistoon ei pystytä enää tekemään suuria muutoksia, vaikka sellaisia saattaisi menetelmää käyttäessä ilmetä. Siksi kehittämisprojektin aikana tulee käyttää joitain muita menetelmiä varmistamaan luotettavuuden saavuttaminen. Toisena ongelmana nopeassa markkinapalautteessa on, että pieni määrä palautettuja puhelimia edustaa todella suurta volyymia. Periaatteessa olettamus on oikea, sillä O´Connorin (1995, 249) mukaan, kun ohjelmistovirhe esiintyy, se esiintyy kaikissa ohjelmiston kopioissa, ja jos se aiheuttaa vian tietyissä olosuhteissa, niin vika aiheutuu aina näissä olosuhteissa.

Ongelmaksi vain muodostuu onko kyseinen pieni otanta tarpeeksi edustava suuresta volyymista.

Komponenttitehtaiden käytännöt

Komponenttitehtaiden estimointikäytännöt liittyvät kaikki koko- ja kompleksisuusmetriikoihin. Tämä ei ole yllätys, nimittäin Fentonin ja Neilin (1999) mukaan suurin osa virheiden ennustamiseen liittyvistä tutkimuksista pohjautuu juuri koko- ja kompleksisuusmetriikoihin. Heidän mukaansa suurin osa koko- ja kompleksisuusmalleista olettaa suoraviivaista suhdetta vikojen syntymiseen, vaikka moni tutkimus todistaa suurta korrelaatiota suunnittelun kompleksisuuden ja vikojen esiintymisen välillä. Ohjelmoijat ja suunnittelijat yhdessä luovat virheet järjestelmään ja kokoon perustuvat menetelmät ottavat vain ohjelmoijien panoksen huomioon.

Komponenttitehtaat ovat käyttäneet menetelmien kehittämiseen aikaisempien kehittämisprojektien historiatietoja apunaan. Estimointimenetelmien toimivuuden oletuksena on aiempien projektien suuri määrä ja niiden samankaltaisuus. Näin saadaan tarkempi keskiarvoon perustuva estimaatti. Koko- ja komponenttimetriikat eivät sovellu kohdeyksikön käyttöön, sillä sen kehittämisprojektit ovat liian erilaisia toisiinsa verrattuina. Lisäksi kohdeyksiköllä ei ole historiatietoja tarjolla.

7.2 Suositus kohdeyksikön jatkotoimenpiteiksi

Mielestäni kohdeyksikön ei kannata lähteä integroimaan mitään kolmesta tarkemmin tarkastellusta luotettavuusmallista. Varsinkin SRE:n ja ODC:n integroiminen muuttaisi suuresti yksikön tämänhetkisiä käytäntöjä. Haastattelun mukaan SRE:tä on juuri kohdeyrityksessä eräässä tuoteprojektissa kokeiltu ja sen tuloksia odotellaan. Kokeilu oli ollut kumminkin työläs, ja elleivät tulokset ole erittäin hyviä, ovat hyödyt verrattuna kustannuksiin pieniä. ODC:n käyttö sopii enemmän projekteihin, joissa tuotetaan vain pelkkää koodia. Esimerkiksi kohdeyrityksessä ODC:n käyttö soveltuu parhaiten komponenttitehtaiden käytettäväksi. Bayes-verkkojen käyttäminen ei vaadi suuria muutoksia. Niiden kehittymistä tulee tarkkailla ja mahdollisesti kokeilla tulevaisuudessa niiden kykyä ohjelmistotyökalujen ilmaisilla kokeiluversioilla. Kokeiluversioilla voisi

ajaa ensimmäisten päättyvien kohdeyksikön projektien historiadataa ja vertailla saatuja tuloksia toteutuneisiin.

Myöskään kohdeyrityksen käytössä olevia menetelmiä en suosittele integroitaviksi. Menetelmät tuntuvat kaikki perustuvan niitä käyttävien yksiköiden historiatietoihin, ja niiden soveltuvuus muualle on näin heikko. Tämä ei ole yllättävää, sillä Neil ja Fenton (1996) kirjoittivat ohjelmistopäälliköiden tekevän yleisesti laatu- ja kustannuspäätökset ennakkoarveluiden perusteella. Arvailuiden pohjana toimivat kokemus ja historiatiedot. Neil ja Fenton toteavat, etteivät tutkijat voi muuta tehdä kuin tunnustaa tosiseikan ja parantaa ’arvailu’ -prosessia. Kohdeyksikölle voi näin olla parasta odottaa kokemuksen ja historiatietojen tuomaa luotettavuuden mallintamiskykyä.

Sitä ennen kohdeyksikkö voi parantaa nykyisiä käytäntöjään ottamalla vaikutteita tutkimuksessa esitetyistä malleista. Seuraavaksi esittelen oman kolmen kohdan parantamisehdotukseni:

1. Kehittämisprojektin alussa määritellään SRE:ssä käytettävät toiminnalliset profiilit.
2. Muutetaan virhetietokantaan vikojen kirjaamista tarkemmaksi. Lisätään kirjauksen yhteyteen vikatyyppikenttä, joka sisältää ODC:n vikatyypit.
3. Käytetään tositestaamista tuotteen kehittämisprosessin loppuvaiheessa.

Toiminnallisen profiilin luominen on osa SRE-mallia. Kuten Everett, Keene ja Nikora (1998) toteavat, toiminnallinen profiili kuvaa käyttäjän suorittamat toiminnot kehitettävällä tuotteella ja kuinka usein näitä toimintoja suoritetaan. Käytännössä tuotteen käyttö jaetaan oletettuihin prosentuaalisiin arvoihin. Everettin, Keenen ja Nikoran mukaan tuotteen testaamisen täytyy jäljitellä tulevien käyttäjien käyttöä. Eli jos arvioidaan, että esimerkiksi 40 prosenttia matkapuhelimen koko käytöstä kuluu soittamiseen, määritellään soittaminen toiminnalliseksi profiiliksi ja sen esiintymistodennäköisyydeksi 0.4. Testausta aloitettaessa voidaan määrätyt testausresurssit jakaa helposti toiminnallisten profiilien avulla. Esimerkiksi testaukseen käytettävästä 500 testitapauksesta, tulee käyttää 200 soittamisen testaamiseen (500 x 0.4 = 200). Musan (2003) mukaan näin taataan asiakastyytyväisyys, sillä asiakkaiden eniten käyttämät ominaisuudet on oikeassa suhteessa testattu. Hän kumminkin muistuttaa

kaikkien kriittisimpien ominaisuuksien tarkan testaamisen, vaikka ominaisuuden käyttötaajuus olisi todella pieni. Esimerkiksi matkapuhelimessa tällainen ominaisuus on hätänumeroon soittaminen.

Tällä hetkellä kohdeyksikön projektit kirjaavat projektissa ilmenneitä vikoja virhetietokantaan. Virheiden kirjaaminen suoritetaan projektin alussa päätettyjen avainsanojen mukaan. Yleisesti avainsanat kuvaavat havaitun vian tilaa: havaittu, tunnistettu, tutkittavana, korjattu, julkistettu, todennettu, suljettu, syrjäytetty tai siirretty. Kun virhe on korjattu ja se kirjataan tietokantaan korjatuksi, voisi samassa yhteydessä kirjata korjatun virheen vikatyypin. Vikatyyppeinä toimisivat ODC:n seitsemän ortogonaalista vikatyyppiä. Näin saataisiin helposti virhetietokannasta analysoitavaksi virheiden jakautuminen vikatyypeittäin. Esimerkiksi KUVIO 17:sta (s. 72) avulla voidaan Chillaregen ym. (1992) mukaan päätellä, missä vaiheessa kehittämisprosessi on. Jos kehittämisprojekti on etenemässä järjestelmätestaukseen, mutta vikatyyppien jakauma osoittaa enemmän yksikkö- tai integrointitestaukseen viittaavaan, tiedetään etenemisen olevan ennenaikaista. Lisäksi TAULUKKO 7:n (s. 71) avulla voidaan päätellä, jos tietyn tyyppisiä virheitä löytyy paljon, missä kehittämisvaiheessa on jotain vialla. Vikatyyppien kirjaaminen ei rasita paljoa tämänhetkistä prosessia ja sen avulla saataisiin arvokasta analysointitietoa. Ongelmana monissa menetelmissä on liian yksinkertaisen tiedon analysointi. Neil ja Fenton (1996) vertaavat tilannetta makrotalouteen, jossa talouden ennusteita ei voitaisi tehdä ilman kompleksisia muuttujia, kuten säästämisaste tai tuottavuus. ODC:n vikatyyppien avulla saadaan enemmän tärkeää informaatiota estimaattien tekemiseen.

Kun kehittämisprosessi on lopuillaan ja tuote on saatu suhteellisen toimivaksi, voidaan käyttää luotettavuuden varmistamiseksi tositestaamista. Näin saataisiin varmistettua kehitettävän sovelluksen yhteentoimivuus muiden matkapuhelinsovellusten kanssa. Kehitettävää sovellusta jaettaisiin kohdeyksikön työntekijöiden käyttöön ja he raportoisivat havaituista virheistä ja parannusehdotuksista. Sovellusta käytettäisiin tositestaamisen ansiosta erilaisissa toimintaympäristöissä ja muiden sovellusten kera. Yllättäviä vikoja voidaan näin havaita.

Kohdeyksikön tulee panostaa tässä tutkimuksessa läpikäytyjen laatujohtamisen menetelmien käyttöön. Etenkin katselmointien, tarkastusten ja metriikoiden oikealla käytöllä päästään lähelle optimitavoitetta – virheetöntä tuotetta. Esimerkiksi Chillarege (1994) toteaa, että oikeasti suurin osa ohjelmiston kehittämisprosessin vioista löydetään suorittamatta itse koodia. Hänen ja Jonesin (1996, 356) mukaan pelkillä koodin tarkastuksilla löydetään 40-60 prosenttia kaikista vioista. Myös Fenton ja Neil (1999) ovat sitä mieltä, että ohjelmiston kehittämisprojektin laatua voidaan huomattavasti parantaa oikeanlaisten menetelmien ja käytänteiden avulla.

7.3 Johtopäätökset

Tutkimukseen valittiin kolme laajassa käytössä olevaa luotettavuusmallia lähempään tarkasteluun. Tarkastelu rajoittui teoreettiselle pohjalle tutkimuksen aikarajoitteen takia. Analyysi olisi ollut tarkempi ja tulokset havainnollisempia, jos jokaista luotettavuusmallia olisi käytetty samalla testidatalla ja mallien antamia tuloksia analysoitu. Analysoitavina malleina olivat SRE, Bayes-verkot ja ODC. Mitään luotettavuusmalleista ei valittu kohdeyksikölle integroitavaksi, koska niiden vaatimat resurssit eivät osoittautuneet verrannollisiksi mahdollisiin hyötyihin. Yleisenä ongelmana tämän hetken malleissa on juuri niiden tuomien hyötyjen vähäinen määrä verrattuna kustannuksiin. Luotettavuusmalleja on kehitetty yli 30 vuotta ja kehitys jatkuu vielä. Voi olla mahdollista, ettei koskaan tulla kehittämään tarkkaa mallia, ei ainakaan sellaista, mikä soveltuisi yleiseen käyttöön. Organisaatioiden käytössä malleja voidaan saada kehitettyä suhteellisen tarkoiksi historiatietoja käyttämällä.

Tutkimuksen kohdeyksikön jatkotoimenpiteiksi suositeltiin analysoitujen luotettavuusmallien tiettyjen käytänteiden soveltamista. Käytänteiden tuoma hyöty verrattuna niiden vaivattomaan käyttöönottamiseen on mielestäni hyvä. Fenton ja Neil (1999) toteavat, että ohjelmiston kehittämisessä laatua voidaan huomattavasti parantaa oikeanlaisten projektikäytäntöjen ja hyvien suunnittelumenetelmien avulla. Ehkä oikeanlainen suuntaus onkin kehittää laatumenetelmiä paremmiksi ja luottaa kehitettävän tuotteen laatutavoitteen saavuttamiseen sen sijaan, että laatua pyrittäisiin ennustamaan. Tutkimuksessa esitettiin laatujohtamisen menetelmistä jatkuva

kehittyminen, ISO 9000 -standardi, katselmoinnit ja metriikat. Näiden menetelmien käyttäminen soveltuu ohjelmistokehitykseen.

Ohjelmiston luotettavuusmalleja on kehitetty paljon, mutta kirjallisuudesta ei löydy teoksia, joissa malleja vertailtaisiin keskenään. Tässä tutkimuksessa vertailtiin kolmea laajalle levinnyttä mallia. Vertailun avuksi kehitettiin viitekehys. Viitekehys analysoi luotettavuusmallien ominaisuuksia teoreettiselta kannalta. Vastaavanlaista viitekehystä en ole alan kirjallisuudesta löytänyt. Kehitetyn viitekehyksen käyttäminen soveltuu eri toimintaympäristöihin ja siksi sitä voidaan soveltaa tulevaisuudessakin. Luotettavuusmallien periaatteita tiivistäviä teoksia löytyy myös vähän kirjallisuudessa. Yleisesti keskitytään vain yhden mallin laajamittaiseen esittelyyn ja analysointiin. Tutkimuksessa tiivistettiin kolmen suositun luotettavuusmallin periaatteet. Tiivistelmien avulla voidaan nopeammin päätellä, soveltuuko malli tiettyyn toimintaympäristöön.

Kohdeyrityksen suurin hyöty tutkimuksesta oli sen sisäisten luotettavuusmallien kartoitus. Itse luotettavuusmalleja ei yrityksessä ollut paljoa käytössä, mutta luotettavuuden parantamiseen tähtääviä menetelmiä joitakin. Menetelmät olivat kaikki olleet vasta vähän aikaa käytössä, joten niiden toiminnasta ja tarkkuudesta ei ollut tarjolla validia tietoa. Tämän vuoksi niiden arviointiin ei kehitetty viitekehys soveltunut. Menetelmien perusteet käytiin tutkimuksessa läpi ja tyydyttiin niiden perusolettamuksien arvioitiin. Suurin osa menetelmistä oli räätälöity tietyn yksikön käyttöön, eivätkä ne sen vuoksi ole yleistettävissä. Kartoitus antoi kuvan, mikä on kohdeyrityksen tilanne tutkimusalueella ja minkälaisten menetelmien käyttöä on sovellettu.

Tutkimuksessa ilmeni, että moni ohjelmiston luotettavuusmalli on vahvassa yhteydessä testaamiseen. Fentonin ja Neilin (1999) mielestä jokaisen vikojen määrää ennustavan mallin tulee ottaa testaus kriittisenä tekijänä huomioon. Juuri testausta parantavien ja sen avulla estimaatteja tekevien mallien tutkiminen olisi hyvä jatkotutkimusaihe. Tutkimusalue aiheessa on yhtä pirstaleinen kuin tässä tutkimuksessa, eli monia erilaisia malleja on kehitetty, mutta mikään niistä ei ole saavuttanut de facto -asemaa. Esimerkiksi Kanin (2003, 271) mukaan testausmetriikoita hyödyntäviä malleja on

paljon ja uusia esitellään jatkuvasti. Näistä harvoja on hänen mukaansa käytetty oikeassa ympäristössä ja tiedot niiden hyödyllisyydestä ovat siksi puutteellisia. Tämän tutkimuksen tuloksena saatiin nivottua yhteen koko ohjelmiston kehittämisprosessille jakautuvien luotettavuusmallien tämänhetkinen tilanne. Jatkotutkimuksen avulla saataisiin vielä testausmenetelmien trendi selville ja näin saataisiin yhteenveto kaikista potentiaalisista ohjelmiston luotettavuusmalleista.

Tutkimuksen ongelmana oli aihealueen pirstaleisuus. Monia erilaisia malleja on kehitetty, mutta niiden perusteita kokoavia tai itse malleja arvioivia teoksia on vähän. Ongelmaksi muodostui löytää parhaat mahdolliset luotettavuusmallit tarkempaan analyysiin. Aihealuetta oli siis vaikea lähestyä ja rajata. Tämän tutkimuksen perusteella ymmärretään miten ohjelmiston luotettavuutta mallinnetaan, saadaan kokonaiskuva kolmen mallin toiminnasta ja niiden toiminnan tarkkuudesta.

LÄHTEET

Bassin K.A., Kratschmer T., Santhanam P. 1998. Evaluating Software Development Objectively. IEEE Software, Vol. 15, No. 6, 66-74.

Bellefeuille J. 1993. Total Quality Management. IEEE Spectrum, Vol. 30, Issue 9, 47-50.

Born G. 1994. Process Management to Quality Improvement. New York: John Wiley & Sons.

Breyfogle F. W. III. 1999. Implementing Six Sigma: Smarter solutions using Statistical Methods. New York: John Wiley & Sons.

Brocklehurst S. & Littlewood B. 1992. New ways to Get Accurate Reliability Measures. IEEE Software, Vol. 9, Issue 4, 34-42.

Chillarege R. 1994. ODC for Process Measurement, Analysis and Control. [Viitattu 01.10.2003]. Saatavilla www-osoitteessa <http://www.chillarege.com/odc/articles/asqc/asqc.html>

Chillarege R. 1996. What is Software Failure? IEEE Transactions on Reliability, Vol. 45, No. 3, 354-355.

Chillarege R. 2000. Orthogonal Defect Classification. Teoksessa ASM Conference Tutorial. SM/ASM 2000 San Jose, USA, March 6-10. New York: ASM.

Chillarege R. 2003 What is ODC? [Viitattu 23.10.2003]. Saatavilla www-osoitteessa <http://www.chillarege.com/odc/odcbackground.html>.

Chillarege R., Bhandari I.S., Chaar J.K., Halliday M.J., Moebus D.S., Ray B.K. & Wong M. 1992. Orthogonal Defect Classification – A Concept for In-Process Measurements. IEEE Transactions on Software Engineering, Vol. 18, No. 11, 943-956.

Darby G. E. 1999. Bridging wireless and wired networks: smart phone operating systems, IP convergence and market segmentation. Info, Vol. 1, No. 6, 563-576.

Deming W. E. 1987. Out of the crisis. Cambridge: MIT, Center of Advanced Engineering Study.

Evans-Pughe C. 2003. Microsoft and the mobile[phone]. IEE Review, Vol. 49, Issue 4, 18-19.

Everett W., Keene S. & Nikora A. 1998. Applying Software Reliability Engineering in the 1990s. IEEE Transactions on reliability, Vol. 47, No. 3, 372-378.

Fenton N.E. 2003. Welcome: Probability Theory and Bayesian Belief Nets [viitattu 04.11.2003]. Saatavilla www-osoitteessa <http://www.dcs.qmul.ac.uk/~norman/BBNs/BBNs.htm>.

Fenton N.E., Krause P. & Neil M. 2002. Software Measurement: Uncertainty and Causal Modeling. IEEE Software, Vol. 19, Issue 4, 116-122.

Fenton N.E. & Ohlsson N. 2000. Quantitative Analysis of Faults and Failures in a Complex Software System. IEEE Transactions on Software Engineering, Vol. 26, No. 8, 797-814.

Fenton N.E. & Neil M. 2000. Software Metrics: Roadmap. Teoksessa Finkelstein A.(toim.) Proceedings of the conference on The future of Software engineering Limerick, Ireland, June 4-11. New York: ACM Press, 357-370.

Fenton N.E. & Neil M. 1999. A Critique of Software Defect Prediction Models. IEEE Transactions on Software Engineering, Vol. 25, No. 5, 675-689.

Garlan D. 2000. Software Architecture: a Roadmap. Teoksessa Finkelstein A. (toim.) Proceedings of the conference on The future of the software engineering Limerick, Ireland, June 4-11. New York: ACM Press, 91-101

Haikala I. & Märijärvi J. 1996. Ohjelmistotuotanto. Espoo: Suomen ATK-kustannus Oy.

HCi 2003. PDCA Cycle. HCi Ltd. [viitattu 28.7.2003]. Saatavilla www-osoitteessa <http://www.hci.com.au/hcisite2/toolkit/pdcacycl.htm#Plan-Do-Check-Act>.

Hirsjärvi S., Remes P. & Sajavaara P. 1997. Tutki ja kirjoita. Tampere: Kirjayhtymä Oy.

Hugin Expert 2003. Users of Hugin Experts Products and Services [viitattu 04.11.2003]. Saatavilla www-osoitteessa <http://www.hugin.com/Products_Services/References/>.

IBM 2003. Orthogonal Defect Classification [viitattu 22.10.2003]. Saatavilla www-osoitteessa <http://www.research.ibm.com/softeng/ODC/ODC.HTM>.

Ireson W.G., Coombs Jr. C.F. & Moss R.Y. 1995. Handbook of reliability engineering and management – 2nd edition. New York: McGraw-Hill.

Jones C. 1996. Applied Software Measurement: assuring productivity and quality - 2nd edition. New York: McGraw - Hill.

Juran J. M. 1988. Juran’s Quality Control Handbook - Fourth Edition. New York: McGraw-Hill.

Järvinen P. & Järvinen A. 2000. Tutkimustyön metodeista. Tampere: Opinpajan kirja.

Kan S.H. 1991. Modeling and software development quality. IBM Systems Journal, Vol. 30, No 3, 351-362.

Kan S.H. 2003. Metrics and Models in Software Quality Engineering. New York: Addison-Wesley.

Kari J. & Kilpeläinen J. M. 2001. IP Convergence Based on SIP. Teoksessa Kivimäki J. (toim.). MITA: Mobile Internet Technical Architecture. Helsinki: IT Press, 287 - 300.

Karvosenoja T. 2000. Integration of GSM Data SW to Mobile Products. Tampere University of Technology. Department of Electrical Engineering. Master's Thesis.

Kytöaho J. 2003. Ohjelmistovirheiden estimointi matkapuhelinprojektissa. Oulun yliopisto, Sähkö- ja tietotekniikan osasto, diplomityö -tutkielma.

Lecklin O. 1999. Laatu yrityksen menestystekijänä. Jyväskylä: Gummerus Kirjapaino Oy.

Lyu M.R, Nikora A. 1992. Applying Reliability Models More Effectively. IEEE Software, Vol. 9, Issue 4, 43-52.

Musa J.D. 1997. Introduction to Software Reliability Engineering and Testing. Teoksessa Werner B. (toim.) Proceedings of the 8th International Symposim on Software Reliability Engineering - Case Studies - Albuquerque, New Mexico, November 2-5. Los Alamitos: IEEE Computer Society, 3-12.

Musa J.D. 1998. Software Reliability Engineering for Mobile Code. Teoksessa Werner B. (toim.) Proceedings of the 9th International Symposim on Software Reliability Engineering Paderborn, Germany, November 4-7. Los Alamitos: IEEE Computer Society, 182.

Musa J.D. 2003. More Reliable Software Faster and Cheaper: An Overview of Software Reliability Engineering. Software Reliability Engineering and Testing Courses. [Viitattu 26.9.2003]. Saatavilla www-osoitteessa <http://members.aol.com/JohnDMusa/ ARTweb.htm>.

Musa J.D, Iannino A. & Okumoto K. 1987. Software Reliability: Measurement, Prediction, Application. New York: McGraw-Hill.

Neil M. & Fenton N.E. 1996. Predicting Software Quality using Bayesian Belief Networks. Proceedings of 21st Annual Software Engineering Workshop, NASA/Goddard Space Flight Centre, December 4-5.

Nokia 2003a. Nokia Mobile Phones. Nokia Corporation [viitattu 17.7.2003]. Saatavilla www-osoitteessa <http://www.nokia.com/nokia/0,8764,33082,00.html>.

Nokia 2003b. ISA Mainstream Plan. Nokia Mobile Phones [viitattu 17.7.2003]. Luottamuksellinen sisäinen dokumentti.

Nokia 2003c. Nokia and Symbian OS. Nokia Corporation [viitattu 28.8.2003]. Saatavilla www-osoitteessa <http://www.nokia.com/downloads/aboutnokia/press/pdf/ symbian.pdf>.

Nokia 2003d. Tietoa Nokiasta. Nokia Corporation [viitattu 10.9.2003]. Saatavilla www-osoitteessa < http://nds2.ir.nokia.com/aboutnokia/downloads/brochures/pdf/ corporate_brochure/Yleisesite.pdf>.

Oakland J.S. 1993. Total Quality Management; The route to improving performance 2th Edition. Oxford: Butterworth Heinemann.

O´Connor P.D.T. 1995. Practical reliability engineering – 3rd edition revised. Chichester: John Wiley & Sons.

Paakkunainen M. 2000. Mobile Phone Software Architecture Development. Tampere university of technology, Department of Information Technology, Master's Thesis.

Pressman R.S. 2001. Software Engineering: a Practitioner's Approach 5th Edition. New York: McGraw-Hill.

Päivärinta S. & Luukka T. 2001. Mobile Terminals of the Future. Teoksessa Kivimäki J. (toim.). MITA: Mobile Internet Technical Architecture. Helsinki: IT Press, 271 – 275.

Sengodan S., Krishnamurthi G., Rajahalme J. & Ravikanth R. 2000. Impact of Different Cellurar-IP Convergence Strategies on Mobile Terminals. Teoksessa Gopal Gupta R. (toim.) IEEE International Conference on Personal Wireless Communications Hyderabad, India, December 17-20. Burlington: Nokia Research Center, 504-508.

Silén, T. 1998. Laatujohtaminen: menetelmiä kilpailukyvyn vahvistamiseksi. Porvoo: WSOY.

Steinbock D. 2001. The Nokia Revolution: The Story of an Extraordinary Company That Transformed an Industry. New York: AMACOM, American Management Association.

Symbian 2003a. About us. Symbian Ltd. [viitattu 28.8.2003]. Saatavilla www-osoitteessa <http://www.symbian.com/about/about.html>.

Symbian 2003b. Intel joins Symbian Platinum Partner Program and provides new Symbian software development kit. Symbian Ltd. [viitattu 28.8.2003]. Saatavilla www-osoitteessa <http://www.symbian.com/news/2003/pr030428.html>.

Viitala M. 2002. Settings Application in Symbian Operating System. Tampere University of Technology, Department of Information Technology, Master's Thesis.

Vonderembse M.A & White G.P. 1996 Operations Management: Concepts, Methods, and Strategies Third Edition. London: West Publishing Company.

Wood A. 1996. Predicting Software Reliability. IEEE Computer, Vol. 29, Issue 11, 69-77.

LIITE 1

SÄHKÖPOSTIKYSELYN RUNKO

Hi,

I'm working in XXX business program and I'm doing my masters thesis about **estimating software errors in software programs**. The main purpose of my research is to develop a method for XXX that estimates **end product errors**.
I'm trying to find out estimation techniques that have been used in NMP. I have bumped into your name when I have discussed about topic or searched material from Intranet. If you have no knowledge about this matter or you aren't the right person to answer, please forward this e-mail to person who knows about this issue.

Questions for you:

- **What kind of software error estimation methods, especially for end products, do you know?**
- **How software reliability is estimated in your unit or program?**
- **Do you use any kind of methods or do you only rely on figures from someone's head (history knowledge)?**
- **Do you think that these kinds of estimates are important for units or programs? Is achieved information good and trustworthy enough in comparison with the effort to get it?**

Responses can be given in English or Finnish. If you don't want to write answers via e-mail, you can call me or we can have a meeting too. **Please answer before week 40.**

More information about the topic:
XXX business program was established in conjunction with NMP's organizational change in spring 2002 and **our focus is on Symbian software**. Our unit acts as a combination of a product program and a SW Component Factory operating mode.

The main concern of my research is to **estimate errors that manage to get to end products**. Benefits with estimation are to get useful information and fast feedback on XXX's software, fast way to correct erroneous end applications, and measure the quality of XXX software in the field. With good estimation method we can perhaps find out how many phone repairs during the warranty period were caused by XXX software.

Figures on conventional quality metrics of NMP products the Field Failure Rate (FFR) and Batch Failure Rate (BFR) have been estimated during the whole product creation process. These figures comprise the whole product including HW. Are these figures good for SW too or is there a better method?

There aren't many scientific techniques for estimating software errors in end products. This research area is quite unexplored. On the other there are many methods how to estimate hardware reliability. With hardware you can make calculations when it probably will collapse or wear-out (like researchers say). Hardware errors don't concern all products. When a software error does exist, it exists in all copies of the program. Therefore software errors can be extremely serious.

I'm trying to find out estimation techniques that have been used in other NMP business units and programs. So far I have found out, for example, that at YYY they have developed a method what estimates the future YYY BFR. The method is based on the Fast Field Feedback (FFFB) analysis. The Fast Field Feedback is the analysis made for a batch of end user returned phones. The returned phones are analyzed in product program to get exact information. The purpose of the analysis is to find the reasons causing field returns from the first production batch as early as possible and to be able to act quickly and even before the volume of the product increases. The FFFB is used with BFR and volume values (both from Cognos) to calculate future YYY BFR. The estimated numbers are weighted against volume and that way results get closer to the real warranty cost.

That was one way to estimate end product errors. How things are done in your unit / program or do you know another method? Please response…

Thanks beforehand,

Anssi Takku